

ELIOT1 HAL

Создано системой Doxygen 1.11.0



1 Topic Index	1
1.1 Topics . . . . .	1
2 Алфавитный указатель структур данных	3
2.1 Структуры данных . . . . .	3
3 Список файлов	7
3.1 Файлы . . . . .	7
4 Topic Documentation	9
4.1 Драйвер модуля CAN . . . . .	9
4.1.1 Подробное описание . . . . .	13
4.1.2 Типы . . . . .	14
4.1.2.1 can_flag_t . . . . .	14
4.1.2.2 can_kind_of_error_t . . . . .	14
4.1.3 Перечисления . . . . .	14
4.1.3.1 _can_bytes_in_datafield . . . . .	14
4.1.3.2 _can_flag . . . . .	15
4.1.3.3 _can_kind_of_error . . . . .	15
4.1.3.4 _can_status . . . . .	16
4.1.3.5 _can_stb_discipline . . . . .	16
4.1.3.6 _canfd_mode . . . . .	16
4.1.3.7 _ttcan_timer_prescaler . . . . .	17
4.1.3.8 _ttcan_trigger_type . . . . .	17
4.1.4 Функции . . . . .	17
4.1.4.1 CAN_AbortPrimaryTxBuffer() . . . . .	17
4.1.4.2 CAN_AbortSecondaryTxBuffer() . . . . .	18
4.1.4.3 CAN_CalculateImprovedTimingValues() . . . . .	18
4.1.4.4 CAN_ClearStatusFlag() . . . . .	18
4.1.4.5 CAN_ClearStatusFlagMask() . . . . .	19
4.1.4.6 CAN_Deinit() . . . . .	19
4.1.4.7 CAN_DisableInterrupt() . . . . .	19
4.1.4.8 CAN_DisableInterruptMask() . . . . .	20
4.1.4.9 CAN_EnableInterrupt() . . . . .	20
4.1.4.10 CAN_EnableInterruptMask() . . . . .	20
4.1.4.11 CAN_EnterNormalMode() . . . . .	21
4.1.4.12 CAN_EnterStandbyMode() . . . . .	21
4.1.4.13 CAN_GetDefaultConfig() . . . . .	21
4.1.4.14 CAN_GetEnabledInterruptMask() . . . . .	21
4.1.4.15 CAN_GetStatusFlag() . . . . .	22
4.1.4.16 CAN_GetStatusFlagMask() . . . . .	22
4.1.4.17 CAN_Init() . . . . .	22
4.1.4.18 CAN_IsInterruptEnabled() . . . . .	23
4.1.4.19 CAN_IsPrimaryTransmitRequestPending() . . . . .	23

4.1.4.20	CAN_IsRxBufferAlmostFull()	23
4.1.4.21	CAN_IsRxBufferEmpty()	24
4.1.4.22	CAN_IsRxBufferFull()	24
4.1.4.23	CAN_IsSecondaryTransmitRequestPending()	24
4.1.4.24	CAN_IsSecondaryTxBufferEmpty()	25
4.1.4.25	CAN_IsSecondaryTxBufferFull()	25
4.1.4.26	CAN_IsSecondaryTxBufferMoreThanHalfFull()	25
4.1.4.27	CAN_ReadRxBuffer()	26
4.1.4.28	CAN_SetArbitrationTimingConfig()	26
4.1.4.29	CAN_SetFilterConfig()	26
4.1.4.30	CAN_SetPrimaryTxBufferConfig()	27
4.1.4.31	CAN_SetRxBufferConfig()	27
4.1.4.32	CAN_SetSecondaryTxBufferConfig()	27
4.1.4.33	CAN_SetTTCANConfig()	27
4.1.4.34	CAN_TransferAbortReceive()	28
4.1.4.35	CAN_TransferAbortSendPrimary()	28
4.1.4.36	CAN_TransferAbortSendSecondary()	28
4.1.4.37	CAN_TransferCreateHandle()	28
4.1.4.38	CAN_TransferGetReceivedCount()	29
4.1.4.39	CAN_TransferGetSentPrimaryCount()	29
4.1.4.40	CAN_TransferGetSentSecondaryCount()	30
4.1.4.41	CAN_TransferHandleIRQ()	30
4.1.4.42	CAN_TransferReceiveFifoBlocking()	30
4.1.4.43	CAN_TransferReceiveFifoNonBlocking()	31
4.1.4.44	CAN_TransferSendPrimaryBlocking()	31
4.1.4.45	CAN_TransferSendPrimaryNonBlocking()	32
4.1.4.46	CAN_TransferSendSecondaryBlocking()	32
4.1.4.47	CAN_TransferSendSecondaryNonBlocking()	33
4.1.4.48	CAN_WritePrimaryTxBuffer()	33
4.1.4.49	CAN_WriteSecondaryTxBuffer()	34
4.2	Драйвер модуля CLKCTR	34
4.2.1	Подробное описание	40
4.2.2	Макросы	40
4.2.2.1	CLKCTR_FCLK_MAX	40
4.2.2.2	CLKCTR_FCLK_MIN	40
4.2.2.3	CLKCTR_FREQ_NOT_SET	40
4.2.2.4	CLKCTR_GNSSCLK_MAX	41
4.2.2.5	CLKCTR_GNSSCLK_MIN	41
4.2.2.6	CLKCTR_HFI_MAX	41
4.2.2.7	CLKCTR_HFI_MIN	41
4.2.2.8	CLKCTR_I2S_EXTCLK_MAX	41
4.2.2.9	CLKCTR_I2S_EXTCLK_MIN	41
4.2.2.10	CLKCTR_I2SCLK_MAX	41

4.2.2.11	CLKCTR_I2SCLK_MIN	41
4.2.2.12	CLKCTR_LFI_MAX	42
4.2.2.13	CLKCTR_LFI_MIN	42
4.2.2.14	CLKCTR_MAN_MAX	42
4.2.2.15	CLKCTR_MAX_FCLK_DIV	42
4.2.2.16	CLKCTR_MAX_GNSSCLK_DIV	42
4.2.2.17	CLKCTR_MAX_I2SCLK_DIV	42
4.2.2.18	CLKCTR_MAX_MCOCLK_DIV	42
4.2.2.19	CLKCTR_MAX_QSPICLK_DIV	43
4.2.2.20	CLKCTR_MAX_SYSCLK_DIV	43
4.2.2.21	CLKCTR_MIN_FCLK_DIV	43
4.2.2.22	CLKCTR_MIN_GNSSCLK_DIV	43
4.2.2.23	CLKCTR_MIN_I2SCLK_DIV	43
4.2.2.24	CLKCTR_MIN_MCOCLK_DIV	43
4.2.2.25	CLKCTR_MIN_QSPICLK_DIV	43
4.2.2.26	CLKCTR_MIN_SYSCLK_DIV	43
4.2.2.27	CLKCTR_NF_MAN_MAX	44
4.2.2.28	CLKCTR_NR_MAN_MAX	44
4.2.2.29	CLKCTR_OD_MAN_MAX	44
4.2.2.30	CLKCTR_PLLCLK_MAX	44
4.2.2.31	CLKCTR_PLLCLK_MIN	44
4.2.2.32	CLKCTR_PLLCLK_OD_MAX	44
4.2.2.33	CLKCTR_PLLCLK_OD_MIN	45
4.2.2.34	CLKCTR_PLLREF_MAX	45
4.2.2.35	CLKCTR_PLLREF_MIN	45
4.2.2.36	CLKCTR_QSPICLK_MAX	45
4.2.2.37	CLKCTR_QSPICLK_MIN	45
4.2.2.38	CLKCTR_SEL_MAX	45
4.2.2.39	CLKCTR_SYSCLK_MAX	45
4.2.2.40	CLKCTR_SYSCLK_MIN	46
4.2.2.41	CLKCTR_XTI32_MAX	46
4.2.2.42	CLKCTR_XTI32_MIN	46
4.2.2.43	CLKCTR_XTI_MAX	46
4.2.2.44	CLKCTR_XTI_MIN	46
4.2.2.45	HFI_FREQUENCY	46
4.2.2.46	PLL_MAX_MULTIPLIER	46
4.2.2.47	PLL_MIN_MULTIPLIER	46
4.2.3	Перечисления	46
4.2.3.1	clkctr_clk_force_type	46
4.2.3.2	clkctr_device_clk_force	47
4.2.3.3	clkctr_extern_freq	47
4.2.3.4	clkctr_hfi_for_main_type	47
4.2.3.5	clkctr_i2sclk	48

4.2.3.6 clkctr_int_freq . . . . .	48
4.2.3.7 clkctr_lpcclk . . . . .	48
4.2.3.8 clkctr_mainclk . . . . .	49
4.2.3.9 clkctr_mcoclk . . . . .	49
4.2.3.10 clkctr_pllref . . . . .	49
4.2.3.11 clkctr_rtclk . . . . .	49
4.2.3.12 clkctr_status . . . . .	50
4.2.3.13 clkctr_usbclk . . . . .	50
4.2.4 Функции . . . . .	50
4.2.4.1 CLKCTR_GetAllDiv() . . . . .	50
4.2.4.2 CLKCTR_GetClk() . . . . .	51
4.2.4.3 CLKCTR_GetClkForce() . . . . .	51
4.2.4.4 CLKCTR_GetDivClk() . . . . .	51
4.2.4.5 CLKCTR_GetFClk() . . . . .	52
4.2.4.6 CLKCTR_GetFClkDiv() . . . . .	52
4.2.4.7 CLKCTR_GetFClkInt() . . . . .	52
4.2.4.8 CLKCTR_GetFrequency() . . . . .	53
4.2.4.9 CLKCTR_GetGNSSClk() . . . . .	53
4.2.4.10 CLKCTR_GetGNSSClkDiv() . . . . .	53
4.2.4.11 CLKCTR_GetHFI() . . . . .	54
4.2.4.12 CLKCTR_GetHFIClk() . . . . .	54
4.2.4.13 CLKCTR_GetHFIClkForMainClk() . . . . .	54
4.2.4.14 CLKCTR_GetHFITrim() . . . . .	55
4.2.4.15 CLKCTR_GetI2SClk() . . . . .	55
4.2.4.16 CLKCTR_GetI2SClkDiv() . . . . .	55
4.2.4.17 CLKCTR_GetI2SExtClk() . . . . .	56
4.2.4.18 CLKCTR_GetLFI() . . . . .	56
4.2.4.19 CLKCTR_GetLPClk() . . . . .	56
4.2.4.20 CLKCTR_GetMainClk() . . . . .	57
4.2.4.21 CLKCTR_GetMCOClk() . . . . .	57
4.2.4.22 CLKCTR_GetMCODiv() . . . . .	57
4.2.4.23 CLKCTR_GetMCOEn() . . . . .	57
4.2.4.24 CLKCTR_GetPLLClk() . . . . .	58
4.2.4.25 CLKCTR_GetPLLConfig() . . . . .	58
4.2.4.26 CLKCTR_GetPLLRef() . . . . .	58
4.2.4.27 CLKCTR_GetQSPIClk() . . . . .	59
4.2.4.28 CLKCTR_GetQSPIClkDiv() . . . . .	59
4.2.4.29 CLKCTR_GetRTCClk() . . . . .	59
4.2.4.30 CLKCTR_GetSwitchClk() . . . . .	60
4.2.4.31 CLKCTR_GetSwitchI2SClk() . . . . .	60
4.2.4.32 CLKCTR_GetSwitchLPClk() . . . . .	60
4.2.4.33 CLKCTR_GetSwitchMainClk() . . . . .	61
4.2.4.34 CLKCTR_GetSwitchMCOClk() . . . . .	61

4.2.4.35 CLKCTR_GetSwitchPLLRef()	61
4.2.4.36 CLKCTR_GetSwitchPMUDIS()	62
4.2.4.37 CLKCTR_GetSwitchRTCClk()	62
4.2.4.38 CLKCTR_GetSwitchUSBClk()	62
4.2.4.39 CLKCTR_GetSysClk()	63
4.2.4.40 CLKCTR_GetSysClkDiv()	63
4.2.4.41 CLKCTR_GetUSBClk()	63
4.2.4.42 CLKCTR_GetXTI()	64
4.2.4.43 CLKCTR_GetXTI32()	64
4.2.4.44 CLKCTR_GetXTIClk()	64
4.2.4.45 CLKCTR_SetClkForce()	65
4.2.4.46 CLKCTR_SetDivClk()	65
4.2.4.47 CLKCTR_SetFClkDiv()	66
4.2.4.48 CLKCTR_SetFrequency()	66
4.2.4.49 CLKCTR_SetGNSSClkDiv()	66
4.2.4.50 CLKCTR_SetHFI()	67
4.2.4.51 CLKCTR_SetHFITrim()	67
4.2.4.52 CLKCTR_SetI2SClkDiv()	68
4.2.4.53 CLKCTR_SetI2SExtClk()	68
4.2.4.54 CLKCTR_SetLFI()	69
4.2.4.55 CLKCTR_SetMCOdiv()	69
4.2.4.56 CLKCTR_SetMCOEn()	70
4.2.4.57 CLKCTR_SetPll()	70
4.2.4.58 CLKCTR_SetPLLConfig()	71
4.2.4.59 CLKCTR_SetPllMan()	71
4.2.4.60 CLKCTR_SetQSPIClkDiv()	72
4.2.4.61 CLKCTR_SetSwitchClk()	72
4.2.4.62 CLKCTR_SetSwitchI2SClk()	73
4.2.4.63 CLKCTR_SetSwitchLPClk()	73
4.2.4.64 CLKCTR_SetSwitchMainClk()	74
4.2.4.65 CLKCTR_SetSwitchMCOClk()	74
4.2.4.66 CLKCTR_SetSwitchPLLRef()	75
4.2.4.67 CLKCTR_SetSwitchPMUDIS()	75
4.2.4.68 CLKCTR_SetSwitchRTCClk()	76
4.2.4.69 CLKCTR_SetSwitchUSBClk()	76
4.2.4.70 CLKCTR_SetSysClkDiv()	77
4.2.4.71 CLKCTR_SetSysDiv()	77
4.2.4.72 CLKCTR_SetXTI()	78
4.2.4.73 CLKCTR_SetXTI32()	78
4.3 Общие определения для библиотеки HAL	78
4.3.1 Подробное описание	79
4.3.2 Макросы	80
4.3.2.1 BE_TO_LE16	80

4.3.2.2 BE_TO_LE24 . . . . .	80
4.3.2.3 BE_TO_LE32 . . . . .	80
4.3.2.4 DIM . . . . .	81
4.3.2.5 FIELD_BIT . . . . .	81
4.3.2.6 MAX . . . . .	81
4.3.2.7 MIN . . . . .	82
4.3.2.8 SUPPRESS_FALL_THROUGH_WARNING . . . . .	82
4.3.2.9 UINT16_MAX . . . . .	82
4.3.2.10 UINT32_MAX . . . . .	82
4.3.2.11 UNUSED . . . . .	83
4.3.3 Функции . . . . .	83
4.3.3.1 LBBLEndian() . . . . .	83
4.4 Драйвер модуля DMA . . . . .	83
4.4.1 Подробное описание . . . . .	87
4.4.2 Макросы . . . . .	87
4.4.2.1 DMA_CHANNEL_CTL . . . . .	87
4.4.2.2 DMA_CHANNEL_CTL_BLOCKSIZE_MASK . . . . .	88
4.4.2.3 DMA_CHANNEL_CTL_BLOCKSIZE_SHIFT . . . . .	88
4.4.3 Перечисления . . . . .	88
4.4.3.1 anonymous enum . . . . .	88
4.4.3.2 anonymous enum . . . . .	88
4.4.3.3 anonymous enum . . . . .	89
4.4.3.4 _dma_int . . . . .	89
4.4.3.5 _dma_priority . . . . .	89
4.4.3.6 _dma_status . . . . .	90
4.4.3.7 _dma_transfer_type . . . . .	90
4.4.4 Функции . . . . .	91
4.4.4.1 DMA_AbortTransfer() . . . . .	91
4.4.4.2 DMA_ChannelIRQHandle() . . . . .	91
4.4.4.3 DMA_ChannelsActive() . . . . .	91
4.4.4.4 DMA_CreateHandle() . . . . .	91
4.4.4.5 DMA_Deinit() . . . . .	92
4.4.4.6 DMA_DisableChannel() . . . . .	92
4.4.4.7 DMA_DisableChannelInterrupt() . . . . .	92
4.4.4.8 DMA_EnableChannel() . . . . .	93
4.4.4.9 DMA_EnableChannelInterrupt() . . . . .	93
4.4.4.10 DMA_EnableChannelPeriphRq() . . . . .	93
4.4.4.11 DMA_GetChannelPriority() . . . . .	93
4.4.4.12 DMA_GetCTLCfgMask() . . . . .	94
4.4.4.13 DMA_GetDescriptorCount() . . . . .	94
4.4.4.14 DMA_HardwareHandshakeEnable() . . . . .	95
4.4.4.15 DMA_Init() . . . . .	95
4.4.4.16 DMA_InitMultiblockDescriptor() . . . . .	95



4.4.4.17 DMA_PrepareChannelTransfer()	95
4.4.4.18 DMA_ScatterGatherEnable()	96
4.4.4.19 DMA_SetCallback()	96
4.4.4.20 DMA_SetChannelPriority()	96
4.4.4.21 DMA_SetupDescriptor()	97
4.4.4.22 DMA_SetupDstScatter()	97
4.4.4.23 DMA_SetupSrcGather()	97
4.4.4.24 DMA_StartTransfer()	98
4.4.4.25 DMA_SubmitChannelDescriptor()	98
4.4.4.26 DMA_SubmitChannelTransfer()	99
4.4.4.27 DMA_SubmitChannelTransferParameter()	99
4.5 Драйвер модуля DTIM	100
4.5.1 Подробное описание	101
4.5.2 Макросы	101
4.5.2.1 DUALTIMER_MAX_INDEX	101
4.5.2.2 DUALTIMER_NUMBER_OF_DUALTIMERS	101
4.5.3 Перечисления	101
4.5.3.1 dualtimer_interrupt_control	101
4.5.3.2 dualtimer_mode	102
4.5.3.3 dualtimer_number_of_repetitions	102
4.5.3.4 dualtimer_prescale	102
4.5.3.5 dualtimer_status	102
4.5.3.6 dualtimer_timer_size	103
4.5.3.7 dualtimer_work_enable	103
4.5.4 Функции	103
4.5.4.1 DUALTIMER_Deinit()	103
4.5.4.2 DUALTIMER_GetAPIStatus()	104
4.5.4.3 DUALTIMER_GetDefaultConfig()	104
4.5.4.4 DUALTIMER_GetRawStatus()	104
4.5.4.5 DUALTIMER_GetStatus()	105
4.5.4.6 DUALTIMER_GetTick()	105
4.5.4.7 DUALTIMER_Init()	105
4.5.4.8 DUALTIMER_IrqClr()	106
4.5.4.9 DUALTIMER_Reload()	106
4.5.4.10 DUALTIMER_Run()	107
4.5.4.11 DUALTIMER_Stop()	107
4.6 Драйвер модуля FLASH	107
4.6.1 Подробное описание	109
4.6.2 Макросы	109
4.6.2.1 FCTR_CMD_ERASE	109
4.6.2.2 FCTR_CMD_MASS_ERASE	109
4.6.2.3 FCTR_CMD_READ	109
4.6.2.4 FCTR_CMD_ROW_WRITE	109

4.6.2.5	FCTR_CMD_WRITE	109
4.6.2.6	FCTR_IRQ_STS_CLR_SUCCESS_FLAGS	109
4.6.2.7	FCTR_IRQ_STS_SET_RESULT_FLAGS	110
4.6.2.8	FLASH_TEST_ADDRESSES	110
4.6.3	Перечисления	110
4.6.3.1	flash_region	110
4.6.3.2	flash_status	110
4.6.4	Функции	111
4.6.4.1	FLASH_Erase()	111
4.6.4.2	FLASH_Init()	111
4.6.4.3	FLASH_MassErase()	112
4.6.4.4	FLASH_Program()	112
4.6.4.5	FLASH_Read()	113
4.6.4.6	FLASH_VerifyErase()	114
4.6.4.7	FLASH_VerifyProgram()	114
4.6.4.8	FLASH_WriteWord()	115
4.7	Драйвер модуля GPIO	116
4.7.1	Подробное описание	116
4.7.2	Макросы	116
4.7.2.1	GPIO_PORTA	116
4.7.2.2	GPIO_PORTB	116
4.7.2.3	GPIO_PORTC	117
4.7.2.4	GPIO_PORTD	117
4.7.2.5	GPIO_PORTPIN	117
4.7.2.6	GPIO_PORTPIN_GET_MASK	117
4.7.2.7	GPIO_PORTPIN_GET_PIN_NUM	117
4.7.2.8	GPIO_PORTPIN_GET_PORT_NUM	117
4.7.3	Перечисления	117
4.7.3.1	gpio_mode_t	117
4.7.3.2	gpio_pin_function_t	118
4.8	Драйвер модуля I2C	118
4.8.1	Подробное описание	122
4.8.2	Макросы	122
4.8.2.1	I2C_CFG_MASK	122
4.8.2.2	I2C_MASTER_TRANSMIT_IGNORE_LAST_NACK	122
4.8.2.3	I2C_RETRY_TIMES	122
4.8.2.4	I2C_STAT_MSTCODE_IDLE	123
4.8.2.5	I2C_STAT_MSTCODE_NACKADR	123
4.8.2.6	I2C_STAT_MSTCODE_NACKDAT	123
4.8.2.7	I2C_STAT_MSTCODE_RXREADY	123
4.8.2.8	I2C_STAT_MSTCODE_TXREADY	123
4.8.2.9	I2C_STAT_SLVST_ADDR	123
4.8.2.10	I2C_STAT_SLVST_RX	123

4.8.2.11 I2C_STAT_SLVST_TX . . . . .	123
4.8.3 Типы . . . . .	123
4.8.3.1 i2c_master_transfer_callback_t . . . . .	123
4.8.3.2 i2c_slave_transfer_callback_t . . . . .	124
4.8.4 Перечисления . . . . .	124
4.8.4.1 i2c_abort_flags . . . . .	124
4.8.4.2 i2c_addr_size_t . . . . .	125
4.8.4.3 i2c_direction_t . . . . .	125
4.8.4.4 i2c_interrupt . . . . .	126
4.8.4.5 i2c_master_transfer_flags_t . . . . .	127
4.8.4.6 i2c_master_transfer_states . . . . .	127
4.8.4.7 i2c_slave_event_transfer_t . . . . .	128
4.8.4.8 i2c_slave_fsm_t . . . . .	128
4.8.4.9 i2c_speed_mode_t . . . . .	128
4.8.4.10 i2c_status_flags . . . . .	129
4.8.4.11 i2c_status_t . . . . .	129
4.8.5 Функции . . . . .	130
4.8.5.1 I2C_ClearAllInterrupts() . . . . .	130
4.8.5.2 I2C_ClearInterrupt() . . . . .	130
4.8.5.3 I2C_DisableInterrupts() . . . . .	131
4.8.5.4 I2C_DMADescriptorInitRX() . . . . .	131
4.8.5.5 I2C_DMADescriptorInitTX() . . . . .	132
4.8.5.6 I2C_Enable() . . . . .	132
4.8.5.7 I2C_EnableInterrupts() . . . . .	133
4.8.5.8 I2C_GetEnabledInterrupts() . . . . .	133
4.8.5.9 I2C_GetInstance() . . . . .	134
4.8.5.10 I2C_IsEnable() . . . . .	135
4.8.5.11 I2C_MasterAddrSet() . . . . .	135
4.8.5.12 I2C_MasterDeinit() . . . . .	135
4.8.5.13 I2C_MasterGetBusActiveState() . . . . .	136
4.8.5.14 I2C_MasterGetDefaultConfig() . . . . .	136
4.8.5.15 I2C_MasterInit() . . . . .	137
4.8.5.16 I2C_MasterReadBlocking() . . . . .	137
4.8.5.17 I2C_MasterSetBaudRate() . . . . .	138
4.8.5.18 I2C_MasterTransferAbort() . . . . .	139
4.8.5.19 I2C_MasterTransferAbortDMA() . . . . .	139
4.8.5.20 I2C_MasterTransferBlocking() . . . . .	140
4.8.5.21 I2C_MasterTransferCreateHandle() . . . . .	140
4.8.5.22 I2C_MasterTransferCreateHandleDMA() . . . . .	141
4.8.5.23 I2C_MasterTransferDMA() . . . . .	141
4.8.5.24 I2C_MasterTransferGetCount() . . . . .	142
4.8.5.25 I2C_MasterTransferHandleIRQ() . . . . .	142
4.8.5.26 I2C_MasterTransferNonBlocking() . . . . .	142

4.8.5.27 I2C_MasterWriteBlocking()	143
4.8.5.28 I2C_Reset()	144
4.9 Драйвер модуля I2S	144
4.9.1 Подробное описание	147
4.9.2 Типы	147
4.9.2.1 i2s_sclk_gating_t	147
4.9.2.2 i2s_transfer_callback_t	147
4.9.3 Перечисления	148
4.9.3.1 _i2s_flag	148
4.9.3.2 _i2s_interrupt_level	148
4.9.3.3 _i2s_resolution	148
4.9.3.4 _i2s_sclk_gating	149
4.9.3.5 _i2s_sclk_per_sample	149
4.9.3.6 _i2s_status	149
4.9.4 Функции	150
4.9.4.1 I2S_AbortTx()	150
4.9.4.2 I2S_ClearDataOverrunFlag()	150
4.9.4.3 I2S_Deinit()	150
4.9.4.4 I2S_DisableInterrupt()	150
4.9.4.5 I2S_DisableInterruptMask()	151
4.9.4.6 I2S_DisableTx()	151
4.9.4.7 I2S_EnableInterrupt()	151
4.9.4.8 I2S_EnableInterruptMask()	151
4.9.4.9 I2S_EnableTx()	152
4.9.4.10 I2S_GetDefaultConfig()	152
4.9.4.11 I2S_GetEnabledInterruptMask()	152
4.9.4.12 I2S_GetInterruptStatus()	152
4.9.4.13 I2S_GetInterruptStatusMask()	153
4.9.4.14 I2S_Init()	153
4.9.4.15 I2S_IsInterruptEnabled()	153
4.9.4.16 I2S_TransferAbort()	154
4.9.4.17 I2S_TransferCreateHandle()	154
4.9.4.18 I2S_TransferHandleIRQ()	154
4.9.4.19 I2S_TransferNonBlocking()	155
4.9.4.20 I2S_WriteLeftFifo()	155
4.9.4.21 I2S_WriteRightFifo()	155
4.10 Драйвер менеджера прерываний IO устройств и DMA каналов	156
4.10.1 Подробное описание	156
4.10.2 Макросы	157
4.10.2.1 IOIM_NA_IRQ_NUM	157
4.10.3 Перечисления	157
4.10.3.1 ioim_status_t	157
4.10.4 Функции	157

4.10.4.1 IOIM_ClearIRQHandler()	157
4.10.4.2 IOIM_ClearIRQHandler_DMA()	157
4.10.4.3 IOIM_GetIRQNumber()	158
4.10.4.4 IOIM_SetIRQHandler()	158
4.10.4.5 IOIM_SetIRQHandler_DMA()	159
4.11 Драйвер модуля JTM	159
4.11.1 Подробное описание	160
4.11.2 Типы	160
4.11.2.1 jtm_callback_t	160
4.11.3 Перечисления	160
4.11.3.1 jtm_parameter_t	160
4.11.3.2 jtm_status_t	161
4.11.4 Функции	161
4.11.4.1 JTM_CreateHandle()	161
4.11.4.2 JTM_GetParameterValue()	162
4.11.4.3 JTM_GetParameterValueNonBlocking()	162
4.11.4.4 JTM_Init()	163
4.12 Драйвер модуля программных прерываний MHU	163
4.12.1 Подробное описание	163
4.12.2 Функции	163
4.12.2.1 MHU_CPU0_ClearInt()	163
4.12.2.2 MHU_CPU0_SetInt()	164
4.12.2.3 MHU_CPU0_StatInt()	164
4.12.2.4 MHU_CPU1_ClearInt()	164
4.12.2.5 MHU_CPU1_SetInt()	164
4.12.2.6 MHU_CPU1_StatInt()	165
4.13 Драйвер модуля POWER	165
4.13.1 Подробное описание	167
4.13.2 Типы	167
4.13.2.1 power_callback_t	167
4.13.3 Перечисления	167
4.13.3.1 power_dcdc_mode	167
4.13.3.2 power_dcdc_threshold	168
4.13.3.3 power_dcdc_vlevel	168
4.13.3.4 power_dcdc_vlevel_value	169
4.13.3.5 power_eco_mode	170
4.13.3.6 power_flash_mode	170
4.13.3.7 power_interrupt	170
4.13.3.8 power_status	170
4.13.3.9 power_test_block	170
4.13.4 Функции	171
4.13.4.1 POWER_ClearInterrupts()	171
4.13.4.2 POWER_CreateHandle()	171

4.13.4.3	POWER_DisableInterrupt()	171
4.13.4.4	POWER_DisableInterruptMask()	172
4.13.4.5	POWER_EnableInterrupt()	172
4.13.4.6	POWER_EnableInterruptMask()	172
4.13.4.7	POWER_GetCurrentConfig()	172
4.13.4.8	POWER_GetEnabledInterruptMask()	173
4.13.4.9	POWER_GetInterruptStatus()	173
4.13.4.10	POWER_GetInterruptStatusMask()	173
4.13.4.11	POWER_GetStatus()	174
4.13.4.12	POWER_IsInterruptEnabled()	174
4.13.4.13	POWER_SetConfig()	174
4.13.4.14	POWER_Shutdown()	175
4.13.4.15	POWER_Standby()	175
4.13.4.16	POWER_StartTestMode()	175
4.13.4.17	POWER_StopTestMode()	176
4.14	Драйвер модуля PPU	176
4.14.1	Подробное описание	178
4.14.2	Типы	178
4.14.2.1	ReqOffForCPU	178
4.14.3	Перечисления	178
4.14.3.1	ppu_add_event_name	178
4.14.3.2	ppu_domain_index	178
4.14.3.3	ppu_event_name	179
4.14.3.4	ppu_opportunities_idr0	179
4.14.3.5	ppu_opportunities_idr1	180
4.14.3.6	ppu_power_mode	180
4.14.3.7	ppu_sense_index	181
4.14.3.8	ppu_status	182
4.14.4	Функции	182
4.14.4.1	PPU_ClrIRQMask()	182
4.14.4.2	PPU_ClrIRQStatus()	182
4.14.4.3	PPU_GetIRQMask()	183
4.14.4.4	PPU_GetIRQStatus()	183
4.14.4.5	PPU_GetLastAPIStatus()	184
4.14.4.6	PPU_GetPDxSenseFromPDy()	184
4.14.4.7	PPU_GetPowerState()	184
4.14.4.8	PPU_Init()	184
4.14.4.9	PPU_SetIRQMask()	185
4.14.4.10	PPU_SetIRQStatus()	185
4.14.4.11	PPU_SetPDCMPPUSense()	186
4.14.4.12	PPU_SetState()	186
4.14.4.13	PPU_SetStateDynamic()	187
4.14.4.14	PPU_StateOffRequestHandler()	187

4.15	Драйвер модуля PWM	188
4.15.1	Подробное описание	191
4.15.2	Макросы	191
4.15.2.1	PWM_COUNT	191
4.15.3	Перечисления	191
4.15.3.1	anonymouse enum	191
4.15.3.2	pwm_chopper_duty [1/2]	191
4.15.3.3	pwm_chopper_duty [2/2]	192
4.15.3.4	pwm_chopper_first_width [1/2]	192
4.15.3.5	pwm_chopper_first_width [2/2]	193
4.15.3.6	pwm_chopper_freq [1/2]	194
4.15.3.7	pwm_chopper_freq [2/2]	194
4.15.3.8	pwm_chopper_work	195
4.15.3.9	pwm_cntmode	195
4.15.3.10	pwm_dirsync	195
4.15.3.11	pwm_dz_mode [1/2]	195
4.15.3.12	pwm_dz_mode [2/2]	196
4.15.3.13	pwm_dz_outx_inv [1/2]	196
4.15.3.14	pwm_dz_outx_inv [2/2]	196
4.15.3.15	pwm_dz_signal	196
4.15.3.16	pwm_eventprd [1/2]	197
4.15.3.17	pwm_eventprd [2/2]	197
4.15.3.18	pwm_int_en	197
4.15.3.19	pwm_int_source [1/2]	197
4.15.3.20	pwm_int_source [2/2]	198
4.15.3.21	pwm_ldcswrf	198
4.15.3.22	pwm_ldxmode	199
4.15.3.23	pwm_loadprd	199
4.15.3.24	pwm_outx_cmd	199
4.15.3.25	pwm_prescaler_cmd	199
4.15.3.26	pwm_prescaler_divmux	200
4.15.3.27	pwm_prescaler_mode	200
4.15.3.28	pwm_prescaler_syncrst	200
4.15.3.29	pwm_run_command	200
4.15.3.30	pwm_scmpxmode	201
4.15.3.31	pwm_status	201
4.15.3.32	pwm_syncosel	201
4.15.3.33	pwm_syncphsen	201
4.15.3.34	pwm_trip_unit_action [1/2]	202
4.15.3.35	pwm_trip_unit_action [2/2]	202
4.15.3.36	pwm_trip_unit_signal	202
4.15.4	Функции	203
4.15.4.1	PWM_ApplyLongSoftOuts()	203

4.15.4.2 PWM_ApplySoftOuts()	203
4.15.4.3 PWM_CmdForAllChannels()	204
4.15.4.4 PWM_Deinit()	204
4.15.4.5 PWM_Enable() [1/2]	205
4.15.4.6 PWM_Enable() [2/2]	205
4.15.4.7 PWM_GetChannelDefaultConfig()	205
4.15.4.8 PWM_GetCntStat()	206
4.15.4.9 PWM_Init()	206
4.15.4.10 PWM_InitChannel()	206
4.15.4.11 PWM_IntCallback()	206
4.15.4.12 PWM_SetPeriod()	207
4.16 Драйвер модуля QSPI	207
4.16.1 Подробное описание	211
4.16.2 Макросы	211
4.16.2.1 DUMMY_BYTE	211
4.16.2.2 FLASH_STAT_BUSY	211
4.16.2.3 FLASH_STAT_WEL	211
4.16.3 Типы	212
4.16.3.1 qspi_config_t	212
4.16.4 Перечисления	212
4.16.4.1 anonymous enum	212
4.16.4.2 _qspi_command_format	212
4.16.4.3 _qspi_command_type	212
4.16.4.4 _qspi_qmode	213
4.16.4.5 _serial_nor_command	213
4.16.4.6 nor_status_t	214
4.16.4.7 qspi_dma_status_t	214
4.16.5 Функции	215
4.16.5.1 NOR_FlashEraseBlock()	215
4.16.5.2 NOR_FlashEraseChip()	215
4.16.5.3 NOR_FlashPageProgram()	216
4.16.5.4 NOR_FlashProgramBlock()	216
4.16.5.5 NOR_FlashRead()	217
4.16.5.6 NOR_FlashReadXIP()	217
4.16.5.7 QSPI_ClearInterrupt()	217
4.16.5.8 QSPI_DeInit()	218
4.16.5.9 QSPI_DisableDMA()	218
4.16.5.10 QSPI_DisableInterrupt()	218
4.16.5.11 QSPI_DisableXIP()	218
4.16.5.12 QSPI_DMADescriptorInitRX()	219
4.16.5.13 QSPI_DMADescriptorInitTX()	219
4.16.5.14 QSPI_DMAReadDescriptorInitRX()	220
4.16.5.15 QSPI_Enable()	220



4.16.5.16	QSPI_EnableDMA()	220
4.16.5.17	QSPI_EnableInterrupt()	220
4.16.5.18	QSPI_EnableXIP()	221
4.16.5.19	QSPI_GetDefaultCommandSet()	221
4.16.5.20	QSPI_GetDefaultConfig()	221
4.16.5.21	QSPI_GetDefaultConfigXIP()	221
4.16.5.22	QSPI_GetDummyDMADescriptorsCount()	222
4.16.5.23	QSPI_GetInstance()	222
4.16.5.24	QSPI_GetReadDMADescriptorsCount()	222
4.16.5.25	QSPI_GetRXLVL()	223
4.16.5.26	QSPI_GetStatusFlag()	224
4.16.5.27	QSPI_GetTXLVL()	224
4.16.5.28	QSPI_Init()	224
4.16.5.29	QSPI_NorFlashInit()	225
4.16.5.30	QSPI_ReadData()	225
4.16.5.31	QSPI_ReadDataByte()	225
4.16.5.32	QSPI_ReadDataDMA()	226
4.16.5.33	QSPI_SetBitSize()	226
4.16.5.34	QSPI_SetInhibitDin()	226
4.16.5.35	QSPI_SetInhibitDout()	227
4.16.5.36	QSPI_SetQMode()	227
4.16.5.37	QSPI_SetSlaveSelect()	227
4.16.5.38	QSPI_TransferCreateHandleDMA()	227
4.16.5.39	QSPI_WriteData()	228
4.16.5.40	QSPI_WriteDataByte()	228
4.16.5.41	QSPI_WriteDataDMA()	228
4.17	Драйвер модуля RWC	229
4.17.1	Подробное описание	232
4.17.2	Макросы	232
4.17.2.1	DAYS_IN_A_YEAR	232
4.17.2.2	RWC_SYNC_RETRY_TIMES	233
4.17.2.3	SECONDS_IN_A_DAY	233
4.17.2.4	SECONDS_IN_A_HOUR	233
4.17.2.5	SECONDS_IN_A_MINUTE	233
4.17.2.6	YEAR_RANGE_END	233
4.17.2.7	YEAR_RANGE_START	233
4.17.3	Перечисления	233
4.17.3.1	rcw_alarm_enable	233
4.17.3.2	rcw_cmd	234
4.17.3.3	rcw_freq_serial	234
4.17.3.4	rcw_internal_register	234
4.17.3.5	rcw_lfe_bypass	235
4.17.3.6	rcw_reset_type	235

4.17.3.7 rwc_rtccclk_divisor . . . . .	235
4.17.3.8 rwc_rtccclk_type . . . . .	236
4.17.3.9 rwc_shutdown_force . . . . .	236
4.17.3.10 rwc_status . . . . .	236
4.17.3.11 rwc_time_clk_sel . . . . .	236
4.17.3.12 rwc_timer_status . . . . .	237
4.17.3.13 rwc_wake_up_irq_enable . . . . .	237
4.17.3.14 rwc_wake_up_polarity . . . . .	237
4.17.3.15 rwc_wkup_enable . . . . .	237
4.17.4 Функции . . . . .	238
4.17.4.1 RWC_Deinit() . . . . .	238
4.17.4.2 RWC_EnableAlarmTimerInterruptFromDPD() . . . . .	238
4.17.4.3 RWC_EnableWakeUpTimerInterruptFromDPD() . . . . .	239
4.17.4.4 RWC_GetAlarm() . . . . .	239
4.17.4.5 RWC_GetDatetime() . . . . .	240
4.17.4.6 RWC_GetDefaultConfig() . . . . .	240
4.17.4.7 RWC_GetInternalRegister() . . . . .	240
4.17.4.8 RWC_GetLastAPIStatus() . . . . .	241
4.17.4.9 RWC_GetRTCClkParam() . . . . .	241
4.17.4.10 RWC_GetSecondsTimerCount() . . . . .	242
4.17.4.11 RWC_GetSecondsTimerMatch() . . . . .	242
4.17.4.12 RWC_GetStatusFlags() . . . . .	242
4.17.4.13 RWC_GetTime() . . . . .	242
4.17.4.14 RWC_Init() . . . . .	243
4.17.4.15 RWC_InterruptClear() . . . . .	243
4.17.4.16 RWC_SetAlarm() . . . . .	244
4.17.4.17 RWC_SetDatetime() . . . . .	244
4.17.4.18 RWC_SetInternalRegister() . . . . .	245
4.17.4.19 RWC_SetLFEBypass() . . . . .	245
4.17.4.20 RWC_SetLFx() . . . . .	245
4.17.4.21 RWC_SetResetCtrl() . . . . .	246
4.17.4.22 RWC_SetRTCClkParam() . . . . .	246
4.17.4.23 RWC_SetSecondsTimerCount() . . . . .	247
4.17.4.24 RWC_SetSecondsTimerMatch() . . . . .	247
4.17.4.25 RWC_SetWakeUpActiveLewel() . . . . .	248
4.17.4.26 RWC_SetWakeUpEnable() . . . . .	248
4.18 Драйвер модуля SDMMC . . . . .	248
4.18.1 Подробное описание . . . . .	251
4.18.2 Макросы . . . . .	251
4.18.2.1 SDMMC_CALC_DIVIDER . . . . .	251
4.18.2.2 SDMMC_CORECFG0_SLOTTYPE_EMBEDDED . . . . .	252
4.18.2.3 SDMMC_CORECFG0_SLOTTYPE_REMOVABLE . . . . .	252
4.18.2.4 SDMMC_CORECFG0_SLOTTYPE_SHARED_BUS . . . . .	252

4.18.2.5	SDMMC_IS_MMC	252
4.18.2.6	SDMMC_IS_SD	252
4.18.2.7	SDMMC_MMC_RCA_ADDR	252
4.18.2.8	SDMMC_SD_OCR_INIT_VALUE	252
4.18.2.9	SDMMC_SD_SEND_IF_COND_PATTERN	253
4.18.2.10	SDMMC_SD_UHS_MODE_DDR50	253
4.18.2.11	SDMMC_SD_UHS_MODE_DEFAULT_SDR12	253
4.18.2.12	SDMMC_SD_UHS_MODE_HIGHSPEED_SDR25	253
4.18.2.13	SDMMC_SD_UHS_MODE_SDR104	253
4.18.2.14	SDMMC_SD_UHS_MODE_SDR50	253
4.18.2.15	SDMMC_SDHC_SECTOR_SIZE	253
4.18.2.16	SDMMC_SDMA_ALIGN	253
4.18.2.17	SDMMC_SDMA_BLOCK_ALIGN	254
4.18.2.18	SDMMC_SDMA_BLOCK_SIZE	254
4.18.2.19	SDMMC_SDMA_IS_BLOCK_ALIGN_ADDR	254
4.18.2.20	SDMMC_TIMEOUTCONTROL_MAX_VALUE	254
4.18.3	Перечисления	254
4.18.3.1	anonymous enum	254
4.18.3.2	anonymous enum	254
4.18.3.3	anonymous enum	254
4.18.3.4	anonymous enum	255
4.18.3.5	anonymous enum	255
4.18.3.6	sdmmc_status_t	255
4.18.3.7	sdmmc_voltage_t	255
4.18.4	Функции	256
4.18.4.1	SDMMC_CalcMemorySpace()	256
4.18.4.2	SDMMC_DisableCard()	256
4.18.4.3	SDMMC_InitCard()	257
4.18.4.4	SDMMC_Read()	257
4.18.4.5	SDMMC_ReadAsync()	258
4.18.4.6	SDMMC_ReadWait()	258
4.18.4.7	SDMMC_Write()	259
4.18.4.8	SDMMC_WriteAsync()	259
4.18.4.9	SDMMC_WriteWait()	260
4.19	Драйвер модуля SMC	260
4.19.1	Подробное описание	262
4.19.2	Перечисления	262
4.19.2.1	smc_adv	262
4.19.2.2	smc_bit_depth	262
4.19.2.3	smc_bls	262
4.19.2.4	smc_burst_align	262
4.19.2.5	smc_cmd_type	263
4.19.2.6	smc_cre	263

4.19.2.7 smc_incr_to_incr4 . . . . .	263
4.19.2.8 smc_packet_lenght . . . . .	263
4.19.2.9 smc_rd_wr_type . . . . .	264
4.19.2.10 smc_status . . . . .	264
4.19.3 Функции . . . . .	264
4.19.3.1 SMC_CheckConfigure() . . . . .	264
4.19.3.2 SMC_DirectCmd() . . . . .	265
4.19.3.3 SMC_PowerSaveOff() . . . . .	265
4.19.3.4 SMC_PowerSaveOn() . . . . .	266
4.19.3.5 SMC_RefreshPeriod() . . . . .	266
4.19.3.6 SMC_SetCycles() . . . . .	267
4.19.3.7 SMC_SetOpmode() . . . . .	267
4.19.3.8 SMC_UserConfig() . . . . .	268
4.20 Драйвер модуля SPI . . . . .	269
4.20.1 Подробное описание . . . . .	274
4.20.2 Макросы . . . . .	274
4.20.2.1 SPI_DUMMYDATA . . . . .	274
4.20.2.2 SPI_RETRY_TIMES . . . . .	274
4.20.3 Перечисления . . . . .	274
4.20.3.1 microwire_busy_ready_check_t . . . . .	274
4.20.3.2 microwire_ctrlword_len_t . . . . .	274
4.20.3.3 microwire_single_serial_t . . . . .	275
4.20.3.4 microwire_tx_rx_t . . . . .	275
4.20.3.5 spi_frame_format_t . . . . .	276
4.20.3.6 spi_frame_width_t . . . . .	276
4.20.3.7 spi_interrupt_enable . . . . .	277
4.20.3.8 spi_mode_t . . . . .	278
4.20.3.9 spi_motorola_cap_data_t . . . . .	278
4.20.3.10 spi_motorola_clk_pol_t . . . . .	278
4.20.3.11 spi_rxfifo_watermark_t . . . . .	279
4.20.3.12 spi_shift_direction_t . . . . .	279
4.20.3.13 spi_status . . . . .	279
4.20.3.14 spi_status_flags . . . . .	280
4.20.3.15 spi_trans_status . . . . .	280
4.20.3.16 spi_txfifo_watermark_t . . . . .	281
4.20.4 Функции . . . . .	281
4.20.4.1 SPI_CurrentStatusInterrupts() . . . . .	281
4.20.4.2 SPI_Deinit() . . . . .	281
4.20.4.3 SPI_DisableInterrupts() . . . . .	282
4.20.4.4 SPI_DMADescriptorInitRX() . . . . .	282
4.20.4.5 SPI_DMADescriptorInitTX() . . . . .	283
4.20.4.6 SPI_Enable() . . . . .	283
4.20.4.7 SPI_EnableInterrupts() . . . . .	283

4.20.4.8	SPI_EnableRxDMA()	284
4.20.4.9	SPI_EnableTxDMA()	284
4.20.4.10	SPI_GetConfig()	284
4.20.4.11	SPI_GetInstance()	284
4.20.4.12	SPI_GetStatusFlags()	285
4.20.4.13	SPI_IsEnable()	285
4.20.4.14	SPI_MasterGetDefaultConfig()	286
4.20.4.15	SPI_MasterHalfDuplexTransferBlocking()	286
4.20.4.16	SPI_MasterHalfDuplexTransferDMA()	286
4.20.4.17	SPI_MasterHalfDuplexTransferNonBlocking()	287
4.20.4.18	SPI_MasterInit()	288
4.20.4.19	SPI_MasterSetBaud()	288
4.20.4.20	SPI_MasterTransferAbort()	289
4.20.4.21	SPI_MasterTransferAbortDMA()	289
4.20.4.22	SPI_MasterTransferBlocking()	289
4.20.4.23	SPI_MasterTransferCreateHandle()	290
4.20.4.24	SPI_MasterTransferCreateHandleDMA()	290
4.20.4.25	SPI_MasterTransferDMA()	291
4.20.4.26	SPI_MasterTransferGetByte()	291
4.20.4.27	SPI_MasterTransferGetRemainingByte()	292
4.20.4.28	SPI_MasterTransferGetTotalByte()	293
4.20.4.29	SPI_MasterTransferHandleIRQ()	293
4.20.4.30	SPI_MasterTransferNonBlocking()	294
4.20.4.31	SPI_ReadData()	294
4.20.4.32	SPI_Reset()	295
4.20.4.33	SPI_SetDummyData()	296
4.20.4.34	SPI_SlaveGetDefaultConfig()	296
4.20.4.35	SPI_SlaveInit()	297
4.20.4.36	SPI_SlaveTransferAbort()	297
4.20.4.37	SPI_SlaveTransferAbortDMA()	297
4.20.4.38	SPI_SlaveTransferCreateHandle()	298
4.20.4.39	SPI_SlaveTransferCreateHandleDMA()	298
4.20.4.40	SPI_SlaveTransferDMA()	299
4.20.4.41	SPI_SlaveTransferGetByte()	299
4.20.4.42	SPI_SlaveTransferHandleIRQ()	300
4.20.4.43	SPI_SlaveTransferNonBlocking()	301
4.20.4.44	SPI_TakeDownInterrupts()	301
4.20.4.45	SPI_WriteData()	301
4.21	Драйвер модуля TIM	302
4.21.1	Подробное описание	303
4.21.2	Макросы	303
4.21.2.1	TIMER_COUNT	303
4.21.2.2	TIMER_HARDWARE_FIELD_MAX	304

4.21.2.3	TIMER_SOFTWARE_FIELD_HIGH_OFFSET	304
4.21.2.4	TIMER_SOFTWARE_FIELD_MAX	304
4.21.3	Перечисления	304
4.21.3.1	timer_status	304
4.21.3.2	timer_type_of_counting	304
4.21.3.3	timer_work_mode	304
4.21.4	Функции	305
4.21.4.1	TIMER_Deinit()	305
4.21.4.2	TIMER_GetAPIStatus()	305
4.21.4.3	TIMER_GetTicks()	305
4.21.4.4	TIMER_GetTimerHardwareValue()	306
4.21.4.5	TIMER_Init()	306
4.21.4.6	TIMER_IRQClear()	307
4.21.4.7	TIMER_IRQDisable()	307
4.21.4.8	TIMER_IRQEnable()	307
4.21.4.9	TIMER_IRQGetStatus()	308
4.21.4.10	TIMER_Reset()	308
4.21.4.11	TIMER_Run()	308
4.21.4.12	TIMER_SetConfig()	309
4.21.4.13	TIMER_SetTick()	309
4.21.4.14	TIMER_Stop()	310
4.22	Драйвер модуля UART	310
4.22.1	Подробное описание	315
4.22.2	Перечисления	315
4.22.2.1	uart_data_len	315
4.22.2.2	uart_interrupt_enable	315
4.22.2.3	uart_lsr_flags	315
4.22.2.4	uart_parity_mode	316
4.22.2.5	uart_rs485_active_state	316
4.22.2.6	uart_rs485_mode	316
4.22.2.7	uart_rxfifo_watermark	317
4.22.2.8	uart_status	317
4.22.2.9	uart_stop_bit_count	317
4.22.2.10	uart_txfifo_watermark	318
4.22.3	Функции	318
4.22.3.1	UART_Deinit()	318
4.22.3.2	UART_DisableInterrupts()	318
4.22.3.3	UART_DMADescriptorInitRX()	319
4.22.3.4	UART_DMADescriptorInitTX()	319
4.22.3.5	UART_EnableInterrupts()	320
4.22.3.6	UART_GetDefaultConfig()	320
4.22.3.7	UART_GetEnabledInterrupts()	320
4.22.3.8	UART_GetRxFifoCount()	321

4.22.3.9	UART_GetStatusFlags()	321
4.22.3.10	UART_GetTxFifoCount()	321
4.22.3.11	UART_Init()	322
4.22.3.12	UART_ReadBlocking()	322
4.22.3.13	UART_ReadByte()	323
4.22.3.14	UART_ReadByteWait()	323
4.22.3.15	UART_Rs485Mode()	324
4.22.3.16	UART_SetBaudRate()	324
4.22.3.17	UART_SetIr()	324
4.22.3.18	UART_SetLoopback()	325
4.22.3.19	UART_SetRs485()	325
4.22.3.20	UART_SetRxFifoWatermark()	325
4.22.3.21	UART_SetTxFifoWatermark()	325
4.22.3.22	UART_TransferAbortReceive()	326
4.22.3.23	UART_TransferAbortReceiveDMA()	326
4.22.3.24	UART_TransferAbortSend()	326
4.22.3.25	UART_TransferAbortSendDMA()	327
4.22.3.26	UART_TransferCreateHandle()	327
4.22.3.27	UART_TransferCreateHandleDMA()	328
4.22.3.28	UART_TransferGetReceiveCount()	328
4.22.3.29	UART_TransferGetRxRingBufferLength()	328
4.22.3.30	UART_TransferGetSendCount()	329
4.22.3.31	UART_TransferGetTxRingBufferLength()	329
4.22.3.32	UART_TransferHandleIRQ()	329
4.22.3.33	UART_TransferReceiveDMA()	330
4.22.3.34	UART_TransferReceiveNonBlocking()	330
4.22.3.35	UART_TransferSendDMA()	331
4.22.3.36	UART_TransferSendNonBlocking()	331
4.22.3.37	UART_TransferStartRingBuffer()	332
4.22.3.38	UART_TransferStartTxRingBuffer()	333
4.22.3.39	UART_TransferStopRingBuffer()	333
4.22.3.40	UART_TransferStopTxRingBuffer()	333
4.22.3.41	UART_WaitWhileActive()	334
4.22.3.42	UART_WriteBlocking()	334
4.22.3.43	UART_WriteByte()	335
4.22.3.44	UART_WriteByteWait()	335
4.22.3.45	UART_WriteTxRing()	335
4.23	Драйвер модуля VTU	336
4.23.1	Подробное описание	338
4.23.2	Перечисления	338
4.23.2.1	timer_num_mode	338
4.23.2.2	vtu_capture_edge_control	338
4.23.2.3	vtu_interrupt_control	338

4.23.2.4	vtu_mode	339
4.23.2.5	vtu_pwm_polarity	339
4.23.2.6	vtu_status	339
4.23.3	Функции	340
4.23.3.1	VTU_ClearTimerIRQ()	340
4.23.3.2	VTU_Deinit()	340
4.23.3.3	VTU_EnableTimer()	341
4.23.3.4	VTU_EnableTimerIRQ()	341
4.23.3.5	VTU_GetCaptureEdgeCtrl()	342
4.23.3.6	VTU_GetCounter()	342
4.23.3.7	VTU_GetDefaultConfig()	342
4.23.3.8	VTU_GetDutyCycleCapture()	343
4.23.3.9	VTU_GetLastAPIStatus()	343
4.23.3.10	VTU_GetPeriodCapture()	343
4.23.3.11	VTU_GetPrescaler()	344
4.23.3.12	VTU_GetPWMPolarity()	344
4.23.3.13	VTU_GetTimerIRQ()	345
4.23.3.14	VTU_Init()	345
4.23.3.15	VTU_SetCallback()	345
4.23.3.16	VTU_SetCaptureEdgeCtrl()	346
4.23.3.17	VTU_SetCounter()	346
4.23.3.18	VTU_SetDutyCycleCapture()	347
4.23.3.19	VTU_SetPeriodCapture()	347
4.23.3.20	VTU_SetPrescaler()	347
4.23.3.21	VTU_SetPWMPolarity()	348
4.24	Драйвер модуля WDT	348
4.24.1	Подробное описание	349
4.24.2	Макросы	350
4.24.2.1	WDT_NUMBER_OF_TIMERS	350
4.24.3	Перечисления	350
4.24.3.1	wdt_inten_type	350
4.24.3.2	wdt_resen_type	350
4.24.3.3	wdt_status	350
4.24.4	Функции	350
4.24.4.1	WDT_ClearStatusFlags()	350
4.24.4.2	WDT_Deinit()	351
4.24.4.3	WDT_Disable()	351
4.24.4.4	WDT_Enable()	351
4.24.4.5	WDT_GetDefaultConfig()	352
4.24.4.6	WDT_GetLastAPIStatus()	352
4.24.4.7	WDT_GetStatusFlagsMsk()	352
4.24.4.8	WDT_GetStatusFlagsRaw()	353
4.24.4.9	WDT_GetWarningValue()	353



4.24.4.10 WDT_Init()	353
4.24.4.11 WDT_Refresh()	354
4.24.4.12 WDT_SetTimeoutValue()	354
4.24.4.13 WDT_SetWarningValue()	355
5 Структуры данных	357
5.1 Структура _can_config	357
5.1.1 Подробное описание	357
5.1.2 Поля	357
5.1.2.1 can_fd_mode	357
5.1.2.2 enable_listen_only	358
5.1.2.3 enable_loopback_ext	358
5.1.2.4 enable_loopback_int	358
5.1.2.5 filter_config	358
5.1.2.6 ptb_config	358
5.1.2.7 rxb_config	358
5.1.2.8 stb_config	358
5.1.2.9 timing_config	359
5.2 Структура _can_frame_filter	359
5.2.1 Подробное описание	359
5.2.2 Поля	359
5.2.2.1 __pad0__	359
5.2.2.2 __pad1__	359
5.2.2.3 accepted_id	359
5.2.2.4 enable_id_check	360
5.2.2.5 id	360
5.2.2.6 mask	360
5.3 Структура _can_frame_filter_config	360
5.3.1 Подробное описание	360
5.3.2 Поля	360
5.3.2.1 filter	360
5.3.2.2 nb_filters_used	361
5.4 Структура _can_handle	361
5.4.1 Подробное описание	361
5.4.2 Поля	361
5.4.2.1 callback	361
5.4.2.2 rx_frames	361
5.4.2.3 rx_nb_frames_all	362
5.4.2.4 rx_nb_frames_rest	362
5.4.2.5 tx_frames_prim	362
5.4.2.6 tx_frames_sec	362
5.4.2.7 tx_nb_frames_all_prim	362
5.4.2.8 tx_nb_frames_all_sec	362

5.4.2.9 tx_nb_frames_rest_prim . . . . .	362
5.4.2.10 tx_nb_frames_rest_sec . . . . .	362
5.4.2.11 user_data . . . . .	363
5.5 Структура _can_ptb_config . . . . .	363
5.5.1 Подробное описание . . . . .	363
5.5.2 Поля . . . . .	363
5.5.2.1 tx_single_shot . . . . .	363
5.6 Структура _can_rx_buffer_frame . . . . .	363
5.6.1 Подробное описание . . . . .	364
5.6.2 Поля . . . . .	364
5.6.2.1 __pad0__ . . . . .	364
5.6.2.2 brs . . . . .	364
5.6.2.3 cycle_time . . . . .	364
5.6.2.4 data . . . . .	365
5.6.2.5 dlc . . . . .	365
5.6.2.6 esi . . . . .	365
5.6.2.7 fdf . . . . .	365
5.6.2.8 id . . . . .	365
5.6.2.9 ide . . . . .	365
5.6.2.10 koer . . . . .	365
5.6.2.11 rtr . . . . .	365
5.6.2.12 rts . . . . .	366
5.6.2.13 tx . . . . .	366
5.7 Структура _can_rx_transfer . . . . .	366
5.7.1 Подробное описание . . . . .	366
5.7.2 Поля . . . . .	366
5.7.2.1 frames . . . . .	366
5.7.2.2 nb_frames . . . . .	366
5.8 Структура _can_rxb_config . . . . .	367
5.8.1 Подробное описание . . . . .	367
5.8.2 Поля . . . . .	367
5.8.2.1 almost_full_level . . . . .	367
5.8.2.2 prohibit_overflow . . . . .	367
5.8.2.3 self_acknowledge . . . . .	367
5.9 Структура _can_stb_config . . . . .	367
5.9.1 Подробное описание . . . . .	368
5.9.2 Поля . . . . .	368
5.9.2.1 tx_discipline . . . . .	368
5.9.2.2 tx_single_shot . . . . .	368
5.10 Структура _can_timing_config . . . . .	368
5.10.1 Подробное описание . . . . .	369
5.10.2 Поля . . . . .	369
5.10.2.1 data_prescaler . . . . .	369

5.10.2.2 data_seg1 . . . . .	369
5.10.2.3 data_seg2 . . . . .	369
5.10.2.4 data_sjw . . . . .	369
5.10.2.5 delay_compensation_enable . . . . .	369
5.10.2.6 prescaler . . . . .	369
5.10.2.7 secondary_sample_point_offset . . . . .	369
5.10.2.8 seg1 . . . . .	370
5.10.2.9 seg2 . . . . .	370
5.10.2.10 sjw . . . . .	370
5.11 Структура _can_tx_buffer_frame . . . . .	370
5.11.1 Подробное описание . . . . .	370
5.11.2 Поля . . . . .	371
5.11.2.1 __pad0__ . . . . .	371
5.11.2.2 brs . . . . .	371
5.11.2.3 data . . . . .	371
5.11.2.4 dlc . . . . .	371
5.11.2.5 fdf . . . . .	371
5.11.2.6 id . . . . .	371
5.11.2.7 ide . . . . .	371
5.11.2.8 rtr . . . . .	372
5.11.2.9 ttsen . . . . .	372
5.12 Структура _can_tx_transfer . . . . .	372
5.12.1 Подробное описание . . . . .	372
5.12.2 Поля . . . . .	372
5.12.2.1 frames . . . . .	372
5.12.2.2 nb_frames . . . . .	372
5.13 Структура _dma_channel_config . . . . .	373
5.13.1 Подробное описание . . . . .	373
5.13.2 Поля . . . . .	373
5.13.2.1 ctx_cfg . . . . .	373
5.13.2.2 dst_addr . . . . .	373
5.13.2.3 is_dst_periph . . . . .	373
5.13.2.4 is_src_periph . . . . .	373
5.13.2.5 next_desc . . . . .	374
5.13.2.6 src_addr . . . . .	374
5.14 Структура _dma_channel_reg . . . . .	374
5.14.1 Подробное описание . . . . .	374
5.14.2 Поля . . . . .	375
5.14.2.1 CFG_HI . . . . .	375
5.14.2.2 CFG_LO . . . . .	375
5.14.2.3 CTL_HI . . . . .	375
5.14.2.4 CTL_LO . . . . .	375
5.14.2.5 DAR_HI . . . . .	375

5.14.2.6	DAR_LO	375
5.14.2.7	DSR_HI	375
5.14.2.8	DSR_LO	376
5.14.2.9	DSTAT_HI	376
5.14.2.10	DSTAT_LO	376
5.14.2.11	DSTATAR_HI	376
5.14.2.12	DSTATAR_LO	376
5.14.2.13	LLP_HI	376
5.14.2.14	LLP_LO	376
5.14.2.15	SAR_HI	376
5.14.2.16	SAR_LO	377
5.14.2.17	SGR_HI	377
5.14.2.18	SGR_LO	377
5.14.2.19	SSTAT_HI	377
5.14.2.20	SSTAT_LO	377
5.14.2.21	SSTATAR_HI	377
5.14.2.22	SSTATAR_LO	377
5.15	Структура <code>_dma_descriptor</code>	378
5.15.1	Подробное описание	378
5.15.2	Поля	378
5.15.2.1	CTL_HI	378
5.15.2.2	CTL_LO	378
5.15.2.3	DAR	378
5.15.2.4	DSTAT	378
5.15.2.5	LLP	379
5.15.2.6	SAR	379
5.15.2.7	SSTAT	379
5.16	Структура <code>_dma_handle</code>	379
5.16.1	Подробное описание	379
5.16.2	Поля	379
5.16.2.1	base	379
5.16.2.2	callback	380
5.16.2.3	channel	380
5.16.2.4	user_data	380
5.17	Структура <code>_dma_multiblock_config</code>	380
5.17.1	Подробное описание	380
5.17.2	Поля	381
5.17.2.1	count	381
5.17.2.2	data_size	381
5.17.2.3	dst_addr	381
5.17.2.4	dst_burst_size	381
5.17.2.5	dst_data_width	381
5.17.2.6	dst_incr	381

5.17.2.7	gather_en	381
5.17.2.8	int_en	382
5.17.2.9	scatter_en	382
5.17.2.10	src_addr	382
5.17.2.11	src_burst_size	382
5.17.2.12	src_data_width	382
5.17.2.13	src_incr	382
5.17.2.14	transfer_type	382
5.18	Структура <code>_i2c_master_dma_handle</code>	383
5.18.1	Подробное описание	383
5.18.2	Поля	383
5.18.2.1	buf	383
5.18.2.2	completion_callback	383
5.18.2.3	dummy_data	383
5.18.2.4	remaining_bytes_DMA	383
5.18.2.5	rx_desc	384
5.18.2.6	rx_dma	384
5.18.2.7	state	384
5.18.2.8	tx_desc	384
5.18.2.9	tx_dma	384
5.18.2.10	user_data	384
5.19	Структура <code>_i2c_master_handle</code>	384
5.19.1	Подробное описание	385
5.19.2	Поля	385
5.19.2.1	buf	385
5.19.2.2	check_addr_nack	385
5.19.2.3	completion_callback	385
5.19.2.4	remaining_bytes	385
5.19.2.5	remaining_subaddr	385
5.19.2.6	state	386
5.19.2.7	subaddr_buf	386
5.19.2.8	transfer	386
5.19.2.9	transfer_count	386
5.19.2.10	user_data	386
5.20	Структура <code>_i2c_slave_handle</code>	386
5.20.1	Подробное описание	387
5.20.2	Поля	387
5.20.2.1	callback	387
5.20.2.2	irq_num	387
5.20.2.3	slave_fsm	387
5.20.2.4	transfer	387
5.20.2.5	user_data	387
5.21	Структура <code>_i2s_config</code>	387

5.21.1	Подробное описание	388
5.21.2	Поля	388
5.21.2.1	interrupt_level	388
5.21.2.2	resolution	388
5.21.2.3	sample_rate	388
5.21.2.4	sclk_gating	388
5.21.2.5	sclk_per_sample	388
5.22	Структура _i2s_handle	389
5.22.1	Подробное описание	389
5.22.2	Поля	389
5.22.2.1	callback	389
5.22.2.2	left_samples	389
5.22.2.3	nb_samples	389
5.22.2.4	right_samples	389
5.22.2.5	user_data	390
5.23	Структура _i2s_transfer	390
5.23.1	Подробное описание	390
5.23.2	Поля	390
5.23.2.1	left_samples	390
5.23.2.2	nb_samples	390
5.23.2.3	right_samples	390
5.24	Структура _jtm_handle	391
5.24.1	Подробное описание	391
5.24.2	Поля	391
5.24.2.1	callback	391
5.24.2.2	parameter	391
5.24.2.3	user_data	391
5.25	Структура _nor_command_set	391
5.25.1	Подробное описание	392
5.25.2	Поля	392
5.25.2.1	erase_chip_cmd	392
5.25.2.2	erase_sector_cmd	392
5.25.2.3	page_write_memory_cmd	392
5.25.2.4	read_memory_command	392
5.25.2.5	read_status_cmd	392
5.25.2.6	write_disable_cmd	393
5.25.2.7	write_enable_cmd	393
5.25.2.8	write_status_cmd	393
5.26	Структура _nor_config	393
5.26.1	Подробное описание	393
5.26.2	Поля	393
5.26.2.1	driver_base_addr	393
5.26.2.2	mem_control_config	394

5.26.2.3 quad_control_config . . . . .	394
5.27 Структура _nor_handle . . . . .	394
5.27.1 Подробное описание . . . . .	394
5.27.2 Поля . . . . .	394
5.27.2.1 device_specific . . . . .	394
5.27.2.2 driver_base_addr . . . . .	395
5.27.2.3 max_chip_erase_time . . . . .	395
5.27.2.4 max_page_program_time . . . . .	395
5.27.2.5 max_sector_erase_time . . . . .	395
5.27.2.6 memory_size_bytes . . . . .	395
5.27.2.7 page_size_bytes . . . . .	395
5.27.2.8 qspi_config . . . . .	395
5.27.2.9 sector_size_bytes . . . . .	395
5.27.2.10 xip_config . . . . .	396
5.28 Структура _pwm_chopper . . . . .	396
5.28.1 Подробное описание . . . . .	396
5.29 Структура _pwm_dz_cfg . . . . .	396
5.29.1 Подробное описание . . . . .	397
5.30 Структура _pwm_handle . . . . .	397
5.30.1 Подробное описание . . . . .	397
5.30.2 Поля . . . . .	397
5.30.2.1 channel . . . . .	397
5.30.2.2 duty_cycle . . . . .	397
5.30.2.3 int_en . . . . .	398
5.30.2.4 int_freq . . . . .	398
5.30.2.5 int_source . . . . .	398
5.30.2.6 out . . . . .	398
5.30.2.7 period_us . . . . .	398
5.30.2.8 pwm_callback . . . . .	398
5.30.2.9 ref_clk . . . . .	398
5.31 Структура _pwm_trip_unit_cfg . . . . .	399
5.32 Структура _qspi_config . . . . .	399
5.32.1 Подробное описание . . . . .	399
5.32.2 Поля . . . . .	399
5.32.2.1 bit_size . . . . .	399
5.32.2.2 cont_trans_en . . . . .	400
5.32.2.3 cont_transfer_ext . . . . .	400
5.32.2.4 cpha . . . . .	400
5.32.2.5 cpol . . . . .	400
5.32.2.6 delay_en . . . . .	400
5.32.2.7 dma_en . . . . .	400
5.32.2.8 inhibit_din . . . . .	400
5.32.2.9 inhibit_dout . . . . .	400

5.32.2.10 mode . . . . .	401
5.32.2.11 msb . . . . .	401
5.32.2.12 slave_pol . . . . .	401
5.32.2.13 slave_select . . . . .	401
5.32.2.14 spi_mode . . . . .	401
5.33 Структура _qspi_nor_config . . . . .	401
5.33.1 Подробное описание . . . . .	402
5.33.2 Поля . . . . .	402
5.33.2.1 is_quad_need_enable . . . . .	402
5.33.2.2 quad_enable_bit_shift . . . . .	402
5.33.2.3 quad_enable_command . . . . .	402
5.33.2.4 quad_read_command . . . . .	402
5.33.2.5 write_two_status_bytes . . . . .	402
5.34 Структура _qspi_nor_handle . . . . .	402
5.34.1 Подробное описание . . . . .	403
5.34.2 Поля . . . . .	403
5.34.2.1 command_set . . . . .	403
5.34.2.2 command_type . . . . .	403
5.34.2.3 intermediate_len . . . . .	403
5.34.2.4 read_cmd_format . . . . .	403
5.35 Структура _qspi_nor_init_config . . . . .	403
5.35.1 Подробное описание . . . . .	404
5.35.2 Поля . . . . .	404
5.35.2.1 cmd_format . . . . .	404
5.35.2.2 quad_mode_setting . . . . .	404
5.36 Структура _qspi_xip_config . . . . .	404
5.36.1 Подробное описание . . . . .	405
5.36.2 Поля . . . . .	405
5.36.2.1 addr4 . . . . .	405
5.36.2.2 cmd . . . . .	405
5.36.2.3 cpha . . . . .	405
5.36.2.4 cpol . . . . .	405
5.36.2.5 dummy_cycles . . . . .	405
5.36.2.6 hp_end_dummy . . . . .	405
5.36.2.7 hp_mode . . . . .	405
5.36.2.8 hpen . . . . .	406
5.36.2.9 le32 . . . . .	406
5.37 Структура _rtc_datetime . . . . .	406
5.37.1 Подробное описание . . . . .	406
5.37.2 Поля . . . . .	406
5.37.2.1 day . . . . .	406
5.37.2.2 hour . . . . .	406
5.37.2.3 minute . . . . .	407



5.37.2.4 month . . . . .	407
5.37.2.5 second . . . . .	407
5.37.2.6 year . . . . .	407
5.38 Структура <code>_spi_dma_handle</code> . . . . .	407
5.38.1 Подробное описание . . . . .	408
5.38.2 Поля . . . . .	408
5.38.2.1 bytes_per_frame . . . . .	408
5.38.2.2 callback . . . . .	408
5.38.2.3 dummy_data . . . . .	408
5.38.2.4 rx_desc . . . . .	408
5.38.2.5 rx_handle . . . . .	408
5.38.2.6 rx_in_progress . . . . .	408
5.38.2.7 state . . . . .	408
5.38.2.8 transfer_size . . . . .	409
5.38.2.9 tx_desc . . . . .	409
5.38.2.10 tx_handle . . . . .	409
5.38.2.11 tx_in_progress . . . . .	409
5.38.2.12 user_data . . . . .	409
5.39 Структура <code>_ttcan_config</code> . . . . .	409
5.39.1 Подробное описание . . . . .	410
5.39.2 Поля . . . . .	410
5.39.2.1 prescaler . . . . .	410
5.39.2.2 reference_id . . . . .	410
5.39.2.3 reference_ide . . . . .	410
5.39.2.4 transmit_enable_window . . . . .	410
5.39.2.5 transmit_trigger_pointer . . . . .	410
5.39.2.6 trigger_time0 . . . . .	410
5.39.2.7 trigger_time1 . . . . .	410
5.39.2.8 trigger_type . . . . .	411
5.39.2.9 watch_trigger_time0 . . . . .	411
5.39.2.10 watch_trigger_time1 . . . . .	411
5.40 Структура <code>_uart_dma_handle</code> . . . . .	411
5.40.1 Подробное описание . . . . .	411
5.40.2 Поля . . . . .	412
5.40.2.1 base . . . . .	412
5.40.2.2 callback . . . . .	412
5.40.2.3 rx_data_size_all . . . . .	412
5.40.2.4 rx_dma_handle . . . . .	412
5.40.2.5 rx_state . . . . .	412
5.40.2.6 tx_data_size_all . . . . .	412
5.40.2.7 tx_dma_handle . . . . .	412
5.40.2.8 tx_state . . . . .	413
5.40.2.9 user_data . . . . .	413

5.41 Структура <code>clkctr_div</code> . . . . .	413
5.41.1 Подробное описание . . . . .	413
5.41.2 Поля . . . . .	413
5.41.2.1 <code>clkctr_fclk_div</code> . . . . .	413
5.41.2.2 <code>clkctr_gnssclk_div</code> . . . . .	413
5.41.2.3 <code>clkctr_i2sclk_div</code> . . . . .	414
5.41.2.4 <code>clkctr_mco_div</code> . . . . .	414
5.41.2.5 <code>clkctr_qspiclk_div</code> . . . . .	414
5.41.2.6 <code>clkctr_sysclk_div</code> . . . . .	414
5.42 Структура <code>clkctr_pll_cfg</code> . . . . .	414
5.42.1 Подробное описание . . . . .	414
5.42.2 Поля . . . . .	415
5.42.2.1 <code>lock</code> . . . . .	415
5.42.2.2 <code>man</code> . . . . .	415
5.42.2.3 <code>nf_man</code> . . . . .	415
5.42.2.4 <code>nr_man</code> . . . . .	415
5.42.2.5 <code>od_man</code> . . . . .	415
5.42.2.6 <code>sel</code> . . . . .	415
5.43 Структура <code>dma_channel_ctl_cfg</code> . . . . .	415
5.43.1 Подробное описание . . . . .	416
5.43.2 Поля . . . . .	416
5.43.2.1 <code>block_size</code> . . . . .	416
5.43.2.2 <code>dst_burst_size</code> . . . . .	416
5.43.2.3 <code>dst_incr</code> . . . . .	416
5.43.2.4 <code>dst_tr_width</code> . . . . .	416
5.43.2.5 <code>gather_en</code> . . . . .	417
5.43.2.6 <code>int_en</code> . . . . .	417
5.43.2.7 <code>llp_dst_en</code> . . . . .	417
5.43.2.8 <code>llp_src_en</code> . . . . .	417
5.43.2.9 <code>scatter_en</code> . . . . .	417
5.43.2.10 <code>src_burst_size</code> . . . . .	417
5.43.2.11 <code>src_incr</code> . . . . .	417
5.43.2.12 <code>src_tr_width</code> . . . . .	417
5.43.2.13 <code>transfer_type</code> . . . . .	418
5.44 Структура <code>dualtimer_hardware_config</code> . . . . .	418
5.44.1 Подробное описание . . . . .	418
5.44.2 Поля . . . . .	418
5.44.2.1 <code>bg_load</code> . . . . .	418
5.44.2.2 <code>cyclicity</code> . . . . .	418
5.44.2.3 <code>enable</code> . . . . .	419
5.44.2.4 <code>int_ctrl</code> . . . . .	419
5.44.2.5 <code>load</code> . . . . .	419
5.44.2.6 <code>mode</code> . . . . .	419

5.44.2.7 prescale . . . . .	419
5.44.2.8 size . . . . .	419
5.45 Структура i2c_master_config_t . . . . .	419
5.45.1 Подробное описание . . . . .	420
5.45.2 Поля . . . . .	420
5.45.2.1 baudrate_bps . . . . .	420
5.45.2.2 enable_master . . . . .	420
5.45.2.3 sda_hold . . . . .	420
5.45.2.4 sda_setup . . . . .	420
5.46 Структура i2c_master_transfer_t . . . . .	421
5.46.1 Подробное описание . . . . .	421
5.46.2 Поля . . . . .	421
5.46.2.1 addr_size . . . . .	421
5.46.2.2 data . . . . .	422
5.46.2.3 data_size . . . . .	422
5.46.2.4 direction . . . . .	422
5.46.2.5 flags . . . . .	422
5.46.2.6 slave_address . . . . .	422
5.46.2.7 subaddress . . . . .	422
5.46.2.8 subaddress_size . . . . .	422
5.47 Структура i2c_slave_config_t . . . . .	423
5.47.1 Подробное описание . . . . .	423
5.47.2 Поля . . . . .	423
5.47.2.1 ack_gen_call . . . . .	423
5.47.2.2 address . . . . .	423
5.47.2.3 enable_slave . . . . .	423
5.47.2.4 i2c_addr_size . . . . .	423
5.48 Структура i2c_slave_transfer_t . . . . .	424
5.48.1 Подробное описание . . . . .	424
5.48.2 Поля . . . . .	424
5.48.2.1 completion_status . . . . .	424
5.48.2.2 event . . . . .	424
5.48.2.3 event_mask . . . . .	424
5.48.2.4 handle . . . . .	424
5.48.2.5 rx_data . . . . .	425
5.48.2.6 rx_size . . . . .	425
5.48.2.7 transferred_count . . . . .	425
5.48.2.8 tx_data . . . . .	425
5.48.2.9 tx_size . . . . .	425
5.49 Структура jtm_config_t . . . . .	425
5.49.1 Подробное описание . . . . .	426
5.49.2 Поля . . . . .	426
5.49.2.1 tcal . . . . .	426

5.49.2.2 wcal	426
5.49.2.3 wtcalconf	426
5.49.2.4 wtconf	426
5.50 Структура power_config	426
5.50.1 Подробное описание	427
5.50.2 Поля	427
5.50.2.1 dcdc_enable	427
5.50.2.2 flash_low_voltage_read_enable	427
5.50.2.3 run_configuration	427
5.50.2.4 standby_configuration	427
5.50.2.5 trim_configuration	427
5.50.2.6 vlevel0	427
5.50.2.7 vlevel1	427
5.50.2.8 vlevel2	428
5.51 Структура power_handle	428
5.51.1 Подробное описание	428
5.51.2 Поля	428
5.51.2.1 callback	428
5.51.2.2 user_data	428
5.52 Структура power_mode_config	429
5.52.1 Подробное описание	429
5.52.2 Поля	429
5.52.2.1 apc_eco_threshold	429
5.52.2.2 apc_low_clk_enable	429
5.52.2.3 dcdc_ccm_enable	429
5.52.2.4 dcdc_level	429
5.52.2.5 dcdc_low_consumption_enable	430
5.52.2.6 dcdc_mode	430
5.52.2.7 dcdc_sink_enable	430
5.52.2.8 dcdc_swdrv	430
5.52.2.9 eco_mode	430
5.52.2.10 flash_power_mode	430
5.53 Структура power_state	430
5.53.1 Подробное описание	431
5.53.2 Поля	431
5.53.2.1 dcdc_is_ready	431
5.53.2.2 flash_low_voltage_read_enabled	431
5.53.2.3 flash_power_mode	431
5.53.2.4 vdda_is_lower_threshold	431
5.54 Структура power_trim_config	431
5.54.1 Подробное описание	432
5.54.2 Поля	432
5.54.2.1 apc_force_trim	432

5.54.2.2	apc_vref_it	432
5.54.2.3	apc_vref_tt	432
5.54.2.4	apc_vref_vt	432
5.54.2.5	dcde_imax	432
5.54.2.6	dcde_imin	433
5.54.2.7	dcde_trimlc	433
5.55	Структура ppu_config	433
5.55.1	Подробное описание	433
5.55.2	Поля	433
5.55.2.1	reqForCPU	433
5.56	Структура pwm_channel_config	434
5.56.1	Подробное описание	435
5.56.2	Поля	435
5.56.2.1	channel	435
5.56.2.2	chopper_duty	435
5.56.2.3	chopper_first_width	435
5.56.2.4	chopper_freq	435
5.56.2.5	chopper_work	435
5.56.2.6	cmd	436
5.56.2.7	cmpa	436
5.56.2.8	cmpb	436
5.56.2.9	cnt_eq_cmpa_dec_outa	436
5.56.2.10	cnt_eq_cmpa_dec_outb	436
5.56.2.11	cnt_eq_cmpa_inc_outa	436
5.56.2.12	cnt_eq_cmpa_inc_outb	436
5.56.2.13	cnt_eq_cmpb_dec_outa	436
5.56.2.14	cnt_eq_cmpb_dec_outb	437
5.56.2.15	cnt_eq_cmpb_inc_outa	437
5.56.2.16	cnt_eq_cmpb_inc_outb	437
5.56.2.17	cnt_eq_prd_outa	437
5.56.2.18	cnt_eq_prd_outb	437
5.56.2.19	cnt_eq_zero_outa	437
5.56.2.20	cnt_eq_zero_outb	437
5.56.2.21	cntmode	437
5.56.2.22	counter	438
5.56.2.23	ctrphs	438
5.56.2.24	dz_falling_edge_delay_clk	438
5.56.2.25	dz_falling_edge_outb_inv	438
5.56.2.26	dz_falling_edge_source	438
5.56.2.27	dz_outa_enable	438
5.56.2.28	dz_outb_enable	438
5.56.2.29	dz_rising_edge_delay_clk	438
5.56.2.30	dz_rising_edge_outa_inv	439

5.56.2.31	dz_rising_edge_source	439
5.56.2.32	eventprd	439
5.56.2.33	inputs_mask_mult	439
5.56.2.34	inputs_mask_one	439
5.56.2.35	ldamode	439
5.56.2.36	ldbmode	439
5.56.2.37	ldcswrf	440
5.56.2.38	loadprd	440
5.56.2.39	period	440
5.56.2.40	prescaler	440
5.56.2.41	prescaler_cmd	440
5.56.2.42	prescaler_divmux	440
5.56.2.43	prescaler_mode	440
5.56.2.44	prescaler_syncrst	441
5.56.2.45	pwm_int_enable	441
5.56.2.46	pwm_int_source	441
5.56.2.47	pwmtn_int_mult	441
5.56.2.48	pwmtn_int_one	441
5.56.2.49	scmpamode	441
5.56.2.50	scmpbmode	441
5.56.2.51	sw_forced_long_outa	441
5.56.2.52	sw_forced_long_outb	442
5.56.2.53	sw_forced_outa	442
5.56.2.54	sw_forced_outb	442
5.56.2.55	syncphsen	442
5.56.2.56	trip_unit_action_outa	442
5.56.2.57	trip_unit_action_outb	442
5.57	Структура qspr_dma_handle_t	442
5.57.1	Подробное описание	443
5.58	Структура rwc_alarm_reg	443
5.58.1	Подробное описание	443
5.58.2	Поля	443
5.58.2.1	alarm	443
5.59	Структура rwc_config	444
5.59.1	Подробное описание	444
5.59.2	Поля	444
5.59.2.1	alarm_en	444
5.59.2.2	alarm_time	444
5.59.2.3	clkdiv	445
5.59.2.4	general	445
5.59.2.5	lfe_bypass	445
5.59.2.6	osc_sel	445
5.59.2.7	pl	445

5.59.2.8 presc . . . . .	445
5.59.2.9 pz . . . . .	445
5.59.2.10 reset_en . . . . .	445
5.59.2.11 shutdown_force . . . . .	446
5.59.2.12 time . . . . .	446
5.59.2.13 time_clk_sel . . . . .	446
5.59.2.14 trim_lfe . . . . .	446
5.59.2.15 trim_lfi . . . . .	446
5.59.2.16 trimload . . . . .	446
5.59.2.17 wake_en . . . . .	446
5.59.2.18 wake_in_en . . . . .	446
5.59.2.19 wake_pol . . . . .	447
5.59.2.20 wake_stat1 . . . . .	447
5.60 Структура rwc_config_reg . . . . .	447
5.60.1 Подробное описание . . . . .	447
5.60.2 Поля . . . . .	448
5.60.2.1 __pad0__ . . . . .	448
5.60.2.2 __pad1__ . . . . .	448
5.60.2.3 __pad2__ . . . . .	448
5.60.2.4 alarm_en . . . . .	448
5.60.2.5 clk_div . . . . .	448
5.60.2.6 osc_sel . . . . .	448
5.60.2.7 pl . . . . .	448
5.60.2.8 pz . . . . .	449
5.60.2.9 reset_en . . . . .	449
5.60.2.10 shutdown_force . . . . .	449
5.60.2.11 time_clk_sel . . . . .	449
5.60.2.12 wake_in_en . . . . .	449
5.60.2.13 wake_stat1 . . . . .	449
5.61 Структура rwc_general_reg . . . . .	449
5.61.1 Подробное описание . . . . .	450
5.61.2 Поля . . . . .	450
5.61.2.1 time . . . . .	450
5.62 Структура rwc_time_reg . . . . .	450
5.62.1 Подробное описание . . . . .	450
5.62.2 Поля . . . . .	451
5.62.2.1 time . . . . .	451
5.63 Структура rwc_trim_reg . . . . .	451
5.63.1 Подробное описание . . . . .	451
5.63.2 Поля . . . . .	451
5.63.2.1 __pad0__ . . . . .	451
5.63.2.2 __pad1__ . . . . .	452
5.63.2.3 lfe_bypass . . . . .	452

5.63.2.4 trim_lfe . . . . .	452
5.63.2.5 trim_lfi . . . . .	452
5.63.2.6 wake_stat2 . . . . .	452
5.63.2.7 wake_stat3 . . . . .	452
5.64 Структура rwc_trimload_reg . . . . .	452
5.64.1 Подробное описание . . . . .	453
5.64.2 Поля . . . . .	453
5.64.2.1 __pad0__ . . . . .	453
5.64.2.2 trimload . . . . .	453
5.65 Объединение rwc_union_reg . . . . .	453
5.65.1 Подробное описание . . . . .	454
5.65.2 Поля . . . . .	454
5.65.2.1 alarm . . . . .	454
5.65.2.2 config . . . . .	454
5.65.2.3 general . . . . .	454
5.65.2.4 reg_value . . . . .	454
5.65.2.5 time . . . . .	454
5.65.2.6 trim . . . . .	454
5.65.2.7 trimload . . . . .	455
5.65.2.8 wake_config . . . . .	455
5.66 Структура rwc_wake_config_reg . . . . .	455
5.66.1 Подробное описание . . . . .	455
5.66.2 Поля . . . . .	455
5.66.2.1 __pad0__ . . . . .	455
5.66.2.2 __pad1__ . . . . .	456
5.66.2.3 wake_en . . . . .	456
5.66.2.4 wake_pol . . . . .	456
5.67 Структура sdmmc_card_t . . . . .	456
5.67.1 Подробное описание . . . . .	457
5.67.2 Поля . . . . .	457
5.67.2.1 cfg . . . . .	457
5.67.2.2 cid . . . . .	457
5.67.2.3 csd . . . . .	457
5.67.2.4 ddr_mode . . . . .	457
5.67.2.5 dev_num . . . . .	457
5.67.2.6 freq_divider . . . . .	457
5.67.2.7 freq_input . . . . .	457
5.67.2.8 gpio_vol . . . . .	458
5.67.2.9 hs_mode . . . . .	458
5.67.2.10 lock . . . . .	458
5.67.2.11 need_1V8en . . . . .	458
5.67.2.12 rca . . . . .	458
5.67.2.13 regs . . . . .	458



5.67.2.14	sdhc_mode	458
5.67.2.15	total_size	458
5.67.2.16	type	459
5.67.2.17	version	459
5.68	Структура sdmmc_cfg_pin_map	459
5.68.1	Подробное описание	459
5.68.2	Поля	459
5.68.2.1	CD	459
5.68.2.2	CK	460
5.68.2.3	CMD	460
5.68.2.4	D0	460
5.68.2.5	D1	460
5.68.2.6	D2	460
5.68.2.7	D3	460
5.68.2.8	D4	460
5.68.2.9	D5	460
5.68.2.10	D6	461
5.68.2.11	D7	461
5.68.2.12	max_current	461
5.68.2.13	pull_en	461
5.69	Структура sdmmc_port_cfg_t	461
5.69.1	Подробное описание	462
5.69.2	Поля	462
5.69.2.1	delay_us	462
5.69.2.2	emmc_8bit_en	462
5.69.2.3	freq_out	462
5.69.2.4	freq_out_init	462
5.69.2.5	hs_en	462
5.69.2.6	pin_map	462
5.69.2.7	reg_base	462
5.69.2.8	sd_uhs_mode	463
5.69.2.9	slot_count	463
5.69.2.10	slot_type	463
5.69.2.11	timeout_cd	463
5.69.2.12	timeout_init	463
5.70	Структура spi_config_internal_t	463
5.70.1	Подробное описание	464
5.70.2	Поля	464
5.70.2.1	frame_width_bits	464
5.70.2.2	frame_width_bytes	464
5.70.2.3	shift_dir	464
5.71	Структура spi_config_t	464
5.71.1	Подробное описание	465

5.71.2 Поля	465
5.71.2.1 baud_rate_bps	465
5.71.2.2 data_width_bits	465
5.71.2.3 direction	465
5.71.2.4 frame_format	465
5.71.2.5 loopback_enable	465
5.71.2.6 [struct]	465
5.71.2.7 microwire_cfg	465
5.71.2.8 motorola_cfg	466
5.72 Структура spi_half_duplex_transfer_t	466
5.72.1 Подробное описание	466
5.72.2 Поля	466
5.72.2.1 rx_data	466
5.72.2.2 rx_data_size	466
5.72.2.3 tx_data	466
5.72.2.4 tx_data_size	467
5.73 Структура spi_handle	467
5.73.1 Подробное описание	467
5.73.2 Поля	467
5.73.2.1 callback	467
5.73.2.2 frame_width_bits	468
5.73.2.3 frame_width_bytes	468
5.73.2.4 instance	468
5.73.2.5 mode	468
5.73.2.6 rx_data	468
5.73.2.7 rx_remaining_bytes	468
5.73.2.8 state	468
5.73.2.9 total_byte_to_transfer	468
5.73.2.10 tx_data	469
5.73.2.11 tx_remaining_bytes	469
5.73.2.12 user_data	469
5.74 Структура spi_microwire_cfg_t	469
5.74.1 Подробное описание	469
5.74.2 Поля	469
5.74.2.1 busy_ready_check	469
5.74.2.2 ctrl_word_len	470
5.74.2.3 single_serial	470
5.74.2.4 tx_rx	470
5.75 Структура spi_motorola_cfg_t	470
5.75.1 Подробное описание	470
5.75.2 Поля	470
5.75.2.1 cap_data	470
5.75.2.2 clk_pol	471

5.76 Структура <code>spi_transfer_t</code> . . . . .	471
5.76.1 Подробное описание . . . . .	471
5.76.2 Поля . . . . .	471
5.76.2.1 <code>data_size</code> . . . . .	471
5.76.2.2 <code>rx_data</code> . . . . .	471
5.76.2.3 <code>tx_data</code> . . . . .	472
5.77 Структура <code>timer_hardware_config</code> . . . . .	472
5.77.1 Подробное описание . . . . .	472
5.77.2 Поля . . . . .	472
5.77.2.1 <code>interrupt_enable</code> . . . . .	472
5.77.2.2 <code>reload_value</code> . . . . .	472
5.77.2.3 <code>start_enable</code> . . . . .	472
5.77.2.4 <code>start_value</code> . . . . .	473
5.77.2.5 <code>work_type</code> . . . . .	473
5.78 Структура <code>uart_config</code> . . . . .	473
5.78.1 Подробное описание . . . . .	473
5.78.2 Поля . . . . .	473
5.78.2.1 <code>baudrate_bps</code> . . . . .	473
5.78.2.2 <code>bit_count_per_char</code> . . . . .	474
5.78.2.3 <code>break_line</code> . . . . .	474
5.78.2.4 <code>enable_hardware_flow_control</code> . . . . .	474
5.78.2.5 <code>enable_infrared</code> . . . . .	474
5.78.2.6 <code>enable_loopback</code> . . . . .	474
5.78.2.7 <code>enable_parity</code> . . . . .	474
5.78.2.8 <code>enable_rxfifo</code> . . . . .	474
5.78.2.9 <code>enable_txfifo</code> . . . . .	474
5.78.2.10 <code>parity_manual</code> . . . . .	475
5.78.2.11 <code>parity_mode</code> . . . . .	475
5.78.2.12 <code>stop_bit_count</code> . . . . .	475
5.79 Структура <code>uart_handle</code> . . . . .	475
5.79.1 Подробное описание . . . . .	476
5.79.2 Поля . . . . .	476
5.79.2.1 <code>callback</code> . . . . .	476
5.79.2.2 <code>rx_data</code> . . . . .	476
5.79.2.3 <code>rx_data_size</code> . . . . .	476
5.79.2.4 <code>rx_data_size_all</code> . . . . .	476
5.79.2.5 <code>rx_ring_buffer</code> . . . . .	476
5.79.2.6 <code>rx_ring_buffer_head</code> . . . . .	476
5.79.2.7 <code>rx_ring_buffer_size</code> . . . . .	476
5.79.2.8 <code>rx_ring_buffer_tail</code> . . . . .	477
5.79.2.9 <code>rx_state</code> . . . . .	477
5.79.2.10 <code>tx_data</code> . . . . .	477
5.79.2.11 <code>tx_data_size</code> . . . . .	477

5.79.2.12 tx_data_size_all . . . . .	477
5.79.2.13 tx_ring_buffer . . . . .	477
5.79.2.14 tx_ring_buffer_head . . . . .	477
5.79.2.15 tx_ring_buffer_size . . . . .	477
5.79.2.16 tx_ring_buffer_tail . . . . .	478
5.79.2.17 tx_state . . . . .	478
5.79.2.18 user_data . . . . .	478
5.80 Структура uart_transfer . . . . .	478
5.80.1 Подробное описание . . . . .	478
5.80.2 Поля . . . . .	479
5.80.2.1 data_size . . . . .	479
5.80.2.2 rx_data . . . . .	479
5.80.2.3 tx_data . . . . .	479
5.81 Структура vtu_config . . . . .	479
5.81.1 Подробное описание . . . . .	479
5.81.2 Поля . . . . .	480
5.81.2.1 capture_edge_control1 . . . . .	480
5.81.2.2 capture_edge_control2 . . . . .	480
5.81.2.3 counter . . . . .	480
5.81.2.4 duty_cycle_capture . . . . .	480
5.81.2.5 interrupt_control . . . . .	480
5.81.2.6 mode . . . . .	480
5.81.2.7 period . . . . .	480
5.81.2.8 prescaler . . . . .	481
5.81.2.9 pwm_polarity . . . . .	481
5.81.2.10 pwm_polarity2 . . . . .	481
5.82 Структура wdt_config . . . . .	481
5.82.1 Подробное описание . . . . .	481
5.82.2 Поля . . . . .	481
5.82.2.1 inten . . . . .	481
5.82.2.2 load . . . . .	481
5.82.2.3 resen . . . . .	481
6 Файлы . . . . .	483
6.1 Файл devices/eliot1/drivers/hal_can.h . . . . .	483
6.1.1 Подробное описание . . . . .	487
6.2 hal_can.h . . . . .	487
6.3 Файл devices/eliot1/drivers/hal_clkctr.h . . . . .	492
6.3.1 Подробное описание . . . . .	497
6.4 hal_clkctr.h . . . . .	497
6.5 hal_common.h . . . . .	501
6.6 Файл devices/eliot1/drivers/hal_dma.h . . . . .	502
6.6.1 Подробное описание . . . . .	506

6.7	<code>hal_dma.h</code>	506
6.8	Файл <code>devices/eliot1/drivers/hal_dualtimer.h</code>	510
6.8.1	Подробное описание	511
6.9	<code>hal_dualtimer.h</code>	511
6.10	Файл <code>devices/eliot1/drivers/hal_flash.h</code>	512
6.10.1	Подробное описание	513
6.11	<code>hal_flash.h</code>	513
6.12	Файл <code>devices/eliot1/drivers/hal_gpio.h</code>	514
6.12.1	Макросы	517
6.12.1.1	<code>GPIO_SEC_INT_PORT_NONSEC_MASK</code>	517
6.12.1.2	<code>GPIO_SEC_INT_SEC_ACC_MASK</code>	517
6.12.2	Перечисления	517
6.12.2.1	<code>gpio_int_type_t</code>	517
6.12.2.2	<code>gpio_pin_direction_t</code>	518
6.12.2.3	<code>gpio_pin_max_current_t</code>	519
6.12.2.4	<code>gpio_pin_pupd_t</code>	519
6.12.3	Функции	519
6.12.3.1	<code>GPIO_GetInstance()</code>	519
6.12.3.2	<code>GPIO_PinIRQ_Clear()</code>	520
6.12.3.3	<code>GPIO_PinIRQ_Disable()</code>	520
6.12.3.4	<code>GPIO_PinIRQ_Enable()</code>	520
6.12.3.5	<code>GPIO_PinIRQ_GetStatus()</code>	520
6.12.3.6	<code>GPIO_PinMode_Function()</code>	521
6.12.3.7	<code>GPIO_PinMode_Get()</code>	521
6.12.3.8	<code>GPIO_PinMode_GPIO()</code>	521
6.12.3.9	<code>GPIO_PinMode_HiZ()</code>	522
6.12.3.10	<code>GPIO_PinRead()</code>	522
6.12.3.11	<code>GPIO_PinSet_MaxCurrent()</code>	522
6.12.3.12	<code>GPIO_PinSet_PUPD()</code>	522
6.12.3.13	<code>GPIO_PinSet_Schmitt()</code>	523
6.12.3.14	<code>GPIO_PinSet_SpeedRaise()</code>	523
6.12.3.15	<code>GPIO_PinToggle()</code>	523
6.12.3.16	<code>GPIO_PinWrite()</code>	523
6.12.3.17	<code>GPIO_PortIRQ_Clear()</code>	524
6.12.3.18	<code>GPIO_PortIRQ_Disable()</code>	524
6.12.3.19	<code>GPIO_PortIRQ_Enable()</code>	524
6.12.3.20	<code>GPIO_PortIRQ_GetStatus()</code>	524
6.12.3.21	<code>GPIO_PortMode_Function()</code>	525
6.12.3.22	<code>GPIO_PortMode_GPIO()</code>	525
6.12.3.23	<code>GPIO_PortMode_HiZ()</code>	525
6.12.3.24	<code>GPIO_PortRead()</code>	526
6.12.3.25	<code>GPIO_PortSecureIRQ_Clear()</code>	526
6.12.3.26	<code>GPIO_PortSecureIRQ_GetInfo1()</code>	526

6.12.3.27	GPIO_PortSecureIRQ_GetInfo2()	526
6.12.3.28	GPIO_PortSecureIRQ_GetStatus()	527
6.12.3.29	GPIO_PortSecureIRQ_SetMask()	527
6.12.3.30	GPIO_PortSet_MaxCurrent()	527
6.12.3.31	GPIO_PortSet_NonsecureMask()	528
6.12.3.32	GPIO_PortSet_PUPD()	528
6.12.3.33	GPIO_PortSet_Schmitt()	528
6.12.3.34	GPIO_PortSet_SpeedRaise()	529
6.12.3.35	GPIO_PortToggle()	529
6.12.3.36	GPIO_PortWrite()	529
6.13	hal_gpio.h	530
6.14	Файл devices/eliot1/drivers/hal_i2c.h	531
6.14.1	Подробное описание	535
6.14.2	Функции	535
6.14.2.1	I2C_SlaveDeinit()	535
6.14.2.2	I2C_SlaveEnable()	536
6.14.2.3	I2C_SlaveGetDefaultConfig()	536
6.14.2.4	I2C_SlaveInit()	536
6.14.2.5	I2C_SlaveReadBlocking()	537
6.14.2.6	I2C_SlaveSetAddress()	537
6.14.2.7	I2C_SlaveSetReceiveBuffer()	538
6.14.2.8	I2C_SlaveSetSendBuffer()	538
6.14.2.9	I2C_SlaveTransferAbort()	539
6.14.2.10	I2C_SlaveTransferCreateHandle()	540
6.14.2.11	I2C_SlaveTransferGetCount()	540
6.14.2.12	I2C_SlaveTransferHandleIRQ()	541
6.14.2.13	I2C_SlaveTransferNonBlocking()	541
6.14.2.14	I2C_SlaveWriteBlocking()	542
6.15	hal_i2c.h	543
6.16	Файл devices/eliot1/drivers/hal_i2c_dma.h	548
6.16.1	Подробное описание	549
6.17	hal_i2c_dma.h	549
6.18	Файл devices/eliot1/drivers/hal_i2s.h	550
6.18.1	Подробное описание	553
6.19	hal_i2s.h	553
6.20	Файл devices/eliot1/drivers/hal_ioim.h	555
6.20.1	Подробное описание	556
6.21	hal_ioim.h	556
6.22	hal_jtm.h	556
6.23	Файл devices/eliot1/drivers/hal_mhu.h	557
6.23.1	Подробное описание	557
6.24	hal_mhu.h	558
6.25	Файл devices/eliot1/drivers/hal_nor_flash.h	558

6.25.1 Подробное описание . . . . .	560
6.26 hal_nor_flash.h . . . . .	560
6.27 Файл devices/eliot1/drivers/hal_power.h . . . . .	561
6.27.1 Подробное описание . . . . .	563
6.28 hal_power.h . . . . .	563
6.29 Файл devices/eliot1/drivers/hal_ppu.h . . . . .	565
6.29.1 Подробное описание . . . . .	567
6.30 hal_ppu.h . . . . .	567
6.31 Файл devices/eliot1/drivers/hal_pwm.h . . . . .	569
6.31.1 Подробное описание . . . . .	571
6.32 hal_pwm.h . . . . .	571
6.33 hal_pwm_newgen.h . . . . .	575
6.34 Файл devices/eliot1/drivers/hal_qspi.h . . . . .	577
6.34.1 Подробное описание . . . . .	578
6.35 hal_qspi.h . . . . .	579
6.36 Файл devices/eliot1/drivers/hal_qspi_dma.h . . . . .	581
6.36.1 Подробное описание . . . . .	582
6.37 hal_qspi_dma.h . . . . .	582
6.38 Файл devices/eliot1/drivers/hal_qspi_nor_flash.h . . . . .	583
6.38.1 Подробное описание . . . . .	584
6.39 hal_qspi_nor_flash.h . . . . .	585
6.40 Файл devices/eliot1/drivers/hal_rwc.h . . . . .	586
6.40.1 Подробное описание . . . . .	589
6.40.2 Функции . . . . .	589
6.40.2.1 RWC_ConvertDatetimeToSeconds() . . . . .	589
6.41 hal_rwc.h . . . . .	590
6.42 Файл devices/eliot1/drivers/hal_sdmmc.h . . . . .	593
6.42.1 Подробное описание . . . . .	596
6.43 hal_sdmmc.h . . . . .	596
6.44 Файл devices/eliot1/drivers/hal_sdmmc_cfg.h . . . . .	597
6.44.1 Подробное описание . . . . .	598
6.45 hal_sdmmc_cfg.h . . . . .	598
6.46 Файл devices/eliot1/drivers/hal_smc.h . . . . .	598
6.46.1 Подробное описание . . . . .	599
6.47 hal_smc.h . . . . .	600
6.48 Файл devices/eliot1/drivers/hal_spi.h . . . . .	601
6.48.1 Подробное описание . . . . .	604
6.49 hal_spi.h . . . . .	605
6.50 Файл devices/eliot1/drivers/hal_spi_dma.h . . . . .	610
6.50.1 Подробное описание . . . . .	611
6.51 hal_spi_dma.h . . . . .	611
6.52 Файл devices/eliot1/drivers/hal_timer.h . . . . .	613
6.52.1 Подробное описание . . . . .	614

6.53	<a href="#">hal_timer.h</a>	614
6.54	<a href="#">Файл devices/eliot1/drivers/hal_uart.h</a>	615
6.54.1	<a href="#">Подробное описание</a>	619
6.55	<a href="#">hal_uart.h</a>	619
6.56	<a href="#">Файл devices/eliot1/drivers/hal_uart_dma.h</a>	623
6.56.1	<a href="#">Подробное описание</a>	624
6.57	<a href="#">hal_uart_dma.h</a>	624
6.58	<a href="#">Файл devices/eliot1/drivers/hal_vtu.h</a>	625
6.58.1	<a href="#">Подробное описание</a>	627
6.59	<a href="#">hal_vtu.h</a>	627
6.60	<a href="#">Файл devices/eliot1/drivers/hal_wdt.h</a>	629
6.60.1	<a href="#">Подробное описание</a>	630
6.61	<a href="#">hal_wdt.h</a>	630
	<a href="#">Предметный указатель</a>	633



# Глава 1

## Topic Index

### 1.1 Topics

Here is a list of all topics with brief descriptions:

Драйвер модуля CAN . . . . .	9
Драйвер модуля CLKCTR . . . . .	34
Общие определения для библиотеки HAL . . . . .	78
Драйвер модуля DMA . . . . .	83
Драйвер модуля DTIM . . . . .	100
Драйвер модуля FLASH. . . . .	107
Драйвер модуля GPIO . . . . .	116
Драйвер модуля I2C . . . . .	118
Драйвер модуля I2S . . . . .	144
Драйвер менеджера прерываний IO устройств и DMA каналов . . . . .	156
Драйвер модуля JTM . . . . .	159
Драйвер модуля программных прерываний MHU . . . . .	163
Драйвер модуля POWER . . . . .	165
Драйвер модуля PPU . . . . .	176
Драйвер модуля PWM . . . . .	188
Драйвер модуля QSPI . . . . .	207
Драйвер модуля RWC . . . . .	229
Драйвер модуля SDMMC . . . . .	248
Драйвер модуля SMC . . . . .	260
Драйвер модуля SPI . . . . .	269
Драйвер модуля TIM . . . . .	302
Драйвер модуля UART . . . . .	310
Драйвер модуля VTU . . . . .	336
Драйвер модуля WDT . . . . .	348



## Глава 2

# Алфавитный указатель структур данных

### 2.1 Структуры данных

Структуры данных с их кратким описанием.

<a href="#">_can_config</a>	Структура конфигурации контроллера CAN . . . . .	357
<a href="#">_can_frame_filter</a>	Фильтр принятых кадров CAN . . . . .	359
<a href="#">_can_frame_filter_config</a>	Конфигурация фильтрации принятых кадров CAN . . . . .	360
<a href="#">_can_handle</a>	Структура дескриптора драйвера CAN . . . . .	361
<a href="#">_can_ptb_config</a>	Параметры высокоприоритетного буфера выдачи . . . . .	363
<a href="#">_can_rx_buffer_frame</a>	Структура буфера приема кадра CAN . . . . .	363
<a href="#">_can_rx_transfer</a>	Структура для приема кадра CAN в неблокирующем режиме (по прерыванию) . . . . .	366
<a href="#">_can_rxb_config</a>	Параметры буфера приема . . . . .	367
<a href="#">_can_stb_config</a>	Параметры низкоприоритетного буфера выдачи . . . . .	367
<a href="#">_can_timing_config</a>	Временные параметры передачи битов CAN . . . . .	368
<a href="#">_can_tx_buffer_frame</a>	Структура буфера передачи кадра CAN . . . . .	370
<a href="#">_can_tx_transfer</a>	Структура для передачи кадра CAN в неблокирующем режиме (по прерыванию) . . . . .	372
<a href="#">_dma_channel_config</a>	Конфигурация канала DMA . . . . .	373
<a href="#">_dma_channel_reg</a>	Дескриптор на регистры канала DMA . . . . .	374
<a href="#">_dma_descriptor</a>	Описатель следующего блока DMA (LLI) . . . . .	378
<a href="#">_dma_handle</a>	Управляющая структура передачи . . . . .	379
<a href="#">_dma_multiblock_config</a>	Конфигурация многоблочной передачи . . . . .	380
<a href="#">_i2c_master_dma_handle</a>	Контекст данных прерывания I2C-DMA . . . . .	383

<a href="#">_i2c_master_handle</a>	Дескриптор для работы по прерыванию . . . . .	384
<a href="#">_i2c_slave_handle</a>	I2C Slave-дескриптор . . . . .	386
<a href="#">_i2s_config</a>	Структура конфигурации контроллера I2S . . . . .	387
<a href="#">_i2s_handle</a>	Структура обработчика драйвера I2S . . . . .	389
<a href="#">_i2s_transfer</a>	Структура для передачи данных I2S в неблокирующем режиме (по прерыванию) . . . . .	390
<a href="#">_jtm_handle</a>	Структура обработчика событий JTM . . . . .	391
<a href="#">_nor_command_set</a>	Основной набор команд для флеш-памяти NOR . . . . .	391
<a href="#">_nor_config</a>	Структура первичной конфигурации флеш-памяти NOR . . . . .	393
<a href="#">_nor_handle</a>	Контекст драйвера флеш-памяти NOR . . . . .	394
<a href="#">_pwm_chopper</a>	Конфигурация дробления сигнала ШИМ . . . . .	396
<a href="#">_pwm_dz_cfg</a>	Конфигурация мертвой зоны ШИМ . . . . .	396
<a href="#">_pwm_handle</a>	Контекст драйвера ШИМ . . . . .	397
<a href="#">_pwm_trip_unit_cfg</a>	. . . . .	399
<a href="#">_qspi_config</a>	Количество бит во фрейме . . . . .	399
<a href="#">_qspi_nor_config</a>	Конфигурационный блок для режима Quad . . . . .	401
<a href="#">_qspi_nor_handle</a>	Контекст драйвера NOR Flash . . . . .	402
<a href="#">_qspi_nor_init_config</a>	Первоначальная конфигурация QSPI . . . . .	403
<a href="#">_qspi_xip_config</a>	Структура параметров конфигурации XIP контроллера QSPI . . . . .	404
<a href="#">_rtc_datetime</a>	Структура используемая для хранения даты и времени . . . . .	406
<a href="#">_spi_dma_handle</a>	Дескриптор SPI-DMA . . . . .	407
<a href="#">_ttcan_config</a>	Временные параметры передачи битов CAN . . . . .	409
<a href="#">_uart_dma_handle</a>	Дескриптор UART-DMA передачи . . . . .	411
<a href="#">clkctr_div</a>	Делители блока . . . . .	413
<a href="#">clkctr_pll_cfg</a>	Коэффициенты PLL . . . . .	414
<a href="#">dma_channel_ctl_cfg</a>	Описание конфигурации пересылки . . . . .	415
<a href="#">dualtimer_hardware_config</a>	Конфигурация аппаратной части двоянного таймера . . . . .	418
<a href="#">i2c_master_config_t</a>	Структура с настройками для инициализации Master-модуля I2C . . . . .	419
<a href="#">i2c_master_transfer_t</a>	Структура дескриптора для неблокирующего обмена . . . . .	421
<a href="#">i2c_slave_config_t</a>	Структура с настройками для инициализации Slave-модуля I2C . . . . .	423

<a href="#">i2c_slave_transfer_t</a>	I2C Slave структура обмена данными . . . . .	424
<a href="#">jtm_config_t</a>	Структура конфигурации JTM . . . . .	425
<a href="#">power_config</a>	Структура конфигурации блока POWER . . . . .	426
<a href="#">power_handle</a>	Структура обработчика драйвера I2S . . . . .	428
<a href="#">power_mode_config</a>	Структура параметров режима питания . . . . .	429
<a href="#">power_state</a>	Структура параметров состояния блока POWER . . . . .	430
<a href="#">power_trim_config</a>	Структура подстроечных параметров APC и DC-DC . . . . .	431
<a href="#">ppu_config</a>	Структура для инициализации . . . . .	433
<a href="#">pwm_channel_config</a>	Конфигурация канала широтно-импульсного модулятора . . . . .	434
<a href="#">qspi_dma_handle_t</a>	Дескриптор QSPI-DMA передачи . . . . .	442
<a href="#">rwc_alarm_reg</a>	Регистр времени пробуждения . . . . .	443
<a href="#">rwc_config</a>	Структура используемая для конфигурирования RWC . . . . .	444
<a href="#">rwc_config_reg</a>	Конфигурационный регистр . . . . .	447
<a href="#">rwc_general_reg</a>	Регистр общего назначения . . . . .	449
<a href="#">rwc_time_reg</a>	Регистр текущего значения счетчика времени . . . . .	450
<a href="#">rwc_trim_reg</a>	Регистр подстройки осцилляторов . . . . .	451
<a href="#">rwc_trimload_reg</a>	Регистр записи значения подстройки из регистра TRIM . . . . .	452
<a href="#">rwc_union_reg</a>	Объединение для доступа к регистрам . . . . .	453
<a href="#">rwc_wake_config_reg</a>	Регистр настройки контроллера пробуждения . . . . .	455
<a href="#">sdmmc_card_t</a>	Контекст драйвера контроллера SDMMC . . . . .	456
<a href="#">sdmmc_cfg_pin_map</a>	Конфигурация выводов контроллера SDMMC . . . . .	459
<a href="#">sdmmc_port_cfg_t</a>	Конфигурация контроллера SDMMC . . . . .	461
<a href="#">spi_config_internal_t</a>	Внутренняя структура конфигурации модуля SPI . . . . .	463
<a href="#">spi_config_t</a>	Структура конфигурации для Master SPI . . . . .	464
<a href="#">spi_half_duplex_transfer_t</a>	Структура SPI для полудуплексной приемо-передачи в режиме Master . . . . .	466
<a href="#">spi_handle</a>	SPI структура дескриптора для работы по прерыванию . . . . .	467
<a href="#">spi_microwire_cfg_t</a>	Конфигурация для протокола Microwire National Semiconductor . . . . .	469
<a href="#">spi_motorola_cfg_t</a>	Конфигурация для протокола Motorola SPI . . . . .	470
<a href="#">spi_transfer_t</a>	Структура SPI для приемо-передачи . . . . .	471

<a href="#">timer_hardware_config</a>	Конфигурация аппаратной части таймера общего назначения . . . . .	472
<a href="#">uart_config</a>	Конфигурация UART . . . . .	473
<a href="#">uart_handle</a>	Дескриптор состояния приема/передачи для неблокирующих функций обмена . .	475
<a href="#">uart_transfer</a>	Указатель на буфер приема или передачи . . . . .	478
<a href="#">vtu_config</a>	Структура для конфигурации VTU . . . . .	479
<a href="#">wdt_config</a>	Структура инициализации сторожевого таймера . . . . .	481

## Глава 3

# Список файлов

### 3.1 Файлы

Полный список документированных файлов.

devices/eliot1/drivers/ <a href="#">hal_can.h</a>	
Интерфейс драйвера модуля ввода-вывода по интерфейсу CAN . . . . .	483
devices/eliot1/drivers/ <a href="#">hal_clkctr.h</a>	
Интерфейс драйвера модуля CLKCTR . . . . .	492
devices/eliot1/drivers/ <a href="#">hal_common.h</a> . . . . .	501
devices/eliot1/drivers/ <a href="#">hal_dma.h</a>	
Интерфейс драйвера прямого доступа к памяти . . . . .	502
devices/eliot1/drivers/ <a href="#">hal_dualtimer.h</a>	
Интерфейс драйвера сдвоенного таймера . . . . .	510
devices/eliot1/drivers/ <a href="#">hal_flash.h</a>	
Интерфейс драйвера модуля FLASH . . . . .	512
devices/eliot1/drivers/ <a href="#">hal_gpio.h</a> . . . . .	514
devices/eliot1/drivers/ <a href="#">hal_i2c.h</a>	
Интерфейс драйвера модуля I2C . . . . .	531
devices/eliot1/drivers/ <a href="#">hal_i2c_dma.h</a>	
Дополнение драйвера I2C с пересылкой данных через DMA . . . . .	548
devices/eliot1/drivers/ <a href="#">hal_i2s.h</a>	
Интерфейс драйвера модуля I2S . . . . .	550
devices/eliot1/drivers/ <a href="#">hal_ioim.h</a>	
Интерфейс менеджера прерываний IO устройств . . . . .	555
devices/eliot1/drivers/ <a href="#">hal_jtm.h</a> . . . . .	556
devices/eliot1/drivers/ <a href="#">hal_mhu.h</a>	
Интерфейс модуля программных прерываний MHU . . . . .	557
devices/eliot1/drivers/ <a href="#">hal_nor_flash.h</a>	
Интерфейс драйвера флеш-памяти NOR . . . . .	558
devices/eliot1/drivers/ <a href="#">hal_power.h</a>	
Интерфейс драйвера модуля POWER . . . . .	561
devices/eliot1/drivers/ <a href="#">hal_ppu.h</a>	
Интерфейс драйвера модуля PPU . . . . .	565
devices/eliot1/drivers/ <a href="#">hal_pwm.h</a>	
Интерфейс драйвера модуля широтно-импульсного модулятора . . . . .	569
devices/eliot1/drivers/ <a href="#">hal_pwm_newgen.h</a> . . . . .	575
devices/eliot1/drivers/ <a href="#">hal_qspi.h</a>	
Интерфейс драйвера модуля QSPI . . . . .	577
devices/eliot1/drivers/ <a href="#">hal_qspi_dma.h</a>	
Дополнение драйвера QSPI для обмена данными с помощью DMA . . . . .	581

devices/eliot1/drivers/ <a href="#">hal_qspi_nor_flash.h</a>	
Интерфейс драйвера модуля QSPI-NOR-FLASH . . . . .	583
devices/eliot1/drivers/ <a href="#">hal_rwc.h</a>	
Интерфейс драйвера модуля RWC . . . . .	586
devices/eliot1/drivers/ <a href="#">hal_sdmmc.h</a>	
Интерфейс драйвера модуля SDMMC . . . . .	593
devices/eliot1/drivers/ <a href="#">hal_sdmmc_cfg.h</a>	
Интерфейс конфигурации модуля SDMMC . . . . .	597
devices/eliot1/drivers/ <a href="#">hal_smc.h</a>	
Интерфейс драйвера внешней статической памяти . . . . .	598
devices/eliot1/drivers/ <a href="#">hal_spi.h</a>	
Интерфейс драйвера модуля SPI . . . . .	601
devices/eliot1/drivers/ <a href="#">hal_spi_dma.h</a>	
Дополнение драйвера SPI с пересылкой данных через DMA . . . . .	610
devices/eliot1/drivers/ <a href="#">hal_timer.h</a>	
Интерфейс драйвера модуля таймеров общего назначения . . . . .	613
devices/eliot1/drivers/ <a href="#">hal_uart.h</a>	
Интерфейс драйвера UART . . . . .	615
devices/eliot1/drivers/ <a href="#">hal_uart_dma.h</a>	
Дополнение драйвера UART с пересылкой данных через DMA . . . . .	623
devices/eliot1/drivers/ <a href="#">hal_vtu.h</a>	
Интерфейс драйвера универсального блока таймеров . . . . .	625
devices/eliot1/drivers/ <a href="#">hal_wdt.h</a>	
Интерфейс драйвера сторожевого таймера . . . . .	629



## Глава 4

# Topic Documentation

### 4.1 Драйвер модуля CAN

Драйвер ввода-вывода по последовательному шинному интерфейсу CAN.

#### Файлы

- файл [hal\\_can.h](#)  
Интерфейс драйвера модуля ввода-вывода по интерфейсу CAN.

#### Структуры данных

- struct [\\_can\\_tx\\_buffer\\_frame](#)  
Структура буфера передачи кадра CAN.
- struct [\\_can\\_rx\\_buffer\\_frame](#)  
Структура буфера приема кадра CAN.
- struct [\\_can\\_frame\\_filter](#)  
Фильтр принятых кадров CAN.
- struct [\\_can\\_frame\\_filter\\_config](#)  
Конфигурация фильтрации принятых кадров CAN.
- struct [\\_tcan\\_config](#)  
Временные параметры передачи битов CAN.
- struct [\\_can\\_timing\\_config](#)  
Временные параметры передачи битов CAN.
- struct [\\_can\\_ptb\\_config](#)  
Параметры высокоприоритетного буфера выдачи
- struct [\\_can\\_stb\\_config](#)  
Параметры низкоприоритетного буфера выдачи
- struct [\\_can\\_rxb\\_config](#)  
Параметры буфера приема
- struct [\\_can\\_config](#)  
Структура конфигурации контроллера CAN.
- struct [\\_can\\_tx\\_transfer](#)  
Структура для передачи кадра CAN в неблокирующем режиме (по прерыванию)
- struct [\\_can\\_rx\\_transfer](#)  
Структура для приема кадра CAN в неблокирующем режиме (по прерыванию)
- struct [\\_can\\_handle](#)  
Структура дескриптора драйвера CAN.

## Макросы

- `#define HAL_CAN_DRIVER_VERSION (MAKE_VERSION(1, 1, 0))`  
Версия драйвера CAN.
- `#define CAN_NB_OF_FILTERS 4`  
Количество входных фильтров CAN.

## Определения типов

- `typedef enum __can_status can_status_t`  
Коды возврата функций драйвера CAN.
- `typedef enum __can_kind_of_error can_kind_of_error_t`  
Виды ошибок на линии CAN.
- `typedef enum __can_flag can_flag_t`  
Флаги прерываний и состояний CAN.
- `typedef enum __canfd_mode canfd_mode_t`  
Режимы работы по протоколу CAN FD (есть отличия в расчете контрольной суммы и формировании битов стаффинга)
- `typedef enum __can_stb_discipline can_stb_discipline_t`  
Дисциплина выдачи из низкоприоритетного буфера
- `typedef enum __can_bytes_in_datafield can_bytes_in_datafield_t`  
Размер данных кадра CAN, указываемый в поле DLC.
- `typedef struct __can_tx_buffer_frame can_tx_buffer_frame_t`  
Структура буфера передачи кадра CAN.
- `typedef struct __can_rx_buffer_frame can_rx_buffer_frame_t`  
Структура буфера приема кадра CAN.
- `typedef struct __can_frame_filter can_frame_filter_t`  
Фильтр принятых кадров CAN.
- `typedef struct __can_frame_filter_config can_frame_filter_config_t`  
Конфигурация фильтрации принятых кадров CAN.
- `typedef enum __ttcan_timer_prescaler ttcan_timer_prescaler_t`  
Символьные константы значений делителя блока TTCAN.
- `typedef enum __ttcan_trigger_type ttcan_trigger_type_t`  
Тип триггера TTCAN.
- `typedef struct __ttcan_config ttcan_config_t`  
Временные параметры передачи битов CAN.
- `typedef struct __can_timing_config can_timing_config_t`  
Временные параметры передачи битов CAN.
- `typedef struct __can_ptb_config can_ptb_config_t`  
Параметры высокоприоритетного буфера выдачи
- `typedef struct __can_stb_config can_stb_config_t`  
Параметры низкоприоритетного буфера выдачи
- `typedef struct __can_rxb_config can_rxb_config_t`  
Параметры буфера приема
- `typedef struct __can_config can_config_t`  
Структура конфигурации контроллера CAN.
- `typedef struct __can_tx_transfer can_tx_transfer_t`  
Структура для передачи кадра CAN в неблокирующем режиме (по прерыванию)
- `typedef struct __can_rx_transfer can_rx_transfer_t`  
Структура для приема кадра CAN в неблокирующем режиме (по прерыванию)
- `typedef struct __can_handle can_handle_t`  
Декларация типа дескриптора драйвера CAN.
- `typedef void(* can_transfer_callback_t)(CAN_Type *base, can_handle_t *handle, can_status_t status, can_flag_t interrupt_flag, void *user_data)`  
Функция обратного вызова CAN.

## Перечисления

- enum `_can_status`  
Коды возврата функций драйвера CAN.
- enum `_can_kind_of_error`  
Виды ошибок на линии CAN.
- enum `_can_flag`  
Флаги прерываний и состояний CAN.
- enum `_canfd_mode`  
Режимы работы по протоколу CAN FD (есть отличия в расчете контрольной суммы и формировании битов стаффинга)
- enum `_can_stb_discipline`  
Дисциплина выдачи из низкоприоритетного буфера
- enum `_can_bytes_in_datafield`  
Размер данных кадра CAN, указываемый в поле DLC.
- enum `_ttcan_timer_prescaler`  
Символьные константы значений делителя блока TTCAN.
- enum `_ttcan_trigger_type`  
Тип триггера TTCAN.

## Инициализация и деинициализация

- `can_status_t CAN_Init (CAN_Type *base, const can_config_t *config)`  
Инициализация драйвера CAN.
- `void CAN_Deinit (CAN_Type *base)`  
Деинициализация драйвера CAN.
- `void CAN_GetDefaultConfig (can_config_t *config, uint32_t source_clock_hz)`  
Получение параметров драйвера CAN по умолчанию
- `void CAN_EnterNormalMode (CAN_Type *base)`  
Переключение контроллера CAN в рабочий режим
- `void CAN_EnterStandbyMode (CAN_Type *base)`  
Переключение контроллера CAN в режим ожидания

## Конфигурирование настроек приемопередачи

- `bool CAN_CalculateImprovedTimingValues (uint32_t baudrate, uint32_t baudrate_data, uint32_t source_clock_hz, can_timing_config_t *pconfig)`  
Расчет рекомендуемых временных параметров (битовых таймингов) для указанных скоростей обмена в сегменте управления и данных
- `void CAN_SetArbitrationTimingConfig (CAN_Type *base, const can_timing_config_t *config)`  
Установка временных параметров (битовых таймингов) CAN.
- `void CAN_SetPrimaryTxBufferConfig (CAN_Type *base, const can_ptb_config_t *config)`  
Установка параметров высокоприоритетного буфера выдачи
- `void CAN_SetSecondaryTxBufferConfig (CAN_Type *base, const can_stb_config_t *config)`  
Установка параметров низкоприоритетного буфера выдачи
- `void CAN_SetRxBufferConfig (CAN_Type *base, const can_rxb_config_t *config)`  
Установка параметров буфера приема
- `void CAN_SetFilterConfig (CAN_Type *base, const can_frame_filter_config_t *config)`  
Установка параметров фильтрации кадров при приеме
- `void CAN_SetTTCANConfig (CAN_Type *base, const ttcan_config_t *config)`  
Установка параметров работы контроллера в режиме TTCAN.

### Флаги статусов и прерываний

- bool [CAN\\_GetStatusFlag](#) (CAN\_Type \*base, can\_flag\_t idx, can\_status\_t \*status)  
Получение состояния флага состояния/прерывания
- uint32\_t [CAN\\_GetStatusFlagMask](#) (CAN\_Type \*base)  
Получение маски активных прерываний
- can\_status\_t [CAN\\_ClearStatusFlag](#) (CAN\_Type \*base, can\_flag\_t idx)  
Сброс флага состояния/прерывания
- void [CAN\\_ClearStatusFlagMask](#) (CAN\_Type \*base, uint32\_t mask)  
Сброс флагов прерываний по маске

### Управление прерываниями

- can\_status\_t [CAN\\_EnableInterrupt](#) (CAN\_Type \*base, can\_flag\_t idx)  
Разрешение прерывания
- void [CAN\\_EnableInterruptMask](#) (CAN\_Type \*base, uint32\_t mask)  
Разрешение прерываний по маске
- bool [CAN\\_IsInterruptEnabled](#) (CAN\_Type \*base, can\_flag\_t idx, can\_status\_t \*status)  
Запрос - разрешено ли прерывание CAN.
- uint32\_t [CAN\\_GetEnabledInterruptMask](#) (CAN\_Type \*base)  
Запрос маски разрешенных прерываний CAN.
- can\_status\_t [CAN\\_DisableInterrupt](#) (CAN\_Type \*base, can\_flag\_t idx)  
Запрет прерывания
- void [CAN\\_DisableInterruptMask](#) (CAN\_Type \*base, uint32\_t mask)  
Запрет прерываний по маске

### Прямое управление выдачей и приемом

- bool [CAN\\_IsPrimaryTransmitRequestPending](#) (CAN\_Type \*base)  
Получение признака требования выдачи для высокоприоритетного буфера
- bool [CAN\\_IsSecondaryTransmitRequestPending](#) (CAN\_Type \*base)  
Получение признака требования выдачи для низкоприоритетного буфера
- bool [CAN\\_IsSecondaryTxBufferEmpty](#) (CAN\_Type \*base)  
Получение признака опустошения низкоприоритетного буфера
- bool [CAN\\_IsSecondaryTxBufferMoreThanHalfFull](#) (CAN\_Type \*base)  
Получение признака заполнения низкоприоритетного буфера выше середины
- bool [CAN\\_IsSecondaryTxBufferFull](#) (CAN\_Type \*base)  
Получение признака заполнения низкоприоритетного буфера.
- can\_status\_t [CAN\\_WritePrimaryTxBuffer](#) (CAN\_Type \*base, const can\_tx\_buffer\_frame\_t \*ptxframe)  
Запись кадра в высокоприоритетный буфер передачи.
- can\_status\_t [CAN\\_WriteSecondaryTxBuffer](#) (CAN\_Type \*base, const can\_tx\_buffer\_frame\_t \*ptxframe)  
Запись кадра в низкоприоритетный буфер передачи
- void [CAN\\_AbortPrimaryTxBuffer](#) (CAN\_Type \*base)  
Отмена выдачи кадра из высокоприоритетного буфера
- void [CAN\\_AbortSecondaryTxBuffer](#) (CAN\_Type \*base)  
Отмена выдачи кадров из низкоприоритетного буфера
- bool [CAN\\_IsRxBufferEmpty](#) (CAN\_Type \*base)  
Получение признака опустошения буфера приема

- bool `CAN_IsRxBufferAlmostFull` (CAN\_Type \*base)  
Получение признака заполнения буфера приема до границы "почти полный".
- bool `CAN_IsRxBufferFull` (CAN\_Type \*base)  
Получение признака заполнения буфера приема
- can\_status\_t `CAN_ReadRxBuffer` (CAN\_Type \*base, can\_rx\_buffer\_frame\_t \*prxframe)  
Чтение принятого кадра из приемной очереди

#### Транзакционные передача и прием

- can\_status\_t `CAN_TransferSendPrimaryBlocking` (CAN\_Type \*base, can\_tx\_buffer\_frame\_t \*ptxframe, size\_t nb\_frames)  
Блокирующая выдача кадра через высокоприоритетный буфер выдачи
- can\_status\_t `CAN_TransferSendSecondaryBlocking` (CAN\_Type \*base, can\_tx\_buffer\_frame\_t \*ptxframe, size\_t nb\_frames)  
Блокирующая выдача кадра через низкоприоритетный буфер выдачи
- can\_status\_t `CAN_TransferReceiveFifoBlocking` (CAN\_Type \*base, can\_rx\_buffer\_frame\_t \*prxframe, size\_t nb\_frames)  
Блокирующий прием кадра
- can\_status\_t `CAN_TransferCreateHandle` (CAN\_Type \*base, can\_handle\_t \*handle, can\_transfer\_callback\_t callback, void \*user\_data)  
Инициализация обработчика событий CAN.
- can\_status\_t `CAN_TransferSendPrimaryNonBlocking` (CAN\_Type \*base, can\_handle\_t \*handle, can\_tx\_transfer\_t \*xfer)  
Неблокирующая выдача через высокоприоритетный буфер
- can\_status\_t `CAN_TransferGetSentPrimaryCount` (CAN\_Type \*base, can\_handle\_t \*handle, uint32\_t \*nb\_frames)  
Возвращает количество кадров, отправленных в шину данных через высокоприоритетный буфер
- void `CAN_TransferAbortSendPrimary` (CAN\_Type \*base, can\_handle\_t \*handle)  
Отмена выдачи из высокоприоритетного буфера
- can\_status\_t `CAN_TransferSendSecondaryNonBlocking` (CAN\_Type \*base, can\_handle\_t \*handle, can\_tx\_transfer\_t \*xfer)  
Неблокирующая выдача через низкоприоритетный буфер
- can\_status\_t `CAN_TransferGetSentSecondaryCount` (CAN\_Type \*base, can\_handle\_t \*handle, uint32\_t \*nb\_frames)  
Возвращает количество кадров, отправленных в шину данных через низкоприоритетный буфер
- void `CAN_TransferAbortSendSecondary` (CAN\_Type \*base, can\_handle\_t \*handle)  
Отмена выдачи из низкоприоритетного буфера
- can\_status\_t `CAN_TransferReceiveFifoNonBlocking` (CAN\_Type \*base, can\_handle\_t \*handle, can\_rx\_transfer\_t \*xfer)  
Неблокирующий прием
- can\_status\_t `CAN_TransferGetReceivedCount` (CAN\_Type \*base, can\_handle\_t \*handle, uint32\_t \*nb\_frames)  
Возвращает количество принятых кадров по прерыванию
- void `CAN_TransferAbortReceive` (CAN\_Type \*base, can\_handle\_t \*handle)  
Отмена приема
- void `CAN_TransferHandleIRQ` (CAN\_Type \*base, can\_handle\_t \*handle)  
Установка обработчика на прерывания от CAN, не связанные с приемом/выдачей

#### 4.1.1 Подробное описание

Драйвер ввода-вывода по последовательному шинному интерфейсу CAN.

Драйвер содержит функции управления контроллером CAN микросхемы ELIOT1.

## 4.1.2 Типы

### 4.1.2.1 can\_flag\_t

```
typedef enum _can_flag can_flag_t
```

Флаги прерываний и состояний CAN.

Флаги в перечислении делятся на флаги прерываний, на которые можно зарегистрировать обработчик прерывания, и флаги состояний, которые только сигнализируют об определенном состоянии контроллера CAN и на которые нельзя прикрепить обработчик. Флаги состояния имеют суффикс State в названии и слово "состояние" в русском пояснении.

### 4.1.2.2 can\_kind\_of\_error\_t

```
typedef enum _can_kind_of_error can_kind_of_error_t
```

Виды ошибок на линии CAN.

Перечисление содержит виды ошибок, которые определяет контроллер CAN. Одно из данных значений устанавливается контроллером в поле кода структуры принятого кадра.

## 4.1.3 Перечисления

### 4.1.3.1 \_can\_bytes\_in\_datafield

```
enum _can_bytes_in_datafield
```

Размер данных кадра CAN, указываемый в поле DLC.

Элементы перечислений

CAN_0ByteDatafield	0 байт
CAN_1ByteDatafield	1 байт
CAN_2ByteDatafield	2 байта
CAN_3ByteDatafield	3 байта
CAN_4ByteDatafield	4 байта
CAN_5ByteDatafield	5 байт
CAN_6ByteDatafield	6 байт
CAN_7ByteDatafield	7 байт
CAN_8ByteDatafield	8 байт
CAN_12ByteDatafield	12 байт
CAN_16ByteDatafield	16 байт
CAN_20ByteDatafield	20 байт
CAN_24ByteDatafield	24 байта
CAN_32ByteDatafield	32 байта
CAN_48ByteDatafield	48 байт
CAN_64ByteDatafield	64 байта

4.1.3.2 `_can_flag`enum `_can_flag`

Флаги прерываний и состояний CAN.

Флаги в перечислении делятся на флаги прерываний, на которые можно зарегистрировать обработчик прерывания, и флаги состояний, которые только сигнализируют об определенном состоянии контроллера CAN и на которые нельзя прикрепить обработчик. Флаги состояния имеют суффикс State в названии и слово "состояние" в русском пояснении.

Элементы перечислений

<code>CAN_FlagAbortState</code>	Состояние отмененной передачи
<code>CAN_FlagError</code>	Флаг ошибки
<code>CAN_FlagTransmissionSecondary</code>	Выполнена передача из низкоприоритетного буфера
<code>CAN_FlagTransmissionPrimary</code>	Выполнена передача из высокоприоритетного буфера
<code>CAN_FlagTransmitBufferFull</code>	Буфер выдачи заполнен
<code>CAN_FlagRBAAlmostFull</code>	Буфер приема почти заполнен
<code>CAN_FlagRBFull</code>	Буфер приема заполнен
<code>CAN_FlagRBOverrun</code>	Переполнение буфера приема
<code>CAN_FlagReceive</code>	Выполнен прием кадра
<code>CAN_FlagBusError</code>	Ошибка шины данных(BUS ERROR)
<code>CAN_FlagArbitrationLost</code>	Проигран арбитраж на шине данных
<code>CAN_FlagErrorPassiveInterrupt</code>	Переход в пассивный режим
<code>CAN_FlagErrorPassiveState</code>	Состояние пассивного режима
<code>CAN_FlagErrorWarningState</code>	Состояние, в котором количество ошибок достигло уровня предупреждения
<code>CAN_FlagTimeTriggered</code>	Сработал триггер TTCAN
<code>CAN_FlagTriggerError</code>	Ошибка триггера TTCAN
<code>CAN_FlagWatchTriggerError</code>	Сработал следящий триггер TTCAN
<code>CAN_FlagsNumber</code>	Константа - количество флагов состояния

4.1.3.3 `_can_kind_of_error`enum `_can_kind_of_error`

Виды ошибок на линии CAN.

Перечисление содержит виды ошибок, которые определяет контроллер CAN. Одно из данных значений устанавливается контроллером в поле кода структуры принятого кадра.

Элементы перечислений

<code>CAN_ErrorNone</code>	Нет ошибки
<code>CAN_ErrorBitError</code>	Ошибка бита
<code>CAN_ErrorFormError</code>	Ошибка формы
<code>CAN_ErrorStuffError</code>	Ошибка вставки дополнительных битов (бит-стаффинга)
<code>CAN_ErrorAckError</code>	Ошибка подтверждения

Элементы перечислений

CAN_ErrorCrcError	Ошибка контрольной суммы
CAN_ErrorOtherError	Прочие ошибки: доминантные биты после передачи собственного признака ошибки, признак ошибки был слишком длинным, доминантный бит при передаче признака Passive-Error после определения ошибки подтверждения
CAN_ErrorReserved	Не используется

#### 4.1.3.4 \_can\_status

enum [\\_can\\_status](#)

Коды возврата функций драйвера CAN.

Элементы перечислений

CAN_Status_Ok	Успешно
CAN_Status_Fail	Провал
CAN_Status_InvalidArgument	Неверный аргумент
CAN_Status_TxBusy	Передатчик занят
CAN_Status_RxEmpty	Приемный буфер пуст
CAN_Status_TxIdle	Заданное число кадров выдано
CAN_Status_RxIdle	Заданное число кадров принято
CAN_Status_RxBusy	Уже запущен прием

#### 4.1.3.5 \_can\_stb\_discipline

enum [\\_can\\_stb\\_discipline](#)

Дисциплина выдачи из низкоприоритетного буфера

Элементы перечислений

CAN_Fifo	Выдача в порядке поступления (по очереди)
CAN_Priority	Выдача в соответствии с приоритетом кадра

#### 4.1.3.6 \_canfd\_mode

enum [\\_canfd\\_mode](#)

Режимы работы по протоколу CAN FD (есть отличия в расчете контрольной суммы и формировании битов стаффинга)



Элементы перечислений

CAN_BoschFd	Режим Bosch CAN FD
CAN_IsoFd	Режим ISO CAN FD

#### 4.1.3.7 \_ttcan\_timer\_prescaler

enum [\\_ttcan\\_timer\\_prescaler](#)

Символьные константы значений делителя блока TTCAN.

Элементы перечислений

CAN_TTCANDiv1	Делитель отсутствует
CAN_TTCANDiv2	Делитель равен 2
CAN_TTCANDiv4	Делитель равен 4
CAN_TTCANDiv8	Делитель равен 8

#### 4.1.3.8 \_ttcan\_trigger\_type

enum [\\_ttcan\\_trigger\\_type](#)

Тип триггера TTCAN.

Элементы перечислений

CAN_TriggerImmediate	Передача запускается сразу
CAN_TriggerTime	Триггер только устанавливает флаг прерывания
CAN_TriggerSingleShotTransmit	Триггер предназначен для использования в окнах выдачи с эксклюзивным доступом только одного узла
CAN_TriggerTransmitStart	Триггер предназначен для запуска передачи в окнах выдачи, в котором могут выдавать несколько узлов
CAN_TriggerTransmitStop	Триггер предназначен для останова передачи в окнах выдачи, в котором могут выдавать несколько узлов

### 4.1.4 Функции

#### 4.1.4.1 CAN\_AbortPrimaryTxBuffer()

```
void CAN_AbortPrimaryTxBuffer (
    CAN_Type * base)
```

Отмена выдачи кадра из высокоприоритетного буфера

## Аргументы

base	Базовый адрес контроллера
------	---------------------------

## 4.1.4.2 CAN\_AbortSecondaryTxBuffer()

```
void CAN_AbortSecondaryTxBuffer (
    CAN_Type * base)
```

Отмена выдачи кадров из низкоприоритетного буфера

Данная функция отменяет выдачу кадров из низкоприоритетного буфера (STB, Secondary Transmit Buffer). Если буфер работает в режиме приоритетной выдачи, то отменяемые кадры определяются маской, если соответствующий номеру ячейки бит установлен, то выполняется отмена выдачи. Если буфер работает в режиме очереди, то выполняется полная очистка буфера, независимо от значения, переданного в маске.

## Аргументы

base	Базовый адрес контроллера
------	---------------------------

## 4.1.4.3 CAN\_CalculateImprovedTimingValues()

```
bool CAN_CalculateImprovedTimingValues (
    uint32_t baudrate,
    uint32_t baudrate_data,
    uint32_t source_clock_hz,
    can_timing_config_t * pconfig)
```

Расчет рекомендуемых временных параметров (битовых таймингов) для указанных скоростей обмена в сегменте управления и данных

## Аргументы

baudrate	Скорость CANFD в битах/с при передаче управляющей части кадра
baudrate_data	Скорость данных в битах/с (для CANFD)
source_clock_hz	Опорная частота в Гц
pconfig	Структура с временными параметрами

## Возвращаемые значения

true	Конфигурация найдена
false	Не удалось найти правильную конфигурацию

## 4.1.4.4 CAN\_ClearStatusFlag()

```
can_status_t CAN_ClearStatusFlag (
    CAN_Type * base,
    can_flag_t idx)
```

Сброс флага состояния/прерывания

## Аргументы

base	Базовый адрес контроллера
idx	Номер флага состояния/прерывания

## Возвращаемые значения

<a href="#">CAN_Status_Ok</a>	
<a href="#">CAN_Status_InvalidArgument</a>	

## 4.1.4.5 CAN\_ClearStatusFlagMask()

```
void CAN_ClearStatusFlagMask (
    CAN_Type * base,
    uint32_t mask)
```

Сброс флагов прерываний по маске

## Заметки

Позиции битов для маски флагов прерываний указаны в перечислении [can\\_flag\\_t](#).

## Аргументы

base	Базовый адрес контроллера
mask	Маска флагов прерываний, которые необходимо сбросить

## 4.1.4.6 CAN\_Deinit()

```
void CAN_Deinit (
    CAN_Type * base)
```

Деинициализация драйвера CAN.

Функция деинициализирует контроллер CAN.

## Аргументы

base	Базовый адрес контроллера
------	---------------------------

## 4.1.4.7 CAN\_DisableInterrupt()

```
can\_status\_t CAN_DisableInterrupt (
    CAN_Type * base,
    can\_flag\_t idx)
```

Запрет прерывания

## Аргументы

base	Базовый адрес контроллера
idx	Номер флага состояния/прерывания

## Возвращаемые значения

<a href="#">CAN_Status_Ok</a>	
<a href="#">CAN_Status_InvalidArgument</a>	

## 4.1.4.8 CAN\_DisableInterruptMask()

```
void CAN_DisableInterruptMask (
    CAN_Type * base,
    uint32_t mask)
```

Запрет прерываний по маске

## Аргументы

base	Базовый адрес контроллера
mask	Маска флагов прерываний, которые необходимо запретить

## 4.1.4.9 CAN\_EnableInterrupt()

```
can_status_t CAN_EnableInterrupt (
    CAN_Type * base,
    can_flag_t idx)
```

Разрешение прерывания

## Аргументы

base	Базовый адрес контроллера
idx	Номер флага состояния/прерывания

## Возвращаемые значения

<a href="#">CAN_Status_Ok</a>	
<a href="#">CAN_Status_InvalidArgument</a>	

## 4.1.4.10 CAN\_EnableInterruptMask()

```
void CAN_EnableInterruptMask (
    CAN_Type * base,
    uint32_t mask)
```

Разрешение прерываний по маске

## Аргументы

base	Базовый адрес контроллера
mask	Маска флагов прерываний, которые необходимо разрешить

## 4.1.4.11 CAN\_EnterNormalMode()

```
void CAN_EnterNormalMode (
    CAN_Type * base)
```

Переключение контроллера CAN в рабочий режим

Данную функцию необходимо вызвать после инициализации, чтобы запустить процесс подключения контроллера CAN к шине данных.

## Аргументы

base	Базовый адрес контроллера
------	---------------------------

## 4.1.4.12 CAN\_EnterStandbyMode()

```
void CAN_EnterStandbyMode (
    CAN_Type * base)
```

Переключение контроллера CAN в режим ожидания

## Аргументы

base	Базовый адрес контроллера
------	---------------------------

## 4.1.4.13 CAN\_GetDefaultConfig()

```
void CAN_GetDefaultConfig (
    can_config_t * config,
    uint32_t source_clock_hz)
```

Получение параметров драйвера CAN по умолчанию

## Аргументы

config	Структура с параметрами конфигурации
source_clock_hz	Опорная частота в Гц

## 4.1.4.14 CAN\_GetEnabledInterruptMask()

```
uint32_t CAN_GetEnabledInterruptMask (
    CAN_Type * base)
```

Запрос маски разрешенных прерываний CAN.

Запрос маски разрешенных прерываний CAN; единицы в соответствующих разрядах соответствуют включенным прерываниям.

## Аргументы

base	Базовый адрес CAN
------	-------------------

## Возвращает

Маска разрешенных прерываний

## 4.1.4.15 CAN\_GetStatusFlag()

```
bool CAN_GetStatusFlag (
    CAN_Type * base,
    can_flag_t idx,
    can_status_t * status)
```

Получение состояния флага состояния/прерывания

## Аргументы

base	Базовый адрес контроллера
idx	Номер флага состояния/прерывания
status	По переданному указателю функция записывает свой статус выполнения. При передаче нулевого указателя запись не производится.

## Возвращаемые значения

true	Состояние активно
false	Состояние неактивно

## 4.1.4.16 CAN\_GetStatusFlagMask()

```
uint32_t CAN_GetStatusFlagMask (
    CAN_Type * base)
```

Получение маски активных прерываний

## Аргументы

base	Базовый адрес контроллера
------	---------------------------

## Возвращает

Маска активных прерываний

## 4.1.4.17 CAN\_Init()

```
can_status_t CAN_Init (
    CAN_Type * base,
    const can_config_t * config)
```

Инициализация драйвера CAN.

Функция инициализирует модуль CAN в соответствии с заданными пользовательскими параметрами.

## Аргументы

base	Базовый адрес контроллера
config	Структура с параметрами конфигурации

## 4.1.4.18 CAN\_IsInterruptEnabled()

```
bool CAN_IsInterruptEnabled (
    CAN_Type * base,
    can_flag_t idx,
    can_status_t * status)
```

Запрос - разрешено ли прерывание CAN.

## Аргументы

base	Базовый адрес CAN
idx	Номер флага состояния/прерывания
status	По переданному указателю функция записывает свой статус выполнения. При передаче нулевого указателя запись не производится.

## Возвращаемые значения

true	Прерывание разрешено
false	Прерывание запрещено

## 4.1.4.19 CAN\_IsPrimaryTransmitRequestPending()

```
bool CAN_IsPrimaryTransmitRequestPending (
    CAN_Type * base)
```

Получение признака требования выдачи для высокоприоритетного буфера

Данная функция возвращает состояние флага требования выдачи для высокоприоритетного буфера. Если флаг установлен, то буфер занят.

## Аргументы

base	Базовый адрес контроллера
------	---------------------------

## Возвращаемые значения

true	Признак выдачи активен (буфер занят)
false	Признак сброшен

## 4.1.4.20 CAN\_IsRxBufferAlmostFull()

```
bool CAN_IsRxBufferAlmostFull (
    CAN_Type * base)
```

Получение признака заполнения буфера приема до границы "почти полный".

Данная функция сообщает, заполнен ли буфер приема до состояния "почти полный".

## Аргументы

base	Базовый адрес контроллера
------	---------------------------

## Возвращаемые значения

true	Буфер заполнен до состояния "почти полный"
false	Буфер не заполнен до состояния "почти полный"

## 4.1.4.21 CAN\_IsRxBufferEmpty()

```
bool CAN_IsRxBufferEmpty (
    CAN_Type * base)
```

Получение признака опустошения буфера приема

Данная функция сообщает, пуст ли буфер приема.

## Аргументы

base	Базовый адрес контроллера
------	---------------------------

## Возвращаемые значения

true	Буфер пуст
false	В буфере есть кадры

## 4.1.4.22 CAN\_IsRxBufferFull()

```
bool CAN_IsRxBufferFull (
    CAN_Type * base)
```

Получение признака заполнения буфера приема

Данная функция сообщает, заполнен ли буфер приема.

## Аргументы

base	Базовый адрес контроллера
------	---------------------------

## Возвращаемые значения

true	Буфер заполнен
false	Буфер не заполнен

## 4.1.4.23 CAN\_IsSecondaryTransmitRequestPending()

```
bool CAN_IsSecondaryTransmitRequestPending (
    CAN_Type * base)
```

Получение признака требования выдачи для низкоприоритетного буфера

Данная функция возвращает состояние флага требования выдачи для ячейки низкоприоритетного буфера (при использовании приоритетного порядка выдачи из низкоприоритетного буфера). Если флаг установлен, то ячейка занята.



## Аргументы

base	Базовый адрес контроллера
------	---------------------------

## Возвращаемые значения

true	Признак выдачи активен (буфер занят)
false	Признак сброшен

## 4.1.4.24 CAN\_IsSecondaryTxBufferEmpty()

```
bool CAN_IsSecondaryTxBufferEmpty (  
    CAN_Type * base)
```

Получение признака опустошения низкоприоритетного буфера

Данная функция сообщает, пуст ли низкоприоритетный буфер.

## Аргументы

base	Базовый адрес контроллера
------	---------------------------

## Возвращаемые значения

true	Буфер пуст
false	В буфере есть кадры

## 4.1.4.25 CAN\_IsSecondaryTxBufferFull()

```
bool CAN_IsSecondaryTxBufferFull (  
    CAN_Type * base)
```

Получение признака заполнения низкоприоритетного буфера.

Данная функция сообщает, заполнен ли низкоприоритетный буфер.

## Аргументы

base	Базовый адрес контроллера
------	---------------------------

## Возвращаемые значения

true	Очередь пуста
false	В очереди есть кадры

## 4.1.4.26 CAN\_IsSecondaryTxBufferMoreThanHalfFull()

```
bool CAN_IsSecondaryTxBufferMoreThanHalfFull (  
    CAN_Type * base)
```

Получение признака заполнения низкоприоритетного буфера выше середины

Данная функция сообщает, пуст ли низкоприоритетный буфер до уровня более половины всех доступных ячеек.

## Аргументы

base	Базовый адрес контроллера
------	---------------------------

## Возвращаемые значения

true	Буфер заполнен выше середины
false	Буфер заполнен меньше середины

## 4.1.4.27 CAN\_ReadRxBuffer()

```
can_status_t CAN_ReadRxBuffer (
    CAN_Type * base,
    can_rx_buffer_frame_t * prxframe)
```

Чтение принятого кадра из приемной очереди

Функция считывает один принятый кадр CAN из приемной очереди. Считанный кадр при этом удаляется из очереди.

## Аргументы

base	Базовый адрес контроллера
prxframe	Принятый кадр

## Возвращаемые значения

CAN_Status_Ok	
CAN_Status_RxEmpty	

## 4.1.4.28 CAN\_SetArbitrationTimingConfig()

```
void CAN_SetArbitrationTimingConfig (
    CAN_Type * base,
    const can_timing_config_t * config)
```

Установка временных параметров (битовых таймингов) CAN.

## Аргументы

base	Базовый адрес контроллера
config	Структура с временными параметрами

## 4.1.4.29 CAN\_SetFilterConfig()

```
void CAN_SetFilterConfig (
    CAN_Type * base,
    const can_frame_filter_config_t * config)
```

Установка параметров фильтрации кадров при приеме

## Аргументы

base	Базовый адрес контроллера
config	Структура с параметрами фильтрации кадров

## 4.1.4.30 CAN\_SetPrimaryTxBufferConfig()

```
void CAN_SetPrimaryTxBufferConfig (
    CAN_Type * base,
    const can_ptb_config_t * config)
```

Установка параметров высокоприоритетного буфера выдачи

## Аргументы

base	Базовый адрес контроллера
config	Структура с параметрами высокоприоритетного буфера выдачи кадров

## 4.1.4.31 CAN\_SetRxBufferConfig()

```
void CAN_SetRxBufferConfig (
    CAN_Type * base,
    const can_rxb_config_t * config)
```

Установка параметров буфера приема

## Аргументы

base	Базовый адрес контроллера
config	Структура с параметрами буфера приема кадров

## 4.1.4.32 CAN\_SetSecondaryTxBufferConfig()

```
void CAN_SetSecondaryTxBufferConfig (
    CAN_Type * base,
    const can_stb_config_t * config)
```

Установка параметров низкоприоритетного буфера выдачи

## Аргументы

base	Базовый адрес контроллера
config	Структура с параметрами низкоприоритетного буфера выдачи кадров

## 4.1.4.33 CAN\_SetTTCANConfig()

```
void CAN_SetTTCANConfig (
    CAN_Type * base,
    const ttcan_config_t * config)
```

Установка параметров работы контроллера в режиме TTCAN.

## Аргументы

base	Базовый адрес контроллера
config	Структура с параметрами TTCAN

## 4.1.4.34 CAN\_TransferAbortReceive()

```
void CAN_TransferAbortReceive (
    CAN_Type * base,
    can_handle_t * handle)
```

Отмена приема

## Аргументы

base	Базовый адрес контроллера
handle	Обработчик

## 4.1.4.35 CAN\_TransferAbortSendPrimary()

```
void CAN_TransferAbortSendPrimary (
    CAN_Type * base,
    can_handle_t * handle)
```

Отмена выдачи из высокоприоритетного буфера

## Аргументы

base	Базовый адрес контроллера
handle	Обработчик

## 4.1.4.36 CAN\_TransferAbortSendSecondary()

```
void CAN_TransferAbortSendSecondary (
    CAN_Type * base,
    can_handle_t * handle)
```

Отмена выдачи из низкоприоритетного буфера

## Аргументы

base	Базовый адрес контроллера
handle	Обработчик

## 4.1.4.37 CAN\_TransferCreateHandle()

```
can_status_t CAN_TransferCreateHandle (
    CAN_Type * base,
    can_handle_t * handle,
    can_transfer_callback_t callback,
    void * user_data)
```

Инициализация обработчика событий CAN.

## Аргументы

base	Базовый адрес контроллера
handle	Обработчик
callback	Функция обратного вызова
user_data	Аргумент функции обратного вызова

## Возвращаемые значения

<a href="#">CAN_Status_Ok</a>	
<a href="#">CAN_Status_Fail</a>	

## 4.1.4.38 CAN\_TransferGetReceivedCount()

```
can_status_t CAN_TransferGetReceivedCount (
    CAN_Type * base,
    can_handle_t * handle,
    uint32_t * nb_frames)
```

Возвращает количество принятых кадров по прерыванию

## Аргументы

base	Базовый адрес контроллера
handle	Обработчик
nb_frames	Указатель для возврата количества отправленных кадров

## Возвращаемые значения

<a href="#">CAN_Status_Ok</a>	
<a href="#">CAN_Status_RxIdle</a>	

## 4.1.4.39 CAN\_TransferGetSentPrimaryCount()

```
can_status_t CAN_TransferGetSentPrimaryCount (
    CAN_Type * base,
    can_handle_t * handle,
    uint32_t * nb_frames)
```

Возвращает количество кадров, отправленных в шину данных через высокоприоритетный буфер

Эта функция возвращает количество кадров, отправленных в шину данных по прерыванию через высокоприоритетный буфер.

## Аргументы

base	Базовый адрес контроллера
handle	Обработчик
nb_frames	Указатель для возврата количества отправленных кадров

Возвращаемые значения

<a href="#">CAN_Status_Ok</a>	
<a href="#">CAN_Status_TxIdle</a>	

#### 4.1.4.40 CAN\_TransferGetSentSecondaryCount()

```
can_status_t CAN_TransferGetSentSecondaryCount (
    CAN_Type * base,
    can_handle_t * handle,
    uint32_t * nb_frames)
```

Возвращает количество кадров, отправленных в шину данных через низкоприоритетный буфер

Эта функция возвращает количество кадров, отправленных в шину данных по прерыванию через низкоприоритетный буфер.

Аргументы

base	Базовый адрес контроллера
handle	Обработчик
nb_frames	Указатель для возврата количества отправленных кадров

Возвращаемые значения

<a href="#">CAN_Status_Ok</a>	
<a href="#">CAN_Status_TxIdle</a>	

#### 4.1.4.41 CAN\_TransferHandleIRQ()

```
void CAN_TransferHandleIRQ (
    CAN_Type * base,
    can_handle_t * handle)
```

Установка обработчика на прерывания от CAN, не связанные с приемом/выдачей

Аргументы

base	Базовый адрес контроллера
handle	Обработчик

#### 4.1.4.42 CAN\_TransferReceiveFifoBlocking()

```
can_status_t CAN_TransferReceiveFifoBlocking (
    CAN_Type * base,
    can_rx_buffer_frame_t * prxframe,
    size_t nb_frames)
```

Блокирующий прием кадра

При использовании блокирующей выдачи не нужно устанавливать обработчик соответствующего события.

## Аргументы

base	Базовый адрес контроллера
prxframe	Буфер для приёма кадров
nb_frames	Максимальное количество кадров в буфере для приёма

## Возвращаемые значения

CAN_Status_Ok	
CAN_Status_Fail	

## 4.1.4.43 CAN\_TransferReceiveFifoNonBlocking()

```
can_status_t CAN_TransferReceiveFifoNonBlocking (
    CAN_Type * base,
    can_handle_t * handle,
    can_rx_transfer_t * xfer)
```

## Неблокирующий прием

Функция принимает кадр CAN по прерыванию. Это неблокирующая функция, она возвращает управление сразу. По приему кадра из шины вызывается функция обратного вызова.

## Аргументы

base	Базовый адрес контроллера
handle	Обработчик
xfer	Структура для передачи кадра в неблокирующем режиме

## Возвращаемые значения

CAN_Status_Ok	
CAN_Status_InvalidArgument	
CAN_Status_RxBusy	

## 4.1.4.44 CAN\_TransferSendPrimaryBlocking()

```
can_status_t CAN_TransferSendPrimaryBlocking (
    CAN_Type * base,
    can_tx_buffer_frame_t * ptxframe,
    size_t nb_frames)
```

## Блокирующая выдача кадра через высокоприоритетный буфер выдачи

При использовании блокирующей выдачи не нужно устанавливать обработчик соответствующего события.

## Аргументы

base	Базовый адрес контроллера
ptxframe	Буфер с кадрами для выдачи
nb_frames	Количество кадров для выдачи

## Возвращаемые значения

<a href="#">CAN_Status_Ok</a>	
<a href="#">CAN_Status_Fail</a>	

## 4.1.4.45 CAN\_TransferSendPrimaryNonBlocking()

```
can_status_t CAN_TransferSendPrimaryNonBlocking (
    CAN_Type * base,
    can_handle_t * handle,
    can_tx_transfer_t * xfer)
```

## Неблокирующая выдача через высокоприоритетный буфер

Функция выдает кадр CAN по прерыванию. Это неблокирующая функция, она возвращает управление сразу. По выдаче кадра в шину данных вызывает функцию обратного вызова.

## Аргументы

base	Базовый адрес контроллера
handle	Обработчик
xfer	Структура для передачи кадра в неблокирующем режиме

## Возвращаемые значения

<a href="#">CAN_Status_Ok</a>	
<a href="#">CAN_Status_Fail</a>	
<a href="#">CAN_Status_TxBusy</a>	

## 4.1.4.46 CAN\_TransferSendSecondaryBlocking()

```
can_status_t CAN_TransferSendSecondaryBlocking (
    CAN_Type * base,
    can_tx_buffer_frame_t * ptxframe,
    size_t nb_frames)
```

## Блокирующая выдача кадра через низкоприоритетный буфер выдачи

При использовании блокирующей выдачи не нужно устанавливать обработчик соответствующего события.



## Аргументы

base	Базовый адрес контроллера
ptxframe	Буфер с кадрами для выдачи
nb_frames	Количество кадров для выдачи

## Возвращаемые значения

CAN_Status_Ok	
CAN_Status_Fail	

## 4.1.4.47 CAN\_TransferSendSecondaryNonBlocking()

```
can_status_t CAN_TransferSendSecondaryNonBlocking (
    CAN_Type * base,
    can_handle_t * handle,
    can_tx_transfer_t * xfer)
```

## Неблокирующая выдача через низкоприоритетный буфер

Функция выдает кадр CAN по прерыванию. Это неблокирующая функция, она возвращает управление сразу. По выдаче кадра в шину вызывается функция обратного вызова.

## Аргументы

base	Базовый адрес контроллера
handle	Обработчик
xfer	Структура для передачи кадра в неблокирующем режиме

## Возвращаемые значения

CAN_Status_Ok	
CAN_Status_Fail	
CAN_Status_TxBusy	

## 4.1.4.48 CAN\_WritePrimaryTxBuffer()

```
can_status_t CAN_WritePrimaryTxBuffer (
    CAN_Type * base,
    const can_tx_buffer_frame_t * ptxframe)
```

## Запись кадра в высокоприоритетный буфер передачи.

Данная функция записывает кадр в высокоприоритетный буфер (РТВ, Primary Transmit Buffer). Выдача из данного буфера "вклинивается" в выдачу из низкоприоритетного буфера. Высокоприоритетный буфер может содержать только один кадр CAN. Функция не блокирующая.

## Аргументы

base	Базовый адрес контроллера
ptxframe	Кадр для выдачи

## Возвращаемые значения

<a href="#">CAN_Status_Ok</a>	
<a href="#">CAN_Status_TxBusy</a>	
<a href="#">CAN_Status_InvalidArgument</a>	

## 4.1.4.49 CAN\_WriteSecondaryTxBuffer()

```
can_status_t CAN_WriteSecondaryTxBuffer (
    CAN_Type * base,
    const can_tx_buffer_frame_t * ptxframe)
```

Запись кадра в низкоприоритетный буфер передачи

Данная функция записывает кадр в низкоприоритетный буфер (STB, Secondary Transmit Buffer), работающий в режиме выдачи кадров в соответствии с их приоритетом.

## Аргументы

base	Базовый адрес контроллера
ptxframe	Кадр для выдачи

## Возвращаемые значения

<a href="#">CAN_Status_Ok</a>	
<a href="#">CAN_Status_TxBusy</a>	

## 4.2 Драйвер модуля CLKCTR

Драйвер модуля управления частотами

## Файлы

- файл [hal\\_clkctr.h](#)  
Интерфейс драйвера модуля CLKCTR.

## Структуры данных

- struct [clkctr\\_div](#)  
Делители блока
- struct [clkctr\\_pll\\_cfg](#)  
Коэффициенты PLL.

## Перечисления

- enum [clkctr\\_mcoclk](#)  
Подаваемая на MCO частота
- enum [clkctr\\_lpclk](#)  
Подаваемая на LPCLK частота
- enum [clkctr\\_rtccclk](#)  
Подаваемая на RTCCLK частота
- enum [clkctr\\_mainclk](#)  
Подаваемая на MAINCLK частота
- enum [clkctr\\_pllref](#)  
Опорная частота PLL.
- enum [clkctr\\_usbclk](#)  
Подаваемая на USBCLK частота
- enum [clkctr\\_i2sclk](#)  
Подаваемая на I2SCLK частота
- enum [clkctr\\_hfi\\_for\\_main\\_type](#)  
Частота, определенная как HFICLK для MAINCLK (CLKCTR\_MainClkTypeHFICLK)
- enum [clkctr\\_extern\\_freq](#)  
Генератор/осциллятор/вход, для которого групповой функцией устанавливается частота
- enum [clkctr\\_int\\_freq](#)  
Имя внутренней частоты тактирования микросхемы, для которой производится действие групповой функцией
- enum [clkctr\\_clk\\_force\\_type](#)  
Тип тактирования
- enum [clkctr\\_device\\_clk\\_force](#)  
Домены, к которым может быть применено динамическое тактирование
- enum [clkctr\\_status](#)  
Статусы драйвера CLKCTR.

## Устаревшие функции для поддержки ранних примеров

- void [CLKCTR\\_SetPll](#) (CLKCTR\_Type \*base, uint32\_t xti\_hz, uint32\_t pll\_mul)  
Установка целочисленного множителя PLL.
- enum [clkctr\\_status](#) [CLKCTR\\_SetPllMan](#) (CLKCTR\_Type \*base, uint32\_t xti\_hz, uint32\_t nr, uint32\_t nf, uint32\_t od)  
Установка множителей PLL в ручном режиме
- void [CLKCTR\\_SetSysDiv](#) (CLKCTR\_Type \*base, uint16\_t fclk\_div, uint16\_t sysclk\_div, uint16\_t gnssclk\_div, uint16\_t qspiclk\_div)  
Установка делителей

## Функции установки и получения частот генераторов/осцилляторов

- enum [clkctr\\_status](#) [CLKCTR\\_SetXTI](#) (CLKCTR\_Type \*base, uint32\_t frequency)  
Установка значения частоты, подаваемой на вход XTI.
- uint32\_t [CLKCTR\\_GetXTI](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты, подаваемой на вход XTI.
- enum [clkctr\\_status](#) [CLKCTR\\_SetXTI32](#) (CLKCTR\_Type \*base, uint32\_t frequency)  
Установка значения частоты, подаваемой на вход XTI32.
- uint32\_t [CLKCTR\\_GetXTI32](#) (CLKCTR\_Type \*base)

- Извлечение значения частоты, подаваемой на вход XT132.
- enum [clkctr\\_status](#) [CLKCTR\\_SetHFI](#) (CLKCTR\_Type \*base, uint32\_t frequency)
  - Установка значения частоты внутреннего генератора APC.
- uint32\_t [CLKCTR\\_GetHFI](#) (CLKCTR\_Type \*base)
  - Извлечение значения частоты внутреннего генератора APC.
- enum [clkctr\\_status](#) [CLKCTR\\_SetLFI](#) (CLKCTR\_Type \*base, uint32\_t frequency)
  - Установка значения частоты внутреннего генератора RWC.
- uint32\_t [CLKCTR\\_GetLFI](#) (CLKCTR\_Type \*base)
  - Извлечение значения частоты внутреннего генератора RWC.
- enum [clkctr\\_status](#) [CLKCTR\\_SetI2SExtClk](#) (CLKCTR\_Type \*base, uint32\_t frequency)
  - Установка значения частоты, поступающей на PA15.
- uint32\_t [CLKCTR\\_GetI2SExtClk](#) (CLKCTR\_Type \*base)
  - Извлечение значения частоты, поступающей на PA15.

#### Функции установки и получения делителей, кроме PLL

- enum [clkctr\\_status](#) [CLKCTR\\_GetAllDiv](#) (CLKCTR\_Type \*base, struct [clkctr\\_div](#) \*divisors)
  - Извлечение всех делителей блока CLKCTR.
- uint32\_t [CLKCTR\\_GetMCOdiv](#) (CLKCTR\_Type \*base)
  - Извлечение значения делителя MCOdiv.
- enum [clkctr\\_status](#) [CLKCTR\\_SetMCOdiv](#) (CLKCTR\_Type \*base, uint32\_t value)
  - Установка значения делителя MCOdiv.
- uint32\_t [CLKCTR\\_GetFCLKdiv](#) (CLKCTR\_Type \*base)
  - Извлечение значения делителя FCLKdiv.
- enum [clkctr\\_status](#) [CLKCTR\\_SetFCLKdiv](#) (CLKCTR\_Type \*base, uint32\_t value)
  - Установка значения делителя FCLKdiv.
- uint32\_t [CLKCTR\\_GetSysClkDiv](#) (CLKCTR\_Type \*base)
  - Извлечение значения делителя SYSCLKdiv.
- enum [clkctr\\_status](#) [CLKCTR\\_SetSysClkDiv](#) (CLKCTR\_Type \*base, uint32\_t value)
  - Установка значения делителя SYSCLKdiv.
- uint32\_t [CLKCTR\\_GetGNSSClkDiv](#) (CLKCTR\_Type \*base)
  - Извлечение значения делителя GNSSCLKdiv.
- enum [clkctr\\_status](#) [CLKCTR\\_SetGNSSClkDiv](#) (CLKCTR\_Type \*base, uint32\_t value)
  - Установка значения делителя GNSSCLKdiv.
- uint32\_t [CLKCTR\\_GetQSPIClkDiv](#) (CLKCTR\_Type \*base)
  - Извлечение значения делителя QSPICLKdiv.
- enum [clkctr\\_status](#) [CLKCTR\\_SetQSPIClkDiv](#) (CLKCTR\_Type \*base, uint32\_t value)
  - Установка значения делителя QSPICLKdiv.
- uint32\_t [CLKCTR\\_GetI2SCLKdiv](#) (CLKCTR\_Type \*base)
  - Извлечение значения делителя I2SCLKdiv.
- enum [clkctr\\_status](#) [CLKCTR\\_SetI2SCLKdiv](#) (CLKCTR\_Type \*base, uint32\_t value)
  - Установка значения делителя I2SCLKdiv.

#### Функции установки и получения коэффициентов PLL

- enum [clkctr\\_status](#) [CLKCTR\\_GetPLLConfig](#) (CLKCTR\_Type \*base, struct [clkctr\\_pll\\_cfg](#) \*config)
  - Извлечение коэффициентов PLL.
- enum [clkctr\\_status](#) [CLKCTR\\_SetPLLConfig](#) (CLKCTR\_Type \*base, struct [clkctr\\_pll\\_cfg](#) config)
  - Установка коэффициентов PLL.

## Функции извлечения частот тактирования

- uint32\_t [CLKCTR\\_GetXTIClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты XTICLK.
- uint32\_t [CLKCTR\\_GetHFIClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты HFICLK.
- uint32\_t [CLKCTR\\_GetRTCClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты RTCCLK.
- uint32\_t [CLKCTR\\_GetLPCLk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты LPCLK.
- uint32\_t [CLKCTR\\_GetPLLClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты PLLCLK.
- uint32\_t [CLKCTR\\_GetPLLRef](#) (CLKCTR\_Type \*base)  
Извлечение значения опорной частоты PLL.
- uint32\_t [CLKCTR\\_GetMainClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты MAINCLK.
- uint32\_t [CLKCTR\\_GetUSBClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты USBCLK, подаваемого на USB PHY.
- uint32\_t [CLKCTR\\_GetFClkInt](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты FCLK\_INT.
- uint32\_t [CLKCTR\\_GetFClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты FCLK.
- uint32\_t [CLKCTR\\_GetSysClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты SYSCLK.
- uint32\_t [CLKCTR\\_GetGNSSClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты GNSSCLK.
- uint32\_t [CLKCTR\\_GetQSPIClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты QSPICLK.
- uint32\_t [CLKCTR\\_GetI2SCLk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты I2SCLK.
- uint32\_t [CLKCTR\\_GetMCOClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты MCOCLK, подаваемой на вывод PA15.
- uint32\_t [CLKCTR\\_GetHFIClkForMainClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты HFICLK для MAINCLK.

## Функции выбора и извлечения источников частот

- enum [clkctr\\_status](#) [CLKCTR\\_SetSwitchMCOClk](#) (CLKCTR\_Type \*base, uint32\_t value)  
Выбор источника частоты MCO.
- enum [clkctr\\_status](#) [CLKCTR\\_SetSwitchPLLRef](#) (CLKCTR\_Type \*base, uint32\_t value)  
Выбор источника опорной частоты PLL.
- enum [clkctr\\_status](#) [CLKCTR\\_SetSwitchMainClk](#) (CLKCTR\_Type \*base, uint32\_t value)  
Выбор источника частоты MAINCLK.
- enum [clkctr\\_status](#) [CLKCTR\\_SetSwitchRTCClk](#) (CLKCTR\_Type \*base, uint32\_t value)  
Выбор источника частоты RTCCLK.
- enum [clkctr\\_status](#) [CLKCTR\\_SetSwitchLPCLk](#) (CLKCTR\_Type \*base, uint32\_t value)  
Выбор источника частоты LPCLK.
- enum [clkctr\\_status](#) [CLKCTR\\_SetSwitchUSBClk](#) (CLKCTR\_Type \*base, uint32\_t value)  
Выбор источника частоты USBCLK.
- enum [clkctr\\_status](#) [CLKCTR\\_SetSwitchI2SCLk](#) (CLKCTR\_Type \*base, uint32\_t value)

- Выбор источника частоты I2SCLK.
- enum [clkctr\\_status](#) [CLKCTR\\_SetSwitchPMUDIS](#) (CLKCTR\_Type \*base, uint32\_t value)
  - Выбор режима работы процессора
- uint32\_t [CLKCTR\\_GetSwitchMCOClk](#) (CLKCTR\_Type \*base)
  - Извлечение источника частоты MCO.
- uint32\_t [CLKCTR\\_GetSwitchPLLRef](#) (CLKCTR\_Type \*base)
  - Извлечение источника частоты PLL.
- uint32\_t [CLKCTR\\_GetSwitchMainClk](#) (CLKCTR\_Type \*base)
  - Извлечение источника частоты MAINCLK.
- uint32\_t [CLKCTR\\_GetSwitchRTCCLK](#) (CLKCTR\_Type \*base)
  - Извлечение источника частоты RTCCLK.
- uint32\_t [CLKCTR\\_GetSwitchLPCLK](#) (CLKCTR\_Type \*base)
  - Извлечение источника частоты LPCLK.
- uint32\_t [CLKCTR\\_GetSwitchUSBClk](#) (CLKCTR\_Type \*base)
  - Извлечение источника частоты USBCLK.
- uint32\_t [CLKCTR\\_GetSwitchI2SCLK](#) (CLKCTR\_Type \*base)
  - Извлечение источника частоты I2SCLK.
- uint32\_t [CLKCTR\\_GetSwitchPMUDIS](#) (CLKCTR\_Type \*base)
  - Определение режима работы процессора

## Прочие функции

- enum [clkctr\\_status](#) [CLKCTR\\_SetHFITrim](#) (CLKCTR\_Type \*base, uint32\_t index)
  - Установка индекса коэффициента подстройки частоты осциллятора HFI.
- uint32\_t [CLKCTR\\_GetHFITrim](#) (CLKCTR\_Type \*base)
  - Извлечение индекса коэффициента подстройки частоты осциллятора HFI.
- uint32\_t [CLKCTR\\_GetMCOEn](#) (CLKCTR\_Type \*base)
  - Извлечение разрешения тактового сигнала MCO.
- enum [clkctr\\_status](#) [CLKCTR\\_SetMCOEn](#) (CLKCTR\_Type \*base, uint32\_t enable)
  - Установка или запрет разрешения тактового сигнала MCO.
- enum [clkctr\\_status](#) [CLKCTR\\_GetClkForce](#) (CLKCTR\_Type \*base, enum [clkctr\\_device\\_clk\\_force](#) device, enum [clkctr\\_clk\\_force\\_type](#) \*clk\_type)
  - Извлечение типа тактирования для модуля
- enum [clkctr\\_status](#) [CLKCTR\\_SetClkForce](#) (CLKCTR\_Type \*base, enum [clkctr\\_device\\_clk\\_force](#) device, enum [clkctr\\_clk\\_force\\_type](#) clk\_type)
  - Установка типа тактирования для модуля

## Функции-обертки

- enum [clkctr\\_status](#) [CLKCTR\\_SetFrequency](#) (CLKCTR\_Type \*base, enum [clkctr\\_extern\\_freq](#) input, uint32\_t frequency)
  - Установка значения частоты, подаваемой на выбранный вход
- uint32\_t [CLKCTR\\_GetFrequency](#) (CLKCTR\_Type \*base, enum [clkctr\\_extern\\_freq](#) input)
  - Извлечение значения частоты, подаваемой на выбранный вход
- enum [clkctr\\_status](#) [CLKCTR\\_GetClk](#) (CLKCTR\_Type \*base, enum [clkctr\\_int\\_freq](#) clk, uint32\_t \*value)
  - Извлечение значения указанной частоты
- enum [clkctr\\_status](#) [CLKCTR\\_SetSwitchClk](#) (CLKCTR\_Type \*base, enum [clkctr\\_int\\_freq](#) clk, uint32\_t value)
  - Выбор источника частоты для указанной частоты

- enum `clkctr_status` `CLKCTR_GetSwitchClk` (`CLKCTR_Type` \*base, enum `clkctr_int_freq` clk, `uint32_t` \*value)

Извлечение источника частоты для указанной частоты

- enum `clkctr_status` `CLKCTR_SetDivClk` (`CLKCTR_Type` \*base, enum `clkctr_int_freq` clk, `uint32_t` value)

Установка коэффициента деления для указанной частоты

- enum `clkctr_status` `CLKCTR_GetDivClk` (`CLKCTR_Type` \*base, enum `clkctr_int_freq` clk, `uint32_t` \*value)

Извлечение коэффициента деления для указанной частоты

Экстремальные значения коэффициентов делителей частот параметров PLL (делитель частоты = коэффициент деления + 1)

- #define `CLKCTR_MAX_SYSCLK_DIV` 31
- #define `CLKCTR_MAX_FCLK_DIV` 31
- #define `CLKCTR_MAX_GNSSCLK_DIV` 7
- #define `CLKCTR_MAX_QSPICLK_DIV` 31
- #define `CLKCTR_MAX_MCOCLK_DIV` 31
- #define `CLKCTR_MAX_I2SCLK_DIV` 31
- #define `CLKCTR_MIN_SYSCLK_DIV` 0
- #define `CLKCTR_MIN_FCLK_DIV` 0
- #define `CLKCTR_MIN_GNSSCLK_DIV` 0
- #define `CLKCTR_MIN_QSPICLK_DIV` 0
- #define `CLKCTR_MIN_MCOCLK_DIV` 0
- #define `CLKCTR_MIN_I2SCLK_DIV` 0
- #define `PLL_MAX_MULTIPLIER` 0x176
- #define `PLL_MIN_MULTIPLIER` 0x0
- #define `CLKCTR_NR_MAN_MAX`
- #define `CLKCTR_NF_MAN_MAX`
- #define `CLKCTR_OD_MAN_MAX`
- #define `CLKCTR_MAN_MAX`
- #define `CLKCTR_SEL_MAX`

Минимально и максимально допустимые внешние частоты блока CLKCTR

- #define `CLKCTR_XTI_MIN` 1
- #define `CLKCTR_XTI_MAX` 50000000
- #define `CLKCTR_XTI32_MIN` 30000
- #define `CLKCTR_XTI32_MAX` 34000
- #define `CLKCTR_HFI_MIN` 4000000
- #define `CLKCTR_HFI_MAX` 25000000
- #define `CLKCTR_LFI_MIN` 32112
- #define `CLKCTR_LFI_MAX` 33423
- #define `CLKCTR_I2S_EXTCLK_MIN` 1
- #define `CLKCTR_I2S_EXTCLK_MAX` 50000000

Минимально и максимально допустимые внутренние частоты блока CLKCTR

- #define CLKCTR\_PLLREF\_MIN 30000
- #define CLKCTR\_PLLREF\_MAX 50000000
- #define CLKCTR\_PLLCLK\_MIN 1880000
- #define CLKCTR\_PLLCLK\_MAX 375000000
- #define CLKCTR\_PLLCLK\_OD\_MIN 30000000
- #define CLKCTR\_PLLCLK\_OD\_MAX 375000000
- #define CLKCTR\_FCLK\_MIN 1
- #define CLKCTR\_FCLK\_MAX 150000000
- #define CLKCTR\_SYSCLK\_MIN 1
- #define CLKCTR\_SYSCLK\_MAX 50000000
- #define CLKCTR\_QSPICLK\_MIN 1
- #define CLKCTR\_QSPICLK\_MAX 96000000
- #define CLKCTR\_GNSSCLK\_MIN 1
- #define CLKCTR\_GNSSCLK\_MAX 80000000
- #define CLKCTR\_I2SCLK\_MIN 1
- #define CLKCTR\_I2SCLK\_MAX 25000000

Служебные определения

- #define HFI\_FREQUENCY 15100000
- #define CLKCTR\_FREQ\_NOT\_SET 0

#### 4.2.1 Подробное описание

Драйвер модуля управления частотами

Драйвер модуля CLKCTR позволяет настраивать частоты тактирования.

#### 4.2.2 Макросы

##### 4.2.2.1 CLKCTR\_FCLK\_MAX

```
#define CLKCTR_FCLK_MAX 150000000
```

Максимальное значение опорной частоты FCLK

##### 4.2.2.2 CLKCTR\_FCLK\_MIN

```
#define CLKCTR_FCLK_MIN 1
```

Минимальное значение опорной частоты FCLK

##### 4.2.2.3 CLKCTR\_FREQ\_NOT\_SET

```
#define CLKCTR_FREQ_NOT_SET 0
```

Частота не установлена или недоступна



## 4.2.2.4 CLKCTR\_GNSSCLK\_MAX

```
#define CLKCTR_GNSSCLK_MAX 80000000
```

Максимальное значение опорной частоты GNSSCLK

## 4.2.2.5 CLKCTR\_GNSSCLK\_MIN

```
#define CLKCTR_GNSSCLK_MIN 1
```

Минимальное значение опорной частоты GNSSCLK

## 4.2.2.6 CLKCTR\_HFI\_MAX

```
#define CLKCTR_HFI_MAX 25000000
```

Максимальное значение частоты HFI

## 4.2.2.7 CLKCTR\_HFI\_MIN

```
#define CLKCTR_HFI_MIN 4000000
```

Минимальное значение частоты HFI

## 4.2.2.8 CLKCTR\_I2S\_EXTCLK\_MAX

```
#define CLKCTR_I2S_EXTCLK_MAX 50000000
```

Максимальное значение частоты I2S\_EXTCLK

## 4.2.2.9 CLKCTR\_I2S\_EXTCLK\_MIN

```
#define CLKCTR_I2S_EXTCLK_MIN 1
```

Минимальное значение частоты I2S\_EXTCLK

## 4.2.2.10 CLKCTR\_I2SCLK\_MAX

```
#define CLKCTR_I2SCLK_MAX 25000000
```

Максимальное значение опорной частоты I2SCLK

## 4.2.2.11 CLKCTR\_I2SCLK\_MIN

```
#define CLKCTR_I2SCLK_MIN 1
```

Минимальное значение опорной частоты I2SCLK

#### 4.2.2.12 CLKCTR\_LFI\_MAX

```
#define CLKCTR_LFI_MAX 33423
```

Максимальное значение частоты LFI

#### 4.2.2.13 CLKCTR\_LFI\_MIN

```
#define CLKCTR_LFI_MIN 32112
```

Минимальное значение частоты LFI

#### 4.2.2.14 CLKCTR\_MAN\_MAX

```
#define CLKCTR_MAN_MAX
```

Макроопределение:  
(CLKCTR\_PLLCFG\_MAN\_Msk \  
» CLKCTR\_PLLCFG\_MAN\_Pos)

Максимальное значение MAN

#### 4.2.2.15 CLKCTR\_MAX\_FCLK\_DIV

```
#define CLKCTR_MAX_FCLK_DIV 31
```

Максимальное для FCLK

#### 4.2.2.16 CLKCTR\_MAX\_GNSSCLK\_DIV

```
#define CLKCTR_MAX_GNSSCLK_DIV 7
```

Максимальное для GNSSCLK

#### 4.2.2.17 CLKCTR\_MAX\_I2SCLK\_DIV

```
#define CLKCTR_MAX_I2SCLK_DIV 31
```

Максимальное для I2SCLK

#### 4.2.2.18 CLKCTR\_MAX\_MCOCLK\_DIV

```
#define CLKCTR_MAX_MCOCLK_DIV 31
```

Максимальное для MCOCLK

## 4.2.2.19 CLKCTR\_MAX\_QSPICLK\_DIV

```
#define CLKCTR_MAX_QSPICLK_DIV 31
```

Максимальное для QSPICLK

## 4.2.2.20 CLKCTR\_MAX\_SYSCLK\_DIV

```
#define CLKCTR_MAX_SYSCLK_DIV 31
```

Максимальное для SYSCLK

## 4.2.2.21 CLKCTR\_MIN\_FCLK\_DIV

```
#define CLKCTR_MIN_FCLK_DIV 0
```

Минимальное для FCLK

## 4.2.2.22 CLKCTR\_MIN\_GNSSCLK\_DIV

```
#define CLKCTR_MIN_GNSSCLK_DIV 0
```

Минимальное для GNSSCLK

## 4.2.2.23 CLKCTR\_MIN\_I2SCLK\_DIV

```
#define CLKCTR_MIN_I2SCLK_DIV 0
```

Минимальное для I2SCLK

## 4.2.2.24 CLKCTR\_MIN\_MCOCLK\_DIV

```
#define CLKCTR_MIN_MCOCLK_DIV 0
```

Минимальное для MCOCLK

## 4.2.2.25 CLKCTR\_MIN\_QSPICLK\_DIV

```
#define CLKCTR_MIN_QSPICLK_DIV 0
```

Минимальное для QSPICLK

## 4.2.2.26 CLKCTR\_MIN\_SYSCLK\_DIV

```
#define CLKCTR_MIN_SYSCLK_DIV 0
```

Минимальное для SYSCLK

#### 4.2.2.27 CLKCTR\_NF\_MAN\_MAX

```
#define CLKCTR_NF_MAN_MAX
```

Макроопределение:

```
(CLKCTR_PLLCFG_NF_MAN_Msk \  
» CLKCTR_PLLCFG_NF_MAN_Pos)
```

Максимальное значение NF\_MAN

#### 4.2.2.28 CLKCTR\_NR\_MAN\_MAX

```
#define CLKCTR_NR_MAN_MAX
```

Макроопределение:

```
(CLKCTR_PLLCFG_NR_MAN_Msk \  
» CLKCTR_PLLCFG_NR_MAN_Pos)
```

Максимальное значение NR\_MAN

#### 4.2.2.29 CLKCTR\_OD\_MAN\_MAX

```
#define CLKCTR_OD_MAN_MAX
```

Макроопределение:

```
(CLKCTR_PLLCFG_OD_MAN_Msk \  
» CLKCTR_PLLCFG_OD_MAN_Pos)
```

Максимальное значение OD\_MAN

#### 4.2.2.30 CLKCTR\_PLLCLK\_MAX

```
#define CLKCTR_PLLCLK_MAX 375000000
```

Максимальное значение выходной частоты PLL полный диапазон

#### 4.2.2.31 CLKCTR\_PLLCLK\_MIN

```
#define CLKCTR_PLLCLK_MIN 1880000
```

Минимальное значение выходной частоты PLL полный диапазон

#### 4.2.2.32 CLKCTR\_PLLCLK\_OD\_MAX

```
#define CLKCTR_PLLCLK_OD_MAX 375000000
```

Максимальное значение выходной частоты PLL без учета делителя OD

## 4.2.2.33 CLKCTR\_PLLCLK\_OD\_MIN

```
#define CLKCTR_PLLCLK_OD_MIN 30000000
```

Минимальное значение выходной частоты PLL без учета делителя OD

## 4.2.2.34 CLKCTR\_PLLREF\_MAX

```
#define CLKCTR_PLLREF_MAX 50000000
```

Максимальное значение опорной частоты PLL

## 4.2.2.35 CLKCTR\_PLLREF\_MIN

```
#define CLKCTR_PLLREF_MIN 30000
```

Минимальное значение опорной частоты PLL

## 4.2.2.36 CLKCTR\_QSPICLK\_MAX

```
#define CLKCTR_QSPICLK_MAX 96000000
```

Максимальное значение опорной частоты QSPICLK

## 4.2.2.37 CLKCTR\_QSPICLK\_MIN

```
#define CLKCTR_QSPICLK_MIN 1
```

Минимальное значение опорной частоты QSPICLK

## 4.2.2.38 CLKCTR\_SEL\_MAX

```
#define CLKCTR_SEL_MAX
```

Макроопределение:

```
(CLKCTR_PLLCFG_SEL_Msk \  
» CLKCTR_PLLCFG_SEL_Pos)
```

Максимальное значение SEL

## 4.2.2.39 CLKCTR\_SYSCLK\_MAX

```
#define CLKCTR_SYSCLK_MAX 50000000
```

Максимальное значение опорной частоты SYSCLK

#### 4.2.2.40 CLKCTR\_SYSCLK\_MIN

```
#define CLKCTR_SYSCLK_MIN 1
```

Минимальное значение опорной частоты SYSCLK

#### 4.2.2.41 CLKCTR\_XTI32\_MAX

```
#define CLKCTR_XTI32_MAX 34000
```

Максимальное значение частоты XTI32

#### 4.2.2.42 CLKCTR\_XTI32\_MIN

```
#define CLKCTR_XTI32_MIN 30000
```

Минимальное значение частоты XTI32

#### 4.2.2.43 CLKCTR\_XTI\_MAX

```
#define CLKCTR_XTI_MAX 50000000
```

Максимальное значение частоты XTI

#### 4.2.2.44 CLKCTR\_XTI\_MIN

```
#define CLKCTR_XTI_MIN 1
```

Минимальное значение частоты XTI

#### 4.2.2.45 HFI\_FREQUENCY

```
#define HFI_FREQUENCY 15100000
```

Частота внутреннего генератора "по умолчанию"

#### 4.2.2.46 PLL\_MAX\_MULTIPLIER

```
#define PLL_MAX_MULTIPLIER 0x176
```

Максимальное для PLL

#### 4.2.2.47 PLL\_MIN\_MULTIPLIER

```
#define PLL_MIN_MULTIPLIER 0x0
```

Минимальное для PLL

### 4.2.3 Перечисления

#### 4.2.3.1 clkctr\_clk\_force\_type

```
enum clkctr\_clk\_force\_type
```

Тип тактирования

Элементы перечислений

CLKCTR_ClkForceTypeDynamic	Динамический
CLKCTR_ClkForceTypeForce	Принудительный

#### 4.2.3.2 clkctr\_device\_clk\_force

enum [clkctr\\_device\\_clk\\_force](#)

Домены, к которым может быть применено динамическое тактирование

Элементы перечислений

CLKCTR_SysSysClkForce	Системный домен общей подсистемы
CLKCTR_SysFClkForce	Системный домен частоты FCLK
CLKCTR_SRAMSysClkForce	Домен подсистемы памяти SRAM
CLKCTR_SRAMFClkForce	Домен SRAM FCLK
CLKCTR_CPUSysClkForce	Домен подсистемы CPU
CLKCTR_CPUFClkForce	Домен CPU FCLK
CLKCTR_CryptoSysClkForce	Домен криптоблока
CLKCTR_CPUDBGPIkClkForce	Домен отладки CPU
CLKCTR_BasePikClkForce	Домен базовый
CLKCTR_SMCClkForce	Домен блока тактирования SMC
CLKCTR_GMSSysClkForce	Домен системного блока GSM

#### 4.2.3.3 clkctr\_extern\_freq

enum [clkctr\\_extern\\_freq](#)

Генератор/осциллятор/вход, для которого групповой функцией устанавливается частота

Элементы перечислений

CLKCTR_ExternFreqXTI	Внешний высокочастотный генератор (вход XTI)
CLKCTR_ExternFreqXTI32	Внешний низкочастотный осциллятор (вход XTI32)
CLKCTR_ExternFreqHFI	Внутренний высокочастотный осциллятор HFI
CLKCTR_ExternFreqLFI	Внутренний высокочастотный осциллятор LFI
CLKCTR_ExternFreqI2SExtClk	Внешняя частота I2S_EXTCLK (PA15)

#### 4.2.3.4 clkctr\_hfi\_for\_main\_type

enum [clkctr\\_hfi\\_for\\_main\\_type](#)

Частота, определенная как HFICLK для MAINCLK (CLKCTR\_MainClkTypeHFICLK)

Заметки

В зависимости от режима работы процессора (задается входом PMUDIS) при переключении MAINCLK на HFICLK частота может быть переключена как на HFICLK, так и на XTICLK.

Элементы перечислений

CLKCTR_HFIForMainTypeHFI	HFI
CLKCTR_HFIForMainTypeXTI	XTI

#### 4.2.3.5 clkctr\_i2sclk

enum [clkctr\\_i2sclk](#)

Подаваемая на I2SCLK частота

Элементы перечислений

CLKCTR_I2SClkTypeSysClk	Внутренний генератор
CLKCTR_I2SClkTypeI2SClk	Внешний вход (PA15)

#### 4.2.3.6 clkctr\_int\_freq

enum [clkctr\\_int\\_freq](#)

Имя внутренней частоты тактирования микросхемы, для которой производится действие групповой функцией

Элементы перечислений

CLKCTR_IntFreqXTIClk	Для внешнего высокочастотного генератора
CLKCTR_IntFreqHFIClk	Для внутреннего высокочастотного генератора
CLKCTR_IntFreqRTCClk	Для генератора реального времени
CLKCTR_IntFreqLPCLk	Для генератора реального времени для тактирования таймеров
CLKCTR_IntFreqPLLClk	Для выходной частоты PLL
CLKCTR_IntFreqPLLRefClk	Для опорной частоты PLL
CLKCTR_IntFreqMainClk	Для главной частоты
CLKCTR_IntFreqUSBClk	Для частоты USB PHY
CLKCTR_IntFreqFClkInt	Для внутренней частоты FCLK_INT
CLKCTR_IntFreqFClk	Для частоты CPU1 и SRAM3
CLKCTR_IntFreqSysClk	Для системной частоты CPU0 и SRAM0-2, AHB/APB
CLKCTR_IntFreqGNSSClk	Для частоты тактирования GNSS
CLKCTR_IntFreqQSPIClk	Для частоты тактирования QSPI
CLKCTR_IntFreqI2SClk	Для частоты тактирования I2S
CLKCTR_IntFreqMCOClk	Для частоты тактирования MCO (выход на PA15)
CLKCTR_IntFreqHFIForMain	Для частоты тактирования MAINCLK при переключении на HFI

#### 4.2.3.7 clkctr\_lpclk

enum [clkctr\\_lpclk](#)

Подаваемая на LPCLK частота



Элементы перечислений

CLKCTR_LPClkTypeRTCClk	От таймера реального времени
CLKCTR_LPClkType500	От ХТІ/500

#### 4.2.3.8 clkctr\_mainclk

enum [clkctr\\_mainclk](#)

Подаваемая на MAINCLK частота

Элементы перечислений

CLKCTR_MainClkTypeHFIClk	Внутренний генератор
CLKCTR_MainClkTypeXTIClk	Внешний генератор
CLKCTR_MainClkTypePLLClk	Блок PLL
CLKCTR_MainClkTypeMax	Максимально допустимое значение

#### 4.2.3.9 clkctr\_mcoclk

enum [clkctr\\_mcoclk](#)

Подаваемая на MCO частота

Элементы перечислений

CLKCTR_MCOClkTypeHFIClk	Частота внутреннего генератора
CLKCTR_MCOClkTypeRTCClk	Частота таймера реального времени
CLKCTR_MCOClkTypeLPClk	Частота таймера реального времени
CLKCTR_MCOClkTypeMainClk	Основная частота
CLKCTR_MCOClkTypePLLClk	Частота от PLL
CLKCTR_MCOClkTypeSysClk	Системная частота
CLKCTR_MCOClkTypeFClkInt	Внутренняя частота процессора F_INTCLK
CLKCTR_MCOClkTypeFClk	Внутренняя частота после динамического управления

#### 4.2.3.10 clkctr\_pllref

enum [clkctr\\_pllref](#)

Опорная частота PLL.

Элементы перечислений

CLKCTR_PLLRefTypeHFIClk	Внутренний генератор
CLKCTR_PLLRefTypeXTIClk	Внешний генератор

#### 4.2.3.11 clkctr\_rtccclk

enum [clkctr\\_rtccclk](#)

Подаваемая на RTCCCLK частота

Элементы перечислений

CLKCTR_RTCClkTypeLFI	От внутреннего RC-осциллятора
CLKCTR_RTCClkTypeLFE	От внешнего RC-осциллятора

#### 4.2.3.12 clkctr\_status

enum [clkctr\\_status](#)

Статусы драйвера CLKCTR.

Элементы перечислений

CLKCTR_Status_Ok	Нет ошибок
CLKCTR_Status_InvalidArgument	Недопустимый аргумент
CLKCTR_Status_CheckError	Получена ошибка от оборудования
CLKCTR_Status_VerifyError	Верификация не прошла
CLKCTR_Status_ConfigureError	Недопустимая конфигурация или ошибка в описании оборудования

#### 4.2.3.13 clkctr\_usbclk

enum [clkctr\\_usbclk](#)

Подаваемая на USBCLK частота

Элементы перечислений

CLKCTR_USBClkTypeHFIClk	Внутренний генератор
CLKCTR_USBClkTypeXTIClk	Внешний генератор

### 4.2.4 Функции

#### 4.2.4.1 CLKCTR\_GetAllDiv()

```
enum clkctr\_status CLKCTR_GetAllDiv (
    CLKCTR_Type * base,
    struct clkctr\_div * divisors)
```

Извлечение всех делителей блока CLKCTR.

Аргументы

base	Блок частот
divisors	Структура с делителями блока

Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_InvalidArgument</a>	

#### 4.2.4.2 CLKCTR\_GetClk()

```
enum clkctr\_status CLKCTR_GetClk (
    CLKCTR_Type * base,
    enum clkctr\_int\_freq clk,
    uint32_t * value)
```

Извлечение значения указанной частоты

Аргументы

base	Блок частот
clk	Частота
value	Значение частоты

Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_InvalidArgument</a>	

#### 4.2.4.3 CLKCTR\_GetClkForce()

```
enum clkctr\_status CLKCTR_GetClkForce (
    CLKCTR_Type * base,
    enum clkctr\_device\_clk\_force device,
    enum clkctr\_clk\_force\_type * clk_type)
```

Извлечение типа тактирования для модуля

Аргументы

base	Блок частот
device	Модуль
clk_type	Тип тактирования

Возвращает

[CLKCTR\\_Status\\_Ok](#)  
[CLKCTR\\_Status\\_InvalidArgument](#)

#### 4.2.4.4 CLKCTR\_GetDivClk()

```
enum clkctr\_status CLKCTR_GetDivClk (
    CLKCTR_Type * base,
    enum clkctr\_int\_freq clk,
    uint32_t * value)
```

Извлечение коэффициента деления для указанной частоты

## Аргументы

base	Блок частот
clk	Конфигурируемая частота
value	Коэффициент деления

## Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_InvalidArgument</a>	

## 4.2.4.5 CLKCTR\_GetFClk()

```
uint32_t CLKCTR_GetFClk (
    CLKCTR_Type * base)
```

Извлечение значения частоты FCLK.

## Заметки

Динамическое тактирование не учитывается.

## Аргументы

base	Блок частот
------	-------------

## Возвращает

Значение частоты в Гц

## 4.2.4.6 CLKCTR\_GetFClkDiv()

```
uint32_t CLKCTR_GetFClkDiv (
    CLKCTR_Type * base)
```

Извлечение значения делителя FCLKDiv.

## Аргументы

base	Блок частот
------	-------------

## Возвращает

Значение делителя

## 4.2.4.7 CLKCTR\_GetFClkInt()

```
uint32_t CLKCTR_GetFClkInt (
    CLKCTR_Type * base)
```

Извлечение значения частоты FCLK\_INT.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение частоты в Гц

#### 4.2.4.8 CLKCTR\_GetFrequency()

```
uint32_t CLKCTR_GetFrequency (  
    CLKCTR_Type * base,  
    enum clkctr\_extern\_freq input)
```

Извлечение значения частоты, подаваемой на выбранный вход

Аргументы

base	Блок частот
input	Вход, для которого определяется частота

Возвращает

Значение частоты в Гц или [CLKCTR\\_FREQ\\_NOT\\_SET](#)

#### 4.2.4.9 CLKCTR\_GetGNSSClk()

```
uint32_t CLKCTR_GetGNSSClk (  
    CLKCTR_Type * base)
```

Извлечение значения частоты GNSSCLK.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение частоты в Гц

#### 4.2.4.10 CLKCTR\_GetGNSSClkDiv()

```
uint32_t CLKCTR_GetGNSSClkDiv (  
    CLKCTR_Type * base)
```

Извлечение значения делителя GNSSCLKDiv.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение делителя

#### 4.2.4.11 CLKCTR\_GetHFI()

```
uint32_t CLKCTR_GetHFI (  
    CLKCTR_Type * base)
```

Извлечение значения частоты внутреннего генератора APC.

Заметки

Частота определена как HFI.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение частоты в Гц или [CLKCTR\\_FREQ\\_NOT\\_SET](#)

#### 4.2.4.12 CLKCTR\_GetHFIClk()

```
uint32_t CLKCTR_GetHFIClk (  
    CLKCTR_Type * base)
```

Извлечение значения частоты HFICLK.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение частоты в Гц

#### 4.2.4.13 CLKCTR\_GetHFIClkForMainClk()

```
uint32_t CLKCTR_GetHFIClkForMainClk (  
    CLKCTR_Type * base)
```

Извлечение значения частоты HFICLK для MAINCLK.

Заметки

HFICLK для MAINCLK зависит от значения сигнала на внешней ножке PMUDIS.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение частоты в Гц

#### 4.2.4.14 CLKCTR\_GetHFITrim()

```
uint32_t CLKCTR_GetHFITrim (  
    CLKCTR_Type * base)
```

Извлечение индекса коэффициента подстройки частоты осциллятора HFI.

Аргументы

base	Блок частот
------	-------------

Возвращает

Индекс коэффициента подстройки частоты

#### 4.2.4.15 CLKCTR\_GetI2SCLK()

```
uint32_t CLKCTR_GetI2SCLK (  
    CLKCTR_Type * base)
```

Извлечение значения частоты I2SCLK.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение частоты в Гц

#### 4.2.4.16 CLKCTR\_GetI2SCLKDiv()

```
uint32_t CLKCTR_GetI2SCLKDiv (  
    CLKCTR_Type * base)
```

Извлечение значения делителя I2SCLKDiv.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение делителя

#### 4.2.4.17 CLKCTR\_GetI2SExtClk()

```
uint32_t CLKCTR_GetI2SExtClk (  
    CLKCTR_Type * base)
```

Извлечение значения частоты, поступающей на PA15.

Заметки

Частота определена как I2S\_EXTCLK.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение частоты в Гц или [CLKCTR\\_FREQ\\_NOT\\_SET](#)

#### 4.2.4.18 CLKCTR\_GetLFI()

```
uint32_t CLKCTR_GetLFI (  
    CLKCTR_Type * base)
```

Извлечение значения частоты внутреннего генератора RWC.

Заметки

Частота определена как LFI.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение частоты в Гц или [CLKCTR\\_FREQ\\_NOT\\_SET](#)

#### 4.2.4.19 CLKCTR\_GetLPCLK()

```
uint32_t CLKCTR_GetLPCLK (  
    CLKCTR_Type * base)
```

Извлечение значения частоты LPCLK.



Аргументы

base	Блок частот
------	-------------

Возвращает

Значение частоты в Гц

#### 4.2.4.20 CLKCTR\_GetMainClk()

```
uint32_t CLKCTR_GetMainClk (  
    CLKCTR_Type * base)
```

Извлечение значения частоты MAINCLK.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение частоты в Гц

#### 4.2.4.21 CLKCTR\_GetMCOClk()

```
uint32_t CLKCTR_GetMCOClk (  
    CLKCTR_Type * base)
```

Извлечение значения частоты MCOCLK, подаваемой на вывод PA15.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение частоты в Гц

#### 4.2.4.22 CLKCTR\_GetMCODiv()

```
uint32_t CLKCTR_GetMCODiv (  
    CLKCTR_Type * base)
```

Извлечение значения делителя MCODiv.

Возвращает

Значение делителя

#### 4.2.4.23 CLKCTR\_GetMCOEn()

```
uint32_t CLKCTR_GetMCOEn (  
    CLKCTR_Type * base)
```

Извлечение разрешения тактового сигнала MCO.

Аргументы

base	Блок частот
------	-------------

Возвращает

- 0 Вывод запрещен
- 1 Вывод разрешен

#### 4.2.4.24 CLKCTR\_GetPLLClk()

```
uint32_t CLKCTR_GetPLLClk (
    CLKCTR_Type * base)
```

Извлечение значения частоты PLLCLK.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение частоты в Гц или [CLKCTR\\_FREQ\\_NOT\\_SET](#), если PLL не вышла в рабочий режим

#### 4.2.4.25 CLKCTR\_GetPLLConfig()

```
enum clkctr\_status CLKCTR_GetPLLConfig (
    CLKCTR_Type * base,
    struct clkctr\_pll\_cfg * config)
```

Извлечение коэффициентов PLL.

Аргументы

base	Блок частот
config	Структура коэффициентов PLL

Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_InvalidArgument</a>	

#### 4.2.4.26 CLKCTR\_GetPLLRef()

```
uint32_t CLKCTR_GetPLLRef (
    CLKCTR_Type * base)
```

Извлечение значения опорной частоты PLL.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение частоты в Гц

#### 4.2.4.27 CLKCTR\_GetQSPIClk()

```
uint32_t CLKCTR_GetQSPIClk (  
    CLKCTR_Type * base)
```

Извлечение значения частоты QSPICLK.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение частоты в Гц

#### 4.2.4.28 CLKCTR\_GetQSPIClkDiv()

```
uint32_t CLKCTR_GetQSPIClkDiv (  
    CLKCTR_Type * base)
```

Извлечение значения делителя QSPICLKDiv.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение делителя

#### 4.2.4.29 CLKCTR\_GetRTCClk()

```
uint32_t CLKCTR_GetRTCClk (  
    CLKCTR_Type * base)
```

Извлечение значения частоты RTCCLK.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение частоты в Гц

#### 4.2.4.30 CLKCTR\_GetSwitchClk()

```
enum clkctr\_status CLKCTR_GetSwitchClk (
    CLKCTR_Type * base,
    enum clkctr\_int\_freq clk,
    uint32_t * value)
```

Извлечение источника частоты для указанной частоты

Аргументы

base	Блок частот
clk	Конфигурируемая частота
value	Индекс источника частоты

Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_InvalidArgument</a>	

#### 4.2.4.31 CLKCTR\_GetSwitchI2SClk()

```
uint32_t CLKCTR_GetSwitchI2SClk (
    CLKCTR_Type * base)
```

Извлечение источника частоты I2SCLK.

Аргументы

base	Блок частот
------	-------------

Возвращает

Источник частоты

#### 4.2.4.32 CLKCTR\_GetSwitchLPCLK()

```
uint32_t CLKCTR_GetSwitchLPCLK (
    CLKCTR_Type * base)
```

Извлечение источника частоты LPCLK.

Аргументы

base	Блок частот
------	-------------

Возвращает

Источник частоты

#### 4.2.4.33 CLKCTR\_GetSwitchMainClk()

```
uint32_t CLKCTR_GetSwitchMainClk (  
    CLKCTR_Type * base)
```

Извлечение источника частоты MAINCLK.

Аргументы

base	Блок частот
------	-------------

Возвращает

Источник частоты

#### 4.2.4.34 CLKCTR\_GetSwitchMCOClk()

```
uint32_t CLKCTR_GetSwitchMCOClk (  
    CLKCTR_Type * base)
```

Извлечение источника частоты MCO.

Аргументы

base	Блок частот
------	-------------

Возвращает

Источник частоты

#### 4.2.4.35 CLKCTR\_GetSwitchPLLRef()

```
uint32_t CLKCTR_GetSwitchPLLRef (  
    CLKCTR_Type * base)
```

Извлечение источника частоты PLL.

Аргументы

base	Блок частот
------	-------------

Возвращает

Источник частоты

#### 4.2.4.36 CLKCTR\_GetSwitchPMUDIS()

```
uint32_t CLKCTR_GetSwitchPMUDIS (
    CLKCTR_Type * base)
```

Определение режима работы процессора

Функция возвращает значение, поданное на вход PMUDIS, которое определяет выбранные источники частоты LPCLK и MAINCLK при выбранном источнике [CLKCTR\\_MainClkTypeHFIClk](#).

Аргументы

base	Блок частот
------	-------------

Возвращаемые значения

0	Логический 0 на входе PMUDIS
1	Логическая 1 на входе PMUDIS

#### 4.2.4.37 CLKCTR\_GetSwitchRTCClk()

```
uint32_t CLKCTR_GetSwitchRTCClk (
    CLKCTR_Type * base)
```

Извлечение источника частоты RTCCLK.

Аргументы

base	Блок частот
------	-------------

Возвращает

Источник частоты

#### 4.2.4.38 CLKCTR\_GetSwitchUSBClk()

```
uint32_t CLKCTR_GetSwitchUSBClk (
    CLKCTR_Type * base)
```

Извлечение источника частоты USBCLK.

Аргументы

base	Блок частот
------	-------------

Возвращает

Источник частоты

#### 4.2.4.39 CLKCTR\_GetSysClk()

```
uint32_t CLKCTR_GetSysClk (  
    CLKCTR_Type * base)
```

Извлечение значения частоты SYSCLK.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение частоты в Гц

#### 4.2.4.40 CLKCTR\_GetSysClkDiv()

```
uint32_t CLKCTR_GetSysClkDiv (  
    CLKCTR_Type * base)
```

Извлечение значения делителя SYSCLKDiv.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение делителя

#### 4.2.4.41 CLKCTR\_GetUSBClk()

```
uint32_t CLKCTR_GetUSBClk (  
    CLKCTR_Type * base)
```

Извлечение значения частоты USBCLK, подаваемого на USB PHY.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение частоты в Гц

#### 4.2.4.42 CLKCTR\_GetXTI()

```
uint32_t CLKCTR_GetXTI (  
    CLKCTR_Type * base)
```

Извлечение значения частоты, подаваемой на вход XTI.

Заметки

Частота определена как XTICK.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение частоты в Гц или [CLKCTR\\_FREQ\\_NOT\\_SET](#)

#### 4.2.4.43 CLKCTR\_GetXTI32()

```
uint32_t CLKCTR_GetXTI32 (  
    CLKCTR_Type * base)
```

Извлечение значения частоты, подаваемой на вход XTI32.

Заметки

Частота определена как LFE.

Аргументы

base	Блок частот
------	-------------

Возвращает

Значение частоты в Гц или [CLKCTR\\_FREQ\\_NOT\\_SET](#)

#### 4.2.4.44 CLKCTR\_GetXTIClk()

```
uint32_t CLKCTR_GetXTIClk (  
    CLKCTR_Type * base)
```

Извлечение значения частоты XTICK.



Аргументы

base	Блок частот
------	-------------

Возвращает

Значение частоты в Гц

#### 4.2.4.45 CLKCTR\_SetClkForce()

```
enum clkctr_status CLKCTR_SetClkForce (
    CLKCTR_Type * base,
    enum clkctr_device_clk_force device,
    enum clkctr_clk_force_type clk_type)
```

Установка типа тактирования для модуля

Аргументы

base	Блок частот
device	Модуль
clk_type	Тип тактирования

Возвращает

CLKCTR\_Status\_Ok  
CLKCTR\_Status\_InvalidArgument

#### 4.2.4.46 CLKCTR\_SetDivClk()

```
enum clkctr_status CLKCTR_SetDivClk (
    CLKCTR_Type * base,
    enum clkctr_int_freq clk,
    uint32_t value)
```

Установка коэффициента деления для указанной частоты

Аргументы

base	Блок частот
clk	Конфигурируемая частота
value	Коэффициент деления

Возвращаемые значения

CLKCTR_Status_Ok	
CLKCTR_Status_InvalidArgument	
CLKCTR_Status_ConfigureError	

## 4.2.4.47 CLKCTR\_SetFClkDiv()

```
enum clkctr_status CLKCTR_SetFClkDiv (
    CLKCTR_Type * base,
    uint32_t value)
```

Установка значения делителя FCLKDiv.

Заметки

Допустимые значения делителя - от 1 до 32.

Аргументы

base	Блок частот
value	Значение делителя

Возвращаемые значения

CLKCTR_Status_Ok	
CLKCTR_Status_InvalidArgument	
CLKCTR_Status_ConfigureError	

## 4.2.4.48 CLKCTR\_SetFrequency()

```
enum clkctr_status CLKCTR_SetFrequency (
    CLKCTR_Type * base,
    enum clkctr_extern_freq input,
    uint32_t frequency)
```

Установка значения частоты, подаваемой на выбранный вход

Аргументы

base	Блок частот
input	Вход, для которого устанавливается частота
frequency	Значение частоты в Гц

Возвращаемые значения

CLKCTR_Status_Ok	
CLKCTR_Status_InvalidArgument	
CLKCTR_Status_ConfigureError	

## 4.2.4.49 CLKCTR\_SetGNSSClkDiv()

```
enum clkctr_status CLKCTR_SetGNSSClkDiv (
    CLKCTR_Type * base,
    uint32_t value)
```

Установка значения делителя GNSSCLKDiv.

Заметки

Допустимые значения делителя - от 1 до 8.

## Аргументы

base	Блок частот
value	Значение делителя

## Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_InvalidArgument</a>	
<a href="#">CLKCTR_Status_ConfigureError</a>	

## 4.2.4.50 CLKCTR\_SetHFI()

```
enum clkctr\_status CLKCTR_SetHFI (
    CLKCTR_Type * base,
    uint32_t frequency)
```

Установка значения частоты внутреннего генератора APC.

## Заметки

Частота определена как HFI.

Установка частоты происходит без учета подстройки HFITRIM.

## Аргументы

base	Блок частот
frequency	Значение частоты в Гц

## Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_ConfigureError</a>	

## 4.2.4.51 CLKCTR\_SetHFITrim()

```
enum clkctr\_status CLKCTR_SetHFITrim (
    CLKCTR_Type * base,
    uint32_t index)
```

Установка индекса коэффициента подстройки частоты осциллятора HFI.

## Заметки

Допустимые значения индекса коэффициента подстройки частоты - от 0 до 0x1f.

## Аргументы

base	Блок частот
index	Индекс коэффициента подстройки частоты

## Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_InvalidArgument</a>	
<a href="#">CLKCTR_Status_ConfigureError</a>	

## 4.2.4.52 CLKCTR\_SetI2SClkDiv()

```
enum clkctr\_status CLKCTR_SetI2SClkDiv (
    CLKCTR_Type * base,
    uint32_t value)
```

Установка значения делителя I2SCLKDiv.

## Заметки

Допустимые значения делителя - от 1 до 32.

## Аргументы

base	Блок частот
value	Значение делителя

## Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_InvalidArgument</a>	
<a href="#">CLKCTR_Status_ConfigureError</a>	

## 4.2.4.53 CLKCTR\_SetI2SExtClk()

```
enum clkctr\_status CLKCTR_SetI2SExtClk (
    CLKCTR_Type * base,
    uint32_t frequency)
```

Установка значения частоты, поступающей на PA15.

## Заметки

Частота определена как I2S\_EXTCLK.

Ножка порта PA15 не переконфигурируется.

## Аргументы

base	Блок частот
frequency	Значение частоты в Гц

## Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_ConfigureError</a>	

## 4.2.4.54 CLKCTR\_SetLFI()

```
enum clkctr\_status CLKCTR_SetLFI (
    CLKCTR_Type * base,
    uint32_t frequency)
```

Установка значения частоты внутреннего генератора RWC.

## Заметки

Частота определена как LFI.

## Аргументы

base	Блок частот
frequency	Значение частоты в Гц

## Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_ConfigureError</a>	

## 4.2.4.55 CLKCTR\_SetMCOdiv()

```
enum clkctr\_status CLKCTR_SetMCOdiv (
    CLKCTR_Type * base,
    uint32_t value)
```

Установка значения делителя MCOdiv.

## Заметки

Допустимые значения делителя - от 1 до 32.

## Аргументы

base	Блок частот
value	Значение делителя

## Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_InvalidArgument</a>	
<a href="#">CLKCTR_Status_ConfigureError</a>	

## 4.2.4.56 CLKCTR\_SetMCOEn()

```
enum clkctr\_status CLKCTR_SetMCOEn (
    CLKCTR_Type * base,
    uint32_t enable)
```

Установка или запрет разрешения тактового сигнала MCO.

## Аргументы

base	Блок частот
enable	0 - запрещает вывод частоты, !=0 - разрешает

## Возвращает

[CLKCTR\\_Status\\_Ok](#)

## 4.2.4.57 CLKCTR\_SetPll()

```
void CLKCTR_SetPll (
    CLKCTR_Type * base,
    uint32_t xti_hz,
    uint32_t pll_mul)
```

Установка целочисленного множителя PLL.

## Заметки

Для изменения делителей перед вызовом этой функции нужно вызвать [CLKCTR\\_SetSysDiv](#).

Для использования внутреннего HFI в качестве частоты внешнего генератора следует передать 0 через xti\_hz.

## Аргументы

base	Блок частот
xti_hz	Частота внешнего генератора в Гц
pll_mul	Целочисленный множитель PLL

## 4.2.4.58 CLKCTR\_SetPLLConfig()

```
enum clkctr\_status CLKCTR_SetPLLConfig (
    CLKCTR_Type * base,
    struct clkctr\_pll\_cfg config)
```

Установка коэффициентов PLL.

Функция устанавливает коэффициенты PLL и ждет завершения конфигурации.

Заметки

В момент использования функции не должно производиться тактирование от PLL.

Аргументы

base	Блок частот
config	Структура коэффициентов PLL

Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_InvalidArgument</a>	
<a href="#">CLKCTR_Status_ConfigureError</a>	

## 4.2.4.59 CLKCTR\_SetPllMan()

```
enum clkctr\_status CLKCTR_SetPllMan (
    CLKCTR_Type * base,
    uint32_t xti_hz,
    uint32_t nr,
    uint32_t nf,
    uint32_t od)
```

Установка множителей PLL в ручном режиме

Заметки

Для изменения делителей перед вызовом этой функции нужно вызвать [CLKCTR\\_SetSysDiv](#).

Для использования внутреннего HFI в качестве частоты внешнего генератора следует передать 0 через xti\_hz.

Умноженная частота перед делителем OD не должна превышать 375МГц  $((xti\_hz / nr) * nf) \leq 375000000$

Предделитель выходной частоты OD может иметь значения 1 или четные значение в диапазоне 2-16.

Аргументы

base	Блок частот
xti_hz	Частота внешнего генератора в Гц
nr	Предделитель опорной частоты
nf	Множитель частоты
od	Предделитель выходной частоты PLL

Возвращаемые значения

<a href="#">CLKCTR_Status_InvalidArgument</a>	
<a href="#">CLKCTR_Status_ConfigureError</a>	
<a href="#">CLKCTR_Status_Ok</a>	

#### 4.2.4.60 CLKCTR\_SetQSPIClkDiv()

```
enum clkctr\_status CLKCTR_SetQSPIClkDiv (
    CLKCTR_Type * base,
    uint32_t value)
```

Установка значения делителя QSPICLKDiv.

Заметки

Допустимые значения делителя - от 1 до 32.

Аргументы

base	Блок частот
value	Значение делителя

Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_InvalidArgument</a>	
<a href="#">CLKCTR_Status_ConfigureError</a>	

#### 4.2.4.61 CLKCTR\_SetSwitchClk()

```
enum clkctr\_status CLKCTR_SetSwitchClk (
    CLKCTR_Type * base,
    enum clkctr\_int\_freq clk,
    uint32_t value)
```

Выбор источника частоты для указанной частоты

Аргументы

base	Блок частот
clk	Конфигурируемая частота
value	Индекс источника частоты

Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_InvalidArgument</a>	
<a href="#">CLKCTR_Status_ConfigureError</a>	



## 4.2.4.62 CLKCTR\_SetSwitchI2SClk()

```
enum clkctr\_status CLKCTR_SetSwitchI2SClk (
    CLKCTR_Type * base,
    uint32_t value)
```

Выбор источника частоты I2SCLK.

Заметки

Индексы источника частоты перечислены в [clkctr\\_i2sclk](#).

Аргументы

base	Блок частот
value	Индекс источника частоты

Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_InvalidArgument</a>	
<a href="#">CLKCTR_Status_ConfigureError</a>	

## 4.2.4.63 CLKCTR\_SetSwitchLPCLK()

```
enum clkctr\_status CLKCTR_SetSwitchLPCLK (
    CLKCTR_Type * base,
    uint32_t value)
```

Выбор источника частоты LPCLK.

Функция переключает источник частоты LPCLK; переключение связано с изменением частоты MAINCLK при выбранном источнике [CLKCTR\\_MainClkTypeHFIClk](#).

Заметки

Индексы источника частоты перечислены в [clkctr\\_lpclk](#).

Аргументы

base	Блок частот
value	Индекс источника частоты

Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_InvalidArgument</a>	
<a href="#">CLKCTR_Status_ConfigureError</a>	

## 4.2.4.64 CLKCTR\_SetSwitchMainClk()

```
enum clkctr\_status CLKCTR_SetSwitchMainClk (
    CLKCTR_Type * base,
    uint32_t value)
```

Выбор источника частоты MAINCLK.

Заметки

Индексы источника частоты перечислены в [clkctr\\_mainclk](#).

Аргументы

base	Блок частот
value	Индекс источника частоты

Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_InvalidArgument</a>	
<a href="#">CLKCTR_Status_ConfigureError</a>	

## 4.2.4.65 CLKCTR\_SetSwitchMCOClk()

```
enum clkctr\_status CLKCTR_SetSwitchMCOClk (
    CLKCTR_Type * base,
    uint32_t value)
```

Выбор источника частоты MCO.

Заметки

Индексы источника частоты перечислены в [clkctr\\_mcoclk](#).

Аргументы

base	Блок частот
value	Индекс источника частоты

Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_InvalidArgument</a>	
<a href="#">CLKCTR_Status_ConfigureError</a>	

## 4.2.4.66 CLKCTR\_SetSwitchPLLRef()

```
enum clkctr\_status CLKCTR_SetSwitchPLLRef (
    CLKCTR_Type * base,
    uint32_t value)
```

Выбор источника опорной частоты PLL.

## Заметки

В момент использования функции не должно производиться тактирование от PLL.

Индексы источника частоты перечислены в [clkctr\\_pllref](#).

## Аргументы

base	Блок частот
value	Индекс источника частоты

## Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_InvalidArgument</a>	
<a href="#">CLKCTR_Status_ConfigureError</a>	

## 4.2.4.67 CLKCTR\_SetSwitchPMUDIS()

```
enum clkctr\_status CLKCTR_SetSwitchPMUDIS (
    CLKCTR_Type * base,
    uint32_t value)
```

Выбор режима работы процессора

Значение, подаваемое на вход PMUDIS, определяет, какие выбраны источники частоты LPCLK и MAINCLK при выбранном источнике [CLKCTR\\_MainClkTypeHFIClk](#).

## Заметки

Физически выбор определяется уровнем напряжения на входе микросхемы.

Допустимые значения, подаваемые на вход PMUDIS - 0 и 1; 0 соответствует логическому нулю на физическом входе PMUDIS, а !0

- единице.

## Аргументы

base	Блок частот
value	Значение, подаваемое на вход PMUDIS

Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
----------------------------------	--

#### 4.2.4.68 CLKCTR\_SetSwitchRTCClk()

```
enum clkctr\_status CLKCTR_SetSwitchRTCClk (
    CLKCTR_Type * base,
    uint32_t value)
```

Выбор источника частоты RTCCLK.

Заметки

Индексы источника частоты перечислены в [clkctr\\_rtccclk](#).

Аргументы

base	Блок частот
value	Индекс источника частоты

Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_InvalidArgument</a>	
<a href="#">CLKCTR_Status_ConfigureError</a>	

#### 4.2.4.69 CLKCTR\_SetSwitchUSBClk()

```
enum clkctr\_status CLKCTR_SetSwitchUSBClk (
    CLKCTR_Type * base,
    uint32_t value)
```

Выбор источника частоты USBCLK.

Заметки

Индексы источника частоты перечислены в [clkctr\\_usbclk](#).

Аргументы

base	Блок частот
value	Индекс источника частоты

Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_InvalidArgument</a>	
<a href="#">CLKCTR_Status_ConfigureError</a>	

#### 4.2.4.70 CLKCTR\_SetSysClkDiv()

```
enum clkctr\_status CLKCTR_SetSysClkDiv (
    CLKCTR_Type * base,
    uint32_t value)
```

Установка значения делителя SYSCLKDiv.

Заметки

Допустимые значения делителя - от 1 до 32.

Аргументы

base	Блок частот
value	Значение делителя

Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_InvalidArgument</a>	
<a href="#">CLKCTR_Status_ConfigureError</a>	

#### 4.2.4.71 CLKCTR\_SetSysDiv()

```
void CLKCTR_SetSysDiv (
    CLKCTR_Type * base,
    uint16_t fclk_div,
    uint16_t sysclk_div,
    uint16_t gnssclk_div,
    uint16_t qspiclk_div)
```

Установка делителей

Аргументы

base	Блок частот
fclk_div	Делитель частоты FCLK
sysclk_div	Делитель частоты SYSCLK
gnssclk_div	Делитель частоты GNSSCLK
qspiclk_div	Делитель частоты QSPICLK

#### 4.2.4.72 CLKCTR\_SetXTI()

```
enum clkctr_status CLKCTR_SetXTI (
    CLKCTR_Type * base,
    uint32_t frequency)
```

Установка значения частоты, подаваемой на вход XTI.

##### Заметки

Частота определена как XTICLK.

##### Аргументы

base	Блок частот
frequency	Значение частоты в Гц

##### Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_ConfigureError</a>	

#### 4.2.4.73 CLKCTR\_SetXTI32()

```
enum clkctr_status CLKCTR_SetXTI32 (
    CLKCTR_Type * base,
    uint32_t frequency)
```

Установка значения частоты, подаваемой на вход XTI32.

##### Заметки

Частота определена как LFE.

##### Аргументы

base	Блок частот
frequency	Значение частоты в Гц

##### Возвращаемые значения

<a href="#">CLKCTR_Status_Ok</a>	
<a href="#">CLKCTR_Status_ConfigureError</a>	

## 4.3 Общие определения для библиотеки HAL

Общие определения, используемые библиотекой HAL.

## Макросы

- `#define UNUSED(x) ((void)(x));`
- `#define MAKE_VERSION(major, minor, bugfix) (((major) << 16) | ((minor) << 8) | (bugfix))`  
Составление номера версии для драйверов
- `#define HAL_COMMON_DRIVER_VERSION (MAKE_VERSION(0, 1, 0))`  
Версия драйвера common.
- `#define BE_TO_LE32(x)`  
Приведение 32-битного Big-endian числа к Little-endian представлению
- `#define BE_TO_LE24(x)`  
Приведение 24-битного Big-endian числа к Little-endian представлению
- `#define BE_TO_LE16(x)`  
Приведение 16-битного Big-endian числа к Little-endian представлению
- `#define MIN(a, b) (((a) < (b)) ? (a) : (b))`  
Выбор минимального значения из двух вариантов
- `#define MAX(a, b) (((a) > (b)) ? (a) : (b))`  
Выбор максимального значения из двух вариантов
- `#define DIM(x) (sizeof(x) / sizeof((x)[0]))`  
Вычисление количества элементов в массиве
- `#define UINT16_MAX ((uint16_t) -1)`
- `#define UINT32_MAX ((uint32_t) -1)`
- `#define FIELD_BIT(top, bottom) ((top) - (bottom) + 1)`  
Вычисление длины битового поля

## Определения типов

- `typedef int32_t status_t`  
Тип, используемый для всех возвращаемых значений статуса и ошибок

## Функции

- `void LBBLEndian (uint32_t *val, uint8_t num_bits)`  
Преобразование формата представления данных Big-endian <-> Little-endian.

## Макрос подавления предупреждения switch-fallthrough

При использовании ARM GCC с включенными флагами `-Wextra` и `-Wimplicit-fallthrough=n` для switch-case блока с разделами case, не оканчивающихся оператором break, будет выдаваться предупреждение. Для подавления данного предупреждения, необходимо использовать предлагаемый макрос в конце каждого case, который не "закрыт" оператором break.

- `#define SUPPRESS_FALL_THROUGH_WARNING() /*!< Подавление предупреждения switch-fallthrough */`

## 4.3.1 Подробное описание

Общие определения, используемые библиотекой HAL.

### 4.3.2 Макросы

#### 4.3.2.1 BE\_TO\_LE16

```
#define BE_TO_LE16(  
    x)
```

Макроопределение:

```
((((unsigned)(x) » 8) & 0xFF) \
 | (((unsigned)(x) « 8) & 0xFF00))
```

Приведение 16-битного Big-endian числа к Little-endian представлению

Аргументы

x	16-битное число с Big-endian порядком байтов
---	--

Возвращает

16-битное число с Little-endian порядком байтов

#### 4.3.2.2 BE\_TO\_LE24

```
#define BE_TO_LE24(  
    x)
```

Макроопределение:

```
((((unsigned)(x) » 16) & 0xFF) \
 | ((unsigned)(x) & 0xFF00) \
 | (((unsigned)(x) « 16) & 0xFF0000))
```

Приведение 24-битного Big-endian числа к Little-endian представлению

Аргументы

x	24-битное число с Big-endian порядком байтов
---	--

Возвращает

24-битное число с Little-endian порядком байтов

#### 4.3.2.3 BE\_TO\_LE32

```
#define BE_TO_LE32(  
    x)
```

Макроопределение:

```
((((unsigned)(x) » 24) & 0xFF) \
 | (((unsigned)(x) « 24) & 0xFF000000) \
 | (((unsigned)(x) » 8) & 0xFF00) \
 | (((unsigned)(x) « 8) & 0xFF0000))
```

Приведение 32-битного Big-endian числа к Little-endian представлению



Аргументы

x	32-битное число с Big-endian порядком байтов
---	--

Возвращает

32-битное число с Little-endian порядком байтов

#### 4.3.2.4 DIM

```
#define DIM(  
    x) (sizeof(x) / sizeof((x)[0]))
```

Вычисление количества элементов в массиве

Аргументы

x	Массив
---	--------

Возвращает

Количество элементов в массиве

#### 4.3.2.5 FIELD\_BIT

```
#define FIELD_BIT(  
    top,  
    bottom) ((top) - (bottom) + 1)
```

Вычисление длины битового поля

Аргументы

top	Старший бит поля
bottom	Младший бит поля

Возвращает

Длина битового поля

#### 4.3.2.6 MAX

```
#define MAX(  
    a,  
    b) (((a) > (b)) ? (a) : (b))
```

Выбор максимального значения из двух вариантов

Аргументы

a	Первое значение
b	Второе значение

Возвращает

Максимальное значение

#### 4.3.2.7 MIN

```
#define MIN(  
    a,  
    b) (((a) < (b)) ? (a) : (b))
```

Выбор минимального значения из двух вариантов

Аргументы

a	Первое значение
b	Второе значение

Возвращает

Минимальное значение

#### 4.3.2.8 SUPPRESS\_FALL\_THROUGH\_WARNING

```
#define SUPPRESS_FALL_THROUGH_WARNING() /*!< Подавление предупреждения switch-fallthrough */
```

Подавление предупреждения switch-fallthrough

#### 4.3.2.9 UINT16\_MAX

```
#define UINT16_MAX ((uint16_t) -1)
```

Максимальное значение 16-битного беззнакового целого числа

#### 4.3.2.10 UINT32\_MAX

```
#define UINT32_MAX ((uint32_t) -1)
```

Максимальное значение 32-битного беззнакового целого числа

## 4.3.2.11 UNUSED

```
#define UNUSED(
    x) ((void)(x));
```

TODO

## 4.3.3 Функции

## 4.3.3.1 LBBLEndian()

```
void LBBLEndian (
    uint32_t * val,
    uint8_t num_bits)
```

Преобразование формата представления данных Big-endian <-> Little-endian.

Заметки

Размер числа может быть от 1 до 32 битов.

Аргументы

val	Преобразуемое число
num_bits	Размер числа в битах

## 4.4 Драйвер модуля DMA

Драйвер прямого доступа к памяти

Файлы

- файл [hal\\_dma.h](#)  
Интерфейс драйвера прямого доступа к памяти

Структуры данных

- struct [\\_dma\\_descriptor](#)  
Описатель следующего блока DMA (LLI)
- struct [\\_dma\\_channel\\_reg](#)  
Дескриптор на регистры канала DMA.
- struct [dma\\_channel\\_ctl\\_cfg](#)  
Описание конфигурации пересылки
- struct [\\_dma\\_channel\\_config](#)  
Конфигурация канала DMA.
- struct [\\_dma\\_handle](#)  
Управляющая структура передачи
- struct [\\_dma\\_multiblock\\_config](#)  
Конфигурация многоблочной передачи

## Макросы

- `#define HAL_DMA_DRIVER_VERSION (MAKE_VERSION(1, 0, 0))`  
Версия драйвера
- `#define DMA_AHB_MAX_BLOCK_SIZE (4095U)`  
Максимальный размер транзакции на AHB шине
- `#define HAL_FEATURE_DMA_NUMBER_OF_CHANNELS (8U)`  
Число каналов одного контроллера DMA.
- `#define HAL_FEATURE_DMA_MAX_NUMBER_HW_HANDSHAKES (16U)`  
Максимальное число аппаратных интерфейсов запросов DMA.
- `#define HAL_FEATURE_DMA_ALL_CHANNELS (HAL_FEATURE_DMA_NUMBER_OF_CHANNELS * 2U)`  
Общее число каналов DMA.
- `#define HAL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE (4U)`  
Размер выравнивания дескриптора
- `#define DMA_ALLOCATE_HEAD_DESCRIPTOR(name, number) dma_descriptor_t name[number] __attribute__((aligned(HAL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE)))`  
Объявление дескриптора и выравнивание адреса дескриптора
- `#define REG_OFFSET (0x58)`  
Разница адресов идентичных регистров из соседних каналов
- `#define DMA_HANDSHAKE_UART0_TX (0U)`  
Номер handshake запроса UART0 TX.
- `#define DMA_HANDSHAKE_UART0_RX (1U)`  
Номер handshake запроса UART0 RX.
- `#define DMA_HANDSHAKE_UART1_TX (2U)`  
Номер handshake запроса UART1 TX.
- `#define DMA_HANDSHAKE_UART1_RX (3U)`  
Номер handshake запроса UART1 RX.
- `#define DMA_HANDSHAKE_UART2_TX (0U)`  
Номер handshake запроса UART2 TX.
- `#define DMA_HANDSHAKE_UART2_RX (1U)`  
Номер handshake запроса UART2 RX.
- `#define DMA_HANDSHAKE_UART3_TX (2U)`  
Номер handshake запроса UART3 TX.
- `#define DMA_HANDSHAKE_UART3_RX (3U)`  
Номер handshake запроса UART3 RX.
- `#define DMA_HANDSHAKE_SPI0_TX (4U)`  
Номер handshake запроса SPI0 TX.
- `#define DMA_HANDSHAKE_SPI0_RX (5U)`  
Номер handshake запроса SPI0 RX.
- `#define DMA_HANDSHAKE_SPI1_TX (6U)`  
Номер handshake запроса SPI1 TX.
- `#define DMA_HANDSHAKE_SPI1_RX (7U)`  
Номер handshake запроса SPI1 RX.
- `#define DMA_HANDSHAKE_SPI2_TX (8U)`  
Номер handshake запроса SPI2 TX.
- `#define DMA_HANDSHAKE_SPI2_RX (9U)`  
Номер handshake запроса SPI2 RX.
- `#define DMA_HANDSHAKE_I2C0_TX (10U)`  
Номер handshake запроса I2C0 TX.
- `#define DMA_HANDSHAKE_I2C0_RX (11U)`

- Номер handshake запроса I2C0 RX.
- `#define DMA_HANDSHAKE_I2C1_TX (12U)`  
Номер handshake запроса I2C1 TX.
- `#define DMA_HANDSHAKE_I2C1_RX (13U)`  
Номер handshake запроса I2C1 RX.
- `#define DMA_HANDSHAKE_QSPI_TX (14U)`  
Номер handshake запроса QSPI TX.
- `#define DMA_HANDSHAKE_QSPI_RX (15U)`  
Номер handshake запроса QSPI RX.

#### CHANNEL\_CTL - Регистр управления каналом

- enum  
Разрядность передачи канала DMA.
- enum  
Размер пакета передачи в словах данных разрядностью DMA\_TransferBitWidth.
- enum  
Тип изменения адреса при пересылке
- enum `_dma_status`  
Статусы возврата из функций для драйвера DMA.
- enum `_dma_priority`  
Приоритеты каналов
- enum `_dma_int`  
Источники прерываний DMA.
- enum `_dma_transfer_type`  
Тип пересылок DMA и управление размером блока
- typedef enum `_dma_status` dma\_status\_t  
Статусы возврата из функций для драйвера DMA.
- typedef enum `_dma_priority` dma\_priority\_t  
Приоритеты каналов
- typedef enum `_dma_int` dma\_irq\_t  
Источники прерываний DMA.
- typedef enum `_dma_transfer_type` dma\_transfer\_type\_t  
Тип пересылок DMA и управление размером блока
- typedef struct `_dma_descriptor` dma\_descriptor\_t  
Описатель следующего блока DMA (LLI)
- typedef struct `_dma_channel_reg` dma\_channel\_regs  
Дескриптор на регистры канала DMA.
- typedef struct `_dma_channel_config` dma\_channel\_config\_t  
Конфигурация канала DMA.
- typedef void(\* dma\_callback) (struct `_dma_handle` \*handle, void \*user\_data, dma\_irq\_t intmode)  
Функция обратного вызова
- typedef struct `_dma_handle` dma\_handle\_t  
Управляющая структура передачи
- typedef struct `_dma_multiblock_config` dma\_multiblock\_config\_t  
Конфигурация многоблочной передачи
- `#define DMA_CHANNEL_CTL_BLOCKSIZE_MASK (0xFFFF00000000ULL)`
- `#define DMA_CHANNEL_CTL_BLOCKSIZE_SHIFT (32ULL)`
- `#define DMA_CHANNEL_CTL_BLOCKSIZE(x) (((uint64_t)((uint64_t)(x)) << DMA_CHANNEL_CTL_BLOCKSIZE_SHIFT) & DMA_CHANNEL_CTL_BLOCKSIZE_MASK)`

- Размер блока
- `#define DMA_CHANNEL_CTL(blockSize, srcEnChain, dstEnChain, transferType, scatter, gather, srcMSize, dstMSize, sInc, dInc, srcTransferWidth, dstTransferWidth, int_en)`  
Конфигурация регистра CTL.

## API HAL для драйвера модуля DMA

- `void DMA_Init (DMA_Type *base)`  
Включение DMA.
- `void DMA_Deinit (DMA_Type *base)`  
Отключение DMA.
- `dma_status_t DMA_CreateHandle (dma_handle_t *handle, DMA_Type *base, uint32_t channel)`  
Инициализация структуры `dma_handle_t`.
- `static void DMA_EnableChannel (DMA_Type *base, uint32_t channel)`  
Активация канала
- `void DMA_DisableChannel (DMA_Type *base, uint32_t channel)`  
Выключение канала
- `static void DMA_EnableChannelPeriphRq (DMA_Type *base, uint32_t channel)`  
Установка запроса на передачу с периферией
- `void DMA_HardwareHandshakeEnable (dma_handle_t *handle, bool enable, uint32_t req_num)`  
Установка аппаратного интерфейса запроса DMA.
- `void DMA_SetChannelPriority (DMA_Type *base, uint32_t channel, dma_priority_t priority)`  
Установка приоритета канала
- `void DMA_SetCallback (dma_handle_t *handle, dma_callback callback, void *user_data)`  
Установка функции обратного вызова
- `static bool DMA_ChannelsActive (DMA_Type *base, uint32_t channel)`  
Функция для проверки работы DMA канала
- `dma_priority_t DMA_GetChannelPriority (DMA_Type *base, uint32_t channel)`  
Получение информации о приоритете канала
- `uint64_t DMA_GetCTLCfgMask (struct dma_channel_ctl_cfg *ctlxcfg)`  
Получение маски для конфигурации пересылки
- `void DMA_PrepareChannelTransfer (dma_channel_config_t *config, void *src_addr, void *dst_addr, uint64_t xfer_cfg)`  
Подготовка канала к передаче
- `dma_status_t DMA_SubmitChannelTransfer (dma_handle_t *handle, dma_channel_config_t *config)`  
Отправление конфигурации передачи
- `void DMA_SubmitChannelDescriptor (dma_handle_t *handle, dma_descriptor_t *descriptor)`  
Отправка дескриптора передачи
- `uint32_t DMA_GetDescriptorCount (uint32_t size_bytes, uint8_t transfer_width)`  
Расчёт количества дескрипторов многоблочной передачи
- `void DMA_AbortTransfer (dma_handle_t *handle)`  
Прерывание передачи без потери данных
- `dma_status_t DMA_StartTransfer (dma_handle_t *handle)`  
Начать транзакцию
- `void DMA_EnableChannelInterrupt (DMA_Type *base, uint32_t channel, uint8_t int_mask)`  
Разрешение источника прерывания
- `void DMA_DisableChannelInterrupt (DMA_Type *base, uint32_t channel, uint8_t int_mask)`  
Отключение источника прерывания

- void `DMA_SetupDescriptor` (`dma_descriptor_t` \*desc, uint64\_t xfercfg, void \*src\_addr, void \*dst\_addr, void \*next\_desc)  
Инициализация дескриптора
- `dma_status_t` `DMA_SubmitChannelTransferParameter` (`dma_handle_t` \*handle, uint64\_t xfercfg, void \*src\_addr, void \*dst\_addr)  
Отправка конфигурации передачи в таблицу дескрипторов
- void `DMA_ChannelIRQHandle` (`dma_handle_t` \*handle)  
Обработчик прерываний
- void `DMA_ScatterGatherEnable` (struct `dma_channel_ctl_cfg` \*ctlxcfg, bool scatter\_en, bool gather\_en)  
Разрешение режимов Разброса/Сбора
- void `DMA_SetupSrcGather` (`dma_handle_t` \*handle, uint32\_t count, uint32\_t interval)  
Настройка режима сбора источника
- void `DMA_SetupDstScatter` (`dma_handle_t` \*handle, uint32\_t count, uint32\_t interval)  
Настройка режима разброса приемника
- void `DMA_InitMultiblockDescriptor` (`dma_descriptor_t` \*desc, `dma_multiblock_config_t` \*config)  
Функция, инициализирующая DMA дескрипторы многоблочной передачи.

#### 4.4.1 Подробное описание

Драйвер прямого доступа к памяти

Драйвер модуля DMA управляет каналами контроллеров DMA0 и DMA1.

#### 4.4.2 Макросы

##### 4.4.2.1 DMA\_CHANNEL\_CTL

```
#define DMA_CHANNEL_CTL(
    blockSize,
    srcEnChain,
    dstEnChain,
    transferType,
    scatter,
    gather,
    srcMSize,
    dstMSize,
    sInc,
    dInc,
    srcTransferWidth,
    dstTransferWidth,
    int_en)
```

Макроопределение:

```
0x0ULL | DMA_CHANNEL_CTL_BLOCKSIZE(blockSize) | \
DMA_CTL0_LO_LLPSRC_EN_VAL(srcEnChain) | \
DMA_CTL0_LO_LLPDST_EN_VAL(dstEnChain) | \
DMA_CTL0_LO_TT_FC_VAL(transferType) | \
DMA_CTL0_LO_DST_SCATTER_EN_VAL(scatter) | \
DMA_CTL0_LO_SRC_GATHER_EN_VAL(gather) | \
DMA_CTL0_LO_SRC_MSIZESIZE_VAL(srcMSize) | \
DMA_CTL0_LO_DEST_MSIZESIZE_VAL(dstMSize) | \
DMA_CTL0_LO_SINC_VAL(sInc) | \
DMA_CTL0_LO_DINC_VAL(dInc) | \
DMA_CTL0_LO_SRC_TR_WIDTH_VAL(srcTransferWidth) | \
DMA_CTL0_LO_DST_TR_WIDTH_VAL(dstTransferWidth) | \
DMA_CTL0_LO_INT_EN_VAL(int_en)
```

Конфигурация регистра CTL.

## Аргументы

blockSize	Размер блока
srcEnChain	Разрешение цепочки для источника
dstEnChain	Разрешение цепочки для источника
transferType	Тип передачи и управление размером блока
scatter	Разрешение разброса приемника
gather	Разрешение сбора источника
srcMSize	Размер пакета источника
dstMSize	Размер пакета приемника
sInc	Тип изменения адреса источника
dInc	Тип изменения адреса приемника
srcTransferWidth	Разрядность обращения источника
dstTransferWidth	Разрядность обращения приемника
int_en	Разрешение прерываний

## 4.4.2.2 DMA\_CHANNEL\_CTL\_BLOCKSIZE\_MASK

```
#define DMA_CHANNEL_CTL_BLOCKSIZE_MASK (0xFFFF00000000ULL)
```

Маска поля 'Размер блока' в регистре CTL

## 4.4.2.3 DMA\_CHANNEL\_CTL\_BLOCKSIZE\_SHIFT

```
#define DMA_CHANNEL_CTL_BLOCKSIZE_SHIFT (32ULL)
```

Смещение поля 'Размер блока' в регистре CTL

## 4.4.3 Перечисления

## 4.4.3.1 anonymous enum

```
anonymous enum
```

Тип изменения адреса при пересылке

Элементы перечислений

DMA_Incr	Инкремент адреса
DMA_Decr	Декремент адреса
DMA_NoChange	Адрес не изменяется

## 4.4.3.2 anonymous enum

```
anonymous enum
```

Разрядность передачи канала DMA.



Элементы перечислений

DMA_Transfer8BitWidth	Ширина одного слова 8 бит
DMA_Transfer16BitWidth	Ширина одного слова 16 бит
DMA_Transfer32BitWidth	Ширина одного слова 32 бит
DMA_Transfer64BitWidth	Ширина одного слова 64 бит
DMA_Transfer128BitWidth	Ширина одного слова 128 бит
DMA_Transfer256BitWidth	Ширина одного слова 256 бит

#### 4.4.3.3 anonymous enum

anonymous enum

Размер пакета передачи в словах данных разрядностью DMA\_TransferBitWidth.

Элементы перечислений

DMA_BurstSize1	Размер пакета 1
DMA_BurstSize4	Размер пакета 4
DMA_BurstSize8	Размер пакета 8
DMA_BurstSize16	Размер пакета 16
DMA_BurstSize32	Размер пакета 32
DMA_BurstSize64	Размер пакета 64
DMA_BurstSize128	Размер пакета 128
DMA_BurstSize256	Размер пакета 256

#### 4.4.3.4 \_dma\_int

enum [\\_dma\\_int](#)

Источники прерываний DMA.

Элементы перечислений

DMA_IntTfr	Завершение DMA пересылки
DMA_IntBlock	Завершение передачи блока
DMA_IntDstTran	Завершение передачи Приемнику
DMA_IntSrcTran	Завершение передачи источнику
DMA_IntError	Ошибка передачи АНВ
DMA_AllIRQ	Маска всех прерываний.

#### 4.4.3.5 \_dma\_priority

enum [\\_dma\\_priority](#)

Приоритеты каналов

Элементы перечислений

DMA_ChannelPriority0	Низший приоритет 0
DMA_ChannelPriority1	Приоритет 1
DMA_ChannelPriority2	Приоритет 2
DMA_ChannelPriority3	Приоритет 3
DMA_ChannelPriority4	Приоритет 4
DMA_ChannelPriority5	Приоритет 5
DMA_ChannelPriority6	Приоритет 6
DMA_ChannelPriority7	Наивысший приоритет 7

#### 4.4.3.6 \_dma\_status

enum [\\_dma\\_status](#)

Статусы возврата из функций для драйвера DMA.

Элементы перечислений

DMA_Status_Success	Функция выполнена успешно
DMA_Status_Busy	Выбранный канал DMA занят
DMA_Status_NoBase	Указанного базового адреса DMA не существует
DMA_Status_Fail	Ошибка выполнения функции

#### 4.4.3.7 \_dma\_transfer\_type

enum [\\_dma\\_transfer\\_type](#)

Тип пересылок DMA и управление размером блока

Элементы перечислений

DMA_MemoryToMemory_DMA	Память-Память, DMA управляет размером блока
DMA_MemoryToPeripheral_DMA	Память-Периферия, DMA управляет размером блока
DMA_PeripheralToMemory_DMA	Периферия-Память, DMA управляет размером блока
DMA_PeripheralToPeripheral_DMA	Периферия-Периферия, DMA управляет размером блока
DMA_PeripheralToMemory_Peripheral	Периферия-Память, периферия управляет размером блока
DMA_PeripheralToPeripheral_SRC	Периферия-Периферия, источник управляет размером блока
DMA_MemoryToPeripheral_Peripheral	Память-Периферия, периферия управляет размером блока
DMA_PeripheralToPeripheral_DST	Периферия-Периферия, приемник управляет размером блока

#### 4.4.4 Функции

##### 4.4.4.1 DMA\_AbortTransfer()

```
void DMA_AbortTransfer (
    dma_handle_t * handle)
```

Прерывание передачи без потери данных

Аргументы

handle	Управляющая структура передачи
--------	--------------------------------

##### 4.4.4.2 DMA\_ChannelIRQHandle()

```
void DMA_ChannelIRQHandle (
    dma_handle_t * handle)
```

Обработчик прерываний

Аргументы

handle	Дескриптор DMA
--------	----------------

##### 4.4.4.3 DMA\_ChannelIsActive()

```
static bool DMA_ChannelIsActive (
    DMA_Type * base,
    uint32_t channel) [inline], [static]
```

Функция для проверки работы DMA канала

Аргументы

base	Базовый адрес DMA
channel	Номер канала

Возвращаемые значения

true	- канал активен
false	- канал не активен

##### 4.4.4.4 DMA\_CreateHandle()

```
dma_status_t DMA_CreateHandle (
    dma_handle_t * handle,
    DMA_Type * base,
    uint32_t channel)
```

Инициализация структуры dma\_handle\_t.

## Аргументы

handle	Управляющая структура передачи
base	Базовый адрес DMA
channel	Номер канала

## Возвращаемые значения

<a href="#">DMA_Status_Success</a>	
<a href="#">DMA_Status_NoBase</a>	

## 4.4.4.5 DMA\_Deinit()

```
void DMA_Deinit (
    DMA_Type * base)
```

Отключение DMA.

## Аргументы

base	Базовый адрес DMA
------	-------------------

## 4.4.4.6 DMA\_DisableChannel()

```
void DMA_DisableChannel (
    DMA_Type * base,
    uint32_t channel)
```

Выключение канала

## Аргументы

base	Базовый адрес DMA
channel	Номер канала

## 4.4.4.7 DMA\_DisableChannelInterrupt()

```
void DMA_DisableChannelInterrupt (
    DMA_Type * base,
    uint32_t channel,
    uint8_t int_mask)
```

Отключение источника прерывания

## Аргументы

base	Базовый адрес DMA
channel	Номер канала
inttype	Тип прерывания

## 4.4.4.8 DMA\_EnableChannel()

```
static void DMA_EnableChannel (
    DMA_Type * base,
    uint32_t channel) [inline], [static]
```

Активация канала

Аргументы

base	Базовый адрес DMA
channel	Номер канала

## 4.4.4.9 DMA\_EnableChannelInterrupt()

```
void DMA_EnableChannelInterrupt (
    DMA_Type * base,
    uint32_t channel,
    uint8_t int_mask)
```

Разрешение источника прерывания

Функция разрешает формирование прерывания типа inttype. Также для возникновения сигнала прерывания важно установить поле INT\_EN в 1 в регистре CTLx.

Аргументы

base	Базовый адрес DMA
channel	Номер канала
inttype	Тип прерывания

## 4.4.4.10 DMA\_EnableChannelPeriphRq()

```
static void DMA_EnableChannelPeriphRq (
    DMA_Type * base,
    uint32_t channel) [inline], [static]
```

Установка запроса на передачу с периферией

Аргументы

base	Базовый адрес DMA
channel	Номер канала

## 4.4.4.11 DMA\_GetChannelPriority()

```
dma_priority_t DMA_GetChannelPriority (
    DMA_Type * base,
    uint32_t channel)
```

Получение информации о приоритете канала

Эта функция считывает приоритет из регистра CFGx.

## Аргументы

base	Базовый адрес DMA
channel	Номер канала

## Возвращаемые значения

<a href="#">DMA_ChannelPriority0</a>	
<a href="#">DMA_ChannelPriority1</a>	
<a href="#">DMA_ChannelPriority2</a>	
<a href="#">DMA_ChannelPriority3</a>	
<a href="#">DMA_ChannelPriority4</a>	
<a href="#">DMA_ChannelPriority5</a>	
<a href="#">DMA_ChannelPriority6</a>	
<a href="#">DMA_ChannelPriority7</a>	

## 4.4.4.12 DMA\_GetCTLCfgMask()

```
uint64_t DMA_GetCTLCfgMask (
    struct dma_channel_ctl_cfg * ctlxcfg)
```

Получение маски для конфигурации пересылки

## Аргументы

ctlxcfg	Описание конфигурации пересылки
---------	---------------------------------

## Возвращает

Маска для конфигурации пересылки

## 4.4.4.13 DMA\_GetDescriptorCount()

```
uint32_t DMA_GetDescriptorCount (
    uint32_t size_bytes,
    uint8_t transfer_width)
```

Расчёт количества дескрипторов многоблочной передачи

## Аргументы

size_bytes	Объем данных в байтах
transfer_width	Ширина одного слова

## Возвращает

Количество дескрипторов многоблочной передачи

## 4.4.4.14 DMA\_HardwareHandshakeEnable()

```
void DMA_HardwareHandshakeEnable (
    dma_handle_t * handle,
    bool enable,
    uint32_t req_num)
```

Установка аппаратного интерфейса запроса DMA.

Аргументы

handle	Управляющая структура передачи
enable	Разрешение аппаратного интерфейса
req_num	Номер периферийного блока

## 4.4.4.15 DMA\_Init()

```
void DMA_Init (
    DMA_Type * base)
```

Включение DMA.

Аргументы

base	Базовый адрес DMA
------	-------------------

## 4.4.4.16 DMA\_InitMultiblockDescriptor()

```
void DMA_InitMultiblockDescriptor (
    dma_descriptor_t * desc,
    dma_multiblock_config_t * config)
```

Функция, инициализирующая DMA дескрипторы многоблочной передачи.

Аргументы

desc	Указатель на начало массива дескрипторов многоблочной передачи
config	Разрешения прерываний

## 4.4.4.17 DMA\_PrepareChannelTransfer()

```
void DMA_PrepareChannelTransfer (
    dma_channel_config_t * config,
    void * src_addr,
    void * dst_addr,
    uint64_t xfer_cfg)
```

Подготовка канала к передаче

Эта функция инициализирует структуру dma\_channel\_config\_t

## Аргументы

config	Указатель на структуру конфигурации канала
src_addr	Начальный адрес Источника
dst_addr	Начальный адрес Приемника
xfer_cfg	Конфигурация регистра CTL

## 4.4.4.18 DMA\_ScatterGatherEnable()

```
void DMA_ScatterGatherEnable (
    struct dma_channel_ctl_cfg * ctlcfg,
    bool scatter_en,
    bool gather_en)
```

Разрешение режимов Разброса/Сбора

## Аргументы

ctlcfg	Конфигурация пересылки
scatter_en	Разрешение режима Разброса
gather_en	Разрешение режима Сбора

## 4.4.4.19 DMA\_SetCallback()

```
void DMA_SetCallback (
    dma_handle_t * handle,
    dma_callback callback,
    void * user_data)
```

Установка функции обратного вызова

## Аргументы

handle	Управляющая структура передачи
callback	Функция обратного вызова
user_data	Данные пользователя

## 4.4.4.20 DMA\_SetChannelPriority()

```
void DMA_SetChannelPriority (
    DMA_Type * base,
    uint32_t channel,
    dma_priority_t priority)
```

Установка приоритета канала



## Аргументы

base	Базовый адрес DMA
channel	Номер канала
priority	Приоритет

## 4.4.4.21 DMA\_SetupDescriptor()

```
void DMA_SetupDescriptor (
    dma_descriptor_t * desc,
    uint64_t xfercfg,
    void * src_addr,
    void * dst_addr,
    void * next_desc)
```

## Инициализация дескриптора

## Аргументы

desc	Дескриптор DMA
xfercfg	Конфигурация передачи для дескриптора
src_addr	Адрес источника
dst_addr	Адрес приемника
next_desc	Следующий дескриптор

## 4.4.4.22 DMA\_SetupDstScatter()

```
void DMA_SetupDstScatter (
    dma_handle_t * handle,
    uint32_t count,
    uint32_t interval)
```

## Настройка режима разброса приемника

## Аргументы

handle	Управляющая структура передачи
count	Счетчик разброса
interval	Интервал разброса

## 4.4.4.23 DMA\_SetupSrcGather()

```
void DMA_SetupSrcGather (
    dma_handle_t * handle,
    uint32_t count,
    uint32_t interval)
```

## Настройка режима сбора источника

## Аргументы

handle	Управляющая структура передачи
count	Счетчик сбора
interval	Интервал сбора

## 4.4.4.24 DMA\_StartTransfer()

```
dma_status_t DMA_StartTransfer (
    dma_handle_t * handle)
```

## Начать транзакцию

Функция выгружает дескриптор из таблицы дескрипторов и инициализирует регистр LLPx адресом выгруженного дескриптора. После чего включается канал и начинается передача.

## Аргументы

handle	Управляющая структура передачи
--------	--------------------------------

## Возвращаемые значения

DMA_Status_Success	
DMA_Status_NoBase	
DMA_Status_Busy	

## 4.4.4.25 DMA\_SubmitChannelDescriptor()

```
void DMA_SubmitChannelDescriptor (
    dma_handle_t * handle,
    dma_descriptor_t * descriptor)
```

## Отправка дескриптора передачи

Функция используется для конфигурации канала дескриптором, который начнет Многоблочную передачу.

```
DMA_SetupDescriptor(next_desc0, xfercfg, src_addr, dst_addr, next_desc1);
DMA_SetupDescriptor(next_desc1, xfercfg, src_addr, dst_addr, next_desc2);
DMA_SetupDescriptor(next_desc2, xfercfg, src_addr, dst_addr, NULL);
DMA_CreateHandle(handle, base, channel)
uint64_t xfercfg = DMA_CHANNEL_CTL(
    LEN, true, true,
    DMA_MemoryToMemory_DMA,
    false, false,
    DMA_BurstSize4, DMA_BurstSize4,
    DMA_Incr, DMA_Incr,
    DMA_Transfer32BitWidth, DMA_Transfer32BitWidth,
    true
);
DMA_SubmitChannelDescriptor(handle, next_desc0)
DMA_StartTransfer(handle)
```

## Аргументы

handle	Управляющая структура передачи
descriptor	Дескриптор передачи

## 4.4.4.26 DMA\_SubmitChannelTransfer()

```
dma_status_t DMA_SubmitChannelTransfer (
    dma_handle_t * handle,
    dma_channel_config_t * config)
```

Отправление конфигурации передачи

Функция инициализирует регистры канала в соответствии с параметрами структуры dma\_channel\_config\_t. Используется для одноканальной передачи:

```
DMA_CreateHandle(handle, base, channel);
uint64_t xfercfg = DMA_CHANNEL_CTL(
    LEN, true, true,
    DMA_MemoryToMemory_DMA,
    false, false,
    DMA_BurstSize4, DMA_BurstSize4,
    DMA_Incr, DMA_Incr,
    DMA_Transfer32BitWidth, DMA_Transfer32BitWidth,
    true
);
DMA_PrepareChannelTransfer(config, src_addr, dst_addr, xfercfg)
DMA_SubmitChannelTransfer(handle, config);
DMA_StartTransfer(handle)
```

Аргументы

handle	Структура передачи
config	Структура конфигурации передачи DMA канала

Возвращаемые значения

DMA_Status_Success	
DMA_Status_Busy	
DMA_Status_NoBase	

## 4.4.4.27 DMA\_SubmitChannelTransferParameter()

```
dma_status_t DMA_SubmitChannelTransferParameter (
    dma_handle_t * handle,
    uint64_t xfer_cfg,
    void * src_addr,
    void * dst_addr)
```

Отправка конфигурации передачи в таблицу дескрипторов

Использование: Одноканальная передача

```
uint64_t xfercfg = DMA_CHANNEL_CTL(
    LEN, false, false,
    DMA_MemoryToMemory_DMA,
    false, false,
    DMA_BurstSize4, DMA_BurstSize4,
    DMA_Incr, DMA_Incr,
    DMA_Transfer32BitWidth, DMA_Transfer32BitWidth,
    true
);
DMA_CreateHandle(handle, base, channel)
DMA_SubmitChannelTransferParameter(handle, xfercfg, src_addr, dst_addr);
DMA_StartTransfer(handle)
```

## Аргументы

handle	Управляющая структура передачи
xfer_cfg	Маска конфигурации пересылки
src_addr	Адрес данных передатчика
dst_addr	Адрес данных приемника

## Возвращаемые значения

<a href="#">DMA_Status_Success</a>	
<a href="#">DMA_Status_Busy</a>	
<a href="#">DMA_Status_NoBase</a>	

## 4.5 Драйвер модуля DTIM

## Драйвер сдвоенного таймера

## Файлы

- файл [hal\\_dualtimer.h](#)  
Интерфейс драйвера сдвоенного таймера

## Структуры данных

- struct [dualtimer\\_hardware\\_config](#)  
Конфигурация аппаратной части сдвоенного таймера

## Макросы

- `#define` [DUALTIMER\\_NUMBER\\_OF\\_DUALTIMERS](#) 1
- `#define` [DUALTIMER\\_MAX\\_INDEX](#) 1

## Перечисления

- enum [dualtimer\\_status](#)  
Статусы драйвера сдвоенного таймера
- enum [dualtimer\\_work\\_enable](#)  
Разрешение работы таймера
- enum [dualtimer\\_mode](#)  
Режим работы таймера
- enum [dualtimer\\_interrupt\\_control](#)  
Управление прерываниями
- enum [dualtimer\\_prescale](#)  
Предделители частоты
- enum [dualtimer\\_timer\\_size](#)  
Размер счетчика
- enum [dualtimer\\_number\\_of\\_repetitions](#)  
Количество запусков

## Интерфейс драйвера

- enum `dualtimer_status` `DUALTIMER_GetDefaultConfig` (struct `dualtimer_hardware_config` \*config)  
Создание конфигурации по умолчанию
- enum `dualtimer_status` `DUALTIMER_Init` (DTIM\_Type \*base, uint32\_t index, struct `dualtimer_hardware_config` config)  
Инициализация сдвоенного таймера
- enum `dualtimer_status` `DUALTIMER_Deinit` (DTIM\_Type \*base, uint32\_t index)  
Деинициализация сдвоенного таймера
- enum `dualtimer_status` `DUALTIMER_Run` (DTIM\_Type \*base, uint32\_t index)  
Запуск сдвоенного таймера
- enum `dualtimer_status` `DUALTIMER_Stop` (DTIM\_Type \*base, uint32\_t index)  
Останов сдвоенного таймера
- uint32\_t `DUALTIMER_GetRawStatus` (DTIM\_Type \*base, uint32\_t index)  
Получение немаскированного статуса сдвоенного таймера
- uint32\_t `DUALTIMER_GetStatus` (DTIM\_Type \*base, uint32\_t index)  
Получение маскированного статуса сдвоенного таймера
- uint32\_t `DUALTIMER_GetTick` (DTIM\_Type \*base, uint32\_t index)  
Получение количества тиков
- enum `dualtimer_status` `DUALTIMER_GetAPIStatus` ()  
Получение результата последнего выполнения функции
- enum `dualtimer_status` `DUALTIMER_Reload` (DTIM\_Type \*base, uint32\_t index, uint32\_t value)  
Немедленная перезапись значения таймера
- enum `dualtimer_status` `DUALTIMER_IrqClr` (DTIM\_Type \*base, uint32\_t index)  
Сброс прерывания от таймера

## 4.5.1 Подробное описание

Драйвер сдвоенного таймера

Драйвер модуля сдвоенного таймера управляет сдвоенным таймером DTIM.

## 4.5.2 Макросы

## 4.5.2.1 DUALTIMER\_MAX\_INDEX

```
#define DUALTIMER_MAX_INDEX 1
```

Максимальный индекс таймера

## 4.5.2.2 DUALTIMER\_NUMBER\_OF\_DUALTIMERS

```
#define DUALTIMER_NUMBER_OF_DUALTIMERS 1
```

Количество сдвоенных таймеров

## 4.5.3 Перечисления

## 4.5.3.1 dualtimer\_interrupt\_control

```
enum dualtimer_interrupt_control
```

Управление прерываниями

Элементы перечислений

DUALTIMER_InterruptDisable	Запрещение прерывания
DUALTIMER_InterruptEnable	Разрешение прерывания

#### 4.5.3.2 dualtimer\_mode

enum [dualtimer\\_mode](#)

Режим работы таймера

Элементы перечислений

DUALTIMER_FreeRunning	Свободный счет (с переполнением)
DUALTIMER_Periodic	Счет с заданным периодом

#### 4.5.3.3 dualtimer\_number\_of\_repetitions

enum [dualtimer\\_number\\_of\\_repetitions](#)

Количество запусков

Элементы перечислений

DUALTIMER_WrappingMode	Многократный автоматический запуск
DUALTIMER_OneShot	Одиночный запуск

#### 4.5.3.4 dualtimer\_prescale

enum [dualtimer\\_prescale](#)

Предделители частоты

Элементы перечислений

DUALTIMER_Prescale1	На 1
DUALTIMER_Prescale16	На 16
DUALTIMER_Prescale256	На 256

#### 4.5.3.5 dualtimer\_status

enum [dualtimer\\_status](#)

Статусы драйвера сдвоенного таймера

Элементы перечислений

DUALTIMER_Status_Ok	Нет ошибок
DUALTIMER_Status_InvalidArgument	Недопустимый аргумент
DUALTIMER_Status_TimerBusy	Таймер уже занят
DUALTIMER_Status_BadConfigure	Недопустимая конфигурация

#### 4.5.3.6 dualtimer\_timer\_size

enum [dualtimer\\_timer\\_size](#)

Размер счетчика

Элементы перечислений

DUALTIMER_TimerSize16	16-битный
DUALTIMER_TimerSize32	32-битный

#### 4.5.3.7 dualtimer\_work\_enable

enum [dualtimer\\_work\\_enable](#)

Разрешение работы таймера

Элементы перечислений

DUALTIMER_Disable	Запрет работы таймера
DUALTIMER_Enable	Разрешение работы таймера

### 4.5.4 Функции

#### 4.5.4.1 DUALTIMER\_Deinit()

```
enum dualtimer\_status DUALTIMER_Deinit (
    DTIM_Type * base,
    uint32_t index)
```

Деинициализация сдвоенного таймера

Останов сдвоенного таймера, ружим таймера становится как после сброса

Аргументы

base	Таймер
index	Индекс таймера в сдвоенном таймере

Возвращаемые значения

<a href="#">DUALTIMER_Status_Ok</a>	
<a href="#">DUALTIMER_Status_InvalidArgument</a>	

#### 4.5.4.2 DUALTIMER\_GetAPIStatus()

enum [dualtimer\\_status](#) DUALTIMER\_GetAPIStatus ()

Получение результата последнего выполнения функции

Получает ошибки выполнения функций, у которых тип возвращаемого результата отличен от enum [dualtimer\\_status](#)

Возвращаемые значения

<a href="#">DUALTIMER_Status_Ok</a>	
<a href="#">DUALTIMER_Status_InvalidArgument</a>	
<a href="#">DUALTIMER_Status_TimerBusy</a>	
<a href="#">DUALTIMER_Status_BadConfigure</a>	

#### 4.5.4.3 DUALTIMER\_GetDefaultConfig()

enum [dualtimer\\_status](#) DUALTIMER\_GetDefaultConfig (  
struct [dualtimer\\_hardware\\_config](#) \* config)

Создание конфигурации по умолчанию

Создание конфигурации по умолчанию заполняет структуру такими значениями, которые находятся в регистрах после сброса.

Заметки

Если применить эти значения, вызвав функцию [DUALTIMER\\_Init](#), то сразу возникнет прерывание, так как значение регистра `LOAD = 0` и прерывания разрешены.

Возвращаемые значения

<a href="#">DUALTIMER_Status_Ok</a>	
<a href="#">DUALTIMER_Status_InvalidArgument</a>	

#### 4.5.4.4 DUALTIMER\_GetRawStatus()

uint32\_t DUALTIMER\_GetRawStatus (  
DTIM\_Type \* base,  
uint32\_t index)

Получение немаскированного статуса сдвоенного таймера

Получение немаскированного статуса сдвоенного таймера. Корректность выполнения функции можно проверить вызовом

[DUALTIMER\\_GetAPIStatus](#)



Аргументы

base	Таймер
index	Индекс таймера в сдвоенном таймере

Возвращает

Статус

#### 4.5.4.5 DUALTIMER\_GetStatus()

```
uint32_t DUALTIMER_GetStatus (  
    DTIM_Type * base,  
    uint32_t index)
```

Получение маскированного статуса сдвоенного таймера

Получение маскированного статуса сдвоенного таймера. Корректность выполнения функции можно проверить вызовом [DUALTIMER\\_GetAPIStatus](#)

Аргументы

base	Таймер
index	Индекс таймера в сдвоенном таймере

Возвращает

Статус

#### 4.5.4.6 DUALTIMER\_GetTick()

```
uint32_t DUALTIMER_GetTick (  
    DTIM_Type * base,  
    uint32_t index)
```

Получение количества тиков

Аргументы

base	Таймер
index	Индекс таймера в сдвоенном таймере

Возвращаемые значения

Количество	подсчитанных тактов
------------	---------------------

#### 4.5.4.7 DUALTIMER\_Init()

```
enum dualtimer_status DUALTIMER_Init (  
    DTIM_Type * base,  
    uint32_t index,  
    struct dualtimer_hardware_config config)
```

Инициализация сдвоенного таймера

## Аргументы

base	Сдвоенный таймер
index	Индекс таймера в сдвоенном таймере
config	Конфигурация таймера в сдвоенном таймере

## Возвращаемые значения

<a href="#">DUALTIMER_Status_Ok</a>	
<a href="#">DUALTIMER_Status_InvalidArgument</a>	
<a href="#">DUALTIMER_Status_TimerBusy</a>	

## 4.5.4.8 DUALTIMER\_IrqClr()

```
enum dualtimer\_status DUALTIMER_IrqClr (
    DTIM_Type * base,
    uint32_t index)
```

Сброс прерывания от таймера

Сброс прерывания от таймера, не влияет на NVIC

## Аргументы

base	Таймер
index	Индекс таймера в сдвоенном таймере

## Возвращаемые значения

<a href="#">DUALTIMER_Status_Ok</a>	
<a href="#">DUALTIMER_Status_InvalidArgument</a>	

## 4.5.4.9 DUALTIMER\_Reload()

```
enum dualtimer\_status DUALTIMER_Reload (
    DTIM_Type * base,
    uint32_t index,
    uint32_t value)
```

Немедленная перезапись значения таймера

Перезаписывает значение таймера. В режиме [DUALTIMER\\_OneShot](#) перезапускает таймер, если тот был остановлен.

## Аргументы

base	Таймер
index	Индекс таймера в сдвоенном таймере
value	Загружаемое значение

Возвращаемые значения

DUALTIMER_Status_Ok	
DUALTIMER_Status_InvalidArgument	

#### 4.5.4.10 DUALTIMER\_Run()

```
enum dualtimer_status DUALTIMER_Run (
    DTIM_Type * base,
    uint32_t index)
```

Запуск сдвоенного таймера

Аргументы

base	Таймер
index	Индекс таймера в сдвоенном таймере

Возвращаемые значения

DUALTIMER_Status_Ok	
DUALTIMER_Status_InvalidArgument	

#### 4.5.4.11 DUALTIMER\_Stop()

```
enum dualtimer_status DUALTIMER_Stop (
    DTIM_Type * base,
    uint32_t index)
```

Останов сдвоенного таймера

Останов сдвоенного таймера режимы таймера остаются прежними

Аргументы

base	Таймер
index	Индекс таймера в сдвоенном таймере

Возвращаемые значения

DUALTIMER_Status_Ok	
DUALTIMER_Status_InvalidArgument	

## 4.6 Драйвер модуля FLASH.

Драйвер накристалльной FLASH-памяти.

## Файлы

- файл [hal\\_flash.h](#)  
Интерфейс драйвера модуля FLASH.

## Макросы

- `#define FCTR_IRQ_STS_SET_RESULT_FLAGS`
- `#define FCTR_IRQ_STS_CLR_SUCCESS_FLAGS`

## Перечисления

- enum [flash\\_status](#)  
Статусы драйвера модуля FLASH.
- enum [flash\\_region](#)  
Регионы накристалльной FLASH-памяти.

## API HAL для драйвера модуля FLASH.

- enum [flash\\_status](#) [FLASH\\_Init](#) (FCTR\_Type \*base)  
Инициализация накристалльной FLASH-памяти.
- enum [flash\\_status](#) [FLASH\\_WriteWord](#) (FCTR\_Type \*base, uint32\_t addr, uint32\_t data)  
Запись 32-битного слова во встроенную FLASH-память.
- enum [flash\\_status](#) [FLASH\\_Program](#) (FCTR\_Type \*base, uint32\_t \*addr, uint32\_t \*src, uint32\_t length)  
Запись данных во встроенную FLASH-память.
- enum [flash\\_status](#) [FLASH\\_VerifyProgram](#) (uint32\_t \*addr, uint32\_t length, uint32\_t \*expected\_data, uint32\_t \*failed\_address, uint32\_t \*failed\_data)  
Проверка корректности данных, записанных во внутреннюю FLASH-память.
- enum [flash\\_status](#) [FLASH\\_Erase](#) (FCTR\_Type \*base, uint32\_t \*addr, uint32\_t length)  
Стирание сектора накристалльной FLASH-памяти.
- enum [flash\\_status](#) [FLASH\\_MassErase](#) (FCTR\_Type \*base, enum [flash\\_region](#) region)  
Стирание раздела накристалльной FLASH-памяти.
- enum [flash\\_status](#) [FLASH\\_VerifyErase](#) (uint32\_t \*addr, uint32\_t length)  
Проверка корректности стирания данных накристалльной FLASH-памяти.
- enum [flash\\_status](#) [FLASH\\_Read](#) (uint32\_t \*addr, uint32\_t \*dest, uint32\_t length)  
Чтение данных из накристалльной FLASH-памяти.
- `#define FLASH_TEST_ADDRESSES(address, length)`  
Проверка корректности адреса и количества записываемых/стираемых байтов.

## Команды для накристалльной FLASH-памяти.

- `#define FCTR_CMD_READ (0x1)`
- `#define FCTR_CMD_WRITE (0x2)`
- `#define FCTR_CMD_ROW_WRITE (0x3)`
- `#define FCTR_CMD_ERASE (0x4)`
- `#define FCTR_CMD_MASS_ERASE (0x7)`

### 4.6.1 Подробное описание

Драйвер накристалльной FLASH-памяти.

Заметки

Драйвер накристалльной FLASH-памяти позволяет стирать и записывать данные в память.

### 4.6.2 Макросы

#### 4.6.2.1 FCTR\_CMD\_ERASE

```
#define FCTR_CMD_ERASE (0x4)
```

Стирание страницы.

#### 4.6.2.2 FCTR\_CMD\_MASS\_ERASE

```
#define FCTR_CMD_MASS_ERASE (0x7)
```

Стирание всего региона FLASH-памяти.

#### 4.6.2.3 FCTR\_CMD\_READ

```
#define FCTR_CMD_READ (0x1)
```

Чтение слова.

#### 4.6.2.4 FCTR\_CMD\_ROW\_WRITE

```
#define FCTR_CMD_ROW_WRITE (0x3)
```

Быстрая запись слова.

#### 4.6.2.5 FCTR\_CMD\_WRITE

```
#define FCTR_CMD_WRITE (0x2)
```

Запись слова.

#### 4.6.2.6 FCTR\_IRQ\_STS\_CLR\_SUCCESS\_FLAGS

```
#define FCTR_IRQ_STS_CLR_SUCCESS_FLAGS
```

Макроопределение:

```
(FCTR_IRQ_STATUS_CLR_CMD_SUCCESS_IRQ_STS_CLR_Msk \
 | FCTR_IRQ_STATUS_CLR_CMD_ACCEPT_IRQ_STS_CLR_Msk)
```

Объединение статусов SUCCESS и ACCEPT.

#### 4.6.2.7 FCTR\_IRQ\_STS\_SET\_RESULT\_FLAGS

```
#define FCTR_IRQ_STS_SET_RESULT_FLAGS
```

Макроопределение:

```
(FCTR_IRQ_STATUS_SET_READ_OVERFLOW_IRQ_STS_SET_Msk \
 | FCTR_IRQ_STATUS_SET_CMD_REJECT_IRQ_STS_SET_Msk \
 | FCTR_IRQ_STATUS_SET_CMD_FAIL_IRQ_STS_SET_Msk \
 | FCTR_IRQ_STATUS_SET_CMD_SUCCESS_IRQ_STS_SET_Msk)
```

Объединение всех статусов, кроме ACCEPT.

#### 4.6.2.8 FLASH\_TEST\_ADDRESSES

```
#define FLASH_TEST_ADDRESSES(
    address,
    length)
```

Проверка корректности адреса и количества записываемых/стираемых байтов.

Аргументы

address	Стартовый адрес записи/стирания.
length	Количество записываемых/стираемых байтов.

Возвращаемые значения

<a href="#">FLASH_Status_Ok</a>	
<a href="#">FLASH_Status_AddressAlignmentError</a>	
<a href="#">FLASH_Status_AddressOutOfRange</a>	

### 4.6.3 Перечисления

#### 4.6.3.1 flash\_region

enum [flash\\_region](#)

Регионы накристалльной FLASH-памяти.

Элементы перечислений

FLASH_MainRegion	Основной раздел.
FLASH_SystemRegion	Системный раздел.

#### 4.6.3.2 flash\_status

enum [flash\\_status](#)

Статусы драйвера модуля FLASH.

Элементы перечислений

FLASH_Status_Ok	Нет ошибок.
FLASH_Status_InvalidArgument	Недопустимый параметр.
FLASH_Status_CheckError	Получена ошибка от оборудования.
FLASH_Status_VerifyError	Ошибка верификации данных.
FLASH_Status_ConfigureError	Недопустимая конфигурация или ошибка в описании оборудования.
FLASH_Status_AddressAlignmentError	Адрес не выровнен по странице/слову.
FLASH_Status_AddressOutOfRange	Адрес вне допустимого диапазона.

#### 4.6.4 Функции

##### 4.6.4.1 FLASH\_Erase()

```
enum flash_status FLASH_Erase (
    FCTR_Type * base,
    uint32_t * addr,
    uint32_t length)
```

Стирание сектора накристалльной FLASH-памяти.

Заметки

К аргументам функции применяются следующие ограничения:

- адрес должен быть выровнен по границе 8 КВ страницы;
- в значении адреса учитываются только младшие биты [20:0] (младшие 21 бит);
- количество стираемых байтов должно быть кратно размеру страницы.

Аргументы

base	Базовый адрес регистров FLASH-памяти.
addr	Начальный адрес стирания.
length	Количество стираемых байтов.

Возвращаемые значения

FLASH_Status_Ok	
FLASH_Status_InvalidArgument	
FLASH_Status_CheckError	
FLASH_Status_AddressAlignmentError	
FLASH_Status_AddressOutOfRange	

##### 4.6.4.2 FLASH\_Init()

```
enum flash_status FLASH_Init (
    FCTR_Type * base)
```

Инициализация накристалльной FLASH-памяти.

## Аргументы

base	Базовый адрес регистров FLASH-памяти.
------	---------------------------------------

## Возвращаемые значения

<a href="#">FLASH_Status_Ok</a>	
<a href="#">FLASH_Status_ConfigureError</a>	

## 4.6.4.3 FLASH\_MassErase()

```
enum flash\_status FLASH_MassErase (
    FCTR_Type * base,
    enum flash\_region region)
```

Стирание раздела накристалльной FLASH-памяти.

## Аргументы

base	Базовый адрес регистров FLASH-памяти.
region	Раздел памяти.

## Возвращаемые значения

<a href="#">FLASH_Status_Ok</a>	
<a href="#">FLASH_Status_CheckError</a>	

## 4.6.4.4 FLASH\_Program()

```
enum flash\_status FLASH_Program (
    FCTR_Type * base,
    uint32_t * addr,
    uint32_t * src,
    uint32_t length)
```

Запись данных во встроенную FLASH-память.

## Заметки

К аргументам функции применяются следующие ограничения:

- в значении адреса учитываются только младшие биты [20:0] (младшие 21 бит);
- записываемые данные должны быть выровнены по границе 32-битного слова;
- количество записываемых байтов должно быть кратно размеру страницы.



## Аргументы

base	Базовый адрес регистров FLASH-памяти.
addr	Начальный адрес записи.
src	Записываемые данные.
length	Количество записываемых байтов.

## Возвращаемые значения

FLASH_Status_Ok	
FLASH_Status_ConfigureError	
FLASH_Status_AddressAlignmentError	
FLASH_Status_AddressOutOfRange	

## 4.6.4.5 FLASH\_Read()

```
enum flash_status FLASH_Read (
    uint32_t * addr,
    uint32_t * dest,
    uint32_t length)
```

Чтение данных из накристалльной FLASH-памяти.

## Заметки

К аргументам функции применяются следующие ограничения:

- адрес чтения должен быть выровнен на размер страницы;
- в значении адреса учитываются только младшие биты [20:0] (младшие 21 бит);
- адрес буфера под данные должен быть выровнен по границе 32-битного слова;
- количество считываемых байтов должно быть кратно размеру страницы.

## Аргументы

addr	Начальный адрес чтения.
dest	Считываемые данные.
length	Количество считываемых байтов.

## Возвращаемые значения

FLASH_Status_Ok	
FLASH_Status_InvalidArgument	
FLASH_Status_CheckError	
FLASH_Status_AddressAlignmentError	
FLASH_Status_AddressOutOfRange	
FLASH_Status_AddressAlignmentError	

## 4.6.4.6 FLASH\_VerifyErase()

```
enum flash_status FLASH_VerifyErase (
    uint32_t * addr,
    uint32_t length)
```

Проверка корректности стирания данных накристалльной FLASH-памяти.

## Заметки

К аргументам функции применяются следующие ограничения:

- адрес должен быть выровнен на размер страницы;
- в значении адреса учитываются только младшие биты [20:0] (младшие 21 бит);
- количество байтов должно быть кратно размеру страницы.

## Аргументы

base	Базовый адрес регистров FLASH-памяти.
addr	Начальный адрес проверяемой памяти.
length	Количество проверяемых байтов.

## Возвращаемые значения

FLASH_Status_Ok	
FLASH_Status_ConfigureError	
FLASH_Status_VerifyError	
FLASH_Status_AddressAlignmentError	
FLASH_Status_AddressOutOfRange	

## 4.6.4.7 FLASH\_VerifyProgram()

```
enum flash_status FLASH_VerifyProgram (
    uint32_t * addr,
    uint32_t length,
    uint32_t * expected_data,
    uint32_t * failed_address,
    uint32_t * failed_data)
```

Проверка корректности данных, записанных во внутреннюю FLASH-память.

## Заметки

К аргументам функции применяются следующие ограничения:

- начальный адрес чтения должен быть выровнен на размер страницы;
- в значении адреса учитываются только младшие биты [20:0] (младшие 21 бит);
- количество проверяемых байтов должно быть кратно размеру страницы;
- адрес эталонных данных должен быть выровнен по границе 32-битного слова.

## Аргументы

base	Базовый адрес регистров FLASH-памяти.
addr	Начальный адрес проверяемых данных.
length	Количество проверяемых байтов.
expected_data	Эталонные данные.
failed_address	Адрес первых несовпадающих данных.
failed_data	Первые несовпадающие данные.

## Возвращаемые значения

<a href="#">FLASH_Status_Ok</a>	
<a href="#">FLASH_Status_InvalidArgument</a>	
<a href="#">FLASH_Status_ConfigureError</a>	
<a href="#">FLASH_Status_AddressAlignmentError</a>	
<a href="#">FLASH_Status_AddressOutOfRange</a>	

## 4.6.4.8 FLASH\_WriteWord()

```
enum flash_status FLASH_WriteWord (
    FCTR_Type * base,
    uint32_t addr,
    uint32_t data)
```

Запись 32-битного слова во встроенную FLASH-память.

## Заметки

К аргументам функции применяются следующие ограничения:

- адрес должен быть выровнен по границе 32-битного слова;
- в значении адреса учитываются только младшие биты [20:0] (младшие 21 бит).

## Аргументы

base	Базовый адрес регистров FLASH-памяти.
addr	Адрес для записи.
data	Записываемое слово.

## Возвращаемые значения

<a href="#">FLASH_Status_Ok</a>	
<a href="#">FLASH_Status_CheckError</a>	
<a href="#">FLASH_Status_AddressAlignmentError</a>	
<a href="#">FLASH_Status_AddressOutOfRange</a>	

## 4.7 Драйвер модуля GPIO

Драйвер для управления внешними выводами

Файлы

- файл [hal\\_gpio.h](#)

Макросы

- `#define GPIO_PORTA 0U`
- `#define GPIO_PORTB 1U`
- `#define GPIO_PORTC 2U`
- `#define GPIO_PORTD 3U`
- `#define GPIO_PORTPIN(port, pin) (((port) & 0xF) << 4) | ((pin) & 0xF)`
- `#define GPIO_PORTPIN_GET_PIN_NUM(portpin) ((portpin) & 0xF)`
- `#define GPIO_PORTPIN_GET_MASK(portpin) (1 << GPIO_PORTPIN_GET_PIN_NUM(portpin))`
- `#define GPIO_PORTPIN_GET_PORT_NUM(portpin) (((portpin) >> 4) & 0xF)`

Режимы работы выводов GPIO

- `enum gpio_mode_t`  
Список режимов работы вывода GPIO.
- `enum gpio_pin_function_t`  
Список альтернативных функций IOCTR выводов устройств

### 4.7.1 Подробное описание

Драйвер для управления внешними выводами

Драйвер содержит функции управления выводами микросхемы Eliot01 в режимах программного управления состоянием вывода (GPIO модуль), а также установкой альтернативной функции драйвера для работы с устройствами (IOCTR модуль).

### 4.7.2 Макросы

#### 4.7.2.1 GPIO\_PORTA

```
#define GPIO_PORTA 0U
```

Номер порта A

#### 4.7.2.2 GPIO\_PORTB

```
#define GPIO_PORTB 1U
```

Номер порта B

#### 4.7.2.3 GPIO\_PORTC

```
#define GPIO_PORTC 2U
```

Номер порта C

#### 4.7.2.4 GPIO\_PORTD

```
#define GPIO_PORTD 3U
```

Номер порта D

#### 4.7.2.5 GPIO\_PORTPIN

```
#define GPIO_PORTPIN(  
    port,  
    pin) (((port) & 0xF) << 4) | ((pin) & 0xF))
```

Создать соответствие порт-вывод

#### 4.7.2.6 GPIO\_PORTPIN\_GET\_MASK

```
#define GPIO_PORTPIN_GET_MASK(  
    portpin) (1 << GPIO_PORTPIN_GET_PIN_NUM(portpin))
```

Получить маску вывода из соответствия порт-вывод

#### 4.7.2.7 GPIO\_PORTPIN\_GET\_PIN\_NUM

```
#define GPIO_PORTPIN_GET_PIN_NUM(  
    portpin) ((portpin) & 0xF)
```

Получить номер вывода из соответствия порт-вывод

#### 4.7.2.8 GPIO\_PORTPIN\_GET\_PORT\_NUM

```
#define GPIO_PORTPIN_GET_PORT_NUM(  
    portpin) (((portpin) >> 4) & 0xF)
```

Получить номер порта из соответствия порт-вывод

### 4.7.3 Перечисления

#### 4.7.3.1 gpio\_mode\_t

```
enum gpio_mode_t
```

Список режимов работы вывода GPIO.

Элементы перечислений

GPIO_MODE_HI_Z	Режим высокоимпендансного состояния вывода
GPIO_MODE_GPIO	Режим программируемого вывода GPIO
GPIO_MODE_AF	Режим альтернативной функции вывода для работы с устройствами
GPIO_MODE_INVALID	Несуществующий, неверный режим вывода

#### 4.7.3.2 gpio\_pin\_function\_t

enum [gpio\\_pin\\_function\\_t](#)

Список альтернативных функций IOCTR выводов устройств

Элементы перечислений

GPIO_ALT_FUNC_TRACE_JTAG_FBIST	Альтернативная функция вывода для работы JTRACE, JTAG и FBIST
GPIO_ALT_FUNC_PWM_VTU	Альтернативная функция вывода для работы PWM и VTU
GPIO_ALT_FUNC_I2C_I2S	Альтернативная функция вывода для работы I2C и I2S
GPIO_ALT_FUNC_SPI0_SPI1	Альтернативная функция вывода для работы SPI0 и SPI1
GPIO_ALT_FUNC_UART	Альтернативная функция вывода для работы UART0, UART1, UART2 и UART3
GPIO_ALT_FUNC_CAN_GNSS_USB	Альтернативная функция вывода для работы CAN, GNSS и USB
GPIO_ALT_FUNC_QSPI_SPI2	Альтернативная функция вывода для работы ASPI и SPI2
GPIO_ALT_FUNC_SDMMC_SMC	Альтернативная функция вывода для работы SDMMC и SMC

## 4.8 Драйвер модуля I2C

Драйвер последовательной асимметричной шины для связи между интегральными схемами внутри приборов.

Файлы

- файл [hal\\_i2c.h](#)  
Интерфейс драйвера модуля I2C.
- файл [hal\\_i2c\\_dma.h](#)  
Дополнение драйвера I2C с пересылкой данных через DMA.

## Структуры данных

- struct `i2c_master_config_t`  
Структура с настройками для инициализации Master-модуля I2C.
- struct `i2c_master_transfer_t`  
Структура дескриптора для неблокирующего обмена.
- struct `_i2c_master_handle`  
Дескриптор для работы по прерыванию.
- struct `i2c_slave_config_t`  
Структура с настройками для инициализации Slave-модуля I2C.
- struct `i2c_slave_transfer_t`  
I2C Slave структура обмена данными
- struct `_i2c_slave_handle`  
I2C Slave-дескриптор
- struct `_i2c_master_dma_handle`  
Контекст данных прерывания I2C-DMA.

## Макросы

- #define `I2C_CFG_MASK` 0x1f
- #define `HAL_I2C_DRIVER_VERSION` (`MAKE_VERSION`(1, 0, 0))  
Версия драйвера.
- #define `I2C_RETRY_TIMES` 0U  
Количество повторов при ожидании флага.
- #define `I2C_MASTER_TRANSMIT_IGNORE_LAST_NACK` 0U  
Игнорировать ли сигнал NACK последнего байта во время передачи Master.
- #define `I2C_STAT_MSTCODE_IDLE` (0U)
- #define `I2C_STAT_MSTCODE_RXREADY` (1U)
- #define `I2C_STAT_MSTCODE_TXREADY` (2U)
- #define `I2C_STAT_MSTCODE_NACKADR` (3U)
- #define `I2C_STAT_MSTCODE_NACKDAT` (4U)
- #define `I2C_STAT_SLVST_ADDR` (0)
- #define `I2C_STAT_SLVST_RX` (1)
- #define `I2C_STAT_SLVST_TX` (2)
- #define `HAL_SPI_DMA_DRIVER_VERSION` (`MAKE_VERSION`(0, 1, 0))  
Версия драйвера
- #define `I2C_MAX_BUFFER_SIZE` (8U)  
Глубина буфера I2C контроллера

## Определения типов

- typedef struct `_i2c_master_handle` `i2c_master_handle_t`  
Тип I2C Master дескриптор
- typedef void(\* `i2c_master_transfer_callback_t`) (I2C\_Type \*base, `i2c_master_handle_t` \*handle, `i2c_status_t` completion\_status, void \*user\_data)  
Указатель на функцию обратного вызова при завершении Master-передачи.
- typedef struct `_i2c_slave_handle` `i2c_slave_handle_t`  
Тип I2C Slave дескриптор
- typedef void(\* `i2c_slave_transfer_callback_t`) (I2C\_Type \*base, volatile `i2c_slave_transfer_t` \*transfer, void \*user\_data)  
Slave тип функции обратного вызова.
- typedef struct `_i2c_master_dma_handle` `i2c_master_dma_handle_t`  
Дескриптор I2C-DMA.
- typedef void(\* `i2c_master_dma_transfer_callback_t`) (I2C\_Type \*base, `i2c_master_dma_handle_t` \*handle, void \*user\_data)  
Функция обратного вызова

## Перечисления

- enum [i2c\\_status\\_t](#)  
Коды возврата функций модуля I2C.
- enum [i2c\\_status\\_flags](#)  
Статусы модуля I2C (read-only регистр IC\_STATUS).
- enum [i2c\\_abort\\_flags](#)  
Причины обрыва передачи (регистр IC\_TX\_ABRT\_SOURCE).
- enum [i2c\\_interrupt](#)  
Флаги прерываний I2C.
- enum [i2c\\_direction\\_t](#)  
Направление передачи.
- enum [i2c\\_addr\\_size\\_t](#)  
Разрядность адреса.
- enum [i2c\\_master\\_transfer\\_states](#)  
Состояния для конечного автомата, используемого при обмене. FSM TODO.
- enum [i2c\\_speed\\_mode\\_t](#)  
Скоростной режим работы модуля в Master-режиме.
- enum [i2c\\_master\\_transfer\\_flags\\_t](#)  
Флаги вариантов передачи.
- enum [i2c\\_slave\\_event\\_transfer\\_t](#)  
События, вызывающие функцию обратного вызова для неблокирующих передач.
- enum [i2c\\_slave\\_fsm\\_t](#)  
I2C Slave состояния конечного автомата

## Функции

- void [I2C\\_MasterTransferCreateHandleDMA](#) (I2C\_Type \*base, [i2c\\_master\\_dma\\_handle\\_t](#) \*handle, [i2c\\_master\\_dma\\_transfer\\_callback\\_t](#) callback, void \*user\_data, [dma\\_handle\\_t](#) \*tx\_dma, [dma\\_handle\\_t](#) \*rx\_dma)  
Функция инициализации дескриптора I2C-DMA.
- [i2c\\_status\\_t](#) [I2C\\_MasterTransferDMA](#) (I2C\_Type \*base, [i2c\\_master\\_dma\\_handle\\_t](#) \*handle, [i2c\\_master\\_transfer\\_t](#) \*xfer)  
Функция, запускающая I2C транзакцию. Данные в буфер/из буфера I2C передаются с помощью I2C.
- void [I2C\\_MasterTransferAbortDMA](#) (I2C\_Type \*base, [i2c\\_master\\_dma\\_handle\\_t](#) \*handle)  
Прекращение передачи I2C.
- static void [I2C\\_DMADescriptorInitTX](#) (I2C\_Type \*base, [dma\\_descriptor\\_t](#) \*desc, uint32\_t count, uint32\_t data\_size, uint8\_t src\_incr, void \*src\_addr)  
Инициализация дескрипторов DMA для многоблочной передачи TX.
- static void [I2C\\_DMADescriptorInitRX](#) (I2C\_Type \*base, [dma\\_descriptor\\_t](#) \*desc, uint32\_t count, uint32\_t data\_size, uint8\_t dst\_incr, void \*dst\_addr)  
Инициализация дескрипторов DMA для многоблочной передачи RX.



Инициализация и деинициализация.

- static bool [I2C\\_IsEnable](#) (I2C\_Type \*base)  
Возврат состояния модуля I2C.
- [i2c\\_status\\_t I2C\\_Enable](#) (I2C\_Type \*base, bool enable)  
Включение и отключение модуля I2C.
- uint32\_t [I2C\\_GetInstance](#) (I2C\_Type \*base)  
Возврат номера экземпляра по базовому адресу.
- [i2c\\_status\\_t I2C\\_Reset](#) (I2C\_Type \*base)  
Сброс модуля I2C.
- void [I2C\\_MasterGetDefaultConfig](#) (i2c\_master\_config\_t \*master\_config)  
Получение конфигурации по умолчанию для Master-режима модуля I2C.
- [i2c\\_status\\_t I2C\\_MasterInit](#) (I2C\_Type \*base, const i2c\_master\_config\_t \*master\_config, uint32\_t src\_clock\_hz)  
Инициализация Master-устройства I2C.
- [i2c\\_status\\_t I2C\\_MasterDeinit](#) (I2C\_Type \*base)  
Деинициализация Master-устройства I2C.

Прерывания.

- static void [I2C\\_EnableInterrupts](#) (I2C\_Type \*base, uint32\_t mask)  
Включение запросов на прерывание.
- static void [I2C\\_DisableInterrupts](#) (I2C\_Type \*base, uint32\_t mask)  
Отключение запросов на прерывание.
- static uint32\_t [I2C\\_GetEnabledInterrupts](#) (I2C\_Type \*base)  
Возврат набора текущих разрешенных запросов на прерывание.
- static void [I2C\\_ClearInterrupt](#) (I2C\_Type \*base, enum i2c\_interrupt irq)  
Очищение флагов прерываний.
- static void [I2C\\_ClearAllInterrupts](#) (I2C\_Type \*base)  
Очищение всех доступных для очистки флагов прерываний.

Операции на шине

- [i2c\\_status\\_t I2C\\_MasterSetBaudRate](#) (I2C\_Type \*base, uint32\_t baudrate\_bps, uint32\_t src↔\_clock\_hz)  
Установка частоты шины для Master-режима модуля I2C.
- [i2c\\_status\\_t I2C\\_MasterAddrSet](#) (I2C\_Type \*base, uint32\_t address, i2c\_addr\_size\_t addr↔\_size)  
Установка адреса Slave-устройства на шине I2C, к которому будет обращение в цикле обмена.
- static bool [I2C\\_MasterGetBusActiveState](#) (I2C\_Type \*base)  
Возврат статуса активность шины.
- [i2c\\_status\\_t I2C\\_MasterWriteBlocking](#) (I2C\_Type \*base, const void \*tx\_buff, size\_t tx\_size, uint32\_t flags)  
Выполнение блокирующей передачи по шине I2C.
- [i2c\\_status\\_t I2C\\_MasterReadBlocking](#) (I2C\_Type \*base, void \*rx\_buff, size\_t rx\_size, uint32↔\_t flags)  
Выполнение блокирующего приема на шине I2C.
- [i2c\\_status\\_t I2C\\_MasterTransferBlocking](#) (I2C\_Type \*base, i2c\_master\_transfer\_t \*xfer)  
Выполнение обмена данными на шине I2C в режиме блокировки.

Неблокирующий обмен (по прерыванию).

- `void I2C_MasterTransferCreateHandle (I2C_Type *base, i2c_master_handle_t *handle, i2c_master_transfer_callback_t callback, void *user_data)`  
Создание нового дескриптора для неблокирующего Master-режима работы.
- `i2c_status_t I2C_MasterTransferNonBlocking (I2C_Type *base, i2c_master_handle_t *handle, i2c_master_transfer_t *xfer)`  
Выполнение неблокирующей транзакции на шине I2C.
- `i2c_status_t I2C_MasterTransferGetCount (I2C_Type *base, i2c_master_handle_t *handle, size_t *count)`  
Получение количества байтов, переданных неблокирующей транзакцией на данный момент.
- `i2c_status_t I2C_MasterTransferAbort (I2C_Type *base, i2c_master_handle_t *handle)`  
Досрочное завершение основной неблокирующей передачи I2C.

Обработчик запросов на прерывание для Master-режима.

- `void I2C_MasterTransferHandleIRQ (I2C_Type *base, i2c_master_handle_t *handle)`  
Обработчик запросов на прерывание для Master-режима.

#### 4.8.1 Подробное описание

Драйвер последовательной асимметричной шины для связи между интегральными схемами внутри приборов.

Драйвер поддерживает обмен по интерфейсу I2C по прерыванию и в режиме опроса.

Используя специальные функции из `hal_i2c_dma.h`, передача данных из буфера / в буфер UART будет происходить с помощью DMA.

#### 4.8.2 Макросы

##### 4.8.2.1 I2C\_CFG\_MASK

```
#define I2C_CFG_MASK 0x1f
```

TODO

##### 4.8.2.2 I2C\_MASTER\_TRANSMIT\_IGNORE\_LAST\_NACK

```
#define I2C_MASTER_TRANSMIT_IGNORE_LAST_NACK 0U
```

Игнорировать ли сигнал NACK последнего байта во время передачи Master.

1 - Master игнорирует NACK последнего байта и считает передачу успешной

##### 4.8.2.3 I2C\_RETRY\_TIMES

```
#define I2C_RETRY_TIMES 0U
```

Количество повторов при ожидании флага.

0 - ожидание до получения значения

#### 4.8.2.4 I2C\_STAT\_MSTCODE\_IDLE

```
#define I2C_STAT_MSTCODE_IDLE (0U)
```

Master код состояния Idle

#### 4.8.2.5 I2C\_STAT\_MSTCODE\_NACKADR

```
#define I2C_STAT_MSTCODE_NACKADR (3U)
```

Master код состояния NACK от Slave при отправке адреса

#### 4.8.2.6 I2C\_STAT\_MSTCODE\_NACKDAT

```
#define I2C_STAT_MSTCODE_NACKDAT (4U)
```

Master код состояния NACK от Slave при отправке данных

#### 4.8.2.7 I2C\_STAT\_MSTCODE\_RXREADY

```
#define I2C_STAT_MSTCODE_RXREADY (1U)
```

Master код состояния Receive Ready

#### 4.8.2.8 I2C\_STAT\_MSTCODE\_TXREADY

```
#define I2C_STAT_MSTCODE_TXREADY (2U)
```

Master код состояния Transmit Ready

#### 4.8.2.9 I2C\_STAT\_SLVST\_ADDR

```
#define I2C_STAT_SLVST_ADDR (0)
```

TODO

#### 4.8.2.10 I2C\_STAT\_SLVST\_RX

```
#define I2C_STAT_SLVST_RX (1)
```

TODO

#### 4.8.2.11 I2C\_STAT\_SLVST\_TX

```
#define I2C_STAT_SLVST_TX (2)
```

TODO

### 4.8.3 Типы

#### 4.8.3.1 i2c\_master\_transfer\_callback\_t

```
typedef void(* i2c_master_transfer_callback_t) (I2C_Type *base, i2c_master_handle_t *handle, i2c_status_t completion_status, void *user_data)
```

Указатель на функцию обратного вызова при завершении Master-передачи.

Используется только для неблокирующей передачи. Для задания функции обратного вызова, вызывающейся для обработки событий, используется [I2C\\_MasterTransferCreateHandle](#).

## Аргументы

base	Базовый адрес модуля I2C
handle	Дескриптор обработчика прерывания
completion_status	Результат завершения операции
user_data	Данные пользователя

## 4.8.3.2 i2c\_slave\_transfer\_callback\_t

```
typedef void(* i2c_slave_transfer_callback_t) (I2C_Type *base, volatile i2c\_slave\_transfer\_t *transfer, void *user_↔ data)
```

Slave тип функции обратного вызова.

Этот тип функции обратного вызова используется только для неблокирующей передачи. Чтобы установить функцию обратного вызова, используйте функцию I2C\_SlaveSetCallback после создания дескриптора.

## Аргументы

base	Базовый адрес экземпляра I2C, на котором произошло событие.
transfer	Дескриптор передачи, содержащий значения, переданные в/из функции обратного вызова.
user_data	Пользовательские данные.

## 4.8.4 Перечисления

## 4.8.4.1 i2c\_abort\_flags

```
enum i2c\_abort\_flags
```

Причины обрыва передачи (регистр IC\_TX\_ABRT\_SOURCE).

## Элементы перечислений

I2C_Abort_7B_Addr_Nack	Master Tx с 7 битной адресацией, NACK от Slave после отправления адреса
I2C_Abort_10B_Addr1_Nack	Master Tx с 10 битной адресацией, NACK от Slave после отправления 1го адреса
I2C_Abort_10B_Addr2_Nack	Master Tx с 10 битной адресацией, NACK от Slave после отправления 1го адреса
I2C_Abort_TxData_Nack	Master Tx не получил ACK от Slave после отправки байта данных
I2C_Abort_GenCall_Nack	Master Tx отправил General Call, и ни один Slave не ответил
I2C_Abort_GenCall_Read	Master Rx попытка чтения с адреса General Call
I2C_Abort_HsCode_Ack	Master в HS режиме получил подтверждение на HS code
I2C_Abort_StartByte_Ack	Master получил подтверждение на [Start] условие

Элементы перечислений

I2C_Abort_HS_RStart_Dis	Master в HS режиме пытается отправить [RStart] условие, но возможность отключена
I2C_Abort_RStart_Dis	Master пытается отправить [RStart] условие, но возможность отключена
I2C_Abort_Read_RStart_Dis	Master пытается осуществить чтение режиме 10 битной адресации, но возможность отключена
I2C_Abort_Master_Dis	Попытка инициализировать Master-обмен при выключенном Master-режиме
I2C_Abort_Arbitr_Lost	Master или Slave (если IC_TX_ABRT_SOURCE[14] == 1) - передатчик проигрывает арбитраж
I2C_Abort_RxFifo_NotEmpty	Slave получил запрос на чтение, но в RxFifo уже есть данные
I2C_Abort_SlaveArbitr_Lost	Slave теряет шину во время передачи данных
I2C_Abort_SlaveDataCmd_Error	Slave есть запрос на передачу данных удаленному Master, но пользователь пытается произвести чтение в режиме мастера (пишет 1 в IC_DATA_CMD.CMD)
I2C_Abort_MasterDetect	Master определил обрыв передачи
I2C_Abort_Tx_FlushCnt	Master/Slave-счетчик оставшихся в TxFifo данны после обрыва передачи

#### 4.8.4.2 i2c\_addr\_size\_t

enum [i2c\\_addr\\_size\\_t](#)

Разрядность адреса.

Элементы перечислений

I2C_Address7Bit	Для обмена используется 7 битная адресация
I2C_Address10Bit	Для обмена используется 10 битная адресация

#### 4.8.4.3 i2c\_direction\_t

enum [i2c\\_direction\\_t](#)

Направление передачи.

Элементы перечислений

I2C_Write	Master передает
I2C_Read	Master принимает

## 4.8.4.4 i2c\_interrupt

enum `i2c_interrupt`

Флаги прерываний I2C.

Заметки

Предназначены для объединения по ИЛИ для формирования битовой маски для регистров:

- `IC_RAW_INTR_STAT`
  - Статус немаскированных прерываний
- `IC_INTR_STAT`
  - Регистр статуса прерываний
- `IC_INTR_MASK`
  - Регистр маскирования прерываний

Прерывания:

- `I2C_IRQ_RdReq`
  - Устанавливается в Slave-режиме при запросе данных удаленным Master. Slave удерживает состояние ожидания ( $SCL = 0$ ), пока прерывание обрабатывается. Процессор должен ответить на это прерывание и начать выдавать данные в `IC_DATA_CMD` регистр. Сброс: чтение `IC_CLR_RD_REQ`
- `I2C_IRQ_TxAbrt`
  - Устанавливается если модуль работает в режиме передатчика и не может произвести передачу. Когда этот бит устанавливается в 1, регистр `IC_TX_ABRT_SOURCE` отображает причину обрыва передачи. Сброс: чтение `IC_CLR_TX_ABRT`
- `I2C_IRQ_Activity`
  - Устанавливается, если модуль проявил какую-либо активность. Сброс:
    - \* Выключение модуля I2C
    - \* Чтение `IC_CLR_ACTIVITY`
    - \* Чтение `IC_CLR_INTR`
    - \* Системный сброс

Элементы перечислений

<code>I2C_IRQ_RxUnder</code>	[0] Чтение из пустого RxFifo. Сброс: чтение <code>IC_CLR_RX_UNDER</code>
<code>I2C_IRQ_RxOver</code>	[1] Переполнение RxFifo. Сброс: чтение <code>IC_CLR_RX_OVER</code>
<code>I2C_IRQ_RxFull</code>	[2] RxFifo заполнен до уровня <code>IC_RX_TL</code> . Сброс: уменьшение уровня RxFifo ниже <code>IC_RX_TL</code>
<code>I2C_IRQ_TxOver</code>	[3] Попытка записать в заполненный TxFifo. Сброс: чтение <code>IC_CLR_TX_OVER</code>
<code>I2C_IRQ_TxEmpty</code>	[4] Опустошение TxFifo ниже уровня <code>IC_TX_TL</code>
<code>I2C_IRQ_RdReq</code>	[5] Устанавливается при запросе данных удаленным Master
<code>I2C_IRQ_TxAbrt</code>	[6] Передача прервана
<code>I2C_IRQ_RxDone</code>	[7] В режиме Slave-передатчика устанавливается в 1, если Master не подтверждает передачу байта

## Элементы перечислений

I2C_IRQ_Activity	[8] Активность на шине I2C
I2C_IRQ_StopDet	[9] В Slave- или Master-режиме устанавливается, если на шине возникает состояние STOP. Сброс: чтение IC_CLR_STOP_DET
I2C_IRQ_StartDet	[10] На шине START или RESTART условия. Сброс: чтение IC_CLR_START_DET
I2C_IRQ_GenCall	[11] Получен адрес General Call и отправлено подтверждение. Сброс: 1) чтение IC_CLR_GEN_CALL 2) выключение модуля

## 4.8.4.5 i2c\_master\_transfer\_flags\_t

```
enum i2c_master_transfer_flags_t
```

Флаги вариантов передачи.

## Заметки

Предназначены для объединения по ИЛИ, чтобы сформировать битовую маску.

## Элементы перечислений

I2C_TransferDataFlag	Передача пакета данных, без установки Start-условия и Stop-условия
I2C_TransferStartFlag	Запуск нового цикла передачи, начинающийся со Start-условия
I2C_TransferStopFlag	Остановка цикла передачи Stop-условием
I2C_TransferReStartFlag	Отправка RStart-условия. Может быть только (!) продолжением передачи. В случае когда предыдущая передача была без I2C_TransferStopFlag и цикл предыдущей передачи не завершился (процесс передачи еще идет)

## 4.8.4.6 i2c\_master\_transfer\_states

```
enum i2c_master_transfer_states
```

Состояния для конечного автомата, используемого при обмене. FSM TODO.

## Элементы перечислений

I2C_MTS_Idle	Состояние: бездействия
I2C_MTS_TransmitSubaddr	Состояние: передача адреса
I2C_MTS_TransmitData	Состояние: передача данных
I2C_MTS_ReceiveDataBegin	Состояние: начало приема данных
I2C_MTS_ReceiveData	Состояние: прием данных
I2C_MTS_ReceiveLastData	Состояние: завершение приема данных
I2C_MTS_Start	Состояние: Start-условие
I2C_MTS_Stop	Состояние: Stop-условие
I2C_MTS_WaitForCompletion	Состояние: Ожидание завершения

## 4.8.4.7 i2c\_slave\_event\_transfer\_t

enum [i2c\\_slave\\_event\\_transfer\\_t](#)

События, вызывающие функцию обратного вызова для неблокирующих передач.

Эти перечисления событий используются для двух взаимосвязанных целей:

- Битовая маска (по ИЛИ) передается в [I2C\\_SlaveTransferNonBlocking](#), чтобы указать, какие события включить.
- При вызове функции обратного вызова, в нее передаются событие через параметр функции.

## Заметки

Эти перечисления предназначены для объединения по ИЛИ, чтобы сформировать битовую маску событий.

## Элементы перечислений

I2C_SlaveEvent_MatchAddrSlave	Получен Slave-адрес после Start или RStart условия
I2C_SlaveEvent_MatchAddrGen	Получен General-адрес
I2C_SlaveEvent_Transmit	Slave-передает: Запрошен вызов функции обратного вызова для предоставления данных на передачу
I2C_SlaveEvent_Receive	Slave-принимает: Запрошен вызов функции обратного вызова для предоставления буфера, в который будут помещены принятые данные
I2C_SlaveEvent_Completion	Текущий обмен был завершен completion_status содержит статус завершения
I2C_SlaveEvents_Ordinary	Битовая маска событий достаточная для большинства задач.
I2C_SlaveEvents_All	Битовая маска всех доступных событий.

## 4.8.4.8 i2c\_slave\_fsm\_t

enum [i2c\\_slave\\_fsm\\_t](#)

I2C Slave состояния конечного автомата

## Элементы перечислений

I2C_SlaveFsm_Idle	Модуль I2C находится в состоянии ожидания, текущего обмена нет
I2C_SlaveFsm_Init	Инициализирован процесс обмена, ожидание событий на шине
I2C_SlaveFsm_Receive	Модуль I2C в фазе получения данных
I2C_SlaveFsm_Transmit	Модуль I2C в фазе передачи данных
I2C_SlaveFsm_Stop	Модуль I2C получил Stop

## 4.8.4.9 i2c\_speed\_mode\_t

enum [i2c\\_speed\\_mode\\_t](#)

Скоростной режим работы модуля в Master-режиме.



Элементы перечислений

I2C_UndefinedSpeedMode	Режим не задан
I2C_StandardSpeedMode	Скорость от 0 до 100 Кб/с
I2C_FastSpeedMode	Скорость до 400 Кб/с
I2C_HighSpeedMode	Скорость меньше либо равна 3.4 Мб/с

#### 4.8.4.10 i2c\_status\_flags

enum [i2c\\_status\\_flags](#)

Статусы модуля I2C (read-only регистр IC\_STATUS).

Элементы перечислений

I2C_Stat_Active	Флаг активности шины
I2C_Stat_TxFifo_NotFull	TxFifo не полон
I2C_Stat_TxFifo_Empty	TxFifo пуст
I2C_Stat_RxFifo_NotEmpty	RxFifo не пуст
I2C_Stat_RxFifo_Full	RxFifo полон
I2C_Stat_Master_Active	Статус активности состояния Master
I2C_Stat_Slave_Active	Статус активности состояния Slave

#### 4.8.4.11 i2c\_status\_t

enum [i2c\\_status\\_t](#)

Коды возврата функций модуля I2C.

Элементы перечислений

I2C_Status_Ok	Успешное завершение
I2C_Status_Busy	Master уже выполняет передачу
I2C_Status_Idle	Slave-драйвер в состоянии ожидания
I2C_Status_Nack	Slave-устройство отправило NACK в ответ на байт
I2C_Status_InvalidParameter	Невозможно продолжить из-за недопустимого параметра
I2C_Status_BreakTransfer	Обрыв обмена
I2C_Status_ArbitrationLost	Потеря арбитража
I2C_Status_NoTransferInProgress	Нет активной передачи
I2C_Status_DmaRequestFail	DMA запрос не выполнен
I2C_Status_Timeout	Тайм-аут при ожидании установки бита статуса в Master/Slave для продолжения передачи
I2C_Status_AddrNack	NACK получен для адреса
I2C_Status_UserError	Некорректная работа с модулем I2C
I2C_Status_SetStartError	Master получил подтверждение на [Start] условие
I2C_Status_HwError	Аппаратная ошибка

## Элементы перечислений

I2C_Status_HsCodeError	Master в HS режиме получил подтверждение на HS code
I2C_Status_UnexpectedState	Неожиданное состояние
I2C_Status_UnexpectedState1	Неожиданное состояние 1
I2C_Status_UnexpectedState2	Неожиданное состояние 2
I2C_Status_UnexpectedState3	Неожиданное состояние 3
I2C_Status_UnexpectedState4	Неожиданное состояние 4
I2C_Status_UnexpectedState5	Неожиданное состояние 5
I2C_Status_UnexpectedState6	Неожиданное состояние 6
I2C_Status_UnexpectedState7	Неожиданное состояние 7
I2C_Status_UnexpectedState8	Неожиданное состояние 8
I2C_Status_UnexpectedState9	Неожиданное состояние 9
I2C_Status_UnDef	Статус не определен

## 4.8.5 Функции

## 4.8.5.1 I2C\_ClearAllInterrupts()

```
static void I2C_ClearAllInterrupts (
    I2C_Type * base)  [inline], [static]
```

Очищение всех доступных для очистки флагов прерываний.

## Заметки

Не все флаги прерываний можно очистить этой функцией. За дополнительной информацией обратитесь к документации на MCU.

## Аргументы

base	Базовый адрес модуля.
------	-----------------------

## 4.8.5.2 I2C\_ClearInterrupt()

```
static void I2C_ClearInterrupt (
    I2C_Type * base,
    enum i2c_interrupt irq)  [inline], [static]
```

Очищение флагов прерываний.

## Заметки

Не все флаги прерываний можно очистить этой функцией. За дополнительной информацией обратитесь к документации на MCU.

## Аргументы

base	Базовый адрес модуля.
irq	Битовая маска прерывания.

## 4.8.5.3 I2C\_DisableInterrupts()

```
static void I2C_DisableInterrupts (
    I2C_Type * base,
    uint32_t mask) [inline], [static]
```

Отключение запросов на прерывание.

## Заметки

Битовая маска прерываний составляется из флагов, [i2c\\_interrupt](#), которые могут быть соединены с использованием оператора побитового ИЛИ.

## Аргументы

base	Базовый адрес модуля.
mask	Битовая маска прерываний.

## 4.8.5.4 I2C\_DMADescriptorInitRX()

```
static void I2C_DMADescriptorInitRX (
    I2C_Type * base,
    dma_descriptor_t * desc,
    uint32_t count,
    uint32_t data_size,
    uint8_t dst_incr,
    void * dst_addr) [inline], [static]
```

Инициализация дескрипторов DMA для многоблочной передачи RX.

## Аргументы

base	Базовый адрес SPI
desc	Указатель на массив дескрипторов
count	Количество DMA дескрипторов
data_size	Общее число передаваемых данных
data_width	Ширина 1ого передаваемого слова
dst_incr	Инкремент адреса Приемника
dst_addr	Адрес Приемника

#### 4.8.5.5 I2C\_DMADescriptorInitTX()

```
static void I2C_DMADescriptorInitTX (
    I2C_Type * base,
    dma_descriptor_t * desc,
    uint32_t count,
    uint32_t data_size,
    uint8_t src_incr,
    void * src_addr)  [inline], [static]
```

Инициализация дескрипторов DMA для многоблочной передачи TX.

Аргументы

base	Базовый адрес SPI
desc	Указатель на массив дескрипторов
count	Количество DMA дескрипторов
data_size	Общее число передаваемых данных
data_width	Ширина 1ого передаваемого слова
src_incr	Инкремент адреса Передатчика
src_addr	Адрес Передатчика

#### 4.8.5.6 I2C\_Enable()

```
i2c_status_t I2C_Enable (
    I2C_Type * base,
    bool enable)
```

Включение и отключение модуля I2C.

Заметки

Общий алгоритм отключения модуля I2C:

1. Определить интервал таймера SYSWAIT, равный 10-кратному периоду высшей скорости I2C. Например: если высшая передача I2C режим 400 кбит/с, то этот интервал SYSWAIT равен 25 мкс.
2. Определить параметр максимального времени ожидания, I2C\_RETRY\_TIMES\_FOR\_DISABLE\_UNITS \* SYSWAIT, при превышении этого значения, сообщать об ошибке.
3. Выполнить блокирующий поток/процесс/функцию, которая предотвращает дальнейшие Master-транзакции I2C.
4. Переменная count инициализируется I2C\_RETRY\_TIMES\_FOR\_DISABLE\_UNITS.
5. Установить бит 0 регистра IC\_ENABLE в 0.
6. Вызвать функцию [I2C\\_IsEnable](#) для проверки текущего состояния модуля. Уменьшить POLL\_COUNT на один. Если POLL\_COUNT == 0, выход с кодом ошибки.
7. Если [I2C\\_IsEnable](#) выдает true, то ожидание времени SYSWAIT и переход к предыдущему шагу. В противном случае, выйти с кодом успеха.

## Аргументы

base	Базовый адрес модуля I2C.
enable	Передайте true, чтобы включить, или false, чтобы отключить указанный модуль I2C.

## Возвращаемые значения

<a href="#">I2C_Status_Ok</a>	
<a href="#">I2C_Status_HwError</a>	

## 4.8.5.7 I2C\_EnableInterrupts()

```
static void I2C_EnableInterrupts (
    I2C_Type * base,
    uint32_t mask) [inline], [static]
```

Включение запросов на прерывание.

## Заметки

Битовая маска прерываний составляется из флагов, [i2c\\_interrupt](#), которые могут быть соединены с использованием оператора побитового ИЛИ.

## Аргументы

base	Базовый адрес модуля.
mask	Битовая маска прерываний.

## 4.8.5.8 I2C\_GetEnabledInterrupts()

```
static uint32_t I2C_GetEnabledInterrupts (
    I2C_Type * base) [inline], [static]
```

Возврат набора текущих разрешенных запросов на прерывание.

## Заметки

Битовая маска прерываний составляется из флагов, [i2c\\_interrupt](#), которые могут быть соединены с использованием оператора побитового ИЛИ.

## Аргументы

base	Базовый адрес модуля.
------	-----------------------

## Возвращает

Битовая маска прерываний.

#### 4.8.5.9 I2C\_GetInstance()

```
uint32_t I2C_GetInstance (  
    I2C_Type * base)
```

Возврат номера экземпляра по базовому адресу.

Если передан недопустимый базовый адрес, то в режиме отладки будет выполнена assert. В режиме релиза будет возвращен номер экземпляра 0.

## Аргументы

base	Базовый адрес модуля I2C.
------	---------------------------

## Возвращает

I2C номер экземпляра, начиная с 0.

## 4.8.5.10 I2C\_IsEnable()

```
static bool I2C_IsEnable (
    I2C_Type * base) [inline], [static]
```

Возврат состояния модуля I2C.

## Аргументы

base	Базовый адрес модуля I2C.
------	---------------------------

## Возвращаемые значения

true	Модуль I2C в состоянии enable.
false	Модуль I2C в состоянии disable.

## 4.8.5.11 I2C\_MasterAddrSet()

```
i2c_status_t I2C_MasterAddrSet (
    I2C_Type * base,
    uint32_t address,
    i2c_addr_size_t addr_size)
```

Установка адреса Slave-устройства на шине I2C, к которому будет обращение в цикле обмена.

Функция устанавливает адрес Slave-устройства и размер адреса. Следующий обмен данными будет происходить с устройством по указанному адресу.

## Аргументы

base	Базовый адрес модуля.
address	Адрес Slave-устройства, если адрес равен 0U to General Call
addr_size	Размер адреса (7- или 10-битный адрес).

## Возвращаемые значения

I2C_Status_Ok	
I2C_Status_Busy	
I2C_Status_HwError	

## 4.8.5.12 I2C\_MasterDeinit()

```
i2c_status_t I2C_MasterDeinit (
    I2C_Type * base)
```

Деинициализация Master-устройства I2C.

## Аргументы

base	Базовый адрес модуля I2C.
------	---------------------------

## Возвращаемые значения

<a href="#">I2C_Status_Ok</a>	
<a href="#">I2C_Status_HwError</a>	

## 4.8.5.13 I2C\_MasterGetBusActiveState()

```
static bool I2C_MasterGetBusActiveState (
    I2C_Type * base)  [inline], [static]
```

Возврат статуса активность шины.

Необходимо, чтобы был включен Master-режим.

## Аргументы

base	Базовый адрес модуля.
------	-----------------------

## Возвращаемые значения

true	Шина занята.
false	Шина свободна.

## 4.8.5.14 I2C\_MasterGetDefaultConfig()

```
void I2C_MasterGetDefaultConfig (
    i2c\_master\_config\_t * master_config)
```

Получение конфигурации по умолчанию для Master-режима модуля I2C.

Эта функция обеспечивает следующую конфигурацию по умолчанию для модуля I2C:

```
master_config->enable_master = true;
master_config->baudrate_bps = 100000U;
```

После вызова этой функции вы можете переопределить любые настройки, перед инициализацией помощью [I2C\\_MasterInit](#).

## Аргументы

master_config	Конфигурация модуля для Master-режима.
---------------	--



## 4.8.5.15 I2C\_MasterInit()

```
i2c_status_t I2C_MasterInit (
    I2C_Type * base,
    const i2c_master_config_t * master_config,
    uint32_t src_clock_hz)
```

Инициализация Master-устройства I2C.

Функция инициализирует Master-устройство I2C, в соответствии с пользовательской конфигурацией. Для инициализации конфигурации значениями по умолчанию используйте функцию [I2C\\_MasterGetDefaultConfig](#).

## Заметки

Частота синхронизации I2C модуля используется для расчета делителей скорости передачи данных и периодов ожидания.

## Аргументы

base	Базовый адрес модуля I2C.
master_config	Конфигурация модуля для Master-режима.
src_clock_hz	Частота синхронизации I2C модуля.

## Возвращаемые значения

<a href="#">I2C_Status_Ok</a>	
<a href="#">I2C_Status_HwError</a>	

## 4.8.5.16 I2C\_MasterReadBlocking()

```
i2c_status_t I2C_MasterReadBlocking (
    I2C_Type * base,
    void * rx_buff,
    size_t rx_size,
    uint32_t flags)
```

Выполнение блокирующего приема на шине I2C.

## Заметки

Флаги управления передачей перечислены в [i2c\\_master\\_transfer\\_flags\\_t](#) и могут быть объединены операцией ИЛИ. Допустимые комбинации:

- I2C\_TransferStartFlag | I2C\_TransferStopFlag
  - Стандартная транзакция. Начинается со Start и заканчивается Stop условием.
- I2C\_TransferStartFlag
  - Начальный пакет транзакции. Начинается со Start и подразумевает продолжение.
- I2C\_TransferDataFlag
  - Пакет передачи данных без Start и Stop условия.

- `I2C_TransferStopFlag`
  - Завершающий пакет транзакции. Не начинается со Start и заканчивается Stop условием.
- `I2C_TransferReStartFlag`
  - Продолжение транзакции с выдачей повторного старта. Подразумевает продолжение. Используется для смены направления обмена.
- `I2C_TransferReStartFlag | I2C_TransferStopFlag`
  - Завершение транзакции с выдачей повторного старта. Не допускает продолжения. Используется для смены направления обмена.

В случае, если Slave не подтверждает последний принятый байт в составной транзакции со сменой направления, статус отказа (`I2C_Status_Nack`) проигнорирован не будет вне зависимости от `I2C_MASTER_TRANSMIT_IGNORE_LAST_NACK`.

#### Аргументы

<code>base</code>	Базовый адрес модуля.
<code>rx_buff</code>	Буфер для хранения принимаемых данных.
<code>rx_size</code>	Размер буфера принимаемых данных в байтах.
<code>flags</code>	Флаги управления передачей для управления специальным поведением.

#### Возвращаемые значения

<code>I2C_Status_Ok</code>	
<code>I2C_Status_Busy</code>	
<code>I2C_Status_ArbitrationLost</code>	
<code>I2C_Status_Timeout</code>	
<code>I2C_Status_UserError</code>	
<code>I2C_Status_AddrNack</code>	
<code>I2C_Status_Nack</code>	
<code>I2C_Status_BreakTransfer</code>	
<code>I2C_Status_UnexpectedState</code>	
<code>I2C_Status_SetStartError</code>	
<code>I2C_Status_HsCodeError</code>	

#### 4.8.5.17 `I2C_MasterSetBaudRate()`

```
i2c_status_t I2C_MasterSetBaudRate (
    I2C_Type * base,
    uint32_t baudrate_bps,
    uint32_t src_clock_hz)
```

Установка частоты шины для Master-режима модуля I2C.

Master-режим модуля I2C автоматически отключается и снова включается при необходимости для настройки скорости передачи данных.

#### Заметки

Не вызывайте эту функцию во время передачи, иначе передача будет прервана.

## Аргументы

base	Базовый адрес модуля.
baudrate_bps	Запрашиваемая частота шины в битах в секунду.
src_clock_hz	Частота синхронизации модуля в Герцах.

## Возвращаемые значения

<a href="#">I2C_Status_Ok</a>	
<a href="#">I2C_Status_HwError</a>	
<a href="#">I2C_Status_InvalidParameter</a>	

## 4.8.5.18 I2C\_MasterTransferAbort()

```
i2c_status_t I2C_MasterTransferAbort (
    I2C_Type * base,
    i2c_master_handle_t * handle)
```

Досрочное завершение основной неблокирующей передачи I2C.

## Заметки

Небезопасно вызывать эту функцию из обработчика IRQ, который имеет более высокий приоритет, чем приоритет IRQ периферийного устройства I2C.

## Аргументы

base	Базовый адрес модуля.
handle	Дескриптор для Master-режима.

## Возвращаемые значения

<a href="#">I2C_Status_Ok</a>	
<a href="#">I2C_Status_Timeout</a>	Тайм-аут во время опроса флагов.

## 4.8.5.19 I2C\_MasterTransferAbortDMA()

```
void I2C_MasterTransferAbortDMA (
    I2C_Type * base,
    i2c_master_dma_handle_t * handle)
```

Прекращение передачи I2C.

## Аргументы

base	Базовый адрес I2C master
handle	Дескриптор SPI-DMA

## 4.8.5.20 I2C\_MasterTransferBlocking()

```
i2c_status_t I2C_MasterTransferBlocking (
    I2C_Type * base,
    i2c_master_transfer_t * xfer)
```

Выполнение обмена данными на шине I2C в режиме блокировки.

## Заметки

Из функции управление не возвращается до тех пор, пока передача не завершится успешно или неуспешно из-за того, что предыдущий обмен еще не завершен или из-за в проигранного арбитража или получения NACK во время передачи адреса или данных.

## Аргументы

base	Базовый адрес модуля.
xfer	Структура передачи.

## Возвращаемые значения

<a href="#">I2C_Status_Ok</a>	
<a href="#">I2C_Status_Busy</a>	
<a href="#">I2C_Status_Nack</a>	
<a href="#">I2C_Status_ArbitrationLost</a>	
<a href="#">I2C_Status_Timeout</a>	
<a href="#">I2C_Status_UserError</a>	
<a href="#">I2C_Status_AddrNack</a>	
<a href="#">I2C_Status_BreakTransfer</a>	
<a href="#">I2C_Status_UnexpectedState</a>	
<a href="#">I2C_Status_SetStartError</a>	
<a href="#">I2C_Status_HsCodeError</a>	

## 4.8.5.21 I2C\_MasterTransferCreateHandle()

```
void I2C_MasterTransferCreateHandle (
    I2C_Type * base,
    i2c_master_handle_t * handle,
    i2c_master_transfer_callback_t callback,
    void * user_data)
```

Создание нового дескриптора для неблокирующего Master-режима работы.

Создание дескриптора предназначено для использования с неблокирующими API. Однажды созданный дескриптор не требуется специально уничтожать отдельной функцией. Если пользователь хочет завершить передачу, должна быть вызвана [I2C\\_MasterTransferAbort](#).

## Аргументы

base	Базовый адрес модуля.
handle	Дескриптор для Master-режима.
callback	Функция обратного вызова.
user_data	Пользовательские данные для функции обратного вызова.

## 4.8.5.22 I2C\_MasterTransferCreateHandleDMA()

```
void I2C_MasterTransferCreateHandleDMA (
    I2C_Type * base,
    i2c_master_dma_handle_t * handle,
    i2c_master_dma_transfer_callback_t callback,
    void * user_data,
    dma_handle_t * tx_dma,
    dma_handle_t * rx_dma)
```

Функция инициализации дескриптора I2C-DMA.

## Аргументы

base	Базовый адрес I2C master
handle	Дескриптор I2C-DMA
callback	Функция обратного вызова
user_data	Пользовательские данные
tx_dma	Дескриптор DMA для передачи
rx_dma	Дескриптор DMA для приема

## 4.8.5.23 I2C\_MasterTransferDMA()

```
i2c_status_t I2C_MasterTransferDMA (
    I2C_Type * base,
    i2c_master_dma_handle_t * handle,
    i2c_master_transfer_t * xfer)
```

Функция, запускающая I2C транзакцию. Данные в буфер/из буфера I2C передаются с помощью I2C.

## Аргументы

base	Базовый адрес I2C master
handle	Дескриптор I2C-DMA
xfer	Структура конфигурации для I2C транзакции

## Возвращаемые значения

#I2C_DMA_Status_Success	
#I2C_DMA_Status_DMA_Busy	

#I2C_DMA_Status_InvalidArgument	
#I2C_DMA_Status_Addr_Nak	
#I2C_DMA_Status_Subaddr_Nak	
#I2C_DMA_Status_Fail	

#### 4.8.5.24 I2C\_MasterTransferGetCount()

```
i2c_status_t I2C_MasterTransferGetCount (
    I2C_Type * base,
    i2c_master_handle_t * handle,
    size_t * count)
```

Получение количества байтов, переданных неблокирующей транзакцией на данный момент.

Аргументы

base	Базовый адрес модуля.
handle	Дескриптор для Master-режима.
count	Количество байтов, переданных на данный момент.

Возвращаемые значения

I2C_Status_Ok	
I2C_Status_Busy	

#### 4.8.5.25 I2C\_MasterTransferHandleIRQ()

```
void I2C_MasterTransferHandleIRQ (
    I2C_Type * base,
    i2c_master_handle_t * handle)
```

Обработчик запросов на прерывание для Master-режима.

Заметки

Эту функцию не нужно вызывать.

Аргументы

base	Базовый адрес модуля.
handle	Дескриптор для Master-режима.

#### 4.8.5.26 I2C\_MasterTransferNonBlocking()

```
i2c_status_t I2C_MasterTransferNonBlocking (
    I2C_Type * base,
    i2c_master_handle_t * handle,
    i2c_master_transfer_t * xfer)
```

Выполнение неблокирующей транзакции на шине I2C.

## Аргументы

base	Базовый адрес модуля.
handle	Дескриптор для Master-режима.
xfer	Структура передачи.

## Возвращаемые значения

I2C_Status_Ok	
I2C_Status_Busy	Либо другой Master в настоящее время использует шину, либо неблокирующая транзакция уже выполняется.

## 4.8.5.27 I2C\_MasterWriteBlocking()

```
i2c_status_t I2C_MasterWriteBlocking (
    I2C_Type * base,
    const void * tx_buff,
    size_t tx_size,
    uint32_t flags)
```

Выполнение блокирующей передачи по шине I2C.

Отправляет до tx\_size байтов на ранее адресованное Slave-устройство. Slave может ответить NACK на любой байт, чтобы досрочно завершить передачу. Если это произойдет, функция возвращает [I2C\\_Status\\_Nack](#).

## Заметки

Флаги управления передачей перечислены в [i2c\\_master\\_transfer\\_flags\\_t](#) и могут быть объединены операцией ИЛИ. Допустимые комбинации:

- I2C\_TransferStartFlag | I2C\_TransferStopFlag
  - Стандартная транзакция. Начинается со Start и заканчивается Stop условием.
- I2C\_TransferStartFlag
  - Начальный пакет транзакции. Начинается со Start и подразумевает продолжение.
- I2C\_TransferDataFlag
  - Пакет передачи данных без Start и Stop условия.
- I2C\_TransferStopFlag
  - Завершающий пакет транзакции. Не начинается со Start и заканчивается Stop условием.
- I2C\_TransferReStartFlag
  - Продолжение транзакции с выдачей повторного старта. Подразумевает продолжение. Используется для смены направления обмена.
- I2C\_TransferReStartFlag | I2C\_TransferStopFlag
  - Завершение транзакции с выдачей повторного старта. Не допускает продолжения. Используется для смены направления обмена.

В случае, если Slave не подтверждает последний принятый байт в составной транзакции со сменой направления, статус отказа ([I2C\\_Status\\_Nack](#)) проигнорирован не будет вне зависимости от [I2C\\_MASTER\\_TRANSMIT\\_IGNORE\\_LAST\\_NACK](#).

## Аргументы

base	Базовый адрес модуля.
tx_buff	Буфер для хранения передаваемых данных.
tx_size	Размер буфера передаваемых данных в байтах.
flags	Флаги управления передачей для управления специальным поведением.

## Возвращаемые значения

<a href="#">I2C_Status_Ok</a>	
<a href="#">I2C_Status_Busy</a>	
<a href="#">I2C_Status_Nack</a>	
<a href="#">I2C_Status_ArbitrationLost</a>	
<a href="#">I2C_Status_Timeout</a>	
<a href="#">I2C_Status_UserError</a>	
<a href="#">I2C_Status_AddrNack</a>	
<a href="#">I2C_Status_BreakTransfer</a>	
<a href="#">I2C_Status_UnexpectedState</a>	
<a href="#">I2C_Status_SetStartError</a>	
<a href="#">I2C_Status_HsCodeError</a>	

## 4.8.5.28 I2C\_Reset()

```
i2c_status_t I2C_Reset (
    I2C_Type * base)
```

Сброс модуля I2C.

Завершение всех операций, очищение TxFifo и RxFifo.

## Аргументы

base	Базовый адрес модуля I2C.
------	---------------------------

## Возвращаемые значения

<a href="#">I2C_Status_Ok</a>	
<a href="#">I2C_Status_HwError</a>	

## 4.9 Драйвер модуля I2S

Драйвер последовательной шины для соединения цифровых аудиоустройств



## Файлы

- файл [hal\\_i2s.h](#)  
Интерфейс драйвера модуля I2S.

## Структуры данных

- struct [\\_i2s\\_config](#)  
Структура конфигурации контроллера I2S.
- struct [\\_i2s\\_transfer](#)  
Структура для передачи данных I2S в неблокирующем режиме (по прерыванию)
- struct [\\_i2s\\_handle](#)  
Структура обработчика драйвера I2S.

## Макросы

- `#define HAL_I2S_DRIVER_VERSION (MAKE_VERSION(0, 1, 0))`  
Версия драйвера I2S.

## Определения типов

- typedef enum [\\_i2s\\_status](#) i2s\_status\_t  
Коды возврата функций драйвера I2S.
- typedef enum [\\_i2s\\_flag](#) i2s\_flag\_t  
Флаги прерываний I2S.
- typedef enum [\\_i2s\\_sclk\\_per\\_sample](#) i2s\_sclk\_per\_sample\_t  
Перечисление возможных значений числа синхроимпульсов sclk на левый и правый поток
- typedef enum [\\_i2s\\_sclk\\_gating](#) i2s\_sclk\_gating\_t  
Перечисление возможных значений обрезания числа синхроимпульсов sclk.
- typedef enum [\\_i2s\\_resolution](#) i2s\_resolution\_t  
Разрядность данных I2S.
- typedef enum [\\_i2s\\_interrupt\\_level](#) i2s\_interrupt\_level\_t  
Уровни срабатывания прерывания по опустошению очереди I2S.
- typedef struct [\\_i2s\\_config](#) i2s\_config\_t  
Структура конфигурации контроллера I2S.
- typedef struct [\\_i2s\\_transfer](#) i2s\_transfer\_t  
Структура для передачи данных I2S в неблокирующем режиме (по прерыванию)
- typedef struct [\\_i2s\\_handle](#) i2s\_handle\_t  
Сокращённое название типа для структуры обработчика драйвера I2S.
- typedef void(\* [i2s\\_transfer\\_callback\\_t](#)) (I2S\_Type \*base, [i2s\\_handle\\_t](#) \*handle, [i2s\\_flag\\_t](#) interrupt\_flag, void \*user\_data)  
Функция обратного вызова I2S.

## Перечисления

- enum [\\_i2s\\_status](#)  
Коды возврата функций драйвера I2S.
- enum [\\_i2s\\_flag](#)  
Флаги прерываний I2S.
- enum [\\_i2s\\_sclk\\_per\\_sample](#)  
Перечисление возможных значений числа синхроимпульсов sclk на левый и правый поток
- enum [\\_i2s\\_sclk\\_gating](#)  
Перечисление возможных значений обрезания числа синхроимпульсов sclk.
- enum [\\_i2s\\_resolution](#)  
Разрядность данных I2S.
- enum [\\_i2s\\_interrupt\\_level](#)  
Уровни срабатывания прерывания по опустошению очереди I2S.

## Инициализация и деинициализация

- [i2s\\_status\\_t](#) [I2S\\_Init](#) (I2S\_Type \*base, const [i2s\\_config\\_t](#) \*config, uint32\_t source\_clock\_hz)  
Инициализация драйвера I2S.
- void [I2S\\_Deinit](#) (I2S\_Type \*base)  
Деинициализация драйвера I2S.
- void [I2S\\_GetDefaultConfig](#) ([i2s\\_config\\_t](#) \*config)  
Получение параметров драйвера I2S по умолчанию

## Управление прерываниями

- static void [I2S\\_EnableInterrupt](#) (I2S\_Type \*base, [i2s\\_flag\\_t](#) idx)  
Разрешение прерывания
- static void [I2S\\_EnableInterruptMask](#) (I2S\_Type \*base, uint32\_t mask)  
Разрешение прерываний по маске
- static bool [I2S\\_IsInterruptEnabled](#) (I2S\_Type \*base, [i2s\\_flag\\_t](#) idx)  
Запрос - разрешено ли прерывание I2S.
- static uint32\_t [I2S\\_GetEnabledInterruptMask](#) (I2S\_Type \*base)  
Запрос маски разрешенных прерываний I2S.
- static void [I2S\\_DisableInterrupt](#) (I2S\_Type \*base, [i2s\\_flag\\_t](#) idx)  
Запрет прерывания
- static void [I2S\\_DisableInterruptMask](#) (I2S\_Type \*base, uint32\_t mask)  
Запрет прерываний по маске
- static bool [I2S\\_GetInterruptStatus](#) (I2S\_Type \*base, [i2s\\_flag\\_t](#) idx)  
Получение состояния флага прерывания
- static uint32\_t [I2S\\_GetInterruptStatusMask](#) (I2S\_Type \*base)  
Получение маски активных прерываний
- static void [I2S\\_ClearDataOverrunFlag](#) (I2S\_Type \*base)  
Сброс флага переполнения очереди выдачи

### Прямое управление выдачей

- void `I2S_WriteLeftFifo` (`I2S_Type *base`, `uint32_t sample`)  
Запись сэмпла в левый канал I2S.
- void `I2S_WriteRightFifo` (`I2S_Type *base`, `uint32_t sample`)  
Запись сэмпла в правый канал I2S.
- static void `I2S_AbortTx` (`I2S_Type *base`)  
Отмена выдачи по I2S.
- static void `I2S_EnableTx` (`I2S_Type *base`)  
Включение передатчика I2S.
- static void `I2S_DisableTx` (`I2S_Type *base`)  
Выключение передатчика I2S.

### Неблокирующая выдача по I2S

- `i2s_status_t` `I2S_TransferCreateHandle` (`I2S_Type *base`, `i2s_handle_t *handle`, `i2s_transfer_callback_t callback`, `void *user_data`)  
Инициализация обработчика событий I2S.
- `i2s_status_t` `I2S_TransferNonBlocking` (`I2S_Type *base`, `i2s_handle_t *handle`, `i2s_transfer_t *xfer`)  
Неблокирующая выдача через высокоприоритетный буфер
- void `I2S_TransferAbort` (`I2S_Type *base`, `i2s_handle_t *handle`)  
Отмена выдачи из высокоприоритетного буфера
- void `I2S_TransferHandleIRQ` (`I2S_Type *base`, `i2s_handle_t *handle`)  
Установка обработчика на прерывания от I2S, не связанных с приемом/выдачей

## 4.9.1 Подробное описание

Драйвер последовательной шины для соединения цифровых аудиоустройств

Драйвер содержит функции управления контроллером I2S микросхемы ELIOT1. Контроллер интерфейса собран со следующими параметрами:

- `COMP_PARAM1` = 0x024C003A
- `COMP_PARAM2` = 0x00000489

## 4.9.2 Типы

### 4.9.2.1 `i2s_sclk_gating_t`

```
typedef enum _i2s_sclk_gating i2s_sclk_gating_t
```

Перечисление возможных значений обрезания числа синхроимпульсов `sclk`.

Если разрешение канала I2S меньше, чем размер слова, то часть импульсов `sclk` может быть обрезана на выходе с помощью установки значения из этого перечисления в параметре драйвер `i2s_config_t::sclk_gating`.

### 4.9.2.2 `i2s_transfer_callback_t`

```
typedef void(* i2s_transfer_callback_t) (I2S_Type *base, i2s_handle_t *handle, i2s_flag_t interrupt_flag, void *user_data)
```

Функция обратного вызова I2S.

## Аргументы

base	Базовый адрес контроллера
handle	Указатель на обработчик
interrupt_flag	Причина вызова
user_data	Данные пользователя

## 4.9.3 Перечисления

## 4.9.3.1 \_i2s\_flag

enum [\\_i2s\\_flag](#)

Флаги прерываний I2S.

Элементы перечислений

I2S_FlagTxFifoEmpty	Опустошение очереди выдачи
I2S_FlagTxFifoOverrun	Переполнение очереди выдачи

## 4.9.3.2 \_i2s\_interrupt\_level

enum [\\_i2s\\_interrupt\\_level](#)

Уровни срабатывания прерывания по опустошению очереди I2S.

Элементы перечислений

I2S_InterruptTriggerLevel_1	Прерывание при 1-м оставшемся слове в очереди
I2S_InterruptTriggerLevel_2	Прерывание при 2-х оставшихся словах в очереди
I2S_InterruptTriggerLevel_3	Прерывание при 3-х оставшихся словах в очереди
I2S_InterruptTriggerLevel_4	Прерывание при 4-х оставшихся словах в очереди
I2S_InterruptTriggerLevel_5	Прерывание при 5-и оставшихся словах в очереди
I2S_InterruptTriggerLevel_6	Прерывание при 6-и оставшихся словах в очереди
I2S_InterruptTriggerLevel_7	Прерывание при 7-и оставшихся словах в очереди
I2S_InterruptTriggerLevel_8	Прерывание при 8-и оставшихся словах в очереди
I2S_InterruptTriggerLevel_9	Прерывание при 9-и оставшихся словах в очереди
I2S_InterruptTriggerLevel_10	Прерывание при 10-и оставшихся словах в очереди
I2S_InterruptTriggerLevel_11	Прерывание при 11-и оставшихся словах в очереди
I2S_InterruptTriggerLevel_12	Прерывание при 12-и оставшихся словах в очереди
I2S_InterruptTriggerLevel_13	Прерывание при 13-и оставшихся словах в очереди
I2S_InterruptTriggerLevel_14	Прерывание при 14-и оставшихся словах в очереди
I2S_InterruptTriggerLevel_15	Прерывание при 15-и оставшихся словах в очереди
I2S_InterruptTriggerLevel_16	Прерывание при 16-и оставшихся словах в очереди

## 4.9.3.3 \_i2s\_resolution

enum [\\_i2s\\_resolution](#)

Разрядность данных I2S.

Элементы перечислений

I2S_ResolutionDefault	По умолчанию (32 бита)
I2S_Resolution_12	12 бит
I2S_Resolution_16	16 бит
I2S_Resolution_20	20 бит
I2S_Resolution_24	24 бита
I2S_Resolution_32	32 бита

#### 4.9.3.4 \_i2s\_sclk\_gating

enum [\\_i2s\\_sclk\\_gating](#)

Перечисление возможных значений обрезания числа синхроимпульсов sclk.

Если разрешение канала I2S меньше, чем размер слова, то часть импульсов sclk может быть обрезана на выходе с помощью установки значения из этого перечисления в параметре драйвер [i2s\\_config\\_t::sclk\\_gating](#).

Элементы перечислений

I2S_NoSclkGating	Нет обрезания sclk
I2S_SclkGatingCycles_12	Обрезание после 12 импульсов sclk
I2S_SclkGatingCycles_16	Обрезание после 16 импульсов sclk
I2S_SclkGatingCycles_20	Обрезание после 20 импульсов sclk
I2S_SclkGatingCycles_24	Обрезание после 24 импульсов sclk

#### 4.9.3.5 \_i2s\_sclk\_per\_sample

enum [\\_i2s\\_sclk\\_per\\_sample](#)

Перечисление возможных значений числа синхроимпульсов sclk на левый и правый поток

Элементы перечислений

I2S_SclkCycles_16	16 синхроимпульсов на значение (левое/правое)
I2S_SclkCycles_24	24 синхроимпульсов на значение (левое/правое)
I2S_SclkCycles_32	32 синхроимпульсов на значение (левое/правое)

#### 4.9.3.6 \_i2s\_status

enum [\\_i2s\\_status](#)

Коды возврата функций драйвера I2S.

Элементы перечислений

I2S_Status_Ok	Успешно
I2S_Status_Fail	Провал
I2S_Status_InvalidArgument	Неверный аргумент
I2S_Status_TxBusy	Передачик занят
I2S_Status_UnsupportedBitRate	Комбинация параметров задана так, что невозможно установить правильную битовую частоту

#### 4.9.4 Функции

##### 4.9.4.1 I2S\_AbortTx()

```
static void I2S_AbortTx (  
    I2S_Type * base)  [inline], [static]
```

Отмена выдачи по I2S.

Аргументы

base	Базовый адрес контроллера
------	---------------------------

##### 4.9.4.2 I2S\_ClearDataOverrunFlag()

```
static void I2S_ClearDataOverrunFlag (  
    I2S_Type * base)  [inline], [static]
```

Сброс флага переполнения очереди выдачи

Аргументы

base	Базовый адрес контроллера
------	---------------------------

##### 4.9.4.3 I2S\_Deinit()

```
void I2S_Deinit (  
    I2S_Type * base)
```

Деинициализация драйвера I2S.

Функция деинициализирует контроллер I2S.

Аргументы

base	Базовый адрес контроллера
------	---------------------------

##### 4.9.4.4 I2S\_DisableInterrupt()

```
static void I2S_DisableInterrupt (  
    I2S_Type * base,  
    i2s_flag_t idx)  [inline], [static]
```

Запрет прерывания

## Аргументы

base	Базовый адрес контроллера
idx	Номер флага состояния/прерывания

## 4.9.4.5 I2S\_DisableInterruptMask()

```
static void I2S_DisableInterruptMask (
    I2S_Type * base,
    uint32_t mask) [inline], [static]
```

Запрет прерываний по маске

## Аргументы

base	Базовый адрес контроллера
mask	Маска флагов прерываний

## 4.9.4.6 I2S\_DisableTx()

```
static void I2S_DisableTx (
    I2S_Type * base) [inline], [static]
```

Выключение передатчика I2S.

## Аргументы

base	I2S Базовый адрес контроллера
------	-------------------------------

## 4.9.4.7 I2S\_EnableInterrupt()

```
static void I2S_EnableInterrupt (
    I2S_Type * base,
    i2s_flag_t idx) [inline], [static]
```

Разрешение прерывания

## Аргументы

base	Базовый адрес контроллера
idx	Номер флага состояния/прерывания

## 4.9.4.8 I2S\_EnableInterruptMask()

```
static void I2S_EnableInterruptMask (
    I2S_Type * base,
    uint32_t mask) [inline], [static]
```

Разрешение прерываний по маске

## Аргументы

base	Базовый адрес контроллера
mask	Маска флагов прерываний

## 4.9.4.9 I2S\_EnableTx()

```
static void I2S_EnableTx (  
    I2S_Type * base)  [inline], [static]
```

Включение передатчика I2S.

## Аргументы

base	I2S Базовый адрес контроллера
------	-------------------------------

## 4.9.4.10 I2S\_GetDefaultConfig()

```
void I2S_GetDefaultConfig (  
    i2s\_config\_t * config)
```

Получение параметров драйвера I2S по умолчанию

## Аргументы

config	Структура с параметрами конфигурации
--------	--------------------------------------

## 4.9.4.11 I2S\_GetEnabledInterruptMask()

```
static uint32_t I2S_GetEnabledInterruptMask (  
    I2S_Type * base)  [inline], [static]
```

Запрос маски разрешенных прерываний I2S.

Запрос маски разрешенных прерываний I2S, единицы в соответствующих разрядах соответствуют включенным прерываниям.

## Аргументы

base	Базовый адрес I2S
------	-------------------

## Возвращает

Маска разрешенных прерываний

## 4.9.4.12 I2S\_GetInterruptStatus()

```
static bool I2S_GetInterruptStatus (  
    I2S_Type * base,  
    i2s\_flag\_t idx)  [inline], [static]
```

Получение состояния флага прерывания



## Аргументы

base	Базовый адрес контроллера
idx	Номер флага состояния/прерывания

## Возвращаемые значения

true	Флаг прерывания установлен
false	Флаг прерывания сброшен

## 4.9.4.13 I2S\_GetInterruptStatusMask()

```
static uint32_t I2S_GetInterruptStatusMask (
    I2S_Type * base) [inline], [static]
```

Получение маски активных прерываний

## Аргументы

base	Базовый адрес контроллера
------	---------------------------

## Возвращает

Маска активных прерываний

## 4.9.4.14 I2S\_Init()

```
i2s_status_t I2S_Init (
    I2S_Type * base,
    const i2s_config_t * config,
    uint32_t source_clock_hz)
```

Инициализация драйвера I2S.

Функция инициализирует модуль I2S в соответствии с заданными пользовательскими параметрами.

## Аргументы

base	Базовый адрес контроллера
config	Структура с параметрами конфигурации
source_clock_hz	Опорная частота в Гц

## Возвращаемые значения

I2S_Status_Ok	
I2S_Status_UnsupportedBitRate	

## 4.9.4.15 I2S\_IsInterruptEnabled()

```
static bool I2S_IsInterruptEnabled (
    I2S_Type * base,
    i2s_flag_t idx) [inline], [static]
```

Запрос - разрешено ли прерывание I2S.

## Аргументы

base	Базовый адрес I2S
idx	Номер флага состояния/прерывания

## Возвращаемые значения

true	Прерывание разрешено
false	Прерывание запрещено

## 4.9.4.16 I2S\_TransferAbort()

```
void I2S_TransferAbort (
    I2S_Type * base,
    i2s_handle_t * handle)
```

Отмена выдачи из высокоприоритетного буфера

## Аргументы

base	Базовый адрес контроллера
handle	Обработчик

## 4.9.4.17 I2S\_TransferCreateHandle()

```
i2s_status_t I2S_TransferCreateHandle (
    I2S_Type * base,
    i2s_handle_t * handle,
    i2s_transfer_callback_t callback,
    void * user_data)
```

Инициализация обработчика событий I2S.

## Аргументы

base	Базовый адрес контроллера
handle	Обработчик
callback	Функция обратного вызова
user_data	Данные пользователя

## Возвращаемые значения

I2S_Status_Ok	
I2S_Status_Fail	

## 4.9.4.18 I2S\_TransferHandleIRQ()

```
void I2S_TransferHandleIRQ (
    I2S_Type * base,
    i2s_handle_t * handle)
```

Установка обработчика на прерывания от I2S, не связанных с приемом/выдачей

## Аргументы

base	Базовый адрес контроллера
handle	Обработчик

## 4.9.4.19 I2S\_TransferNonBlocking()

```
i2s_status_t I2S_TransferNonBlocking (
    I2S_Type * base,
    i2s_handle_t * handle,
    i2s_transfer_t * xfer)
```

Неблокирующая выдача через высокоприоритетный буфер

Функция выдает кадр I2S по прерыванию. Это неблокирующая функция, она возвращает управление сразу. По выдаче кадра в шину вызывает функция обратного вызова.

## Аргументы

base	Базовый адрес контроллера
handle	Обработчик
xfer	Структура для передачи кадра в неблокирующем режиме

## Возвращаемые значения

I2S_Status_Ok	
I2S_Status_Fail	
I2S_Status_TxBusy	

## 4.9.4.20 I2S\_WriteLeftFifo()

```
void I2S_WriteLeftFifo (
    I2S_Type * base,
    uint32_t sample)
```

Запись сэмпла в левый канал I2S.

## Аргументы

base	Базовый адрес контроллера
sample	Значение сэмпла

## 4.9.4.21 I2S\_WriteRightFifo()

```
void I2S_WriteRightFifo (
    I2S_Type * base,
    uint32_t sample)
```

Запись сэмпла в правый канал I2S.

## Аргументы

base	Базовый адрес контроллера
sample	Значение сэмпла

## 4.10 Драйвер менеджера прерываний IO устройств и DMA каналов

Менеджер прерываний IO устройств и каналов DMA.

## Файлы

- файл [hal\\_ioim.h](#)  
Интерфейс менеджера прерываний IO устройств

## Макросы

- `#define IOIM_NA_IRQ_NUM (-16)`

## Перечисления

- enum [ioim\\_status\\_t](#)  
Возвращаемые статусы IOIM.

## Функции

- `int32_t IOIM_GetIRQNumber (void *base)`  
Получение номера прерывания в системе
- `ioim_status_t IOIM_SetIRQHandler (void *base, void *handler, void *handle)`  
Установка обработчика прерывания для устройства IO.
- `ioim_status_t IOIM_ClearIRQHandler (void *base)`  
Сброс обработчика прерывания для устройства IO.
- `ioim_status_t IOIM_SetIRQHandler_DMA (void *base, uint32_t channel, void *handler, void *handle)`  
Установка обработчика прерывания для DMA.
- `ioim_status_t IOIM_ClearIRQHandler_DMA (void *base, uint32_t channel)`  
Сброс обработчика прерывания для DMA.

### 4.10.1 Подробное описание

Менеджер прерываний IO устройств и каналов DMA.

Менеджер прерываний выполняет регистрацию векторов прерываний устройств IO (UART, I2C, I2S, SPI) и каналов DMA. При срабатывании прерывания вызывает обработчик из драйвера устройства с передачей указателей на базовый адрес и контекст. Обработчик драйвера должен перед этим быть зарегистрирован соответствующей функцией.

## 4.10.2 Макросы

### 4.10.2.1 IOIM\_NA\_IRQ\_NUM

```
#define IOIM_NA_IRQ_NUM (-16)
```

Несуществующий номер прерывания

## 4.10.3 Перечисления

### 4.10.3.1 ioim\_status\_t

```
enum ioim_status_t
```

Возвращаемые статусы IOIM.

Элементы перечислений

IOIM_Status_Ok	Ошибок нет
IOIM_Status_UnknownBase	Неизвестный базовый адрес устройства
IOIM_Status_NullHandler	Адрес обработчика прерывания равен 0

## 4.10.4 Функции

### 4.10.4.1 IOIM\_ClearIRQHandler()

```
ioim_status_t IOIM_ClearIRQHandler (
    void * base)
```

Сброс обработчика прерывания для устройства IO.

Обработчик прерывания устройства удаляется из таблицы, отключается вектор прерывания для данного устройства.

Аргументы

base	Базовый адрес устройства
------	--------------------------

Возвращаемые значения

IOIM_Status_Ok	
IOIM_Status_UnknownBase	

### 4.10.4.2 IOIM\_ClearIRQHandler\_DMA()

```
ioim_status_t IOIM_ClearIRQHandler_DMA (
    void * base,
    uint32_t channel)
```

Сброс обработчика прерывания для DMA.

Обработчик прерывания DMA удаляется из таблицы, отключается вектор прерывания для данного DMA.

## Аргументы

base	Базовый адрес DMA
channel	Номер канала DMA

## Возвращаемые значения

<a href="#">IOIM_Status_Ok</a>	
<a href="#">IOIM_Status_UnknownBase</a>	

## 4.10.4.3 IOIM\_GetIRQNumber()

```
int32_t IOIM_GetIRQNumber (
    void * base)
```

Получение номера прерывания в системе

## Аргументы

base	Базовый адрес устройства
------	--------------------------

## Возвращает

Номер прерывания устройства

## 4.10.4.4 IOIM\_SetIRQHandler()

```
ioim_status_t IOIM_SetIRQHandler (
    void * base,
    void * handler,
    void * handle)
```

Установка обработчика прерывания для устройства IO.

Функция вносит в свою таблицу прерываний обработчик handler и включает вектор прерывания в системе. При срабатывании прерывания в обработчик будут переданы аргументы base и handle.

## Аргументы

base	Базовый адрес устройства
handler	Указатель на функцию обработчик прерывания
handle	Контекст драйвера устройства

## Возвращаемые значения

<a href="#">IOIM_Status_Ok</a>	
<a href="#">IOIM_Status_UnknownBase</a>	
<a href="#">IOIM_Status_NullHandler</a>	

## 4.10.4.5 IOIM\_SetIRQHandler\_DMA()

```
ioim_status_t IOIM_SetIRQHandler_DMA (
    void * base,
    uint32_t channel,
    void * handler,
    void * handle)
```

Установка обработчика прерывания для DMA.

Аргументы

base	Базовый адрес DMA
channel	Номер канала DMA
handler	Указатель на обработчик прерывания
handle	Контекст драйвера DMA

Возвращаемые значения

IOIM_Status_Ok	
IOIM_Status_UnknownBase	
IOIM_Status_NullHandler	

## 4.11 Драйвер модуля JTM

Драйвер модуля встроенных датчиков температуры и напряжения

Структуры данных

- struct `jtm_config_t`  
Структура конфигурации JTM.
- struct `_jtm_handle`  
Структура обработчика событий JTM.

Макросы

- `#define HAL_JTM_DRIVER_VERSION (MAKE_VERSION(0, 1, 0))`  
Версия драйвера CAN.

Определения типов

- `typedef struct _jtm_handle jtm_handle_t`  
Декларация типа дескриптора драйвера JTM.
- `typedef void(* jtm_callback_t) (jtm_handle_t *handle, jtm_parameter_t parameter, int32_t value, void *user_data)`  
Функция обратного вызова JTM.

## Перечисления

- enum [jtm\\_parameter\\_t](#)  
Перечень параметров, значения которых можно прочитать с помощью драйвера JTM.
- enum [jtm\\_status\\_t](#)  
Коды возврата функций драйвера JTM.

## Функции

- void [JTM\\_Init](#) (JTM\_Type \*base, [jtm\\_config\\_t](#) \*config)  
Инициализация драйвера JTM.
- [jtm\\_status\\_t JTM\\_GetParameterValue](#) (JTM\_Type \*base, [jtm\\_parameter\\_t](#) parameter, int32\_t \*value)  
Блокирующее чтение параметра JTM.
- [jtm\\_status\\_t JTM\\_CreateHandle](#) (JTM\_Type \*base, [jtm\\_handle\\_t](#) \*handle, [jtm\\_callback\\_t](#) callback, void \*user\_data)  
Инициализация обработчика событий JTM.
- [jtm\\_status\\_t JTM\\_GetParameterValueNonBlocking](#) (JTM\_Type \*base, [jtm\\_handle\\_t](#) \*handle, [jtm\\_parameter\\_t](#) parameter)  
Неблокирующее чтение параметра JTM.

### 4.11.1 Подробное описание

Драйвер модуля встроенных датчиков температуры и напряжения

Драйвер содержит функции для измерения температуры кристалла и напряжения в контрольных точках с помощью АЦП и интегрированного датчика температуры

### 4.11.2 Типы

#### 4.11.2.1 jtm\_callback\_t

```
typedef void(* jtm_callback_t) (jtm\_handle\_t *handle, jtm\_parameter\_t parameter, int32_t value, void *user_data)
```

Функция обратного вызова JTM.

#### Аргументы

handle	Дескриптор драйвера JTM
parameter	Параметр, прочитанный с помощью драйвера JTM
value	Значение параметра. Температура выдается в тысячных долях градуса Цельсия, напряжение - в мВ.
user_data	Указатель на произвольные пользовательские данные

### 4.11.3 Перечисления

#### 4.11.3.1 jtm\_parameter\_t

```
enum jtm\_parameter\_t
```

Перечень параметров, значения которых можно прочитать с помощью драйвера JTM.



Элементы перечислений

JTM_Temperature	Температура
JTM_Vcasn	Напряжение Vcasn
JTM_Vcore	Напряжение питания ядра с вывода VDDC

#### 4.11.3.2 jtm\_status\_t

enum [jtm\\_status\\_t](#)

Коды возврата функций драйвера JTM.

Элементы перечислений

JTM_Status_Ok	Успешно
JTM_Status_Fail	Провал
JTM_Status_BadParameter	Неправильный параметр
JTM_Status_Busy	Контроллер занят (идет преобразование)

### 4.11.4 Функции

#### 4.11.4.1 JTM\_CreateHandle()

```
jtm\_status\_t JTM_CreateHandle (
    JTM_Type * base,
    jtm\_handle\_t * handle,
    jtm\_callback\_t callback,
    void * user_data)
```

Инициализация обработчика событий JTM.

Аргументы

base	Базовый адрес контроллера
handle	Обработчик
callback	Функция обратного вызова
user_data	Аргумент функции обратного вызова

Возвращаемые значения

<a href="#">JTM_Status_Ok</a>	
<a href="#">JTM_Status_Fail</a>	

## 4.11.4.2 JTM\_GetParameterValue()

```
jtm_status_t JTM_GetParameterValue (
    JTM_Type * base,
    jtm_parameter_t parameter,
    int32_t * value)
```

Блокирующее чтение параметра JTM.

Функция возвращает значение указанного параметра JTM, прочитанного с помощью встроенного АЦП, через указатель в параметре value. Температура выдается в тысячных долях градуса Цельсия, напряжение - в мВ.

Аргументы

base	Базовый адрес контроллера
parameter	Тип параметра для чтения
value	Значение параметра

Возвращаемые значения

JTM_Status_Ok	
JTM_Status_Busy	
JTM_Status_BadParameter	

## 4.11.4.3 JTM\_GetParameterValueNonBlocking()

```
jtm_status_t JTM_GetParameterValueNonBlocking (
    JTM_Type * base,
    jtm_handle_t * handle,
    jtm_parameter_t parameter)
```

Неблокирующее чтение параметра JTM.

Функция инициирует чтение указанного параметра JTM с помощью встроенного АЦП. После завершения процедуры чтения вызывается функция обратного вызова, указанная при создании дескриптора JTM.

Аргументы

base	Базовый адрес контроллера
handle	Функция обратного вызова
parameter	Тип параметра для чтения

Возвращаемые значения

JTM_Status_Ok	
JTM_Status_Busy	
JTM_Status_BadParameter	

## 4.11.4.4 JTM\_Init()

```
void JTM_Init (
    JTM_Type * base,
    jtm_config_t * config)
```

Инициализация драйвера JTM.

Аргументы

base	Базовый адрес контроллера
config	Структура с параметрами конфигурации

## 4.12 Драйвер модуля программных прерываний MHU

Модуль программных прерываний MHU.

Файлы

- файл [hal\\_mhu.h](#)  
Интерфейс модуля программных прерываний MHU.

Функции

- static void [MHU\\_CPU0\\_SetInt](#) (MHU\_Type \*base, uint32\_t mask)  
Установка прерывания CPU0 по маске
- static void [MHU\\_CPU0\\_ClearInt](#) (MHU\_Type \*base, uint32\_t mask)  
Сброс прерывания CPU0 по маске
- static uint32\_t [MHU\\_CPU0\\_StatInt](#) (MHU\_Type \*base)  
Чтение статуса прерываний CPU0.
- static void [MHU\\_CPU1\\_SetInt](#) (MHU\_Type \*base, uint32\_t mask)  
Установка прерывания CPU1 по маске
- static void [MHU\\_CPU1\\_ClearInt](#) (MHU\_Type \*base, uint32\_t mask)  
Сброс прерывания CPU1 по маске
- static uint32\_t [MHU\\_CPU1\\_StatInt](#) (MHU\_Type \*base)  
Чтение статуса прерываний CPU1.

## 4.12.1 Подробное описание

Модуль программных прерываний MHU.

Модуль программных прерываний MHU (Message Handling Units) реализует механизм обмена прерываниями между ядрами CPU0 и CPU1.

## 4.12.2 Функции

## 4.12.2.1 MHU\_CPU0\_ClearInt()

```
static void MHU_CPU0_ClearInt (
    MHU_Type * base,
    uint32_t mask) [inline], [static]
```

Сброс прерывания CPU0 по маске

## Аргументы

base	Базовый адрес MHU
mask	Маска прерываний

## 4.12.2.2 MHU\_CPU0\_SetInt()

```
static void MHU_CPU0_SetInt (  
    MHU_Type * base,  
    uint32_t mask) [inline], [static]
```

Установка прерывания CPU0 по маске

## Аргументы

base	Базовый адрес MHU
mask	Маска прерываний

## 4.12.2.3 MHU\_CPU0\_StatInt()

```
static uint32_t MHU_CPU0_StatInt (  
    MHU_Type * base) [inline], [static]
```

Чтение статуса прерываний CPU0.

## Аргументы

base	Базовый адрес MHU
------	-------------------

Возвращает

Значение статуса прерываний

## 4.12.2.4 MHU\_CPU1\_ClearInt()

```
static void MHU_CPU1_ClearInt (  
    MHU_Type * base,  
    uint32_t mask) [inline], [static]
```

Сброс прерывания CPU1 по маске

## Аргументы

base	Базовый адрес MHU
mask	Маска прерываний

## 4.12.2.5 MHU\_CPU1\_SetInt()

```
static void MHU_CPU1_SetInt (  
    MHU_Type * base,  
    uint32_t mask) [inline], [static]
```

Установка прерывания CPU1 по маске

## Аргументы

base	Базовый адрес MHU
mask	Маска прерываний

## 4.12.2.6 MHU\_CPU1\_StatInt()

```
static uint32_t MHU_CPU1_StatInt (
    MHU_Type * base)  [inline], [static]
```

Чтение статуса прерываний CPU1.

## Аргументы

base	Базовый адрес MHU
------	-------------------

## Возвращает

Значение статуса прерываний

## 4.13 Драйвер модуля POWER

Драйвер модуля управления питанием и режимами работы

## Файлы

- файл [hal\\_power.h](#)  
Интерфейс драйвера модуля POWER.

## Структуры данных

- struct [power\\_mode\\_config](#)  
Структура параметров режима питания
- struct [power\\_trim\\_config](#)  
Структура подстроечных параметров APC и DC-DC.
- struct [power\\_config](#)  
Структура конфигурации блока POWER.
- struct [power\\_state](#)  
Структура параметров состояния блока POWER.
- struct [power\\_handle](#)  
Структура обработчика драйвера I2S.

## Определения типов

- typedef void(\* [power\\_callback\\_t](#)) (PWRCTR\_Type \*base, struct [power\\_handle](#) \*handle, uint8\_t interrupt\_mask, void \*user\_data)  
Функция обратного вызова для обработки прерывания POWER.

## Перечисления

- enum [power\\_status](#)  
Коды возврата функций драйвера POWER.
- enum [power\\_dcdc\\_vlevel](#)  
Перечисление выходных напряжений встроенного регулятора DC-DC.
- enum [power\\_dcdc\\_vlevel\\_value](#)  
Перечисление значений для уровней 0-2 выходных напряжений встроенного регулятора DC-DC.
- enum [power\\_dcdc\\_mode](#)  
Перечисление режимов работы встроенного регулятора DC-DC.
- enum [power\\_eco\\_mode](#)  
Перечисление режимов ECO DC-DC и APC.
- enum [power\\_dcdc\\_threshold](#)  
Перечисление пороговых напряжений DC-DC.
- enum [power\\_flash\\_mode](#)  
Перечисление пороговых напряжений DC-DC.
- enum [power\\_test\\_block](#)  
Перечисление блоков для тестирования
- enum [power\\_interrupt](#)  
Перечисление типов фронтов прерываний

## Функции конфигурирования и чтения состояния

- void [POWER\\_GetCurrentConfig](#) (PWRCTR\_Type \*base, struct [power\\_config](#) \*config)  
Получение текущих значений параметров блока POWER.
- void [POWER\\_SetConfig](#) (PWRCTR\_Type \*base, struct [power\\_config](#) \*config)  
Установка параметров блока POWER.
- void [POWER\\_GetStatus](#) (PWRCTR\_Type \*base, struct [power\\_state](#) \*status)  
Установка параметров блока POWER.

## Функции управления прерываниями

- void [POWER\\_EnableInterrupt](#) (PWRCTR\_Type \*base, enum [power\\_interrupt](#) idx)  
Разрешение прерывания
- void [POWER\\_EnableInterruptMask](#) (PWRCTR\_Type \*base, uint8\_t mask)  
Разрешение прерываний по маске
- bool [POWER\\_IsInterruptEnabled](#) (PWRCTR\_Type \*base, enum [power\\_interrupt](#) idx)  
Запрос - разрешено ли прерывание
- uint8\_t [POWER\\_GetEnabledInterruptMask](#) (PWRCTR\_Type \*base)  
Запрос маски разрешенных прерываний
- void [POWER\\_DisableInterrupt](#) (PWRCTR\_Type \*base, enum [power\\_interrupt](#) idx)  
Запрет прерывания
- void [POWER\\_DisableInterruptMask](#) (PWRCTR\_Type \*base, uint8\_t mask)  
Запрет прерываний по маске
- bool [POWER\\_GetInterruptStatus](#) (PWRCTR\_Type \*base, enum [power\\_interrupt](#) idx)  
Получение состояния флага прерывания
- uint8\_t [POWER\\_GetInterruptStatusMask](#) (PWRCTR\_Type \*base)  
Получение маски активных прерываний
- void [POWER\\_ClearInterrupts](#) (PWRCTR\_Type \*base)  
Сброс признаков активных прерываний
- enum [power\\_status](#) [POWER\\_CreateHandle](#) (PWRCTR\_Type \*base, struct [power\\_handle](#) \*handle, [power\\_callback\\_t](#) callback, void \*user\_data)  
Инициализация обработчика прерываний блока POWER.

#### Функции управления тестовыми режимами

- void `POWER_StartTestMode` (`PWRCTR_Type *base`, enum `power_test_block` `test_block`)  
Запуск тестового режима
- void `POWER_StopTestMode` (`PWRCTR_Type *base`)  
Останов тестового режима

#### Функции переключения режимов

- void `POWER_DeepSleepThisCpu` ()  
Функция погружения ядра процессора в сон Погружается в сон то ядро, на котором выполнится данная функция.
- enum `power_status` `POWER_Standby` (`PWRCTR_Type *base`)  
Функция погружения процессора в сон
- enum `power_status` `POWER_Shutdown` (`PWRCTR_Type *base`)  
Функция погружения процессора в глубокий сон

### 4.13.1 Подробное описание

Драйвер модуля управления питанием и режимами работы

Драйвер модуля POWER позволяет управлять режимами питания микросхемы.

### 4.13.2 Типы

#### 4.13.2.1 `power_callback_t`

```
typedef void(* power_callback_t) (PWRCTR_Type *base, struct power_handle *handle, uint8_t interrupt_mask, void *user_data)
```

Функция обратного вызова для обработки прерывания POWER.

#### Аргументы

<code>base</code>	Базовый адрес блока POWER
<code>handle</code>	Указатель на обработчик
<code>interrupt_mask</code>	Причина вызова
<code>user_data</code>	Данные пользователя

### 4.13.3 Перечисления

#### 4.13.3.1 `power_dcdc_mode`

```
enum power_dcdc_mode
```

Перечисление режимов работы встроенного регулятора DC-DC.

Элементы перечислений

POWER_DcdcModeAuto	Автовыбор PWM/PFM
POWER_DcdcModePwm	PWM
POWER_DcdcModePwmFccm	PWM FCCM
POWER_DcdcModePfm	PFM (рекомендуется для режима ECO)

#### 4.13.3.2 power\_dcdc\_threshold

enum [power\\_dcdc\\_threshold](#)

Перечисление пороговых напряжений DC-DC.

Элементы перечислений

POWER_DcdcThreshold_0_60V	0,60 В
POWER_DcdcThreshold_0_62V	0,62 В
POWER_DcdcThreshold_0_64V	0,64 В
POWER_DcdcThreshold_0_66V	0,66 В
POWER_DcdcThreshold_0_68V	0,68 В
POWER_DcdcThreshold_0_70V	0,70 В
POWER_DcdcThreshold_0_72V	0,72 В
POWER_DcdcThreshold_0_74V	0,74 В
POWER_DcdcThreshold_0_76V	0,76 В
POWER_DcdcThreshold_0_78V	0,78 В
POWER_DcdcThreshold_0_80V	0,80 В
POWER_DcdcThreshold_0_82V	0,82 В
POWER_DcdcThreshold_0_84V	0,84 В
POWER_DcdcThreshold_0_86V	0,86 В
POWER_DcdcThreshold_0_88V	0,88 В
POWER_DcdcThreshold_0_90V	0,90 В

#### 4.13.3.3 power\_dcdc\_vlevel

enum [power\\_dcdc\\_vlevel](#)

Перечисление выходных напряжений встроенного регулятора DC-DC.

Элементы перечислений

POWER_DcdcVLevel0	VLEVEL0
POWER_DcdcVLevel1	VLEVEL1
POWER_DcdcVLevel2	VLEVEL2



## 4.13.3.4 power\_dcdc\_vlevel\_value

enum power\_dcdc\_vlevel\_value

Перечисление значений для уровней 0-2 выходных напряжений встроенного регулятора DC-DC.

Параметры VLEVEL0-2 должны выбираться так, чтобы выполнялись условия:

- VLEVEL0 < VLEVEL1 < VLEVEL2;
- VLEVEL0 < 0,99 В.

Элементы перечислений

POWER_DcdcVLevel_0_85V	0,85 В
POWER_DcdcVLevel_0_86V	0,86 В
POWER_DcdcVLevel_0_87V	0,87 В
POWER_DcdcVLevel_0_88V	0,88 В
POWER_DcdcVLevel_0_89V	0,89 В
POWER_DcdcVLevel_0_90V	0,90 В
POWER_DcdcVLevel_0_91V	0,91 В
POWER_DcdcVLevel_0_92V	0,92 В
POWER_DcdcVLevel_0_93V	0,93 В
POWER_DcdcVLevel_0_94V	0,94 В
POWER_DcdcVLevel_0_95V	0,95 В
POWER_DcdcVLevel_0_96V	0,96 В
POWER_DcdcVLevel_0_97V	0,97 В
POWER_DcdcVLevel_0_98V	0,98 В
POWER_DcdcVLevel_0_99V	0,99 В
POWER_DcdcVLevel_1_00V	1,00 В
POWER_DcdcVLevel_1_01V	1,01 В
POWER_DcdcVLevel_1_02V	1,02 В
POWER_DcdcVLevel_1_03V	1,03 В
POWER_DcdcVLevel_1_04V	1,04 В
POWER_DcdcVLevel_1_05V	1,05 В
POWER_DcdcVLevel_1_06V	1,06 В
POWER_DcdcVLevel_1_07V	1,07 В
POWER_DcdcVLevel_1_08V	1,08 В
POWER_DcdcVLevel_1_09V	1,09 В
POWER_DcdcVLevel_1_10V	1,10 В
POWER_DcdcVLevel_1_11V	1,11 В
POWER_DcdcVLevel_1_12V	1,12 В
POWER_DcdcVLevel_1_13V	1,13 В
POWER_DcdcVLevel_1_14V	1,14 В
POWER_DcdcVLevel_1_15V	1,15 В
POWER_DcdcVLevel_1_16V	1,16 В

## 4.13.3.5 power\_eco\_mode

enum [power\\_eco\\_mode](#)

Перечисление режимов ECO DC-DC и APC.

Элементы перечислений

POWER_EcoOff	Режим ECO выключен
POWER_EcoDcdc	Включен режим ECO DC-DC
POWER_EcoReserved	Зарезервировано (не используется)
POWER_EcoDcdcAndApc	Включен режим ECO DC-DC и APC

## 4.13.3.6 power\_flash\_mode

enum [power\\_flash\\_mode](#)

Перечисление пороговых напряжений DC-DC.

Элементы перечислений

POWER_FlashModeNormal	Рабочий режим
POWER_FlashModeSleep	Сон
POWER_FlashModePowerDown	Флеш отключена

## 4.13.3.7 power\_interrupt

enum [power\\_interrupt](#)

Перечисление типов фронтов прерываний

Элементы перечислений

POWER_VmonRising	Прерывание по нарастающему фронту сигнала монитора питания
POWER_VmonFalling	Прерывание по спадающему фронту сигнала монитора питания

## 4.13.3.8 power\_status

enum [power\\_status](#)

Коды возврата функций драйвера POWER.

Элементы перечислений

POWER_Status_Ok	Успешно
POWER_Status_Fail	Провал

## 4.13.3.9 power\_test\_block

enum [power\\_test\\_block](#)

Перечисление блоков для тестирования

Элементы перечислений

POWER_TestBlockApc	Тестирование APC
POWER_TestBlockDcdc	Тестирование DC-DC
POWER_TestBlockJtm	Тестирование JTM
POWER_TestBlockRwc	Тестирование RWC

#### 4.13.4 Функции

##### 4.13.4.1 POWER\_ClearInterrupts()

```
void POWER_ClearInterrupts (
    PWRCTR_Type * base)
```

Сброс признаков активных прерываний

Аргументы

base	Базовый адрес блока POWER
------	---------------------------

##### 4.13.4.2 POWER\_CreateHandle()

```
enum power_status POWER_CreateHandle (
    PWRCTR_Type * base,
    struct power_handle * handle,
    power_callback_t callback,
    void * user_data)
```

Инициализация обработчика прерываний блока POWER.

Аргументы

base	Базовый адрес блока POWER
handle	Обработчик
callback	Функция обратного вызова
user_data	Данные пользователя

Возвращаемые значения

POWER_Status_Ok	
POWER_Status_Fail	

##### 4.13.4.3 POWER\_DisableInterrupt()

```
void POWER_DisableInterrupt (
    PWRCTR_Type * base,
    enum power_interrupt idx)
```

Запрет прерывания

## Аргументы

base	Базовый адрес блока POWER
idx	Номер флага состояния/прерывания

## 4.13.4.4 POWER\_DisableInterruptMask()

```
void POWER_DisableInterruptMask (  
    PWRCTR_Type * base,  
    uint8_t mask)
```

Запрет прерываний по маске

## Аргументы

base	Базовый адрес блока POWER
mask	Маска флагов прерываний

## 4.13.4.5 POWER\_EnableInterrupt()

```
void POWER_EnableInterrupt (  
    PWRCTR_Type * base,  
    enum power_interrupt idx)
```

Разрешение прерывания

## Аргументы

base	Базовый адрес блока POWER
idx	Номер прерывания

## 4.13.4.6 POWER\_EnableInterruptMask()

```
void POWER_EnableInterruptMask (  
    PWRCTR_Type * base,  
    uint8_t mask)
```

Разрешение прерываний по маске

## Аргументы

base	Базовый адрес блока POWER
mask	Маска флагов прерываний

## 4.13.4.7 POWER\_GetCurrentConfig()

```
void POWER_GetCurrentConfig (  
    PWRCTR_Type * base,  
    struct power_config * config)
```

Получение текущих значений параметров блока POWER.

## Аргументы

base	Базовый адрес блока POWER
config	Структура с параметрами конфигурации

## 4.13.4.8 POWER\_GetEnabledInterruptMask()

```
uint8_t POWER_GetEnabledInterruptMask (
    PWRCTR_Type * base)
```

Запрос маски разрешенных прерываний

Запрос маски разрешенных прерываний, единицы в соответствующих разрядах соответствуют включенным прерываниям.

## Аргументы

base	Базовый адрес блока POWER
------	---------------------------

Возвращает

Маска разрешенных прерываний

## 4.13.4.9 POWER\_GetInterruptStatus()

```
bool POWER_GetInterruptStatus (
    PWRCTR_Type * base,
    enum power_interrupt idx)
```

Получение состояния флага прерывания

## Аргументы

base	Базовый адрес блока POWER
idx	Номер флага состояния/прерывания

Возвращаемые значения

true	Флаг прерывания установлен
false	Флаг прерывания сброшен

## 4.13.4.10 POWER\_GetInterruptStatusMask()

```
uint8_t POWER_GetInterruptStatusMask (
    PWRCTR_Type * base)
```

Получение маски активных прерываний

## Аргументы

base	Базовый адрес блока POWER
------	---------------------------

Возвращает

Маска активных прерываний

## 4.13.4.11 POWER\_GetStatus()

```
void POWER_GetStatus (  
    PWRCTR_Type * base,  
    struct power_state * status)
```

Установка параметров блока POWER.

## Аргументы

base	Базовый адрес блока POWER
status	Структура с параметрами состояния блока POWER

## 4.13.4.12 POWER\_IsInterruptEnabled()

```
bool POWER_IsInterruptEnabled (  
    PWRCTR_Type * base,  
    enum power_interrupt idx)
```

Запрос - разрешено ли прерывание

## Аргументы

base	Базовый адрес блока POWER
idx	Номер флага состояния/прерывания

Возвращаемые значения

true	Прерывание разрешено
false	Прерывание запрещено

## 4.13.4.13 POWER\_SetConfig()

```
void POWER_SetConfig (  
    PWRCTR_Type * base,  
    struct power_config * config)
```

Установка параметров блока POWER.

## Аргументы

base	Базовый адрес блока POWER
config	Структура с параметрами конфигурации

## 4.13.4.14 POWER\_Shutdown()

```
enum power_status POWER_Shutdown (
    PWRCTR_Type * base)
```

Функция погружения процессора в глубокий сон

## Аргументы

base	Базовый адрес блока POWER
------	---------------------------

## Возвращаемые значения

POWER_Status_Ok	
POWER_Status_Fail	

## 4.13.4.15 POWER\_Standby()

```
enum power_status POWER_Standby (
    PWRCTR_Type * base)
```

Функция погружения процессора в сон

## Аргументы

base	Базовый адрес блока POWER
------	---------------------------

## Возвращаемые значения

POWER_Status_Ok	
POWER_Status_Fail	

## 4.13.4.16 POWER\_StartTestMode()

```
void POWER_StartTestMode (
    PWRCTR_Type * base,
    enum power_test_block test_block)
```

Запуск тестового режима

## Аргументы

base	Базовый адрес блока POWER
test_block	Тестируемый блок

## 4.13.4.17 POWER\_StopTestMode()

```
void POWER_StopTestMode (
    PWRCTR_Type * base)
```

Останов тестового режима

## Аргументы

base	Базовый адрес блока POWER
------	---------------------------

## 4.14 Драйвер модуля PPU

Драйвер модуля управления питанием доменов устройств

## Файлы

- файл [hal\\_ppu.h](#)  
Интерфейс драйвера модуля PPU.

## Структуры данных

- struct [ppu\\_config](#)  
Структура для инициализации

## Определения типов

- typedef void(\* [ReqOffForCPU](#)) (void)  
Тип функции запроса для выключения смежного ядра

## Перечисления

- enum [ppu\\_status](#)  
Статусы драйвера PPU.
- enum [ppu\\_domain\\_index](#)  
Индексы блоков PPU.
- enum [ppu\\_sense\\_index](#)  
Индексы бит блоков PPU, от которых зависят другие домены
- enum [ppu\\_power\\_mode](#)  
Состояние домена питания



## Функции

- void `PPU_StateOffRequestHandler` (void)  
Функция отключения питания текущего ядра

## Регистры масок прерывания

- enum `ppu_event_name`  
Имена масок прерываний
- enum `ppu_add_event_name`  
Имена масок дополнительных прерываний

## Идентификационные регистры

- enum `ppu_opportunities_idr0`  
Идентификационный регистр PPU\_IDR0.
- enum `ppu_opportunities_idr1`  
Идентификационный регистр PPU\_IDR1.

## Функции установки состояний

- enum `ppu_status` `PPU_SetState` (PPU\_Type \*base, enum `ppu_power_mode` mode)  
Функция запроса установки статического режима работы
- enum `ppu_status` `PPU_SetStateDynamic` (PPU\_Type \*base, enum `ppu_power_mode` mode)  
Функция запроса установки динамического режима работы
- enum `ppu_status` `PPU_SetPDCMPPUSense` (enum `ppu_domain_index` pd\_dst, enum `ppu_sense_index` pd\_src, uint32\_t sense)  
Функция установки зависимости доменов питания
- enum `ppu_status` `PPU_SetIRQStatus` (PPU\_Type \*base, enum `ppu_event_name` irq, enum `ppu_add_event_name` add\_irq)  
Функция установки состояний прерываний
- enum `ppu_status` `PPU_SetIRQMask` (PPU\_Type \*base, enum `ppu_event_name` irq, enum `ppu_add_event_name` add\_irq)  
Функция установки масок прерываний
- enum `ppu_status` `PPU_ClrIRQStatus` (PPU\_Type \*base, enum `ppu_event_name` irq, enum `ppu_add_event_name` add\_irq)  
Функция сброса состояний прерываний
- enum `ppu_status` `PPU_ClrIRQMask` (PPU\_Type \*base, enum `ppu_event_name` irq, enum `ppu_add_event_name` add\_irq)  
Функция сброса масок прерываний

## Функции получения состояния

- enum `ppu_power_mode` `PPU_GetPowerState` (PPU\_Type \*base)  
Функция получения состояния домена
- uint32\_t `PPU_GetPDxSenseFromPDy` (enum `ppu_domain_index` pd\_dst, enum `ppu_sense_index` pd\_src)  
Функция получения зависимости доменов питания
- enum `ppu_status` `PPU_GetLastAPIStatus` (void)  
Получение статуса выполнения функции, тип результата которой отличен от enum `ppu_status`.
- enum `ppu_status` `PPU_GetIRQStatus` (PPU\_Type \*base, enum `ppu_event_name` \*irq, enum `ppu_add_event_name` \*add\_irq)  
Получение статуса прерываний
- enum `ppu_status` `PPU_GetIRQMask` (PPU\_Type \*base, enum `ppu_event_name` \*irq, enum `ppu_add_event_name` \*add\_irq)  
Получение масок прерываний

## Функции инициализации

- enum `ppu_status` `PPU_Init` (`PPU_Type *base`, `struct ppu_config *cfg`)  
Функция инициализации блока PPU.

### 4.14.1 Подробное описание

Драйвер модуля управления питанием доменов устройств

Драйвер модуля PPU позволяет управлять питанием доменов чипа.

### 4.14.2 Типы

#### 4.14.2.1 ReqOffForCPU

```
typedef void(* ReqOffForCPU) (void)
```

Тип функции запроса для выключения смежного ядра

Механизм отключения домена питания одного ядра из другого ядра определяется пользователем. Этот механизм вызывается драйвером при запросе отключения домена питания одним из одного ядра из другого ядра. Пример механизма приведён в `boards/eliot1_bub/driver_examples/ppu/core1_↔startup/`

### 4.14.3 Перечисления

#### 4.14.3.1 ppu\_add\_event\_name

```
enum ppu_add_event_name
```

Имена масок дополнительных прерываний

Элементы перечислений

<code>PPU_StaPolicyOp</code>	Статус события завершения перехода статической операционной политики
<code>PPU_STA_PolicyPwr</code>	Статус события завершения перехода политики статического питания
<code>PPU_DynDeny</code>	Маска события отказа динамического перехода
<code>PPU_DynAccept</code>	Маска события принятия динамического перехода
<code>PPU_UnsptPolicy</code>	Маска события неподдерживаемой политики
<code>PPU_AddEventNameAll</code>	Все доступные маски дополнительных событий

#### 4.14.3.2 ppu\_domain\_index

```
enum ppu_domain_index
```

Индексы блоков PPU.

Элементы перечислений

PPU_DomainCPU0	Домен питания подсистемы CPU0
PPU_DomainCPU1	Домен питания подсистемы CPU1
PPU_DomainDEBUG	Домен питания подсистемы отладки
PPU_DomainCRYPTO	Домен питания крипто-ускорителя CryptoCell
PPU_DomainGMS	Домен питания крипто-ускорителя GMS
PPU_DomainGNSS	Домен питания навигационного ядра GNSS
PPU_DomainSRAM0	Домен питания банков памяти SRAM0
PPU_DomainSRAM1	Домен питания банков памяти SRAM1
PPU_DomainSRAM2	Домен питания банков памяти SRAM2
PPU_DomainSRAM3	Домен питания банков памяти SRAM3
PPU_DomainSYS	Домен питания системный
PPU_DomainMax	Максимальное значение индекса домена

#### 4.14.3.3 ppu\_event\_name

enum [ppu\\_event\\_name](#)

Имена масок прерываний

Элементы перечислений

PPU_Locked	Маска события блокировки
PPU_EmuDeny	Маска события отказа от эмуляции перехода
PPU_EmuAccept	Маска события принятия перехода эмуляции
PPU_StaDeny	Маска события запрета статического перехода
PPU_StaAccept	Маска события принятия статического перехода
PPU_StaPolicyTrn	Маска события полного завершения статического перехода к политике
PPU_EventNameAll	Все доступные маски событий

#### 4.14.3.4 ppu\_opportunities\_idr0

enum [ppu\\_opportunities\\_idr0](#)

Идентификационный регистр PPU\_IDR0.

Элементы перечислений

PPU_DYN_WRM_RST_SPT	Динамическая поддержка WARM_RST
PPU_DYN_ON_SPT	Динамическая поддержка ON
PPU_DYN_FUNC_RET_SPT	Динамическая поддержка FUNC_RET
PPU_DYN_FULL_RET_SPT	Динамическая поддержка FULL_RET
PPU_DYN_MEM_OFF_SPT	Динамическая поддержка MEM_OFF
PPU_DYN_LGC_RET_SPT	Динамическая поддержка LOGIC_RET
PPU_DYN_MEM_RET_EMU_SPT	Динамическая поддержка MEM_RET_EMU

Элементы перечислений

PPU_DYN_MEM_RET_SPT	Динамическая поддержка MEM_RET
PPU_DYN_OFF_EMU_SPT	Динамическая поддержка OFF_EMU
PPU_DYN_OFF_SPT	Динамическая поддержка OFF
PPU_STA_DBG_RECOV_SPT	Поддержка DBG_RECOV
PPU_STA_WRM_RST_SPT	Поддержка WARM_RST
PPU_STA_ON_SPT	Поддержка ON
PPU_STA_FUNC_RET_SPT	Поддержка FUNC_RET
PPU_STA_FULL_RET_SPT	Поддержка FULL_RET
PPU_STA_MEM_OFF_SPT	Поддержка MEM_OFF
PPU_STA_LGC_RET_SPT	Поддержка LOGIC_RET
PPU_STA_MEM_RET_EMU_SPT	Поддержка MEM_RET_EMU
PPU_STA_MEM_RET_SPT	Поддержка MEM_RET
PPU_STA_OFF_EMU_SPT	Поддержка OFF_EMU
PPU_STA_OFF_SPT	Поддержка OFF

#### 4.14.3.5 ppu\_opportunities\_idr1

enum [ppu\\_opportunities\\_idr1](#)

Идентификационный регистр PPU\_IDR1.

Элементы перечислений

PPU_OFF_MEM_RET_TRANS	Прямой переход от OFF к MEM_RET
PPU_OP_ACTIVE	Модель использования режима работы для динамических переходов
PPU_STA_POLICY_OP_IRQ_SPT	Статус события завершения перехода операционной политики
PPU_STA_POLICY_PWR_IRQ_SPT	Статус события завершения перехода к политике электропитания
PPU_FUNC_RET_RAM_REG	Указывает, присутствует ли регистр PPU_FUNRR или зарезервирован
PPU_FULL_RET_RAM_REG	Указывает, присутствует или зарезервирован регистр PPU_FULRR
PPU_MEM_RET_RAM_REG	Указывает, присутствует ли регистр PPU_MEMRR или зарезервирован
PPU_LOCK_SPT	Блокировка и событие прерывания блокировки поддерживаются
PPU_SW_DEV_DEL_SPT	Поддержка конфигурации управления задержкой программного устройства
PPU_PWR_MODE_ENTRY_DEL_SPT	Поддержка задержки входа в режим питания

#### 4.14.3.6 ppu\_power\_mode

enum [ppu\\_power\\_mode](#)

Состояние домена питания

## Заметки

Приоритет состояния домена питания возрастает с возрастанием значения

## Элементы перечислений

PPU_PowerModeOn	Логика включена, оперативная память включена, компонент работает
PPU_PowerModeOff	Логика выключена и оперативная память выключена
PPU_PowerModeMemRet	Логика выключена, ОЗУ сохранено
PPU_PowerModeWarmRst	Схема в состоянии сброса с включенной логикой и оперативной памятью
PPU_PowerModeDbgRecov	Схема в состоянии сброса с включенной логикой и оперативной памятью. Этот режим используется для включения сброса компонента, как правило, когда он заблокирован или не работает, при этом сохраняется часть или все состояние компонента во время сброса для последующего анализа отладки
PPU_PowerModeFuncRet	Логика включена, оперативная память сохранена, компонент работает
PPU_PowerModeMemOff	Логика включена, ОЗУ выключено, компонент работает
PPU_PowerModeFullRet	Логика и оперативная память в сохранении
PPU_PowerModeLogicRet	Сохранение логики при отключенной оперативной памяти
PPU_PowerModeMemRetEmu	Логика включена, оперативная память включена. Этот режим используется для имитации функционального состояния MEM_RET без отключения питания
PPU_PowerModeOffEmu	Логика включена, оперативная память включена. Этот режим используется для имитации функционального состояния OFF без отключения питания
PPU_PowerModeMax	Максимальное допустимое значение

## 4.14.3.7 ppu\_sense\_index

enum [ppu\\_sense\\_index](#)

Индексы бит блоков PPU, от которых зависят другие домены

## Элементы перечислений

PPU_SenseCPU0	Домен питания подсистемы CPU0
PPU_SenseCPU1	Домен питания подсистемы CPU1
PPU_SenseCRYPTO	Домен питания крипто-ускорителя CryptoCell
PPU_SenseGMS	Домен питания крипто-ускорителя GMS
PPU_SenseGNSS	Домен питания навигационного ядра GNSS
PPU_SenseSRAM0	Домен питания банков памяти SRAM0
PPU_SenseSRAM1	Домен питания банков памяти SRAM1
PPU_SenseSRAM2	Домен питания банков памяти SRAM2
PPU_SenseSRAM3	Домен питания банков памяти SRAM3
PPU_SenseSYS	Домен питания системный

## 4.14.3.8 ppu\_status

enum [ppu\\_status](#)

Статусы драйвера PPU.

Элементы перечислений

<a href="#">PPU_Status_Ok</a>	Нет ошибок
<a href="#">PPU_Status_InvalidArgument</a>	Недопустимый аргумент
<a href="#">PPU_Status_FeatureNotSupport</a>	Не поддерживается
<a href="#">PPU_Status_DriverError</a>	Ошибка драйвера
<a href="#">PPU_Status_ConfigError</a>	Ошибка конфигурации

## 4.14.4 Функции

## 4.14.4.1 PPU\_ClrIRQMask()

```
enum ppu\_status PPU_ClrIRQMask (
    PPU_Type * base,
    enum ppu\_event\_name irq,
    enum ppu\_add\_event\_name add_irq)
```

Функция сброса масок прерываний

Аргументы

base	Базовый адрес блока PPU
irq	Маски основных прерываний
add_irq	Маски дополнительных прерываний

Возвращаемые значения

<a href="#">PPU_Status_Ok</a>	
<a href="#">PPU_Status_InvalidArgument</a>	

## 4.14.4.2 PPU\_ClrIRQStatus()

```
enum ppu\_status PPU_ClrIRQStatus (
    PPU_Type * base,
    enum ppu\_event\_name irq,
    enum ppu\_add\_event\_name add_irq)
```

Функция сброса состояний прерываний

## Аргументы

base	Базовый адрес блока PPU
irq	Основные прерывания
add_irq	Дополнительные прерывания

## Возвращаемые значения

PPU_Status_Ok	
PPU_Status_InvalidArgument	

## 4.14.4.3 PPU\_GetIRQMask()

```
enum ppu_status PPU_GetIRQMask (
    PPU_Type * base,
    enum ppu_event_name * irq,
    enum ppu_add_event_name * add_irq)
```

## Получение масок прерываний

## Аргументы

base	Базовый адрес блока PPU
irq	Маски прерываний
add_irq	Маски дополнительных прерываний

## Возвращает

Статус

## 4.14.4.4 PPU\_GetIRQStatus()

```
enum ppu_status PPU_GetIRQStatus (
    PPU_Type * base,
    enum ppu_event_name * irq,
    enum ppu_add_event_name * add_irq)
```

## Получение статуса прерываний

## Аргументы

base	Базовый адрес блока PPU
irq	Состояния прерываний
add_irq	Состояния дополнительных прерываний

## Возвращает

Статус

## 4.14.4.5 PPU\_GetLastAPIStatus()

```
enum ppu_status PPU_GetLastAPIStatus (
    void )
```

Получение статуса выполнения функции, тип результата которой отличен от enum ppu\_status.

Возвращает

Статус

## 4.14.4.6 PPU\_GetPDxSenseFromPDy()

```
uint32_t PPU_GetPDxSenseFromPDy (
    enum ppu_domain_index pd_dst,
    enum ppu_sense_index pd_src)
```

Функция получения зависимости доменов питания

Аргументы

pd_dst	Домен который зависит от состояние другого домена
pd_src	Домен от которого зависит состояние другого домена

Возвращает

1 - Есть зависимость

0 - Нет зависимости или выполняемое действие не валидно. Узнать валидность можно вызвав ( PPU\_GetLastAPIStatus )

## 4.14.4.7 PPU\_GetPowerState()

```
enum ppu_power_mode PPU_GetPowerState (
    PPU_Type * base)
```

Функция получения состояния домена

Аргументы

base	Базовый адрес блока PPU
------	-------------------------

Возвращает

Состояние домена

Заметки

Если задан некорректный адрес, возвращается PPU\_PowerModeOff,

## 4.14.4.8 PPU\_Init()

```
enum ppu_status PPU_Init (
    PPU_Type * base,
    struct ppu_config * cfg)
```

Функция инициализации блока PPU.



## Аргументы

base	Базовый адрес блока PPU
cfg	Конфигурация

## Возвращаемые значения

PPU_Status_Ok	
PPU_Status_InvalidArgument	

## 4.14.4.9 PPU\_SetIRQMask()

```
enum ppu_status PPU_SetIRQMask (
    PPU_Type * base,
    enum ppu_event_name irq,
    enum ppu_add_event_name add_irq)
```

## Функция установки масок прерываний

## Аргументы

base	Базовый адрес блока PPU
irq	Маски основных прерываний
add_irq	Маски дополнительных прерываний

## Возвращаемые значения

PPU_Status_Ok	
PPU_Status_InvalidArgument	

## 4.14.4.10 PPU\_SetIRQStatus()

```
enum ppu_status PPU_SetIRQStatus (
    PPU_Type * base,
    enum ppu_event_name irq,
    enum ppu_add_event_name add_irq)
```

## Функция установки состояний прерываний

## Аргументы

base	Базовый адрес блока PPU
irq	Основные прерывания
add_irq	Дополнительные прерывания

Возвращаемые значения

<a href="#">PPU_Status_Ok</a>	
<a href="#">PPU_Status_InvalidArgument</a>	

#### 4.14.4.11 PPU\_SetPDCMPPUSense()

```
enum ppu\_status PPU_SetPDCMPPUSense (
    enum ppu\_domain\_index pd_dst,
    enum ppu\_sense\_index pd_src,
    uint32_t sense)
```

Функция установки зависимости доменов питания

Аргументы

pd_dst	Домен который зависит от состояние другого домена
pd_src	Домен от которого зависит состояние другого домена
sense	Зависимость: 1 - есть, 0 - нет

Возвращаемые значения

<a href="#">PPU_Status_Ok</a>	
<a href="#">PPU_Status_FeatureNotSupport</a>	

#### 4.14.4.12 PPU\_SetState()

```
enum ppu\_status PPU_SetState (
    PPU_Type * base,
    enum ppu\_power\_mode mode)
```

Функция запроса установки статического режима работы

Аргументы

base	Базовый адрес блока PPU
mode	Устанавливаемое состояние

Заметки

Статический переход доменов CPUx в режим OFF недопустим и приведёт к возникновению BusFault.

Возвращаемые значения

<a href="#">PPU_Status_Ok</a>	
<a href="#">PPU_Status_InvalidArgument</a>	
<a href="#">PPU_Status_FeatureNotSupport</a>	

#### 4.14.4.13 PPU\_SetStateDynamic()

```
enum ppu\_status PPU_SetStateDynamic (
    PPU_Type * base,
    enum ppu\_power\_mode mode)
```

Функция запроса установки динамического режима работы

Аргументы

base	Базовый адрес блока PPU
mode	Устанавливаемое состояние

Заметки

Переход доменов CPUx в режим OFF произойдет только после вызова инструкции WFI, в режим ON допускается переводить, когда ядро будет настроено для работы(у ядра будет исполняемая программа)

Возвращаемые значения

<a href="#">PPU_Status_Ok</a>	
<a href="#">PPU_Status_InvalidArgument</a>	
<a href="#">PPU_Status_FeatureNotSupport</a>	

#### 4.14.4.14 PPU\_StateOffRequestHandler()

```
void PPU_StateOffRequestHandler (
    void )
```

Функция отключения питания текущего ядра

Заметки

Переводит ядро, на котором выполняется функция в динамический режим Off: 1) Устанавливает признак глубокого сна; 2) Разрешает работу WIC/EWC, но не настраивает NVIC; 3) Разрешает отключение FPU, если он есть в ядре; 4) Запрашивает динамически Off; 5) Исполняет \_\_WFI(); После этих действий ядро должно уснуть, а домен отключиться.

Эта функция должна быть использована для отключения одного ядра из другого ядра процессора. Например, если CPU0 пытается отключить CPU1, то эта функция должна вызываться на ядре CPU1 в механизме, который обрабатывает запрос остановки от CPU0 к CPU1 ( см. [ppu\\_config](#) ).

## 4.15 Драйвер модуля PWM

Драйвер широтно-импульсного модулятора

Файлы

- файл [hal\\_pwm.h](#)  
Интерфейс драйвера модуля широтно-импульсного модулятора

Структуры данных

- struct [pwm\\_channel\\_config](#)  
Конфигурация канала широтно-импульсного модулятора
- struct [\\_pwm\\_chopper](#)  
Конфигурация дробления сигнала ШИМ
- struct [\\_pwm\\_dz\\_cfg](#)  
Конфигурация мертвой зоны ШИМ
- struct [\\_pwm\\_trip\\_unit\\_cfg](#)
- struct [\\_pwm\\_handle](#)  
Контекст драйвера ШИМ

Макросы

- `#define PWM_COUNT (3)`
- `#define PWM_CHANNEL_COUNT 4`
- `#define PWM_UNITS { (PWM_Type *)0x40111000, (PWM_Type *)0x40111100, (PWM_Type *)0x40111200, (PWM_Type *)0x40111300 }`
- `#define PWM_FREQ_HZ_TO_PERIOD_US(x) (1000000 / (x))`  
Макрос для преобразования частоты в период
- `#define PWM_PERIOD_MS(x) ((x) * 1000UL)`  
Макрос для преобразования времени из миллисекунд в микросекунды
- `#define PWM_PERIOD_S(x) ((x) * 1000000UL)`  
Макрос для преобразования времени из секунд в микросекунды

Определения типов

- `typedef struct _pwm_chopper pwm_chopper_cfg_t`  
Конфигурация дробления сигнала ШИМ
- `typedef struct _pwm_dz_cfg pwm_dz_cfg_t`  
Конфигурация мертвой зоны ШИМ
- `typedef struct _pwm_trip_unit_cfg pwm_trip_unit_cfg_t`
- `typedef struct _pwm_handle pwm_handle_t`  
Контекст драйвера ШИМ

## Перечисления

- enum [pwm\\_status](#)  
Статусы драйвера широтно-импульсного модулятора
- enum [pwm\\_prescaler\\_mode](#)  
Управление режимом работы предделителя канала
- enum [pwm\\_prescaler\\_cmd](#)  
Управление состоянием предделителя канала
- enum [pwm\\_run\\_command](#)  
Управление пуском/остановкой канала
- enum [pwm\\_prescaler\\_divmux](#)  
Управление мультиплексором делителя частоты (деление частоты после делителя)
- enum [pwm\\_prescaler\\_syncrst](#)  
Разрешения сброса предделителя при возникновении событий SYNCI или SWFSYNC.
- enum [pwm\\_dirsync](#)  
Направление счета после синхронизации
- enum [pwm\\_syncosel](#)  
Выбор источника выходного сигнала SYNCO.
- enum [pwm\\_loadprd](#)  
Управление моментом переписи данных из теневого регистра периода в активный
- enum [pwm\\_syncphsen](#)  
Сигнал разрешения загрузки счетчика из регистра фазы
- enum [pwm\\_cntmode](#)  
Режим работы счетчика CRTCNT.
- enum [pwm\\_scmpxmode](#)  
Режим работы регистра CMPx.
- enum [pwm\\_ldxmode](#)  
Выбор режима загрузки данных из теневого регистра в активный CMPx.
- enum [pwm\\_outx\\_cmd](#)  
Управление выходом OUTx.
- enum [pwm\\_ldcswrf](#)  
Механизм загрузки активного регистра из теневого регистра для регистра программного управления выходами
- enum [pwm\\_dz\\_signal](#)  
Источник сигнала для генерации запрещенной зоны
- enum [pwm\\_dz\\_outx\\_inv](#)  
Полярность OUTx после генерации запрещенной зоны
- enum [pwm\\_dz\\_mode](#)  
Выбор режима работы блока запрещенной зоны при формировании OUTx.
- enum [pwm\\_chopper\\_duty](#)  
Скважность дробящего сигнала
- enum [pwm\\_chopper\\_freq](#)  
Частота дробящего сигнала
- enum [pwm\\_chopper\\_first\\_width](#)  
Ширина первого импульса
- enum [pwm\\_chopper\\_work](#)  
Работа блока Chopper.
- enum [pwm\\_trip\\_unit\\_signal](#)  
Работа блока trip unit.
- enum [pwm\\_trip\\_unit\\_action](#)  
Реакции на событие блока trip unit.

- enum [pwm\\_int\\_en](#)  
Разрешение прерывания блока
- enum [pwm\\_eventprd](#)  
Выбор периода прерываний PWM\_INT.
- enum [pwm\\_int\\_source](#)  
Источник прерывания
- enum  
Выходы каналов ШИМ
- enum [pwm\\_eventprd](#)  
Частота возникновения прерываний ШИМ
- enum [pwm\\_int\\_source](#)  
Источники прерывания ШИМ
- enum [pwm\\_chopper\\_duty](#)  
Скважность дробящего сигнала
- enum [pwm\\_chopper\\_freq](#)  
Частота дробящего сигнала
- enum [pwm\\_chopper\\_first\\_width](#)  
Ширина первого импульса
- enum [pwm\\_dz\\_outx\\_inv](#)  
Полярность OUTx после генерации запрещенной зоны
- enum [pwm\\_dz\\_mode](#)  
Выбор режима работы блока запрещенной зоны при формировании OUTx.
- enum [pwm\\_trip\\_unit\\_action](#)  
Реакции на событие блока trip unit.

## Функции

- void [PWM\\_Init](#) ([pwm\\_handle\\_t](#) \*hpwm)  
Инициализация канала ШИМ и выводов
- void [PWM\\_Enable](#) ([pwm\\_handle\\_t](#) \*hpwm, bool enable)  
Включение вывода канала ШИМ
- void [PWM\\_IntCallback](#) ([pwm\\_handle\\_t](#) \*hpwm)  
Общая функция обратного вызова при возникновении прерываний

## Интерфейс драйвера

- enum [pwm\\_status](#) [PWM\\_GetChannelDefaultConfig](#) ([struct pwm\\_channel\\_config](#) \*cfg)  
Инициализация структуры "по умолчанию" для канала блока ШИМ
- enum [pwm\\_status](#) [PWM\\_InitChannel](#) ([PWM\\_Type](#) \*base, [struct pwm\\_channel\\_config](#) cfg)  
Инициализация канала блока широтно-импульсного модулятора
- enum [pwm\\_status](#) [PWM\\_Deinit](#) ([PWM\\_Type](#) \*base)  
Деинициализация блока широтно-импульсного модулятора
- enum [pwm\\_status](#) [PWM\\_Enable](#) ([PWM\\_Type](#) \*base, [uint32\\_t](#) channel, enum [pwm\\_run\\_command](#) cmd)  
Запуск/останов канала блока широтно-импульсного модулятора
- enum [pwm\\_status](#) [PWM\\_CmdForAllChannels](#) ([PWM\\_Type](#) \*base, [uint32\\_t](#) channel\_mask, enum [pwm\\_run\\_command](#) cmd0, enum [pwm\\_run\\_command](#) cmd1, enum [pwm\\_run\\_command](#) cmd2, enum [pwm\\_run\\_command](#) cmd3)  
Запуск/останов всех каналов блока широтно-импульсного модулятора

- enum `pwm_status PWM_ApplySoftOuts` (PWM\_Type \*base, uint32\_t channel, int8\_t mask↔\_outs)  
Программная не длительная установка значения выходов канала
- enum `pwm_status PWM_ApplyLongSoftOuts` (PWM\_Type \*base, uint32\_t channel, int8\_t↔ mask\_outs, enum `pwm_outx_cmd` outa, enum `pwm_outx_cmd` outb)  
Программная длительная установка значения выходов канала
- uint32\_t `PWM_GetCntStat` (PWM\_Type \*base, uint32\_t channel)  
Определяет работает ли счетчик в канале
- enum `pwm_status PWM_SetPeriod` (PWM\_Type \*base, uint32\_t channel, uint32\_t period)  
Устанавливает значение периода

### 4.15.1 Подробное описание

Драйвер широтно-импульсного модулятора

Драйвер модуля широтно-импульсного модулятора управляет блоком генерации широтно-импульсного модулированного сигнала

Драйвер модуля широтно-импульсного модулятора управляет блоком генерации широтно-импульсного модулированного сигнала

### 4.15.2 Макросы

#### 4.15.2.1 PWM\_COUNT

```
#define PWM_COUNT (3)
```

Количество блоков широтно-импульсного модулятора

### 4.15.3 Перечисления

#### 4.15.3.1 anonymous enum

anonymous enum

Выходы каналов ШИМ

Элементы перечислений

PWM_OutA	Канал A
PWM_OutB	Канал B

#### 4.15.3.2 pwm\_chopper\_duty [1/2]

enum `pwm_chopper_duty`

Скважность дробящего сигнала

Элементы перечислений

pwm_ChopperDuty_1↔ _8	1/8
pwm_ChopperDuty_2↔ _8	2/8
pwm_ChopperDuty_3↔ _8	3/8
pwm_ChopperDuty_4↔ _8	4/8
pwm_ChopperDuty_5↔ _8	5/8
pwm_ChopperDuty_6↔ _8	6/8
pwm_ChopperDuty_7↔ _8	7/8

#### 4.15.3.3 pwm\_chopper\_duty [2/2]

enum [pwm\\_chopper\\_duty](#)

Скважность дробящего сигнала

Элементы перечислений

PWM_ChopperDuty_1↔ _8	1/8
PWM_ChopperDuty_2↔ _8	2/8
PWM_ChopperDuty_3↔ _8	3/8
PWM_ChopperDuty_4↔ _8	4/8
PWM_ChopperDuty_5↔ _8	5/8
PWM_ChopperDuty_6↔ _8	6/8
PWM_ChopperDuty_7↔ _8	7/8

#### 4.15.3.4 pwm\_chopper\_first\_width [1/2]

enum [pwm\\_chopper\\_first\\_width](#)

Ширина первого импульса



Элементы перечислений

pwm_ChopperFirstWidth_0_8	0xCLK/8
pwm_ChopperFirstWidth_1_8	1xCLK/8
pwm_ChopperFirstWidth_2_8	2xCLK/8
pwm_ChopperFirstWidth_3_8	3xCLK/8
pwm_ChopperFirstWidth_4_8	4xCLK/8
pwm_ChopperFirstWidth_5_8	5xCLK/8
pwm_ChopperFirstWidth_6_8	6xCLK/8
pwm_ChopperFirstWidth_7_8	7xCLK/8
pwm_ChopperFirstWidth_8_8	8xCLK/8
pwm_ChopperFirstWidth_9_8	9xCLK/8
pwm_ChopperFirstWidth_10↔ _8	10xCLK/8
pwm_ChopperFirstWidth_11↔ _8	11xCLK/8
pwm_ChopperFirstWidth_12↔ _8	12xCLK/8
pwm_ChopperFirstWidth_13↔ _8	13xCLK/8
pwm_ChopperFirstWidth_14↔ _8	14xCLK/8
pwm_ChopperFirstWidth_15↔ _8	15xCLK/8

#### 4.15.3.5 pwm\_chopper\_first\_width [2/2]

enum [pwm\\_chopper\\_first\\_width](#)

Ширина первого импульса

Элементы перечислений

PWM_ChopperFirstWidth_0_8	0xCLK/8
PWM_ChopperFirstWidth_1_8	1xCLK/8
PWM_ChopperFirstWidth_2_8	2xCLK/8
PWM_ChopperFirstWidth_3_8	3xCLK/8
PWM_ChopperFirstWidth_4_8	4xCLK/8
PWM_ChopperFirstWidth_5_8	5xCLK/8
PWM_ChopperFirstWidth_6_8	6xCLK/8
PWM_ChopperFirstWidth_7_8	7xCLK/8
PWM_ChopperFirstWidth_8_8	8xCLK/8
PWM_ChopperFirstWidth_9_8	9xCLK/8
PWM_ChopperFirstWidth_10↔ _8	10xCLK/8
PWM_ChopperFirstWidth_11↔ _8	11xCLK/8

Элементы перечислений

PWM_ChopperFirstWidth_12↔ _8	12xCLK/8
PWM_ChopperFirstWidth_13↔ _8	13xCLK/8
PWM_ChopperFirstWidth_14↔ _8	14xCLK/8
PWM_ChopperFirstWidth_15↔ _8	15xCLK/8

#### 4.15.3.6 pwm\_chopper\_freq [1/2]

enum [pwm\\_chopper\\_freq](#)

Частота дробящего сигнала

Элементы перечислений

pwm_ChopperFreqClk_8	CLK/8
pwm_ChopperFreqClk_16	CLK/16
pwm_ChopperFreqClk_24	CLK/24
pwm_ChopperFreqClk_32	CLK/32
pwm_ChopperFreqClk_40	CLK/40
pwm_ChopperFreqClk_48	CLK/48
pwm_ChopperFreqClk_56	CLK/56
pwm_ChopperFreqClk_64	CLK/64

#### 4.15.3.7 pwm\_chopper\_freq [2/2]

enum [pwm\\_chopper\\_freq](#)

Частота дробящего сигнала

Элементы перечислений

PWM_ChopperFreqClk_8	CLK/8
PWM_ChopperFreqClk_16	CLK/16
PWM_ChopperFreqClk_24	CLK/24
PWM_ChopperFreqClk_32	CLK/32
PWM_ChopperFreqClk_40	CLK/40
PWM_ChopperFreqClk_48	CLK/48
PWM_ChopperFreqClk_56	CLK/56
PWM_ChopperFreqClk_64	CLK/64

## 4.15.3.8 pwm\_chopper\_work

enum [pwm\\_chopper\\_work](#)

Работа блока Chopper.

Элементы перечислений

pwm_ChopperWorkOff	Выключена функция дробления
pwm_ChopperWorkOn	Включена функция дробления

## 4.15.3.9 pwm\_cntmode

enum [pwm\\_cntmode](#)

Режим работы счетчика CRTCNT.

Элементы перечислений

pwm_CntModeUp	Up-count режим
pwm_CntModeDown	Down-count режим
pwm_CntModeUpDown	Up-down-count режим
pwm_CntModeOff	Счет не осуществляется

## 4.15.3.10 pwm\_dirsync

enum [pwm\\_dirsync](#)

Направление счета после синхронизации

Заметки

Этот бит используется, только когда счетчик работает в up-down режиме.

Элементы перечислений

pwm_DirSyncDown	После синхронизации счетчик декрементируется
pwm_DirSyncUp	После синхронизации счетчик инкрементируется

## 4.15.3.11 pwm\_dz\_mode [1/2]

enum [pwm\\_dz\\_mode](#)

Выбор режима работы блока запрещенной зоны при формировании OUTx.

Элементы перечислений

<code>pwm_DzModeOff</code>	Генератор выключен
<code>pwm_DzModeOn</code>	Генератор включен

#### 4.15.3.12 `pwm_dz_mode` [2/2]

enum `pwm_dz_mode`

Выбор режима работы блока запрещенной зоны при формировании OUTx.

Элементы перечислений

<code>PWM_DzModeOff</code>	Генератор выключен
<code>PWM_DzModeOn</code>	Генератор включен

#### 4.15.3.13 `pwm_dz_outx_inv` [1/2]

enum `pwm_dz_outx_inv`

Полярность OUTx после генерации запрещенной зоны

Элементы перечислений

<code>pwm_DzSignalOutxInvOff</code>	Нет инверсии
<code>pwm_DzSignalOutxInvOn</code>	Есть инверсия

#### 4.15.3.14 `pwm_dz_outx_inv` [2/2]

enum `pwm_dz_outx_inv`

Полярность OUTx после генерации запрещенной зоны

Элементы перечислений

<code>PWM_DzSignalOutxInvOff</code>	Нет инверсии
<code>PWM_DzSignalOutxInvOn</code>	Есть инверсия

#### 4.15.3.15 `pwm_dz_signal`

enum `pwm_dz_signal`

Источник сигнала для генерации запрещенной зоны

Элементы перечислений

pwm_DzSignalOutA	Сигнал OUTA in
pwm_DzSignalOutB	Сигнал OUTB in

#### 4.15.3.16 pwm\_eventprd [1/2]

enum [pwm\\_eventprd](#)

Выбор периода прерываний PWM\_INT.

Элементы перечислений

pwm_EventPrdNo	Нет генерации прерывания
pwm_EventPrdOne	Генерация прерывания каждый раз при возникновении события
pwm_EventPrdTwo	Генерация прерывания каждый второй раз при возникновении события
pwm_EventPrdThree	Генерация прерывания каждый третий раз при возникновении события

#### 4.15.3.17 pwm\_eventprd [2/2]

enum [pwm\\_eventprd](#)

Частота возникновения прерываний ШИМ

Элементы перечислений

PWM_EventPrdNo	Нет генерации прерывания
PWM_EventPrdOne	Генерация прерывания каждый раз при возникновении события
PWM_EventPrdTwo	Генерация прерывания каждый второй раз при возникновении события
PWM_EventPrdThree	Генерация прерывания каждый третий раз при возникновении события

#### 4.15.3.18 pwm\_int\_en

enum [pwm\\_int\\_en](#)

Разрешение прерывания блока

Элементы перечислений

pwm_IntEnNo	Прерывание запрещено
pwm_IntEnYes	Прерывание разрешено

#### 4.15.3.19 pwm\_int\_source [1/2]

enum [pwm\\_int\\_source](#)

Источник прерывания

Элементы перечислений

<code>pwm_IntSourceNo</code>	Резерв
<code>pwm_IntSourceCtrcntEquZero</code>	Признак равенства счетчика нулю $CTRCNT=0$
<code>pwm_IntSourceCtrcntEquCtrprd</code>	Признак равенства счетчика периоду счета $CTRCNT=CTRPRD$
<code>pwm_IntSourceCtrcntEquCmpAInc</code>	Признак равенства счетчика регистру сравнения CMPA во время инкремента
<code>pwm_IntSourceCtrcntEquCmpADec</code>	Признак равенства счетчика регистру сравнения CMPA во время декремента
<code>pwm_IntSourceCtrcntEquCmpBInc</code>	Признак равенства счетчика регистру сравнения CMPB во время инкремента
<code>pwm_IntSourceCtrcntEquCmpBDec</code>	Признак равенства счетчика регистру сравнения CMPB во время декремента

#### 4.15.3.20 `pwm_int_source` [2/2]

enum `pwm_int_source`

Источники прерывания ШИМ

Элементы перечислений

<code>PWM_IntSourceCtrcntEquZero</code>	Признак равенства счетчика ШИМ нулю
<code>PWM_IntSourceCtrcntEquCtrprd</code>	Признак равенства счетчика ШИМ периоду счета
<code>PWM_IntSourceCtrcntEquCmpAInc</code>	Признак равенства счетчика ШИМ регистру сравнения выхода А во время инкремента
<code>PWM_IntSourceCtrcntEquCmpADec</code>	Признак равенства счетчика ШИМ регистру сравнения выхода А во время декремента
<code>PWM_IntSourceCtrcntEquCmpBInc</code>	Признак равенства счетчика ШИМ регистру сравнения выхода В во время инкремента
<code>PWM_IntSourceCtrcntEquCmpBDec</code>	Признак равенства счетчика ШИМ регистру сравнения выхода В во время декремента

#### 4.15.3.21 `pwm_ldcswrf`

enum `pwm_ldcswrf`

Механизм загрузки активного регистра из теневого регистра для регистра программного управления выходами

Элементы перечислений

<code>pwm_LdcswrfCtrcntZero</code>	Загрузка при $CTRCNT=0$
<code>pwm_LdcswrfCtrcntEquCtrprd</code>	Загрузка при $CTRCNT=CTRPRD$
<code>pwm_LdcswrfCtrcntZeroEquCtrprd</code>	Загрузка при $CTRCNT=0$ или $CTRCNT=CTRPRD$
<code>pwm_LdcswrfDirect</code>	Загружать напрямую при обращении CPU без использования теневого регистра

## 4.15.3.22 pwm\_ldxmode

enum `pwm_ldxmode`

Выбор режима загрузки данных из теневого регистра в активный CMPx.

Элементы перечислений

<code>pwm_LdxModeCtrcntZero</code>	Загрузка при <code>CTRCNT=0</code>
<code>pwm_LdxModeCtrcntEquCtrprd</code>	Загрузка при <code>CTRCNT=CTRPRD</code>
<code>pwm_LdxModeCtrcntZeroOrEquCtrprd</code>	Загрузка при <code>CTRCNT=0</code> или <code>CTRCNT=CTRPRD</code>
<code>pwm_LdxModeNoLoad</code>	Загрузка не осуществляется

## 4.15.3.23 pwm\_loadprd

enum `pwm_loadprd`

Управление моментом переписи данных из теневого регистра периода в активный

Элементы перечислений

<code>pwm_LoadPrdRegister</code>	Регистр периода ( <code>CTRPRD</code> ) загружается из теневого регистра, когда счетчик ( <code>CTRCNT</code> ) равен нулю (запись или чтение <code>CTRPRD</code> осуществляется через теневой регистр)
<code>pwm_LoadPrdDirect</code>	<code>CTRPRD</code> загружается напрямую без использования теневого регистра (запись или чтение <code>CTRPRD</code> осуществляется напрямую)

## 4.15.3.24 pwm\_outx\_cmd

enum `pwm_outx_cmd`

Управление выходом OUTx.

Элементы перечислений

<code>pwm_OutxCmdNo</code>	Не выполнять действий
<code>pwm_OutxCmdClear</code>	Сбросить OUTx в 0
<code>pwm_OutxCmdSet</code>	Установить OUTx в 1
<code>pwm_OutxCmdToggle</code>	Инвертировать OUTx

## 4.15.3.25 pwm\_prescaler\_cmd

enum `pwm_prescaler_cmd`

Управление состоянием делителя канала

Элементы перечислений

<code>pwm_PrescCmdReset</code>	Счетчик предделителя сбрасывается в «0»
<code>pwm_PrescCmdSave</code>	Счетчик предделителя сохраняет состояние на момент останова

#### 4.15.3.26 `pwm_prescaler_divmux`

enum `pwm_prescaler_divmux`

Управление мультиплексором делителя частоты (деление частоты после делителя)

Элементы перечислений

<code>pwm_PrescalerDivMux1</code>	Частота от предделителя делится на 1
<code>pwm_PrescalerDivMux2</code>	Частота от предделителя делится на 2
<code>pwm_PrescalerDivMux4</code>	Частота от предделителя делится на 4
<code>pwm_PrescalerDivMux8</code>	Частота от предделителя делится на 8
<code>pwm_PrescalerDivMux16</code>	Частота от предделителя делится на 16
<code>pwm_PrescalerDivMuxPWMClk</code>	Используется внешняя частота <code>PWM_CLK</code>

#### 4.15.3.27 `pwm_prescaler_mode`

enum `pwm_prescaler_mode`

Управление режимом работы предделителя канала

Элементы перечислений

<code>pwm_PrescModeTimerIsRun</code>	Предделитель формирует частоту только при включенном таймере
<code>pwm_PrescModeAlways</code>	Предделитель формирует частоту не зависимо от включенности таймера

#### 4.15.3.28 `pwm_prescaler_syncrst`

enum `pwm_prescaler_syncrst`

Разрешения сброса предделителя при возникновении событий `SYNCI` или `SWFSYNC`.

Элементы перечислений

<code>pwm_PrescalerSyncRstDis</code>	Сброс запрещен
<code>pwm_PrescalerSyncRstEn</code>	Сброс разрешен

#### 4.15.3.29 `pwm_run_command`

enum `pwm_run_command`

Управление пуском/остановкой канала



Элементы перечислений

pwm_RunCmdStop	Остановка после следующего переключения счетчика CTCNT
pwm_RunCmdStopEvent	Остановка при совершении следующих событий: up-count режим: остановка при CTCNT==CTRPRD down-count режим: остановка при CTCNT==0 up-down-count режим: остановка при CTCNT==0
pwm_RunCmdRun	Запуск

#### 4.15.3.30 pwm\_scmpxmode

enum [pwm\\_scmpxmode](#)

Режим работы регистра CMPx.

Элементы перечислений

pwm_SCmpxModeReg	Работа с теневым регистром, все запросы CPU проходят через теневой регистр
pwm_SCmpxModeDirect	Прямой режим, используется только активный регистр CMPx

#### 4.15.3.31 pwm\_status

enum [pwm\\_status](#)

Статусы драйвера широтно-импульсного модулятора

Элементы перечислений

PWM_Status_Ok	Нет ошибок
PWM_Status_InvalidArgument	Недопустимый аргумент
PWM_Status_BadConfigure	Недопустимая конфигурация

#### 4.15.3.32 pwm\_syncosel

enum [pwm\\_syncosel](#)

Выбор источника выходного сигнала SYNCO.

Элементы перечислений

pwm_SyncoSelSyncl	SYNCL
pwm_SyncoSelCtrcntZero	Счетчик равен нулю (CTRCNT==0)
pwm_SyncoSelCtrcntEquCmpb	Счетчик равен значению регистра сравнения В (CTRCNT==CMPB)
pwm_SyncoSelOff	SYNCO отключен

#### 4.15.3.33 pwm\_syncphsen

enum [pwm\\_syncphsen](#)

Сигнал разрешения загрузки счетчика из регистра фазы

Элементы перечислений

pwm_SyncPhsEnDis	Загрузка CTRCNT из регистра фазы CTRPHS запрещена
pwm_SyncPhsEnEn	Загрузка CTRCNT из регистра фазы CTRPHS во время синхронизации разрешена

#### 4.15.3.34 pwm\_trip\_unit\_action [1/2]

enum [pwm\\_trip\\_unit\\_action](#)

Реакции на событие блока trip unit.

Элементы перечислений

pwm_TripUnitActionHigh	OUTx переводится в высокоомное состояние
pwm_TripUnitActionOne	OUTx подтягивается к 1
pwm_TripUnitActionZero	OUTx подтягивается к 0
pwm_TripUnitActionNo	Действий не производится

#### 4.15.3.35 pwm\_trip\_unit\_action [2/2]

enum [pwm\\_trip\\_unit\\_action](#)

Реакции на событие блока trip unit.

Элементы перечислений

PWM_TripUnitActionHigh	OUTx переводится в высокоомное состояние
PWM_TripUnitActionOne	OUTx подтягивается к 1
PWM_TripUnitActionZero	OUTx подтягивается к 0
PWM_TripUnitActionNo	Действий не производится

#### 4.15.3.36 pwm\_trip\_unit\_signal

enum [pwm\\_trip\\_unit\\_signal](#)

Работа блока trip unit.

Элементы перечислений

pwm_TripUnitSignalNotUsed	Вход не используется
pwm_TripUnitSignalUsed	Вход используется

#### 4.15.4 Функции

##### 4.15.4.1 PWM\_ApplyLongSoftOuts()

```
enum pwm_status PWM_ApplyLongSoftOuts (
    PWM_Type * base,
    uint32_t channel,
    int8_t mask_outs,
    enum pwm_outx_cmd outa,
    enum pwm_outx_cmd outb)
```

Программная длительная установка значения выходов канала

Аргументы

base	Блок широтно-импульсного модулятора
channel	Канал
mask_outs	Маски выводов: бит 0 - OutA; 1 - OutB
outa	Значение вывода OutA
outb	Значение вывода OutB

Заметки

Значения pwm\_OutxCmdToggle для outa и outb имеют эффект как pwm\_OutxCmdNo.

Возвращаемые значения

PWM_Status_Ok	
PWM_Status_InvalidArgument	
PWM_Status_BadConfigure	

##### 4.15.4.2 PWM\_ApplySoftOuts()

```
enum pwm_status PWM_ApplySoftOuts (
    PWM_Type * base,
    uint32_t channel,
    int8_t mask_outs)
```

Программная не длительная установка значения выходов канала

Заметки

В отличие от длительной приводит к переключению значения выходов которое будет изменено аппаратным событием. Значения устанавливаются ранее предопределённые в конфигурации.

Аргументы

base	Блок широтно-импульсного модулятора
channel	Канал
mask_outs	Маски выводов: бит 0 - OutA; 1 - OutB

Возвращаемые значения

<a href="#">PWM_Status_Ok</a>	
<a href="#">PWM_Status_InvalidArgument</a>	
<a href="#">PWM_Status_BadConfigure</a>	

#### 4.15.4.3 PWM\_CmdForAllChannels()

```
enum pwm\_status PWM_CmdForAllChannels (
    PWM_Type * base,
    uint32_t channel_mask,
    enum pwm\_run\_command cmd0,
    enum pwm\_run\_command cmd1,
    enum pwm\_run\_command cmd2,
    enum pwm\_run\_command cmd3)
```

Запуск/останов всех каналов блока широтно-импульсного модулятора

Аргументы

base	Блок широтно-импульсного модулятора
channel_mask	Каналы - 1 бит означает управление каналом
cmd0	Команда запуска 0 канала широтно-импульсного модулятора
cmd1	Команда запуска 1 канала широтно-импульсного модулятора
cmd2	Команда запуска 2 канала широтно-импульсного модулятора
cmd3	Команда запуска 3 канала широтно-импульсного модулятора

Возвращаемые значения

<a href="#">PWM_Status_Ok</a>	
<a href="#">PWM_Status_InvalidArgument</a>	

#### 4.15.4.4 PWM\_Deinit()

```
enum pwm\_status PWM_Deinit (
    PWM_Type * base)
```

Деинициализация блока широтно-импульсного модулятора

Аргументы

base	Блок широтно-импульсного модулятора
------	-------------------------------------

Возвращаемые значения

<a href="#">PWM_Status_Ok</a>	
<a href="#">PWM_Status_InvalidArgument</a>	

## 4.15.4.5 PWM\_Enable() [1/2]

```
void PWM_Enable (
    pwm_handle_t * hpwm,
    bool enable)
```

Включение вывода канала ШИМ

Аргументы

hpwm	Контекст драйвера ШИМ
bool	1 - включение, 0 - выключение

## 4.15.4.6 PWM\_Enable() [2/2]

```
enum pwm_status PWM_Enable (
    PWM_Type * base,
    uint32_t channel,
    enum pwm_run_command cmd)
```

Запуск/останов канала блока широтно-импульсного модулятора

Аргументы

base	Блок широтно-импульсного модулятора
channel	Канал
cmd	Команда запуска для канала широтно-импульсного модулятора

Возвращаемые значения

PWM_Status_Ok	
PWM_Status_InvalidArgument	

## 4.15.4.7 PWM\_GetChannelDefaultConfig()

```
enum pwm_status PWM_GetChannelDefaultConfig (
    struct pwm_channel_config * cfg)
```

Инициализация структуры "по умолчанию" для канала блока ШИМ

Аргументы

cfg	Конфигурация
-----	--------------

Возвращаемые значения

PWM_Status_Ok	
PWM_Status_InvalidArgument	

## 4.15.4.8 PWM\_GetCntStat()

```
uint32_t PWM_GetCntStat (
    PWM_Type * base,
    uint32_t channel)
```

Определяет работает ли счетчик в канале

Аргументы

base	Блок широтно-импульсного модулятора
channel	Канал

Возвращаемые значения

1	- Счетчик работает
0	- Счетчик не работает

## 4.15.4.9 PWM\_Init()

```
void PWM_Init (
    pwm_handle_t * hpwm)
```

Инициализация канала ШИМ и выводов

Аргументы

hpwm	Контекст драйвера ШИМ
------	-----------------------

## 4.15.4.10 PWM\_InitChannel()

```
enum pwm_status PWM_InitChannel (
    PWM_Type * base,
    struct pwm_channel_config cfg)
```

Инициализация канала блока широтно-импульсного модулятора

Аргументы

base	Блок широтно-импульсного модулятора
cfg	Конфигурация

Возвращаемые значения

<a href="#">PWM_Status_Ok</a>	
<a href="#">PWM_Status_InvalidArgument</a>	

## 4.15.4.11 PWM\_IntCallback()

```
void PWM_IntCallback (
    pwm_handle_t * hpwm)
```

Общая функция обратного вызова при возникновении прерываний

## Аргументы

hpwm	Контекст драйвера ШИМ
------	-----------------------

## 4.15.4.12 PWM\_SetPeriod()

```
enum pwm_status PWM_SetPeriod (
    PWM_Type * base,
    uint32_t channel,
    uint32_t period)
```

Устанавливает значение периода

## Аргументы

base	Блок широтно-импульсного модулятора
channel	Канал
period	Период в клоках частоты тактирования

## Возвращаемые значения

PWM_Status_Ok	
PWM_Status_InvalidArgument	
PWM_Status_BadConfigure	

## 4.16 Драйвер модуля QSPI

Драйвер контроллера QSPI.

## Файлы

- файл [hal\\_nor\\_flash.h](#)  
Интерфейс драйвера флеш-памяти NOR.
- файл [hal\\_qspi.h](#)  
Интерфейс драйвера модуля QSPI.
- файл [hal\\_qspi\\_dma.h](#)  
Дополнение драйвера QSPI для обмена данными с помощью DMA.
- файл [hal\\_qspi\\_nor\\_flash.h](#)  
Интерфейс драйвера модуля QSPI-NOR-FLASH.

## Структуры данных

- struct [\\_nor\\_command\\_set](#)  
Основной набор команд для флеш-памяти NOR.
- struct [\\_nor\\_config](#)  
Структура первичной конфигурации флеш-памяти NOR.
- struct [\\_nor\\_handle](#)  
Контекст драйвера флеш-памяти NOR.
- struct [\\_qspi\\_config](#)  
Количество бит во фрейме
- struct [\\_qspi\\_xip\\_config](#)  
Структура параметров конфигурации XIP контроллера QSPI.
- struct [qspi\\_dma\\_handle\\_t](#)  
Дескриптор QSPI-DMA передачи
- struct [\\_qspi\\_nor\\_config](#)  
Конфигурационный блок для режима Quad.
- struct [\\_qspi\\_nor\\_init\\_config](#)  
Первоначальная конфигурация QSPI.
- struct [\\_qspi\\_nor\\_handle](#)  
Контекст драйвера NOR Flash.

## Макросы

- `#define` [DUMMY\\_BYTE](#) 0x00

## Определения типов

- typedef struct [\\_nor\\_command\\_set](#) nor\_command\_set\_t  
Основной набор команд для флеш-памяти NOR.
- typedef struct [\\_nor\\_config](#) nor\_config\_t  
Структура первичной конфигурации флеш-памяти NOR.
- typedef struct [\\_nor\\_handle](#) nor\_handle\_t  
Контекст драйвера флеш-памяти NOR.
- typedef enum [\\_qspi\\_qmode](#) qspi\_qmode\_t  
Режим работы контроллера QSPI.
- typedef struct [\\_qspi\\_config](#) qspi\_config\_t  
Количество бит во фрейме
- typedef struct [\\_qspi\\_xip\\_config](#) qspi\_xip\_config\_t  
Структура параметров конфигурации XIP контроллера QSPI.
- typedef struct [\\_qspi\\_nor\\_config](#) qspi\_nor\_config\_t  
Конфигурационный блок для режима Quad.
- typedef enum [\\_qspi\\_command\\_format](#) qspi\_command\_format\_t  
Формат команды QSPI.
- typedef struct [\\_qspi\\_nor\\_init\\_config](#) qspi\_nor\_init\_config\_t  
Первоначальная конфигурация QSPI.
- typedef enum [\\_qspi\\_command\\_type](#) qspi\_command\_type\_t  
Тип команды QSPI.
- typedef struct [\\_qspi\\_nor\\_handle](#) qspi\_nor\_handle\_t  
Контекст драйвера NOR Flash.



## Перечисления

- enum [nor\\_status\\_t](#)  
Статусы драйвера флеш-памяти NOR.
- enum [\\_qspi\\_qmode](#)  
Режим работы контроллера QSPI.
- enum [qspi\\_dma\\_status\\_t](#)  
Статусы выполнения функций
- enum [\\_serial\\_nor\\_command](#)  
Коды операций микросхемы флеш-памяти
- enum  
Требования для включения режима Quad.
- enum [\\_qspi\\_command\\_format](#)  
Формат команды QSPI.
- enum [\\_qspi\\_command\\_type](#)  
Тип команды QSPI.

## Функции

- void [QSPI\\_GetDefaultConfigXIP](#) ([qspi\\_xip\\_config\\_t](#) \*qspi\_xip\_config)  
Получение конфигурации XIP QSPI по умолчанию
- void [QSPI\\_GetDefaultCommandSet](#) ([nor\\_command\\_set\\_t](#) \*command\_set)  
Получение стандартного набора команд SPI Flash.
- [nor\\_status\\_t](#) [QSPI\\_EnableXIP](#) ([nor\\_handle\\_t](#) \*handle)  
Заполнение регистра XIPCFG контроллера QSPI.
- void [QSPI\\_DisableXIP](#) ([nor\\_handle\\_t](#) \*handle)  
Выключение режима QSPI XIP.
- [nor\\_status\\_t](#) [QSPI\\_NorFlashInit](#) ([nor\\_config\\_t](#) \*config, [nor\\_handle\\_t](#) \*handle)  
Инициализация устройства флеш-памяти NOR.
- [nor\\_status\\_t](#) [NOR\\_FlashRead](#) ([nor\\_handle\\_t](#) \*handle, [uint32\\_t](#) address, [uint8\\_t](#) \*buffer, [uint32\\_t](#) length)  
Чтение данных с флеш-памяти NOR.
- [nor\\_status\\_t](#) [NOR\\_FlashPageProgram](#) ([nor\\_handle\\_t](#) \*handle, [uint32\\_t](#) address, [uint8\\_t](#) \*buffer, [uint32\\_t](#) length)  
Программирование страницы флеш-памяти NOR.
- [nor\\_status\\_t](#) [NOR\\_FlashEraseBlock](#) ([nor\\_handle\\_t](#) \*handle, [uint32\\_t](#) address, [uint32\\_t](#) size)  
Очистка блока памяти
- [nor\\_status\\_t](#) [NOR\\_FlashEraseChip](#) ([nor\\_handle\\_t](#) \*handle)  
Очистка чипа флеш-памяти NOR.
- void [NOR\\_FlashReadXIP](#) ([uint32\\_t](#) address, [uint8\\_t](#) \*buffer, [uint32\\_t](#) length)  
Чтение в режиме XIP.
- [nor\\_status\\_t](#) [NOR\\_FlashProgramBlock](#) ([nor\\_handle\\_t](#) \*handle, [uint32\\_t](#) address, [uint8\\_t](#) \*page\_pointer, [uint32\\_t](#) length)  
Заполнение блока флеш-памяти NOR.
- [uint32\\_t](#) [QSPI\\_GetInstance](#) ([QSPI\\_Type](#) \*base)  
Получение номера блока QSPI.
- void [QSPI\\_GetDefaultConfig](#) ([qspi\\_config\\_t](#) \*config)  
Получение конфигурации QSPI по умолчанию
- void [QSPI\\_Init](#) ([QSPI\\_Type](#) \*base, const [qspi\\_config\\_t](#) \*config)  
Инициализация контроллера QSPI.
- void [QSPI\\_SetBitSize](#) ([QSPI\\_Type](#) \*base, [qspi\\_bit\\_size\\_t](#) bit\_size)

- Установка количества передаваемых бит
  - void [QSPI\\_SetQMode](#) (QSPI\_Type \*base, [qspi\\_qmode\\_t](#) spi\_mode)
- Установка режима SPI.
  - void [QSPI\\_SetInhibitDin](#) (QSPI\_Type \*base, bool inhibit\_din)
- Установка запрета записи в Tx FIFO.
  - void [QSPI\\_SetInhibitDout](#) (QSPI\_Type \*base, bool inhibit\_dout)
- Установка запрета чтения из Rx FIFO.
  - static void [QSPI\\_Enable](#) (QSPI\_Type \*base)
- Включение контроллера QSPI.
  - static void [QSPI\\_DeInit](#) (QSPI\_Type \*base)
- Деинициализация контроллера QSPI.
  - static void [QSPI\\_EnableDMA](#) (QSPI\_Type \*base)
- Включение DMA.
  - static void [QSPI\\_DisableDMA](#) (QSPI\_Type \*base)
- Выключение DMA.
  - static uint32\_t [QSPI\\_GetStatusFlag](#) (QSPI\_Type \*base)
- Получение значения статусного регистра
  - static void [QSPI\\_SetSlaveSelect](#) (QSPI\_Type \*base, uint32\_t slave\_select)
- Переключение ведомого устройства
  - static void [QSPI\\_EnableInterrupt](#) (QSPI\_Type \*base, uint32\_t mask)
- Включение прерываний
  - static void [QSPI\\_DisableInterrupt](#) (QSPI\_Type \*base, uint32\_t mask)
- Отключение прерываний
  - static void [QSPI\\_ClearInterrupt](#) (QSPI\_Type \*base, uint32\_t mask)
- Сброс прерываний
  - static void [QSPI\\_WriteData](#) (QSPI\_Type \*base, uint32\_t data)
- Передача 32-битного слова в Tx FIFO.
  - static void [QSPI\\_WriteDataByte](#) (QSPI\_Type \*base, uint8\_t data)
- Передача байта данных в Tx FIFO.
  - static uint32\_t [QSPI\\_ReadData](#) (QSPI\_Type \*base)
- Чтение 32-битного слова из Rx FIFO.
  - static uint8\_t [QSPI\\_ReadDataByte](#) (QSPI\_Type \*base)
- Чтение байта данных из Rx FIFO.
  - static uint32\_t [QSPI\\_GetTXLVL](#) (QSPI\_Type \*base)
- Чтение уровня заполнения буфера передачи Tx FIFO.
  - static uint32\_t [QSPI\\_GetRXLVL](#) (QSPI\_Type \*base)
- Чтение уровня заполнения буфера приема Rx FIFO.
  - void [QSPI\\_TransferCreateHandleDMA](#) (QSPI\_Type \*base, [qspi\\_dma\\_handle\\_t](#) \*handle, [dma\\_handle\\_t](#) \*tx\_handle, [dma\\_handle\\_t](#) \*rx\_handle)
- Инициализация контекста передачи QSPI-DMA.
  - [qspi\\_dma\\_status\\_t](#) [QSPI\\_WriteDataDMA](#) ([qspi\\_dma\\_handle\\_t](#) \*handle, void \*addr, uint8\_t incr, uint8\_t transfer\_width, uint32\_t size)
- Запись данных в QSPI TX.
  - [qspi\\_dma\\_status\\_t](#) [QSPI\\_ReadDataDMA](#) ([qspi\\_dma\\_handle\\_t](#) \*handle, void \*addr, uint8\_t incr, uint8\_t transfer\_width, uint32\_t size)
- Чтение данных из QSPI RX.
  - uint32\_t [QSPI\\_GetReadDMADescriptorsCount](#) (uint32\_t size\_in\_bytes)
- Расчет количества дескрипторов многоблочной передачи, нужных для считывания данных из NOR Flash памяти
  - void [QSPI\\_DMAREadDescriptorInitRX](#) (QSPI\_Type \*base, [dma\\_descriptor\\_t](#) \*desc, uint32\_t data\_size, void \*dst\_addr, uint8\_t dst\_addr\_incr)

Инициализация группы дескрипторов DMA (многоблочная передача) для приема данных в RX буфер.

- static uint32\_t `QSPI_GetDummyDMADescriptorsCount` (uint32\_t size\_in\_bytes)

Расчет количества дескрипторов многоблочной передачи, нужных для считывания данных из NOR Flash памяти при послыке фиктивных данных

- static void `QSPI_DMADescriptorInitTX` (QSPI\_Type \*base, dma\_descriptor\_t \*desc, uint32\_t count, uint32\_t data\_size, uint32\_t data\_width, void \*src\_addr, uint8\_t src\_addr\_incr)

Инициализация группы дескрипторов DMA (многоблочная передача) для отправки данных в TX буфер.

- static void `QSPI_DMADescriptorInitRX` (QSPI\_Type \*base, dma\_descriptor\_t \*desc, uint32\_t count, uint32\_t data\_size, uint32\_t data\_width, void \*dst\_addr)

Инициализация группы дескрипторов DMA (многоблочная передача) для приема данных в RX буфер.

Поля внутреннего статусного регистра

- #define `FLASH_STAT_BUSY` (1 << 0)
- #define `FLASH_STAT_WEL` (1 << 1)

#### 4.16.1 Подробное описание

Драйвер контроллера QSPI.

Драйвер поддерживает обмен данными с устройствами, подключенными к выходам QSPI в режимах Normal, Dual, Quad SPI и шириной поля данных от 4 до 32 битов.

Работа контроллера QSPI с флеш-памятью NOR.

#### 4.16.2 Макросы

##### 4.16.2.1 DUMMY\_BYTE

```
#define DUMMY_BYTE 0x00
```

Фиктивные данные

##### 4.16.2.2 FLASH\_STAT\_BUSY

```
#define FLASH_STAT_BUSY (1 << 0)
```

Производится очистка/запись

##### 4.16.2.3 FLASH\_STAT\_WEL

```
#define FLASH_STAT_WEL (1 << 1)
```

Бит Write Enable Latch

### 4.16.3 Типы

#### 4.16.3.1 qspi\_config\_t

typedef struct [\\_qspi\\_config](#) qspi\_config\_t

Количество бит во фрейме

Структура, определяющая параметры конфигурации контроллера QSPI

### 4.16.4 Перечисления

#### 4.16.4.1 anonymous enum

anonymous enum

Требования для включение режима Quad.

Указывает смещение бита Quad Enable.

Элементы перечислений

NOR_QuadModeNotConfig	Режим Quad не сконфигурирован
NOR_QuadModeStatusReg1_Bit6	Статусный регистр 1 бит 6
NOR_QuadModeStatusReg2_Bit1	Статусный регистр 2 бит 1
NOR_QuadModeStatusReg2_Bit7	Статусный регистр 2 бит 7
NOR_QuadModeStatusReg2_Bit1_0x31	Статусный регистр 2 бит 1 (специальная команда)

#### 4.16.4.2 \_qspi\_command\_format

enum [\\_qspi\\_command\\_format](#)

Формат команды QSPI.

Элементы перечислений

QSPI_CommandAllSerial	Все составляющие команды передаются последовательно
QSPI_CommandDataQuad	Только данные передаются в режиме Quad
QSPI_CommandOpcodeSerial	Только код операции передается последовательно

#### 4.16.4.3 \_qspi\_command\_type

enum [\\_qspi\\_command\\_type](#)

Тип команды QSPI.

Элементы перечислений

QSPI_CommandOpcodeOnly	Команда имеет только код операции
QSPI_CommandOpcodeAddrOneByte	Команда имеет код операции и один байт адреса
QSPI_CommandOpcodeAddrTwoBytes	Команда имеет код операции и два байта адреса
QSPI_CommandOpcodeAddrThreeBytes	Команда имеет код операции и три байта адреса
QSPI_CommandOpcodeAddrFourBytes	Команда имеет код операции и четыре байта адреса
QSPI_CommandNoOpcodeAddrThreeBytes	Команда не имеет кода операции и имеет три байта адреса
QSPI_CommandNoOpcodeAddrFourBytes	Команда не имеет кода операции и имеет четыре байта адреса

#### 4.16.4.4 \_qspi\_qmode

enum [\\_qspi\\_qmode](#)

Режим работы контроллера QSPI.

Элементы перечислений

QSPI_NormalSPI	Стандартный режим SPI
QSPI_DualSPI	DUAL SPI
QSPI_QuadSPI	QUAD SPI

#### 4.16.4.5 \_serial\_nor\_command

enum [\\_serial\\_nor\\_command](#)

Коды операций микросхемы флеш-памяти

Элементы перечислений

NOR_CmdInvalid	Неверный код операции
NOR_CmdWriteStatus	WRSR: Запись в статусный регистр
NOR_CmdWriteSecStatus_31	WRSR: Запись во второй статусный регистр
NOR_CmdWriteSecStatus_3E	WRSR: Запись во второй статусный регистр
NOR_CmdWriteMemory	WRITE: Запись байта/страницы в массив флеш-памяти с адресом меньше четырех байт
NOR_CmdWriteMemoryA32	4PP: Запись байта/страницы в массив флеш-памяти с адресом четыре байта
NOR_CmdWriteEnable	WREN: Установка бита Write Enable Latch
NOR_CmdWriteDisable	WRDI: Сброс бита Write Enable Latch
NOR_CmdReadStatus	RDSR: Чтение статусного регистра
NOR_CmdReadSecStatus_35	RDSR: Чтение второго статусного регистра
NOR_CmdReadSecStatus_3F	RDSR: Чтение второго статусного регистра

## Элементы перечислений

NOR_CmdReadMemory	READ: Чтение данных из массива флеш-памяти с адресом меньше четырех байт
NOR_CmdReadMemoryA32	4READ: Чтение данных из массива флеш-памяти с адресом четыре байта
NOR_CmdReadMemorySDR_1_1_1	READ: Чтение данных из массива флеш-памяти в режиме 1_1_1
NOR_CmdReadMemorySDR_1_1_2	READ: Чтение данных из массива флеш-памяти в режиме 1_1_2
NOR_CmdReadMemorySDR_1_2_2	READ: Чтение данных из массива флеш-памяти в режиме 1_2_2
NOR_CmdReadMemorySDR_1_1_4	READ: Чтение данных из массива флеш-памяти в режиме 1_1_4
NOR_CmdReadMemorySDR_1_4_4	READ: Чтение данных из массива флеш-памяти в режиме 1_4_4
NOR_CmdReadMemorySDR_1_4_4_A32	READ: Чтение данных из массива флеш-памяти в режиме 1_4_4 с адресом четыре байта
NOR_CmdReadMemorySDR_1_1_4_A32	READ: Чтение данных из массива флеш-памяти в режиме 1_1_4 с адресом четыре байта
NOR_CmdEraseChipNor	CE: Очистка всего массива флеш-памяти
NOR_CmdEraseChip	CE: Очистка всего массива флеш-памяти
NOR_CmdErasePage	PE: Очистка страницы
NOR_CmdEraseSector4KB	SE4KB: Очистка сектора размером 4КБ
NOR_CmdEraseSector32KB	SE32KB: Очистка сектора размером 32 КБ
NOR_CmdEraseSector	SE: Очистка сектора
NOR_CmdEraseSector4KBA32	4SE4KB: Очистка сектора размером 4КБ с адресом четыре байта
NOR_CmdEraseSectorA32	4SE: Очистка сектора с адресом четыре байта

## 4.16.4.6 nor\_status\_t

enum [nor\\_status\\_t](#)

Статусы драйвера флеш-памяти NOR.

## Элементы перечислений

NOR_Status_Success	Успешное завершение операции
NOR_Status_Fail	Ошибка при выполнении операции
NOR_Status_InvalidArgument	Неверный аргумент
NOR_Status_Timeout	Превышено время ожидания

## 4.16.4.7 qspi\_dma\_status\_t

enum [qspi\\_dma\\_status\\_t](#)

Статусы выполнения функций

Элементы перечислений

QSPI_DMA_Status_Success	Успешное выполнение функции
QSPI_DMA_Status_Fail	Неудачное выполнение функции
QSPI_DMA_Status_InvalidArgument	Неверно переданные аргументы в функцию

## 4.16.5 Функции

### 4.16.5.1 NOR\_FlashEraseBlock()

```
nor_status_t NOR_FlashEraseBlock (
    nor_handle_t * handle,
    uint32_t address,
    uint32_t size)
```

Очистка блока памяти

Аргументы

handle	Контекст драйвера флеш-памяти NOR
address	Начальный адрес для очистки
size	Размер блока для очистки

Возвращаемые значения

NOR_Status_Success	
NOR_Status_Fail	
NOR_Status_InvalidArgument	
NOR_Status_Timeout	

### 4.16.5.2 NOR\_FlashEraseChip()

```
nor_status_t NOR_FlashEraseChip (
    nor_handle_t * handle)
```

Очистка чипа флеш-памяти NOR.

Аргументы

handle	Контекст драйвера флеш-памяти NOR
--------	-----------------------------------

Возвращаемые значения

NOR_Status_Success	
NOR_Status_InvalidArgument	
NOR_Status_Timeout	

## 4.16.5.3 NOR\_FlashPageProgram()

```
nor_status_t NOR_FlashPageProgram (
    nor_handle_t * handle,
    uint32_t address,
    uint8_t * buffer,
    uint32_t length)
```

Программирование страницы флеш-памяти NOR.

Аргументы

handle	Контекст драйвера флеш-памяти NOR
address	Адрес программируемой страницы
buffer	Буфер памяти

Возвращаемые значения

NOR_Status_Success	
NOR_Status_Fail	
NOR_Status_InvalidArgument	
NOR_Status_Timeout	

## 4.16.5.4 NOR\_FlashProgramBlock()

```
nor_status_t NOR_FlashProgramBlock (
    nor_handle_t * handle,
    uint32_t address,
    uint8_t * page_pointer,
    uint32_t length)
```

Заполнение блока флеш-памяти NOR.

Аргументы

handle	Контекст драйвера NOR Flash
address	Адрес блока для записи
page_pointer	Указатель на буфер для записи
length	Размер буфера для записи

Возвращаемые значения

NOR_Status_Success	
NOR_Status_Fail	
NOR_Status_InvalidArgument	



## 4.16.5.5 NOR\_FlashRead()

```
nor_status_t NOR_FlashRead (
    nor_handle_t * handle,
    uint32_t address,
    uint8_t * buffer,
    uint32_t length)
```

Чтение данных с флеш-памяти NOR.

Аргументы

handle	Контекст драйвера флеш-памяти NOR
address	Начальный адрес памяти
buffer	Буфер памяти
length	Размер данных в байтах

Возвращаемые значения

NOR_Status_Success	
NOR_Status_Fail	
NOR_Status_InvalidArgument	

## 4.16.5.6 NOR\_FlashReadXIP()

```
void NOR_FlashReadXIP (
    uint32_t address,
    uint8_t * buffer,
    uint32_t length)
```

Чтение в режиме XIP.

Аргументы

address	Адрес чтения XIP
buffer	Буфер записи прочитанных данных
length	Размер буфера для чтения

## 4.16.5.7 QSPI\_ClearInterrupt()

```
static void QSPI_ClearInterrupt (
    QSPI_Type * base,
    uint32_t mask) [inline], [static]
```

Сброс прерываний

## Аргументы

base	Базовый адрес контроллера QSPI
mask	Маска прерываний

## 4.16.5.8 QSPI\_DeInit()

```
static void QSPI_DeInit (
    QSPI_Type * base)  [inline], [static]
```

Деинициализация контроллера QSPI.

## Аргументы

base	Базовый адрес контроллера QSPI
------	--------------------------------

## 4.16.5.9 QSPI\_DisableDMA()

```
static void QSPI_DisableDMA (
    QSPI_Type * base)  [inline], [static]
```

Выключение DMA.

## Аргументы

base	Базовый адрес контроллера QSPI
------	--------------------------------

## 4.16.5.10 QSPI\_DisableInterrupt()

```
static void QSPI_DisableInterrupt (
    QSPI_Type * base,
    uint32_t mask)  [inline], [static]
```

Отключение прерываний

## Аргументы

base	Базовый адрес контроллера QSPI
mask	Маска прерываний

## 4.16.5.11 QSPI\_DisableXIP()

```
void QSPI_DisableXIP (
    nor_handle_t * handle)
```

Выключение режима QSPI XIP.

## Аргументы

handle	Контекст драйвера флеш-памяти NOR
--------	-----------------------------------

## 4.16.5.12 QSPI\_DMADescriptorInitRX()

```
static void QSPI_DMADescriptorInitRX (
    QSPI_Type * base,
    dma_descriptor_t * desc,
    uint32_t count,
    uint32_t data_size,
    uint32_t data_width,
    void * dst_addr) [inline], [static]
```

Инициализация группы дескрипторов DMA (многоблочная передача) для приема данных в RX буфер.

## Аргументы

base	Базовый адрес DMA
desc	Указатель на массив дескрипторов DMA
data_size	Общий размер передаваемых данных
dst_addr	Адрес Приемника
src_addr_incr	Инкремент адреса

## 4.16.5.13 QSPI\_DMADescriptorInitTX()

```
static void QSPI_DMADescriptorInitTX (
    QSPI_Type * base,
    dma_descriptor_t * desc,
    uint32_t count,
    uint32_t data_size,
    uint32_t data_width,
    void * src_addr,
    uint8_t src_addr_incr) [inline], [static]
```

Инициализация группы дескрипторов DMA (многоблочная передача) для отправки данных в TX буфер.

## Аргументы

base	Базовый адрес DMA
desc	Указатель на массив дескрипторов DMA
count	Число дескрипторов многоблочной передачи
data_size	Общий размер передаваемых данных
data_width	Размер одного слова данных
src_addr	Адрес Источника

#### 4.16.5.14 QSPI\_DMAReadDescriptorInitRX()

```
void QSPI_DMAReadDescriptorInitRX (
    QSPI_Type * base,
    dma_descriptor_t * desc,
    uint32_t data_size,
    void * dst_addr,
    uint8_t dst_addr_incr)
```

Инициализация группы дескрипторов DMA (многоблочная передача) для приема данных в RX буфер.

Аргументы

base	Базовый адрес DMA
desc	Указатель на массив дескрипторов DMA
data_size	Общий размер передаваемых данных
dst_addr	Адрес Приемника
src_addr_incr	Инкремент адреса

#### 4.16.5.15 QSPI\_Enable()

```
static void QSPI_Enable (
    QSPI_Type * base) [inline], [static]
```

Включение контроллера QSPI.

Аргументы

base	Базовый адрес контроллера QSPI
------	--------------------------------

#### 4.16.5.16 QSPI\_EnableDMA()

```
static void QSPI_EnableDMA (
    QSPI_Type * base) [inline], [static]
```

Включение DMA.

Аргументы

base	Базовый адрес контроллера QSPI
------	--------------------------------

#### 4.16.5.17 QSPI\_EnableInterrupt()

```
static void QSPI_EnableInterrupt (
    QSPI_Type * base,
    uint32_t mask) [inline], [static]
```

Включение прерываний

## Аргументы

base	Базовый адрес контроллера QSPI
mask	Маска прерываний

## 4.16.5.18 QSPI\_EnableXIP()

```
nor_status_t QSPI_EnableXIP (
    nor_handle_t * handle)
```

Заполнение регистра XIPCFG контроллера QSPI.

## Аргументы

base	Базовый адрес контроллера QSPI
qspi_xip_config	Конфигурационная структура режима XIP QSPI

## Включение и конфигурация режима QSPI XIP

## Аргументы

handle	Контекст драйвера флеш-памяти NOR
--------	-----------------------------------

## 4.16.5.19 QSPI\_GetDefaultCommandSet()

```
void QSPI_GetDefaultCommandSet (
    nor_command_set_t * command_set)
```

Получение стандартного набора команд SPI Flash.

## Аргументы

command_set	Набор команд SPI Flash
-------------	------------------------

## 4.16.5.20 QSPI\_GetDefaultConfig()

```
void QSPI_GetDefaultConfig (
    qspi_config_t * config)
```

Получение конфигурации QSPI по умолчанию

## Аргументы

config	Конфигурационная структура QSPI
--------	---------------------------------

## 4.16.5.21 QSPI\_GetDefaultConfigXIP()

```
void QSPI_GetDefaultConfigXIP (
    qspi_xip_config_t * qspi_xip_config)
```

Получение конфигурации XIP QSPI по умолчанию

Аргументы

qspi_xip_config	Конфигурационная структура режима XIP QSPI
-----------------	--

#### 4.16.5.22 QSPI\_GetDummyDMADescriptorsCount()

```
static uint32_t QSPI_GetDummyDMADescriptorsCount (
    uint32_t size_in_bytes) [inline], [static]
```

Расчет количества дескрипторов многоблочной передачи, нужных для считывания данных из NOR Flash памяти при послыке фиктивных данных

Аргументы

size_in_bytes	Число данных в байтах
---------------	-----------------------

Возвращает

Количество дескрипторов многоблочной передачи

#### 4.16.5.23 QSPI\_GetInstance()

```
uint32_t QSPI_GetInstance (
    QSPI_Type * base)
```

Получение номера блока QSPI.

Аргументы

base	Адрес QSPI
------	------------

Возвращает

Номер блока QSPI

#### 4.16.5.24 QSPI\_GetReadDMADescriptorsCount()

```
uint32_t QSPI_GetReadDMADescriptorsCount (
    uint32_t size_in_bytes)
```

Расчет количества дескрипторов многоблочной передачи, нужных для считывания данных из NOR Flash памяти

Аргументы

size_in_bytes	Число данных в байтах
---------------	-----------------------

Возвращает

Количество дескрипторов многоблочной передачи

## 4.16.5.25 QSPI\_GetRXLVL()

```
static uint32_t QSPI_GetRXLVL (  
    QSPI_Type * base)  [inline], [static]
```

Чтение уровня заполнения буфера приема Rx FIFO.

Аргументы

base	Базовый адрес контроллера QSPI
------	--------------------------------

Возвращает

Количество фреймов данных в буфере приема

#### 4.16.5.26 QSPI\_GetStatusFlag()

```
static uint32_t QSPI_GetStatusFlag (  
    QSPI_Type * base)  [inline], [static]
```

Получение значения статусного регистра

Аргументы

base	Базовый адрес контроллера QSPI
------	--------------------------------

Возвращает

Значение регистра STAT

#### 4.16.5.27 QSPI\_GetTXLVL()

```
static uint32_t QSPI_GetTXLVL (  
    QSPI_Type * base)  [inline], [static]
```

Чтение уровня заполнения буфера передачи Tx FIFO.

Аргументы

base	Базовый адрес контроллера QSPI
------	--------------------------------

Возвращает

Количество фреймов данных в буфере передачи

#### 4.16.5.28 QSPI\_Init()

```
void QSPI_Init (  
    QSPI_Type * base,  
    const qspi\_config\_t * config)
```

Инициализация контроллера QSPI.



## Аргументы

base	Базовый адрес контроллера QSPI
config	Структура с настройками контроллера по умолчанию

## 4.16.5.29 QSPI\_NorFlashInit()

```
nor_status_t QSPI_NorFlashInit (
    nor_config_t * config,
    nor_handle_t * handle)
```

Инициализация устройства флеш-памяти NOR.

Функция инициализирует контроллер QSPI и флеш-память NOR.

## Аргументы

config	Конфигурация флеш-памяти NOR
handle	Контекст драйвера флеш-памяти NOR

## Возвращаемые значения

NOR_Status_Success	
NOR_Status_Fail	
NOR_Status_InvalidArgument	

## 4.16.5.30 QSPI\_ReadData()

```
static uint32_t QSPI_ReadData (
    QSPI_Type * base) [inline], [static]
```

Чтение 32-битного слова из Rx FIFO.

## Аргументы

base	Базовый адрес контроллера QSPI
------	--------------------------------

## Возвращает

Считанное 32-битное слово

## 4.16.5.31 QSPI\_ReadDataByte()

```
static uint8_t QSPI_ReadDataByte (
    QSPI_Type * base) [inline], [static]
```

Чтение байта данных из Rx FIFO.

## Аргументы

base	Базовый адрес контроллера QSPI
------	--------------------------------

## Возвращает

Считанный байт данных

## 4.16.5.32 QSPI\_ReadDataDMA()

```
qspi_dma_status_t QSPI_ReadDataDMA (
    qspi_dma_handle_t * handle,
    void * addr,
    uint8_t incr,
    uint8_t transfer_width,
    uint32_t size)
```

Чтение данных из QSPI RX.

## Аргументы

handle	Дескриптор DMA-QSPI передачи
addr	Адрес Приемника
incr	Инкремент адреса
transfer_width	Ширина одного слова
size	Общий размер данных (в байтах)

## Возвращаемые значения

QSPI_DMA_Status_Success	
QSPI_DMA_Status_Fail	
QSPI_DMA_Status_InvalidArgument	

## 4.16.5.33 QSPI\_SetBitSize()

```
void QSPI_SetBitSize (
    QSPI_Type * base,
    qspi_bit_size_t bit_size)
```

Установка количества передаваемых бит

## Аргументы

base	Базовый адрес контроллера QSPI
bit_size	Количество передаваемых бит

## 4.16.5.34 QSPI\_SetInhibitDin()

```
void QSPI_SetInhibitDin (
    QSPI_Type * base,
    bool inhibit_din)
```

Установка запрета записи в Tx FIFO.

## Аргументы

base	Базовый адрес контроллера QSPI
inhibit_din	Запрет (1) или нет запрета (0) на запись

## 4.16.5.35 QSPI\_SetInhibitDout()

```
void QSPI_SetInhibitDout (
    QSPI_Type * base,
    bool inhibit_dout)
```

Установка запрета чтения из Rx FIFO.

## Аргументы

base	Базовый адрес контроллера QSPI
inhibit_dout	Запрет (1) или нет запрета (0) на чтение

## 4.16.5.36 QSPI\_SetQMode()

```
void QSPI_SetQMode (
    QSPI_Type * base,
    qspi_qmode_t spi_mode)
```

Установка режима SPI.

## Аргументы

base	Базовый адрес контроллера QSPI
spi_mode	Режим SPI

## 4.16.5.37 QSPI\_SetSlaveSelect()

```
static void QSPI_SetSlaveSelect (
    QSPI_Type * base,
    uint32_t slave_select) [inline], [static]
```

Переключение ведомого устройства

## Аргументы

base	Базовый адрес контроллера QSPI
slave_select	Выбор ведомого устройства по битовой маске

## 4.16.5.38 QSPI\_TransferCreateHandleDMA()

```
void QSPI_TransferCreateHandleDMA (
    QSPI_Type * base,
    qspi_dma_handle_t * handle,
    dma_handle_t * tx_handle,
    dma_handle_t * rx_handle)
```

Инициализация контекста передачи QSPI-DMA.

## Аргументы

base	Базовый адрес QSPI
handle	Указатель на контекст
tx_handle	Указатель на контекст драйвера DMA для передачи данных
rx_handle	Указатель на контекст драйвера DMA для приема данных

## 4.16.5.39 QSPI\_WriteData()

```
static void QSPI_WriteData (
    QSPI_Type * base,
    uint32_t data) [inline], [static]
```

Передача 32-битного слова в Tx FIFO.

## Аргументы

base	Базовый адрес контроллера QSPI
data	32-битное слово для передачи

## 4.16.5.40 QSPI\_WriteDataByte()

```
static void QSPI_WriteDataByte (
    QSPI_Type * base,
    uint8_t data) [inline], [static]
```

Передача байта данных в Tx FIFO.

## Аргументы

base	Базовый адрес контроллера QSPI
data	Байт данных для передачи

## 4.16.5.41 QSPI\_WriteDataDMA()

```
qspi_dma_status_t QSPI_WriteDataDMA (
    qspi_dma_handle_t * handle,
    void * addr,
    uint8_t incr,
    uint8_t transfer_width,
    uint32_t size)
```

Запись данных в QSPI TX.

## Аргументы

handle	Дескриптор DMA-QSPI передачи
addr	Адрес Источника
incr	Инкремент адреса
transfer_width	Ширина одного слова
size	Общий размер данных (в байтах)

Возвращаемые значения

QSPI_DMA_Status_Success	
QSPI_DMA_Status_Fail	
QSPI_DMA_Status_InvalidArgument	

## 4.17 Драйвер модуля RWC

Драйвер счетчика реального времени и Wake-контроллера

Файлы

- файл [hal\\_rwc.h](#)  
Интерфейс драйвера модуля RWC.

Структуры данных

- struct [rwc\\_trimload\\_reg](#)  
Регистр записи значения подстройки из регистра TRIM.
- struct [rwc\\_time\\_reg](#)  
Регистр текущего значения счетчика времени
- struct [rwc\\_alarm\\_reg](#)  
Регистр времени пробуждения
- struct [rwc\\_trim\\_reg](#)  
Регистр подстройки осцилляторов
- struct [rwc\\_config\\_reg](#)  
Конфигурационный регистр
- struct [rwc\\_general\\_reg](#)  
Регистр общего назначения
- struct [rwc\\_wake\\_config\\_reg](#)  
Регистр настройки контроллера пробуждения
- union [rwc\\_union\\_reg](#)  
Объединение для доступа к регистрам
- struct [\\_rtc\\_datetime](#)  
Структура используемая для хранения даты и времени
- struct [rwc\\_config](#)  
Структура используемая для конфигурирования RWC.

Макросы

- `#define RWC_SYNC_RETRY_TIMES 0U`  
Количество циклов ожидания
- `#define RWC_SET_RETRY_TIMES 10U`  
Количество циклов установки времени

## Определения типов

- typedef struct [\\_rtc\\_datetime](#) rtc\_datetime\_t  
Структура используемая для хранения даты и времени

## Перечисления

- enum [rwc\\_time\\_clk\\_sel](#)  
Выбор сигнала для тактирования счетчика времени
- enum [rwc\\_wake\\_up\\_polarity](#)  
Уровень активного сигнала WKUP для генерирования прерывания
- enum [rwc\\_wake\\_up\\_irq\\_enable](#)  
Разрешение прерывания RWC\_WKUP.
- enum [rwc\\_lfe\\_bypass](#)  
Режимы работы осциллятора LFE.
- enum [rwc\\_internal\\_register](#)  
Внутренние регистры с батарейным питанием
- enum [rwc\\_rtclk\\_type](#)  
Источник частоты, подаваемой на RTCCLK.
- enum [rwc\\_reset\\_type](#)  
Виды сбросов внутренних регистров при сигнале SRSTn.
- enum [rwc\\_rtclk\\_divisor](#)  
Делители частоты RTCCLK.
- enum [rwc\\_cmd](#)  
Команды доступа к внутренним регистрам
- enum [rwc\\_status](#)  
Статусы драйвера CLKCTR.
- enum [rwc\\_timer\\_status](#)  
Статусы таймера
- enum [rwc\\_alarm\\_enable](#)  
Разрешение прерывания ALARM.
- enum [rwc\\_wkup\\_enable](#)  
Разрешение работы входа WKUP.
- enum [rwc\\_shutdown\\_force](#)  
Принудительный переход в SHUTDOWN.
- enum [rwc\\_freq\\_serial](#)  
Значения делителей частоты внутреннего интерфейса

## Функции

- enum [rwc\\_status](#) [RWC\\_SetSecondsTimerMatch](#) (RWC\_Type \*base, uint32\_t match\_value)  
Устанавливает время будильника в секундах
- uint32\_t [RWC\\_GetSecondsTimerMatch](#) (RWC\_Type \*base)  
Получает актуальное время срабатывания будильника в секундах
- enum [rwc\\_status](#) [RWC\\_SetSecondsTimerCount](#) (RWC\_Type \*base, uint32\_t count\_value)  
Устанавливает текущее время в секундах
- enum [rwc\\_status](#) [RWC\\_GetSecondsTimerCount](#) (RWC\_Type \*base, uint32\_t \*sec)  
Получение текущее время в секундах
- enum [rwc\\_status](#) [RWC\\_GetDefaultConfig](#) (struct [rwc\\_config](#) \*config)  
Получение конфигурации таймера по умолчанию

- enum `rw_status RWC_SetLFEBypass` (`RWC_Type *base`, enum `rw_lfe_bypass` value)  
Выбор режима работы осциллятора LFE.
- enum `rw_status RWC_SetLFx` (`RWC_Type *base`, enum `rw_rtclk_type` value)  
Выбор генератора LFE или LFI.
- enum `rw_status RWC_GetLastAPIStatus` ()  
Получение статуса выполнения функции, тип результата которой отличен от enum `rw_status`.
- enum `rw_status RWC_SetResetCtrl` (`RWC_Type *base`, enum `rw_reset_type` value)  
Управление сбросом регистров при сигнале на входе SRSTn.

#### Функции чтения/записи внутренних регистров

- enum `rw_status RWC_GetInternalRegister` (`RWC_Type *base`, enum `rw_internal_register` reg, union `rw_union_reg` \*value)  
Чтение значения из внутреннего регистра RWC.
- enum `rw_status RWC_SetInternalRegister` (`RWC_Type *base`, enum `rw_internal_register` reg, union `rw_union_reg` value)  
Запись значения во внутренний регистр RWC.

#### Функции для работы с частотой тактирования

- enum `rw_status RWC_GetRTCClkParam` (`uint32_t *div`, enum `rw_rtclk_type` \*src)  
Чтение делителя и источника частоты RTCCLK.
- enum `rw_status RWC_SetRTCClkParam` (`uint32_t div`, enum `rw_rtclk_type` src)  
Запись делителя и источника частоты RTCCLK.
- `uint32_t RWC_GetTime` (`RWC_Type *base`)  
Чтение значения счетчика реального времени

#### Инициализация и деинициализация

- enum `rw_status RWC_Init` (`RWC_Type *base`, struct `rw_config` cfg)  
Инициализирует таймер реального времени
- enum `rw_status RWC_Deinit` (`RWC_Type *base`)  
Деинициализирует таймер реального времени

#### Текущее время и предупреждение

- enum `rw_status RWC_SetDatetime` (`RWC_Type *base`, const `rtc_datetime_t` \*datetime)  
Устанавливает текущее время и дату согласно заданной структуре
- enum `rw_status RWC_GetDatetime` (`RWC_Type *base`, `rtc_datetime_t` \*datetime)  
Получает текущее дату/время и сохраняет в указанную структуру
- enum `rw_status RWC_SetAlarm` (`RWC_Type *base`, const `rtc_datetime_t` \*alarmTime)  
Устанавливает время будильника
- enum `rw_status RWC_GetAlarm` (`RWC_Type *base`, `rtc_datetime_t` \*datetime)  
Возвращает время будильника

## Interrupt Interface

- enum `rcw_status` `RCW_EnableWakeUpTimerInterruptFromDPD` (`RCW_Type *base`, `bool enable`)  
Разрешение прерывания по сигналу wake-up из режима глубокого отключения питания
- enum `rcw_status` `RCW_EnableAlarmTimerInterruptFromDPD` (`RCW_Type *base`, `bool enable`)  
Разрешение прерывания по сигналу будильника из режима глубокого отключения питания
- enum `rcw_status` `RCW_InterruptClear` (`RCW_Type *base`)  
Сброс прерывания `RCW_ALARM`.

## Status Interface

- uint32\_t `RCW_GetStatusFlags` (`RCW_Type *base`)  
Получение статусов таймера реального времени

## Функции работы с внешним сигналом пробуждения (wake-up)

- enum `rcw_status` `RCW_SetWakeUpEnable` (`RCW_Type *base`, `bool enable`)  
Разрешение работы входа `wake_up`.
- enum `rcw_status` `RCW_SetWakeUpActiveLevel` (`RCW_Type *base`, enum `rcw_wake_up_polarity` value)  
Установка полярности сигнала `wake_up`.

## Макросы для операций перевода времени

- #define `SECONDS_IN_A_DAY` (86400U)
- #define `SECONDS_IN_A_HOUR` (3600U)
- #define `SECONDS_IN_A_MINUTE` (60U)
- #define `DAYS_IN_A_YEAR` (365U)
- #define `YEAR_RANGE_START` (1970U)
- #define `YEAR_RANGE_END` (2099U)

### 4.17.1 Подробное описание

#### Драйвер счетчика реального времени и Wake-контроллера

##### Заметки

Драйвер модуля `RCW` управляет счетчиком реального времени и Wake-контроллером.

### 4.17.2 Макросы

#### 4.17.2.1 `DAYS_IN_A_YEAR`

```
#define DAYS_IN_A_YEAR (365U)
```

Дней в одном году



## 4.17.2.2 RWC\_SYNC\_RETRY\_TIMES

```
#define RWC_SYNC_RETRY_TIMES 0U
```

Количество циклов ожидания

Заметки

0 - максимальное ожидание. Если необходимо бесконечное ожидание, прокомментируйте этот макрос

## 4.17.2.3 SECONDS\_IN\_A\_DAY

```
#define SECONDS_IN_A_DAY (86400U)
```

Секунд в одном дне

## 4.17.2.4 SECONDS\_IN\_A\_HOUR

```
#define SECONDS_IN_A_HOUR (3600U)
```

Секунд в одном часе

## 4.17.2.5 SECONDS\_IN\_A\_MINUTE

```
#define SECONDS_IN_A_MINUTE (60U)
```

Секунд в одной минуте

## 4.17.2.6 YEAR\_RANGE\_END

```
#define YEAR_RANGE_END (2099U)
```

Максимально допустимый год

## 4.17.2.7 YEAR\_RANGE\_START

```
#define YEAR_RANGE_START (1970U)
```

Начальный год

## 4.17.3 Перечисления

## 4.17.3.1 rwc\_alarm\_enable

```
enum rwc\_alarm\_enable
```

Разрешение прерывания ALARM.

Элементы перечислений

RWC_AlarmDisable	Запретить прерывание ALARM
RWC_AlarmEnable	Разрешить прерывание ALARM

#### 4.17.3.2 rwc\_cmd

enum [rwc\\_cmd](#)

Команды доступа к внутренним регистрам

Элементы перечислений

RWC_CmdWait	Признак завершения выполнения команды, записывать нельзя
RWC_CmdWrite	Команда записи регистра
RWC_CmdRead	Команда чтения регистра

#### 4.17.3.3 rwc\_freq\_serial

enum [rwc\\_freq\\_serial](#)

Значения делителей частоты внутреннего интерфейса

Элементы перечислений

RWC_FS8MHz	8 МГц
RWC_FS4MHz	4 МГц
RWC_FS2MHz	2 МГц
RWC_FS1MHz	1 МГц
RWC_FS500kHz	500 кГц
RWC_FS250kHz	250 кГц
RWC_FS125kHz	125 кГц
RWC_FS625hHz	62,5 кГц

#### 4.17.3.4 rwc\_internal\_register

enum [rwc\\_internal\\_register](#)

Внутренние регистры с батарейным питанием

Элементы перечислений

RWC_TrimLoad	Регистр записи значения подстройки из регистра TRIM
RWC_Time	Регистр текущего значения счетчика времени
RWC_Alarm	Регистр времени пробуждения
RWC_Trim	Регистр подстройки осцилляторов
RWC_Config	Конфигурационный регистр
RWC_GeneralReg	Регистр общего назначения
RWC_WakeConfig	Регистр настройки контроллера пробуждения

## 4.17.3.5 rwc\_lfe\_bypass

enum [rwc\\_lfe\\_bypass](#)

Режимы работы осциллятора LFE.

Элементы перечислений

RWC_QuartzResonator	Режим осциллятора совместно с внешним кварцевым резонатором
RWC_CMOSSignal	Режим буфера для внешнего КМОП сигнала на выводе XT132

## 4.17.3.6 rwc\_reset\_type

enum [rwc\\_reset\\_type](#)

Виды сбросов внутренних регистров при сигнале SRSTn.

Элементы перечислений

RWC_ResetOnlyWakeConfigAndConfig	Сбрасываются только регистры WakeConfig и Config
RWC_ResetAllExpectedGeneralReg	Сбрасываются все регистры кроме GeneralReg

## 4.17.3.7 rwc\_rtclk\_divisor

enum [rwc\\_rtclk\\_divisor](#)

Делители частоты RTCCLK.

Элементы перечислений

RWC_Div1	Делитель LFI на 1
RWC_Div2	Делитель LFI на 2
RWC_Div4	Делитель LFI на 4
RWC_Div8	Делитель LFI на 8
RWC_Div16	Делитель LFI на 16
RWC_Div32	Делитель LFI на 32
RWC_Div64	Делитель LFI на 64
RWC_Div128	Делитель LFI на 128
RWC_Div256	Делитель LFI на 256
RWC_Div512	Делитель LFI на 512
RWC_Div1024	Делитель LFI на 1024
RWC_Div2048	Делитель LFI на 2048
RWC_Div4096	Делитель LFI на 4096
RWC_Div8192	Делитель LFI на 8192
RWC_Div16384	Делитель LFI на 16384
RWC_Div32768	Делитель LFI на 32768
RWC_DivMax	Максимальный делитель

## 4.17.3.8 rwc\_rtcclk\_type

enum [rwc\\_rtcclk\\_type](#)

Источник частоты, подаваемой на RTCCLK.

Элементы перечислений

RWC_RTCClkTypeLFI	RTCClk переключена на LFI
RWC_RTCClkTypeLFE	RTCClk переключена на LFE

## 4.17.3.9 rwc\_shutdown\_force

enum [rwc\\_shutdown\\_force](#)

Принудительный переход в SHUTDOWN.

Элементы перечислений

RWC_ShutdownNoSet	Не переходить в SHUTDOWN
RWC_ShutdownSet	Переходить в SHUTDOWN

## 4.17.3.10 rwc\_status

enum [rwc\\_status](#)

Статусы драйвера CLKCTR.

Элементы перечислений

RWC_Status_Ok	Нет ошибок
RWC_Status_InvalidArgument	Недопустимый параметр
RWC_Status_CheckError	Получена ошибка от оборудования
RWC_Status_VerifyError	Верификация не прошла
RWC_Status_ConfigureError	Недопустимая конфигурация или ошибка в описании оборудования
RWC_Status_HardwareBusy	Оборудование не готово
RWC_Status_Timeout	Тайм-аут

## 4.17.3.11 rwc\_time\_clk\_sel

enum [rwc\\_time\\_clk\\_sel](#)

Выбор сигнала для тактирования счетчика времени

Элементы перечислений

RWC_TimeClock32kHz	32 кГц (RTCCLK)
RWC_TimeClock1Hz	1 Гц

#### 4.17.3.12 rwc\_timer\_status

enum [rwc\\_timer\\_status](#)

Статусы таймера

Элементы перечислений

RWC_WakeStat3	Бит устанавливается после первичной подачи питания на RWC. Бит сбрасывается при переходе в режим SHUTDOWN.
RWC_WakeStat2	Бит устанавливается при выходе из режима SHUTDOWN по внешнему событию WKUP либо по сбросу SRSTn. Бит сбрасывается при переходе в режим SHUTDOWN.
RWC_WakeStat1	Бит устанавливается при выходе из режима SHUTDOWN. Бит сбрасывается при переходе в режим SHUTDOWN.

#### 4.17.3.13 rwc\_wake\_up\_irq\_enable

enum [rwc\\_wake\\_up\\_irq\\_enable](#)

Разрешение прерывания RWC\_WKUP.

Элементы перечислений

RWC_IRQWkUpDisable	Запрещено
RWC_IRQWkUpEnable	Разрешено

#### 4.17.3.14 rwc\_wake\_up\_polarity

enum [rwc\\_wake\\_up\\_polarity](#)

Уровень активного сигнала WKUP для генерирования прерывания

Элементы перечислений

RWC_WkUpPolarityHigh	Высокий уровень
RWC_WkUpPolarityLow	Низкий уровень

#### 4.17.3.15 rwc\_wkup\_enable

enum [rwc\\_wkup\\_enable](#)

Разрешение работы входа WKUP.

Элементы перечислений

RWC_WkUpDisable	Прерывание WKUP запрещено
RWC_WkUpEnable	Прерывание WKUP разрешено

#### 4.17.4 Функции

##### 4.17.4.1 RWC\_Deinit()

```
enum rwc\_status RWC_Deinit (
    RWC_Type * base)
```

Деинициализирует таймер реального времени

Заметки

Сбрасывает регистры RWC\_Config и RWC\_WakeConfig таймера к значениям по умолчанию

Аргументы

base	Таймер
------	--------

Возвращаемые значения

<a href="#">RWC_Status_Ok</a>	
<a href="#">RWC_Status_InvalidArgument</a>	
<a href="#">RWC_Status_CheckError</a>	

##### 4.17.4.2 RWC\_EnableAlarmTimerInterruptFromDPD()

```
enum rwc\_status RWC_EnableAlarmTimerInterruptFromDPD (
    RWC_Type * base,
    bool enable)
```

Разрешение прерывания по сигналу будильника из режима глубокого отключения питания

Аргументы

base	Таймер
enable	Управление разрешением. <ul style="list-style-type: none"> <li>• true: разрешено.</li> <li>• false: запрещено.</li> </ul>

Возвращаемые значения

<a href="#">RWC_Status_Ok</a>	
<a href="#">RWC_Status_InvalidArgument</a>	
<a href="#">RWC_Status_CheckError</a>	

#### 4.17.4.3 RWC\_EnableWakeUpTimerInterruptFromDPD()

```
enum rwc\_status RWC_EnableWakeUpTimerInterruptFromDPD (
    RWC_Type * base,
    bool enable)
```

Разрешение прерывания по сигналу wake-up из режима глубокого отключения питания

Аргументы

base	Таймер
enable	Управление разрешением. <ul style="list-style-type: none"> <li>• true: разрешено.</li> <li>• false: запрещено.</li> </ul>

Возвращаемые значения

<a href="#">RWC_Status_Ok</a>	
<a href="#">RWC_Status_InvalidArgument</a>	
<a href="#">RWC_Status_CheckError</a>	

#### 4.17.4.4 RWC\_GetAlarm()

```
enum rwc\_status RWC_GetAlarm (
    RWC_Type * base,
    rtc\_datetime\_t * datetime)
```

Возвращает время будильника

Аргументы

base	Таймер
datetime	Структура дата/время

Возвращаемые значения

<a href="#">RWC_Status_Ok</a>	
<a href="#">RWC_Status_InvalidArgument</a>	
<a href="#">RWC_Status_CheckError</a>	

#### 4.17.4.5 RWC\_GetDatetime()

```
enum rwc\_status RWC_GetDatetime (  
    RWC_Type * base,  
    rtc\_datetime\_t * datetime)
```

Получает текущее дату/время и сохраняет в указанную структуру

Аргументы

base	Таймер
datetime	Структура дата/время

Возвращает

[RWC\\_Status\\_Ok](#)  
[RWC\\_Status\\_InvalidArgument](#)

#### 4.17.4.6 RWC\_GetDefaultConfig()

```
enum rwc\_status RWC_GetDefaultConfig (  
    struct rwc\_config * config)
```

Получение конфигурации таймера по умолчанию

Аргументы

config	Конфигурация
--------	--------------

Возвращаемые значения

<a href="#">RWC_Status_Ok</a>	
<a href="#">RWC_Status_InvalidArgument</a>	

#### 4.17.4.7 RWC\_GetInternalRegister()

```
enum rwc\_status RWC_GetInternalRegister (  
    RWC_Type * base,  
    enum rwc\_internal\_register reg,  
    union rwc\_union\_reg * value)
```

Чтение значения из внутреннего регистра RWC.

Заметки

Функцию не рекомендуется использовать без понимания функционирования блока



## Аргументы

base	Таймер
reg	Внутренний регистр
value	Значение регистра

## Возвращаемые значения

RWC_Status_Ok	
RWC_Status_InvalidArgument	
RWC_Status_Timeout	

## 4.17.4.8 RWC\_GetLastAPIStatus()

```
enum rwc_status RWC_GetLastAPIStatus ()
```

Получение статуса выполнения функции, тип результата которой отличен от enum rwc\_status.

## Возвращаемые значения

RWC_Status_Ok	
RWC_Status_InvalidArgument	
RWC_Status_CheckError	

## 4.17.4.9 RWC\_GetRTCClkParam()

```
enum rwc_status RWC_GetRTCClkParam (
    uint32_t * div,
    enum rwc_rtclk_type * src)
```

Чтение делителя и источника частоты RTCCLK.

## Заметки

Функцию не рекомендуется использовать без понимания функционирования блока

## Аргументы

div	Делитель частоты
src	Источник частоты

## Возвращаемые значения

RWC_Status_Ok	
RWC_Status_InvalidArgument	
RWC_Status_CheckError	

## 4.17.4.10 RWC\_GetSecondsTimerCount()

```
enum rwc\_status RWC_GetSecondsTimerCount (
    RWC_Type * base,
    uint32_t * sec)
```

Получение текущее время в секундах

Аргументы

base	Таймер
sec	Считанное значение секунд

Возвращаемые значения

<a href="#">RWC_Status_Ok</a>	
<a href="#">RWC_Status_InvalidArgument</a>	

## 4.17.4.11 RWC\_GetSecondsTimerMatch()

```
uint32_t RWC_GetSecondsTimerMatch (
    RWC_Type * base)
```

Получает актуальное время срабатывания будильника в секундах

Аргументы

base	Таймер
------	--------

Возвращает

Время будильника в секундах

## 4.17.4.12 RWC\_GetStatusFlags()

```
uint32_t RWC_GetStatusFlags (
    RWC_Type * base)
```

Получение статусов таймера реального времени

Аргументы

base	Таймер
------	--------

Возвращает

Объединение статусов `rwc_timer_status`

## 4.17.4.13 RWC\_GetTime()

```
uint32_t RWC_GetTime (
    RWC_Type * base)
```

Чтение значения счетчика реального времени

## Аргументы

base	Таймер
------	--------

## Возвращаемые значения

Значение	регистра времени TIME
----------	-----------------------

## 4.17.4.14 RWC\_Init()

```
enum rwc\_status RWC_Init (
    RWC_Type * base,
    struct rwc\_config cfg)
```

Инициализирует таймер реального времени

## Заметки

Эту функцию следует вызывать перед использованием драйвера таймера реального времени.

## Аргументы

base	Таймер
cfg	Конфигурация

## Возвращаемые значения

<a href="#">RWC_Status_Ok</a>	
<a href="#">RWC_Status_InvalidArgument</a>	
<a href="#">RWC_Status_CheckError</a>	

## 4.17.4.15 RWC\_InterruptClear()

```
enum rwc\_status RWC_InterruptClear (
    RWC_Type * base)
```

Сброс прерывания RWC\_ALARM.

## Аргументы

base	Таймер
------	--------

## Возвращаемые значения

<a href="#">RWC_Status_Ok</a>	
<a href="#">RWC_Status_InvalidArgument</a>	

## 4.17.4.16 RWC\_SetAlarm()

```
enum rwc\_status RWC_SetAlarm (  
    RWC_Type * base,  
    const rtc\_datetime\_t * alarmTime)
```

Устанавливает время будильника

Заметки

Функция проверяет, превышает ли указанное время срабатывания будильника текущее время. Если нет, то функция не устанавливает сигнал будильника и возвращает сообщение об ошибке.

Аргументы

base	Таймер
alarmTime	Структура дата/время

Возвращаемые значения

<a href="#">RWC_Status_Ok</a>	
<a href="#">RWC_Status_InvalidArgument</a>	
<a href="#">RWC_Status_ConfigureError</a>	

## 4.17.4.17 RWC\_SetDatetime()

```
enum rwc\_status RWC_SetDatetime (  
    RWC_Type * base,  
    const rtc\_datetime\_t * datetime)
```

Устанавливает текущее время и дату согласно заданной структуре

Заметки

Процедура может приводить к зависанию.

Аргументы

base	Таймер
datetime	Структура дата/время

Возвращает

[RWC\\_Status\\_Ok](#)  
[RWC\\_Status\\_InvalidArgument](#)  
[RWC\\_Status\\_Timeout](#)

## 4.17.4.18 RWC\_SetInternalRegister()

```
enum rwc_status RWC_SetInternalRegister (
    RWC_Type * base,
    enum rwc_internal_register reg,
    union rwc_union_reg value)
```

Запись значения во внутренний регистр RWC.

Заметки

Функцию не рекомендуется использовать без понимания функционирования блока

Аргументы

base	Таймер
reg	Внутренний регистр
value	Записываемое значение

Возвращаемые значения

RWC_Status_Ok	
RWC_Status_InvalidArgument	
RWC_Status_Timeout	

## 4.17.4.19 RWC\_SetLFEBypass()

```
enum rwc_status RWC_SetLFEBypass (
    RWC_Type * base,
    enum rwc_lfe_bypass value)
```

Выбор режима работы осциллятора LFE.

Аргументы

base	Базовый адрес
value	Режим работы осциллятора LFE

Возвращаемые значения

RWC_Status_Ok	
RWC_Status_InvalidArgument	
RWC_Status_CheckError	

## 4.17.4.20 RWC\_SetLFx()

```
enum rwc_status RWC_SetLFx (
    RWC_Type * base,
    enum rwc_rtclk_type value)
```

Выбор генератора LFE или LFI.

## Аргументы

base	Базовый адрес
value	Генератор

## Возвращаемые значения

<a href="#">RWC_Status_Ok</a>	
<a href="#">RWC_Status_InvalidArgument</a>	
<a href="#">RWC_Status_CheckError</a>	

## 4.17.4.21 RWC\_SetResetCtrl()

```
enum rwc\_status RWC_SetResetCtrl (
    RWC_Type * base,
    enum rwc\_reset\_type value)
```

Управление сбросом регистров при сигнале на входе SRSTn.

## Аргументы

base	Базовый адрес
value	Вид сброса

## Возвращаемые значения

<a href="#">RWC_Status_Ok</a>	
<a href="#">RWC_Status_InvalidArgument</a>	
<a href="#">RWC_Status_CheckError</a>	

## 4.17.4.22 RWC\_SetRTCClkParam()

```
enum rwc\_status RWC_SetRTCClkParam (
    uint32_t div,
    enum rwc\_rtclk\_type src)
```

Запись делителя и источника частоты RTCCCLK.

## Заметки

Допустимые значения для делителя частоты - степени 2 от  $2^0$  до  $2^{\text{RWC\_DivMax}}$ .

Функцию не рекомендуется использовать без понимания функционирования блока

## Аргументы

div	Делитель частоты
src	Источник частоты

Возвращаемые значения

<a href="#">RWC_Status_Ok</a>	
<a href="#">RWC_Status_InvalidArgument</a>	
<a href="#">RWC_Status_CheckError</a>	

#### 4.17.4.23 RWC\_SetSecondsTimerCount()

```
enum rwc\_status RWC_SetSecondsTimerCount (
    RWC_Type * base,
    uint32_t count_value)
```

Устанавливает текущее время в секундах

Заметки

Время устанавливается до тех пор, пока не будет установлено либо не будет превышено максимальное число попыток записи.

Аргументы

base	Таймер
countValue	Время в секундах

Возвращаемые значения

<a href="#">RWC_Status_Ok</a>	
<a href="#">RWC_Status_InvalidArgument</a>	
<a href="#">RWC_Status_Timeout</a>	

#### 4.17.4.24 RWC\_SetSecondsTimerMatch()

```
enum rwc\_status RWC_SetSecondsTimerMatch (
    RWC_Type * base,
    uint32_t match_value)
```

Устанавливает время будильника в секундах

Аргументы

base	Таймер
matchValue	Время будильника в секундах

Возвращаемые значения

<a href="#">RWC_Status_Ok</a>	
<a href="#">RWC_Status_InvalidArgument</a>	
<a href="#">RWC_Status_CheckError</a>	

## 4.17.4.25 RWC\_SetWakeUpActiveLewel()

```
enum rwc\_status RWC_SetWakeUpActiveLewel (
    RWC_Type * base,
    enum rwc\_wake\_up\_polarity value)
```

Установка полярности сигнала wake\_up.

Аргументы

base	Базовый адрес
value	Полярность сигнала

Возвращаемые значения

<a href="#">RWC_Status_Ok</a>	
<a href="#">RWC_Status_InvalidArgument</a>	
<a href="#">RWC_Status_CheckError</a>	

## 4.17.4.26 RWC\_SetWakeUpEnable()

```
enum rwc\_status RWC_SetWakeUpEnable (
    RWC_Type * base,
    bool enable)
```

Разрешение работы входа wake\_up.

Аргументы

base	Базовый адрес
enable	Разрешение

Возвращаемые значения

<a href="#">RWC_Status_Ok</a>	
<a href="#">RWC_Status_InvalidArgument</a>	
<a href="#">RWC_Status_CheckError</a>	

## 4.18 Драйвер модуля SDMMC

Драйвер SDMMC контроллера SD и MMC карт

Файлы

- файл [hal\\_sdmmc.h](#)  
Интерфейс драйвера модуля SDMMC.
- файл [hal\\_sdmmc\\_cfg.h](#)  
Интерфейс конфигурации модуля SDMMC.



## Структуры данных

- struct `sdmmc_card_t`  
Контекст драйвера контроллера SDMMC.
- struct `sdmmc_cfg_pin_map`  
Конфигурация выводов контроллера SDMMC.
- struct `sdmmc_port_cfg_t`  
Конфигурация контроллера SDMMC.

## Макросы

- `#define SDMMC_CALC_DIVIDER(input_freq_hz, required_freq_hz) (((input_freq_hz) / (required_freq_hz)) / 2)`  
Формула подсчета делителя выходной частоты контроллера SDMMC.

## Перечисления

- enum `sdmmc_status_t`  
Статусы драйвера SDMMC.
- enum `sdmmc_voltage_t`  
Рабочие напряжения питания карты

## Функции

- `sdmmc_status_t SDMMC_InitCard (sdmmc_card_t *sd, uint32_t num, sdmmc_voltage_t vol, void *init_buf)`  
Инициализация SDMMC контроллера и вставленной карты SD или MMC.
- `void SDMMC_DisableCard (sdmmc_card_t *sd)`  
Остановка SDMMC контроллера и выключение питания вставленной карты
- `sdmmc_status_t SDMMC_CalcMemorySpace (sdmmc_card_t *sd, void *sector_buf, bool unsafe)`  
Подсчет размера пространства памяти карты
- `sdmmc_status_t SDMMC_Read (sdmmc_card_t *sd, uint32_t start_block, void *data, uint32_t nblocks)`  
Чтение карты блоками размером 512 байт
- `sdmmc_status_t SDMMC_ReadAsync (sdmmc_card_t *sd, uint32_t start_block, void *data, uint32_t nblocks)`  
Асинхронное чтение карты блоками размером 512 байт
- `sdmmc_status_t SDMMC_ReadWait (sdmmc_card_t *sd)`  
Ожидание завершения операции асинхронного чтения памяти карты
- `sdmmc_status_t SDMMC_Write (sdmmc_card_t *sd, uint32_t start_block, const void *data, uint32_t nblocks)`  
Запись карты блоками размером 512 байт
- `sdmmc_status_t SDMMC_WriteAsync (sdmmc_card_t *sd, uint32_t start_block, const void *data, uint32_t nblocks)`  
Асинхронная запись карты блоками размером 512 байт
- `sdmmc_status_t SDMMC_WriteWait (sdmmc_card_t *sd)`  
Ожидание завершения операции асинхронной записи памяти карты

Количество слотов под карты и их типы

- enum
- #define [SDMMC\\_IS\\_MMC\(x\)](#) ((x)->type == [SDMMC\\_TypeMMC](#))
- #define [SDMMC\\_IS\\_SD\(x\)](#) ((x)->type == [SDMMC\\_TypeSD](#))

Рабочие напряжения контроллера SDMMC

- enum

Направления передачи SDMA канала контроллера SDMMC

- enum

Типы и размеры ответов карты

- enum

Ширина шины данных карты в битах

- enum

Константы контроллера SDMMC

- #define [SDMMC\\_SDHC\\_SECTOR\\_SIZE](#) 512
- #define [SDMMC\\_SD\\_SEND\\_IF\\_COND\\_PATTERN](#) 0x1AA
- #define [SDMMC\\_SD\\_OCR\\_INIT\\_VALUE](#) 0xFF80
- #define [SDMMC\\_MMC\\_RCA\\_ADDR](#) 0x00010000
- #define [SDMMC\\_TIMEOUTCONTROL\\_MAX\\_VALUE](#) 0xE

Типы слота карты контроллера SDMMC

- #define [SDMMC\\_CORECFG0\\_SLOTTYPE\\_REMOVABLE](#) 0
- #define [SDMMC\\_CORECFG0\\_SLOTTYPE\\_EMBEDDED](#) 1
- #define [SDMMC\\_CORECFG0\\_SLOTTYPE\\_SHARED\\_BUS](#) 2

Режимы UHS-I карты SD

- #define [SDMMC\\_SD\\_UHS\\_MODE\\_DEFAULT\\_SDR12](#) 0
- #define [SDMMC\\_SD\\_UHS\\_MODE\\_HIGHSPEED\\_SDR25](#) 1
- #define [SDMMC\\_SD\\_UHS\\_MODE\\_SDR50](#) 2
- #define [SDMMC\\_SD\\_UHS\\_MODE\\_SDR104](#) 3
- #define [SDMMC\\_SD\\_UHS\\_MODE\\_DDR50](#) 4

## Выравнивание адреса буфера данных SDMA

Возможные варианты значений:

- 0 - 4KB (Detects A11 Carry out)
  - 1 - 8KB (Detects A12 Carry out)
  - 2 - 16KB (Detects A13 Carry out)
  - 3 - 32KB (Detects A14 Carry out)
  - 4 - 64KB (Detects A15 Carry out)
  - 5 - 128KB (Detects A16 Carry out)
  - 6 - 256KB (Detects A17 Carry out)
  - 7 - 512KB (Detects A18 Carry out)
- 
- `#define SDMMC_SDMA_ALIGN 0`
  - `#define SDMMC_SDMA_BLOCK_SIZE (4096 << SDMMC_SDMA_ALIGN)`
  - `#define SDMMC_SDMA_BLOCK_ALIGN __attribute__((aligned(SDMMC_SDMA_BLOCK_SIZE)))`
  - `#define SDMMC_SDMA_IS_BLOCK_ALIGN_ADDR(x) (((uint32_t) (x) & (SDMMC_SDMA_BLOCK_SIZE - 1)) == 0)`

### 4.18.1 Подробное описание

Драйвер SDMMC контроллера SD и MMC карт

Драйвер содержит функции инициализации карты, подсчета размера пространства памяти карты, синхронные и асинхронные операции чтения и записи карты.

### 4.18.2 Макросы

#### 4.18.2.1 SDMMC\_CALC\_DIVIDER

```
#define SDMMC_CALC_DIVIDER(
    input_freq_hz,
    required_freq_hz) (((input_freq_hz) / (required_freq_hz)) / 2)
```

Формула подсчета делителя выходной частоты контроллера SDMMC.

Аргументы

<code>input_freq_hz</code>	Входная частота контроллера SDMMC
<code>required_freq_hz</code>	Необходимая выходная частота контроллера SDMMC

Возвращает

Делитель частоты

#### 4.18.2.2 SDMMC\_CORECFG0\_SLOTTYPE\_EMBEDDED

```
#define SDMMC_CORECFG0_SLOTTYPE_EMBEDDED 1
```

Встроенная карта

#### 4.18.2.3 SDMMC\_CORECFG0\_SLOTTYPE\_REMOVABLE

```
#define SDMMC_CORECFG0_SLOTTYPE_REMOVABLE 0
```

Извлекаемая карта

#### 4.18.2.4 SDMMC\_CORECFG0\_SLOTTYPE\_SHARED\_BUS

```
#define SDMMC_CORECFG0_SLOTTYPE_SHARED_BUS 2
```

Разделяемая шина

#### 4.18.2.5 SDMMC\_IS\_MMC

```
#define SDMMC_IS_MMC(  
    x) ((x)->type == SDMMC_TypeMMC)
```

Проверка на тип MMC

#### 4.18.2.6 SDMMC\_IS\_SD

```
#define SDMMC_IS_SD(  
    x) ((x)->type == SDMMC_TypeSD)
```

Проверка на тип SD

#### 4.18.2.7 SDMMC\_MMC\_RCA\_ADDR

```
#define SDMMC_MMC_RCA_ADDR 0x00010000
```

Относительный адрес MMC карты

#### 4.18.2.8 SDMMC\_SD\_OCR\_INIT\_VALUE

```
#define SDMMC_SD_OCR_INIT_VALUE 0xFF80
```

Значение OCR регистра при инициализации

## 4.18.2.9 SDMMC\_SD\_SEND\_IF\_COND\_PATTERN

```
#define SDMMC_SD_SEND_IF_COND_PATTERN 0x1AA
```

Начальный паттерн инициализации SD карты

## 4.18.2.10 SDMMC\_SD\_UHS\_MODE\_DDR50

```
#define SDMMC_SD_UHS_MODE_DDR50 4
```

DDR50

## 4.18.2.11 SDMMC\_SD\_UHS\_MODE\_DEFAULT\_SDR12

```
#define SDMMC_SD_UHS_MODE_DEFAULT_SDR12 0
```

Default/SDR12

## 4.18.2.12 SDMMC\_SD\_UHS\_MODE\_HIGHSPEED\_SDR25

```
#define SDMMC_SD_UHS_MODE_HIGHSPEED_SDR25 1
```

HighSpeed/SDR25

## 4.18.2.13 SDMMC\_SD\_UHS\_MODE\_SDR104

```
#define SDMMC_SD_UHS_MODE_SDR104 3
```

SDR104

## 4.18.2.14 SDMMC\_SD\_UHS\_MODE\_SDR50

```
#define SDMMC_SD_UHS_MODE_SDR50 2
```

SDR50

## 4.18.2.15 SDMMC\_SDHC\_SECTOR\_SIZE

```
#define SDMMC_SDHC_SECTOR_SIZE 512
```

Размер сектора High Capacity карты

## 4.18.2.16 SDMMC\_SDMA\_ALIGN

```
#define SDMMC_SDMA_ALIGN 0
```

Выравнивание адреса

## 4.18.2.17 SDMMC\_SDMA\_BLOCK\_ALIGN

```
#define SDMMC_SDMA_BLOCK_ALIGN __attribute__((aligned(SDMMC_SDMA_BLOCK_SIZE)))
```

Атрибут GCC выравнивания по размеру блока SDMA

## 4.18.2.18 SDMMC\_SDMA\_BLOCK\_SIZE

```
#define SDMMC_SDMA_BLOCK_SIZE (4096 << SDMMC_SDMA_ALIGN)
```

Размер блока данных SDMA в байтах

## 4.18.2.19 SDMMC\_SDMA\_IS\_BLOCK\_ALIGN\_ADDR

```
#define SDMMC_SDMA_IS_BLOCK_ALIGN_ADDR(  
    x) (((uint32_t) (x) & (SDMMC_SDMA_BLOCK_SIZE - 1)) == 0)
```

Проверка выравнивания адреса по размеру блока данных SDMA

## 4.18.2.20 SDMMC\_TIMEOUTCONTROL\_MAX\_VALUE

```
#define SDMMC_TIMEOUTCONTROL_MAX_VALUE 0xE
```

Максимальное значение таймера ожидания сигнала на линиях DAT

## 4.18.3 Перечисления

## 4.18.3.1 anonymous enum

anonymous enum

Элементы перечислений

SDMMC_TypeMMC	Тип карты MMC
SDMMC_TypeSD	Тип карты SD

## 4.18.3.2 anonymous enum

anonymous enum

Элементы перечислений

SDMMC_HostPWR_3V3	3,3 вольт
SDMMC_HostPWR_3V0	3,0 вольт
SDMMC_HostPWR_1V8	1,8 вольт

## 4.18.3.3 anonymous enum

anonymous enum

Элементы перечислений

SDMMC_SDMA_TransferWrite	Запись
SDMMC_SDMA_TransferRead	Чтение

#### 4.18.3.4 anonymous enum

anonymous enum

Элементы перечислений

SDMMC_NoResponse	Без ответа
SDMMC_ResponseLength136	Длина ответа - 136 бит
SDMMC_ResponseLength48	Длина ответа - 48 бит
SDMMC_ResponseLength48_Check	Длина ответа - 48 бит с проверкой

#### 4.18.3.5 anonymous enum

anonymous enum

Элементы перечислений

SDMMC_DataBusTransferWidth_1Bit	1 бит
SDMMC_DataBusTransferWidth_4bit	4 бит
SDMMC_ExtDataBusTransferWidth_8bit	8 бит

#### 4.18.3.6 sdmmc\_status\_t

enum [sdmmc\\_status\\_t](#)

Статусы драйвера SDMMC.

Элементы перечислений

SDMMC_Status_Ok	Ошибок нет
SDMMC_Status_Err	Ошибка исполнения

#### 4.18.3.7 sdmmc\_voltage\_t

enum [sdmmc\\_voltage\\_t](#)

Рабочие напряжения питания карты

Элементы перечислений

SDMMC_3v3	3,3 В (для карт MMC и SD в режиме Default Speed или High Speed)
SDMMC_1v8	1,8 В (для карт MMC)
SDMMC_3v3To1v8	3,3 В с переключением на 1,8 В (для карт SD в режимах UHS-I)

## 4.18.4 Функции

### 4.18.4.1 SDMMC\_CalcMemorySpace()

```
sdmmc_status_t SDMMC_CalcMemorySpace (
    sdmmc_card_t * sd,
    void * sector_buf,
    bool unsafe)
```

Подсчет размера пространства памяти карты

Подсчет общего пространства памяти карты через регистры параметров, если подсчет по параметрам не удался, то может применяться итеративный метод определения размера, но с повреждением информации на карте. Подсчитанный размер записывается в поле `total_size` контекста `sd` в байтах.

Заметки

Буфер памяти для SDMA должен быть выровнен по границе блока SDMA и размером 512 байт.

Аргументы

<code>sd</code>	Контекст драйвера SDMMC
<code>sector_buf</code>	Буфер памяти для SDMA
<code>unsafe</code>	Использовать (1) или не использовать (0) небезопасные методы определения емкости карты

Возвращаемые значения

<code>SDMMC_Status_Ok</code>	
<code>SDMMC_Status_Err</code>	

### 4.18.4.2 SDMMC\_DisableCard()

```
void SDMMC_DisableCard (
    sdmmc_card_t * sd)
```

Остановка SDMMC контроллера и выключение питания вставленной карты



## Аргументы

sd	Контекст драйвера SDMMC
----	-------------------------

## 4.18.4.3 SDMMC\_InitCard()

```
sdmmc_status_t SDMMC_InitCard (
    sdmmc_card_t * sd,
    uint32_t num,
    sdmmc_voltage_t vol,
    void * init_buf)
```

Инициализация SDMMC контроллера и вставленной карты SD или MMC.

## Заметки

Буфер памяти для SDMA должен быть выровнен по границе блока SDMA и размером 512 байт.

## Аргументы

sd	Контекст драйвера SDMMC
num	Номер контроллера SDMMC
vol	Выбор напряжения питания карты
init_buf	Буфер памяти для SDMA

## Возвращаемые значения

SDMMC_Status_Ok	
SDMMC_Status_Err	

## 4.18.4.4 SDMMC\_Read()

```
sdmmc_status_t SDMMC_Read (
    sdmmc_card_t * sd,
    uint32_t start_block,
    void * data,
    uint32_t nblocks)
```

Чтение карты блоками размером 512 байт

## Заметки

Буфер памяти для SDMA должен быть выровнен по границе блока SDMA.

## Аргументы

sd	Контекст драйвера SDMMC
start_block	Номер первого блока памяти карты
data	Буфер памяти для SDMA
nblocks	Количество считываемых блоков памяти

Возвращаемые значения

<a href="#">SDMMC_Status_Ok</a>	
<a href="#">SDMMC_Status_Err</a>	

#### 4.18.4.5 SDMMC\_ReadAsync()

```
sdmmc_status_t SDMMC_ReadAsync (
    sdmmc_card_t * sd,
    uint32_t start_block,
    void * data,
    uint32_t nblocks)
```

Асинхронное чтение карты блоками размером 512 байт

Заметки

Буфер памяти для SDMA должен быть выровнен по границе блока SDMA.

Аргументы

sd	Контекст драйвера SDMMC
start_block	Номер первого блока памяти карты
data	Буфер памяти для SDMA
nblocks	Количество считываемых блоков памяти

Возвращаемые значения

<a href="#">SDMMC_Status_Ok</a>	
<a href="#">SDMMC_Status_Err</a>	

#### 4.18.4.6 SDMMC\_ReadWait()

```
sdmmc_status_t SDMMC_ReadWait (
    sdmmc_card_t * sd)
```

Ожидание завершения операции асинхронного чтения памяти карты

Аргументы

sd	Контекст драйвера SDMMC
----	-------------------------

Возвращаемые значения

<a href="#">SDMMC_Status_Ok</a>	
<a href="#">SDMMC_Status_Err</a>	

## 4.18.4.7 SDMMC\_Write()

```
sdmmc_status_t SDMMC_Write (
    sdmmc_card_t * sd,
    uint32_t start_block,
    const void * data,
    uint32_t nblocks)
```

Запись карты блоками размером 512 байт

Заметки

Буфер памяти для SDMA должен быть выровнен по границе блока SDMA.

Аргументы

sd	Контекст драйвера SDMMC
start_block	Номер первого блока памяти карты
data	Буфер памяти для SDMA
nblocks	Количество записываемых блоков памяти

Возвращаемые значения

SDMMC_Status_Ok	
SDMMC_Status_Err	

## 4.18.4.8 SDMMC\_WriteAsync()

```
sdmmc_status_t SDMMC_WriteAsync (
    sdmmc_card_t * sd,
    uint32_t start_block,
    const void * data,
    uint32_t nblocks)
```

Асинхронная запись карты блоками размером 512 байт

Заметки

Буфер памяти для SDMA должен быть выровнен по границе блока SDMA.

Аргументы

sd	Контекст драйвера SDMMC
start_block	Номер первого блока памяти карты
data	Буфер памяти для SDMA
nblocks	Количество записываемых блоков памяти

Возвращаемые значения

<a href="#">SDMMC_Status_Ok</a>	
<a href="#">SDMMC_Status_Err</a>	

#### 4.18.4.9 SDMMC\_WriteWait()

```
sdmmc_status_t SDMMC_WriteWait (
    sdmmc_card_t * sd)
```

Ожидание завершения операции асинхронной записи памяти карты

Аргументы

sd	Контекст драйвера SDMMC
----	-------------------------

Возвращаемые значения

<a href="#">SDMMC_Status_Ok</a>	
<a href="#">SDMMC_Status_Err</a>	

## 4.19 Драйвер модуля SMC

Драйвер внешней статической памяти

Файлы

- файл [hal\\_smc.h](#)  
Интерфейс драйвера внешней статической памяти

Макросы

- `#define SMC_NO_DEACTIVATION 0`  
Деактивация не выполняется

## Перечисления

- enum `smc_status`  
Статусы драйвера SMC.
- enum `smc_cmd_type`  
Тип конфигурационной команды
- enum `smc_cre`  
Значение выхода CRE при выполнении команды ModeReg.
- enum `smc_burst_align`  
Граница пакета памяти
- enum `smc_bls`  
Поведение выводов SMC\_NBLS.
- enum `smc_adv`  
Использование сигнала NADV.
- enum `smc_packet_lenght`  
Длина пакета данных в 16-битных словах при записи или чтении
- enum `smc_rd_wr_type`  
Тип интерфейса при записи или чтении
- enum `smc_bit_depth`  
Разрядность интерфейса памяти
- enum `smc_incr_to_incr4`  
Управление преобразованием AHB-пакетов типа INCR в пакеты типа INCR4.

## Интерфейс драйвера

- enum `smc_status SMC_PowerSaveOn` (SMC\_Type \*base)  
Включение энергосберегающего режима
- enum `smc_status SMC_PowerSaveOff` (SMC\_Type \*base)  
Выключение энергосберегающего режима
- enum `smc_status SMC_DirectCmd` (SMC\_Type \*base, uint32\_t chip\_select, enum `smc_cmd_type` cmd\_type, enum `smc_cre` set\_cre, uint32\_t addr)  
Отправка конфигурационных команд
- enum `smc_status SMC_SetCycles` (SMC\_Type \*base, uint32\_t ttr, uint32\_t tpc, uint32\_t twp, uint32\_t tceoe, uint32\_t twc, uint32\_t trc)  
Хранение новой конфигурации временных параметров интерфейса.
- enum `smc_status SMC_SetOpmode` (SMC\_Type \*base, enum `smc_burst_align` align, enum `smc_bls` bls, enum `smc_adv` adv, enum `smc_packet_lenght` wr\_lenght, enum `smc_rd_wr_type` wr\_sync, enum `smc_packet_lenght` rd\_lenght, enum `smc_rd_wr_type` rd\_sync, enum `smc_bit_depth` depth)  
Установка регистра SET\_OPMODE.
- enum `smc_status SMC_RefreshPeriod` (SMC\_Type \*base, uint32\_t period)  
Управление деактивацией микросхемы
- enum `smc_status SMC_UserConfig` (SMC\_Type \*base, enum `smc_incr_to_incr4` bank0, enum `smc_incr_to_incr4` bank1, uint32\_t smccldiv)  
Запись в регистр USER\_CONFIG.
- bool `SMC_CheckConfigure` (SMC\_Type \*base, uint32\_t chip\_select, uint32\_t cycles, uint32\_t mode)  
Проверка завершения установки конфигурации

### 4.19.1 Подробное описание

Драйвер внешней статической памяти

Драйвер модуля внешней статической памяти управляет внешней статической памяти .

### 4.19.2 Перечисления

#### 4.19.2.1 smc\_adv

enum [smc\\_adv](#)

Использование сигнала NADV.

Элементы перечислений

SMC_AdvNotUsed	Сигнал не используется.
SMC_AdvUsed	Сигнал используется, шины адреса и данных мультиплексируются.
SMC_AdvLcd	При работе с LCD дисплеем ADV должен быть сброшен.
SMC_AdvMemory	ADV должен быть установлен при работе с микросхемами памяти.

#### 4.19.2.2 smc\_bit\_depth

enum [smc\\_bit\\_depth](#)

Разрядность интерфейса памяти

Элементы перечислений

SMC_BitDepth8	8 бит - не поддерживается
SMC_BitDepth16	16 бит
SMC_BitDepth24	24 бит - не поддерживается
SMC_BitDepth32	32 бит - не поддерживается

#### 4.19.2.3 smc\_bls

enum [smc\\_bls](#)

Поведение выводов SMC\_NBLS.

Элементы перечислений

SMC_BlsAsNcs	Выводы SMC_NBLS переключаются так же, как выводы SMC_NCS.
SMC_BlsAsNwe	Выводы SMC_NBLS переключаются так же, как вывод SMC_NWE.

#### 4.19.2.4 smc\_burst\_align

enum [smc\\_burst\\_align](#)

Граница пакета памяти

Элементы перечислений

SMC_BurstAlignNo	Нет границы
SMC_BurstAlign32	32 слова по 16 бит
SMC_BurstAlign64	64 слова по 16 бит
SMC_BurstAlign128	128 слов по 16 бит
SMC_BurstAlign256	256 слов по 16 бит

#### 4.19.2.5 smc\_cmd\_type

enum [smc\\_cmd\\_type](#)

Тип конфигурационной команды

Элементы перечислений

SMC_CmdTypeUpdateRegsAndAHBCommand	UpdateRegs + AHB command
SMC_CmdTypeModeReg	ModeReg
SMC_CmdTypeUpdateRegs	UpdateRegs
SMC_CmdTypeModeRegAndUpdateRegs	ModeReg + UpdateRegs

#### 4.19.2.6 smc\_cre

enum [smc\\_cre](#)

Значение выхода CRE при выполнении команды ModeReg.

Элементы перечислений

SMC_CRE0	Выход CRE сброшен
SMC_CRE1	Выход CRE установлен

#### 4.19.2.7 smc\_incr\_to\_incr4

enum [smc\\_incr\\_to\\_incr4](#)

Управление преобразованием AHB-пакетов типа INCR в пакеты типа INCR4.

Элементы перечислений

SMC_IncrToIncr4Enable	Пакеты типа INCR преобразуются в пакеты типа INCR4.
SMC_IncrToIncr4Disable	Пакеты типа INCR обрабатываются как одиночные обращения типа SINGLE.

#### 4.19.2.8 smc\_packet\_lenght

enum [smc\\_packet\\_lenght](#)

Длина пакета данных в 16-битных словах при записи или чтении

Элементы перечислений

SMC_PacketLenght1	1 слово
SMC_PacketLenght4	4 слово
SMC_PacketLenght8	8 слово
SMC_PacketLenghtEndless	Непрерывный пакет

#### 4.19.2.9 smc\_rd\_wr\_type

enum [smc\\_rd\\_wr\\_type](#)

Тип интерфейса при записи или чтении

Элементы перечислений

SMC_RdWrAsync	Асинхронный
SMC_RdWrSync	Синхронный

#### 4.19.2.10 smc\_status

enum [smc\\_status](#)

Статусы драйвера SMC.

Элементы перечислений

SMC_Status_Ok	Нет ошибок
SMC_Status_InvalidArgument	Недопустимый аргумент

### 4.19.3 Функции

#### 4.19.3.1 SMC\_CheckConfigure()

```
bool SMC_CheckConfigure (
    SMC_Type * base,
    uint32_t chip_select,
    uint32_t cycles,
    uint32_t mode)
```

Проверка завершения установки конфигурации

Аргументы

base	Адрес блока SMC
chip_select	Банк памяти. Может быть 0 или 1.
cycles	Временные параметры
mode	Режима работы



Возвращаемые значения

true	Конфигурация установлена
false	Конфигурация не установлена

Заметки

При некорректном base возвращается false

#### 4.19.3.2 SMC\_DirectCmd()

```
enum smc_status SMC_DirectCmd (
    SMC_Type * base,
    uint32_t chip_select,
    enum smc_cmd_type cmd_type,
    enum smc_cre set_cre,
    uint32_t addr)
```

Отправка конфигурационных команд

Предназначена для отправки конфигурационных команд во внешнюю память и для управления обновлением конфигурационных регистров контроллера значениями из регистров SMC\_SET\_ ↔ OPMODE и SMC\_SET\_CYCLES

Аргументы

base	Адрес блока SMC
chip_select	Банк памяти. Может быть 0 или 1.
cmd_type	Тип конфигурационной команды.
set_cre	При выполнении команды ModeReg задает значение выхода SMC_CRE
addr	При выполнении команды ModeReg поле используется в качестве разрядов [19:0] адреса внешней памяти. При выполнении команды UpdateRegs+AHB command поле ADDR[15:0] используется для сопоставления со значением на шине SMC_DA[15:0] для синхронного обновления конфигурации.

Возвращаемые значения

SMC_Status_Ok	
SMC_Status_InvalidArgument	

#### 4.19.3.3 SMC\_PowerSaveOff()

```
enum smc_status SMC_PowerSaveOff (
    SMC_Type * base)
```

Выключение энергосберегающего режима

Иницирует запрос на переход контроллера из энергосберегающего состояния. Не дожидается завершения перехода из энергосберегающего состояния.

## Аргументы

base	Адрес блока SMC
------	-----------------

## Возвращаемые значения

<a href="#">SMC_Status_Ok</a>	
<a href="#">SMC_Status_InvalidArgument</a>	

## 4.19.3.4 SMC\_PowerSaveOn()

```
enum smc\_status SMC_PowerSaveOn (
    SMC_Type * base)
```

Включение энергосберегающего режима

Иницирует запрос на переход контроллера в энергосберегающее состояние. Не дожидается завершения перехода в энергосберегающее состояние.

## Аргументы

base	Адрес блока SMC
------	-----------------

## Возвращаемые значения

<a href="#">SMC_Status_Ok</a>	
<a href="#">SMC_Status_InvalidArgument</a>	

## 4.19.3.5 SMC\_RefreshPeriod()

```
enum smc\_status SMC_RefreshPeriod (
    SMC_Type * base,
    uint32_t period)
```

Управление деактивацией микросхемы

Задаёт количество последовательных пакетов перед деактивацией микросхемы памяти для возможности ее обновления

## Заметки

Используется перед началом работы с блоком

## Аргументы

base	Адрес блока SMC
period	Количество пакетов перед деактивацией. Допустимые значения: SMC_NO_DEACTIVATION - нет деактивации, от 1 до 15 - деактивация после 1, .. 15 пакета

Возвращаемые значения

<a href="#">SMC_Status_Ok</a>	
<a href="#">SMC_Status_InvalidArgument</a>	

#### 4.19.3.6 SMC\_SetCycles()

```
enum smc\_status SMC_SetCycles (
    SMC_Type * base,
    uint32_t ttr,
    uint32_t tpc,
    uint32_t twp,
    uint32_t tceoe,
    uint32_t twc,
    uint32_t trc)
```

Хранение новой конфигурации временных параметров интерфейса.

Предназначена для хранения новой конфигурации временных параметров интерфейса в регистре SMC\_SET\_CYCLES. Значение этого регистра переписывается в регистр SMC\_CYCLESh выбранного банка при выполнении команды UpdateRegs ( [SMC\\_DirectCmd](#) )

Аргументы

base	Адрес блока SMC
ttr	Задержка между последовательными пакетами (turnaround)
tpc	Длительность цикла доступа к странице
twp	Задержка активации вывода CMS_NWE
tceoe	Задержка активации вывода SMC_NOE
twc	Длительность цикла записи
trc	Длительность цикла чтения

Возвращаемые значения

<a href="#">SMC_Status_Ok</a>	
<a href="#">SMC_Status_InvalidArgument</a>	

#### 4.19.3.7 SMC\_SetOpmode()

```
enum smc\_status SMC_SetOpmode (
    SMC_Type * base,
    enum smc\_burst\_align align,
    enum smc\_bls bls,
    enum smc\_adv adv,
    enum smc\_packet\_lenght wr_lenght,
    enum smc\_rd\_wr\_type wr_sync,
    enum smc\_packet\_lenght rd_lenght,
    enum smc\_rd\_wr\_type rd_sync,
    enum smc\_bit\_depth depth)
```

Установка регистра SET\_OPMODE.

Устанавливает все поля регистра SET\_OPMODE. Если хотя бы одно поле имеет недопустимое значение, то записи в регистр не происходит.

## Заметки

Используется перед началом работы с блоком

## Аргументы

base	Адрес блока SMC
align	Выравнивание
bls	Бит задает поведение выводов SMC_NBLS[1:0]
adv	Задаёт использование сигнала SMC_NADV
wr_lenght	Задаёт длину пакета данных при записи
wr_sync	Задаёт тип интерфейса при записи
rd_lenght	Задаёт длину пакета данных при чтении
rd_sync	Задаёт тип интерфейса при чтении
depth	Задаёт разрядность интерфейса памяти

## Возвращаемые значения

<a href="#">SMC_Status_Ok</a>	
<a href="#">SMC_Status_InvalidArgument</a>	

## 4.19.3.8 SMC\_UserConfig()

```
enum smc\_status SMC_UserConfig (
    SMC_Type * base,
    enum smc\_incr\_to\_incr4 bank0,
    enum smc\_incr\_to\_incr4 bank1,
    uint32_t smcclkdiv)
```

Запись в регистр USER\_CONFIG.

Управляет преобразованием АHB-пакетов типа INCR в пакеты типа INCR4. Задаёт коэффициент деления для формирования тактового сигнала.

## Заметки

Используется перед началом работы с блоком

## Аргументы

base	Адрес блока SMC
bank0	Разрешение преобразования АHB-пакетов типа INCR в пакеты типа INCR4 для банка памяти 1.
bank1	Разрешение преобразования АHB-пакетов типа INCR в пакеты типа INCR4 для банка памяти 0.
smcclkdiv	Коэффициент деления SMCCLKDIV. Допустимые значения от 1 до 32

Возвращаемые значения

<a href="#">SMC_Status_Ok</a>	
<a href="#">SMC_Status_InvalidArgument</a>	

## 4.20 Драйвер модуля SPI

Последовательный синхронный стандарт передачи данных в режиме полного дуплекса, полудуплекса или симплекса

Файлы

- файл [hal\\_spi.h](#)  
Интерфейс драйвера модуля SPI.
- файл [hal\\_spi\\_dma.h](#)  
Дополнение драйвера SPI с пересылкой данных через DMA.

Структуры данных

- struct [spi\\_microwire\\_cfg\\_t](#)  
Конфигурация для протокола Microwire National Semiconductor.
- struct [spi\\_motorola\\_cfg\\_t](#)  
Конфигурация для протокола Motorola SPI.
- struct [spi\\_config\\_t](#)  
Структура конфигурации для Master SPI.
- struct [spi\\_transfer\\_t](#)  
Структура SPI для приемо-передачи
- struct [spi\\_half\\_duplex\\_transfer\\_t](#)  
Структура SPI для полудуплексной приемо-передачи в режиме Master.
- struct [spi\\_config\\_internal\\_t](#)  
Внутренняя структура конфигурации модуля SPI.
- struct [spi\\_handle](#)  
SPI структура дескриптора для работы по прерыванию
- struct [\\_spi\\_dma\\_handle](#)  
Дескриптор SPI-DMA.

Макросы

- `#define HAL_SPI_DRIVER_VERSION (MAKE\_VERSION(1, 1, 0))`  
Версия драйвера SPI.
- `#define SPI\_DUMMYDATA (0xA5U)`  
SPI фиктивные данные для передачи по умолчанию.
- `#define SPI\_RETRY\_TIMES 0U`
- `#define HAL_SPI_DMA_DRIVER_VERSION (MAKE\_VERSION(1, 0, 0))`  
Версия драйвера

## Определения типов

- typedef struct [spi\\_handle](#) spi\_master\_handle\_t  
Дескриптор Master SPI для работы по прерыванию
- typedef struct [spi\\_handle](#) spi\_slave\_handle\_t  
Дескриптор Slave SPI для работы по прерыванию
- typedef void(\* spi\_master\_callback\_t) (SPI\_Type \*base, [spi\\_master\\_handle\\_t](#) \*handle, uint32\_t status, void \*user\_data)  
Прототип пользовательской функции обратного вызова Master SPI для вызова по окончании обмена
- typedef void(\* spi\_slave\_callback\_t) (SPI\_Type \*base, [spi\\_slave\\_handle\\_t](#) \*handle, uint32\_t status, void \*user\_data)  
Прототип пользовательской функции обратного вызова Slave SPI для вызова по окончании обмена
- typedef struct [\\_spi\\_dma\\_handle](#) spi\_dma\_handle\_t  
Дескриптор SPI-DMA.
- typedef void(\* spi\_dma\_callback\_t) (SPI\_Type \*base, [spi\\_dma\\_handle\\_t](#) \*handle, void \*user\_data, [dma\\_irq\\_t](#) inttype)  
Функция обратного вызова

## Перечисления

- enum [spi\\_status](#)  
Статусы возврата из функций для драйвера SPI.
- enum [spi\\_shift\\_direction\\_t](#)  
Формат передачи данных (MSB или LSB)
- enum [spi\\_txfifo\\_watermark\\_t](#)  
Триггер уровня заполнения TxFIFO.
- enum [spi\\_rxfifo\\_watermark\\_t](#)  
Триггер уровня заполнения RxFIFO.
- enum [spi\\_frame\\_width\\_t](#)  
Размер кадра данных в 32-х битном режиме передачи данных (CTRLR0.DFS\_32)
- enum [spi\\_frame\\_format\\_t](#)  
Формат кадра передачи данных
- enum [spi\\_status\\_flags](#)  
SPI флаги статусов
- enum [spi\\_mode\\_t](#)  
Режим передачи (CTRLR0.TMOD)
- enum [spi\\_trans\\_status](#)  
Состояния приемо-передачи SPI.
- enum [spi\\_interrupt\\_enable](#)  
Источники прерываний SPI.

## Функции

- uint8\_t [SPI\\_GetInstance](#) (SPI\_Type \*base)  
Получение индекса модуля SPI.
- enum [spi\\_status](#) [SPI\\_MasterTransferCreateHandleDMA](#) (SPI\_Type \*base, [spi\\_dma\\_callback\\_t](#) callback, void \*user\_data, [spi\\_dma\\_handle\\_t](#) \*handle, [dma\\_handle\\_t](#) \*tx\_handle, [dma\\_handle\\_t](#) \*rx\_handle)  
Функция инициализации дескриптора SPI-DMA.

- enum `spi_status SPI_MasterTransferDMA` (`SPI_Type *base`, `spi_dma_handle_t *handle`, `spi_transfer_t *xfer`)  
Функция для SPI master приема/передачи данных в полнодуплексном режиме через DMA каналы в порт SPI.
- enum `spi_status SPI_MasterHalfDuplexTransferDMA` (`SPI_Type *base`, `spi_dma_handle_t *handle`, `spi_half_duplex_transfer_t *xfer`)  
Функция приема/передачи данных в полудуплексном режиме через DMA каналы в порт SPI.
- static enum `spi_status SPI_SlaveTransferCreateHandleDMA` (`SPI_Type *base`, `spi_dma_callback_t callback`, `void *user_data`, `spi_dma_handle_t *handle`, `dma_handle_t *tx_handle`, `dma_handle_t *rx_handle`)  
Функция инициализации дескриптора SPI-DMA.
- static enum `spi_status SPI_SlaveTransferDMA` (`SPI_Type *base`, `spi_dma_handle_t *handle`, `spi_transfer_t *xfer`)  
Функция для SPI slave приема/передачи данных в полнодуплексном режиме через DMA каналы в порт SPI.
- void `SPI_MasterTransferAbortDMA` (`spi_dma_handle_t *handle`)  
Прекращение передачи SPI.
- static void `SPI_SlaveTransferAbortDMA` (`spi_dma_handle_t *handle`)  
Прекращение передачи SPI.
- static void `SPI_DMADescriptorInitTX` (`SPI_Type *base`, `dma_descriptor_t *desc`, `uint32_t count`, `uint32_t data_size`, `uint8_t src_incr`, `uint32_t data_width`, `void *src_addr`)  
Инициализация дескрипторов DMA для многоблочной передачи TX.
- static void `SPI_DMADescriptorInitRX` (`SPI_Type *base`, `dma_descriptor_t *desc`, `uint32_t count`, `uint32_t data_size`, `uint8_t dst_incr`, `uint32_t data_width`, `void *dst_addr`)  
Инициализация дескрипторов DMA для многоблочной передачи RX.

## National Semiconductor Microwire

- enum `microwire_ctrlword_len_t`  
Выбор длины управляющего слова для формата кадра передачи данных National Semiconductor Microwire.
- enum `microwire_busy_ready_check_t`  
Включить/отключить проверку busy/ready флага (регистр SR) для формата кадра передачи данных National Semiconductor Microwire.
- enum `microwire_tx_rx_t`  
Направление передачи слова данных для формата кадра передачи данных National Semiconductor Microwire.
- enum `microwire_single_serial_t`  
Выбор типа передачи (одиночная или последовательная) для формата кадра передачи данных National Semiconductor Microwire.

## Motorola SPI

- enum `spi_motorola_clk_pol_t`  
Выбор полярности тактового сигнала при отсутствия передаваемых данных в режиме Master для формата кадра передачи данных Motorola SPI.
- enum `spi_motorola_cap_data_t`  
Выбор фронта для захвата данных для формата кадра передачи данных Motorola SPI.

## Инициализация и деинициализация

- void [SPI\\_MasterGetDefaultConfig](#) ([spi\\_config\\_t](#) \*config)  
Инициализация конфигурации SPI значениями по умолчанию
- enum [spi\\_status](#) [SPI\\_MasterInit](#) ([SPI\\_Type](#) \*base, const [spi\\_config\\_t](#) \*config, [uint32\\_t](#) src\_ ← clock\_hz)  
Инициализация SPI модуля как Master с заданной конфигурацией
- void [SPI\\_SlaveGetDefaultConfig](#) ([spi\\_config\\_t](#) \*config)  
Заполнение структуры конфигурации SPI Slave-устройства значениями по умолчанию
- enum [spi\\_status](#) [SPI\\_SlaveInit](#) ([SPI\\_Type](#) \*base, const [spi\\_config\\_t](#) \*config)  
Инициализация SPI заданной конфигурацией
- void [SPI\\_Deinit](#) ([SPI\\_Type](#) \*base)  
Деинициализация SPI.
- static void [SPI\\_Enable](#) ([SPI\\_Type](#) \*base, bool enable)  
Включение или отключение модуль SPI Master или Slave.
- static bool [SPI\\_IsEnable](#) ([SPI\\_Type](#) \*base)  
Получение статуса модуля SPI включено/выключено
- static void [SPI\\_Reset](#) ([SPI\\_Type](#) \*base)  
Останов всех операций SPI.

## Статусы

- static [uint32\\_t](#) [SPI\\_GetStatusFlags](#) ([SPI\\_Type](#) \*base)  
Получение флага состояния SPI.

## Прерывания

- static void [SPI\\_EnableInterrupts](#) ([SPI\\_Type](#) \*base, [uint32\\_t](#) irqs)  
Включение прерываний SPI.
- static void [SPI\\_DisableInterrupts](#) ([SPI\\_Type](#) \*base, [uint32\\_t](#) irqs)  
Отключение прерываний SPI.
- static [uint32\\_t](#) [SPI\\_CurrentStatusInterrupts](#) ([SPI\\_Type](#) \*base)  
Получение статуса прерываний
- static void [SPI\\_TakeDownInterrupts](#) ([SPI\\_Type](#) \*base, [uint32\\_t](#) irqs)  
Сброс флагов прерывания

## DMA управление

- void [SPI\\_EnableTxDMA](#) ([SPI\\_Type](#) \*base, bool enable)  
Включение или отключение запроса DMA от SPI TxFIFO.
- void [SPI\\_EnableRxDMA](#) ([SPI\\_Type](#) \*base, bool enable)  
Включение или отключение запроса DMA от SPI RxFIFO.



## Операции на шине SPI

- `spi_config_internal_t * SPI_GetConfig` (`SPI_Type *base`)  
Получение внутренней конфигурации
- `enum spi_status SPI_MasterSetBaud` (`SPI_Type *base`, `uint32_t baudrate_bps`, `uint32_t src_clock_hz`)  
Установка скорости передачи для SPI Master.
- `void SPI_WriteData` (`SPI_Type *base`, `uint32_t data`)  
Запись данных в регистр данных SPI.
- `uint32_t SPI_ReadData` (`SPI_Type *base`)  
Получение данных из регистра данных SPI.
- `void SPI_SetDummyData` (`SPI_Type *base`, `uint8_t dummy_data`)  
Запись фиктивных данных для передачи

## Приемо-передача

- `enum spi_status SPI_MasterTransferCreateHandle` (`SPI_Type *base`, `spi_master_handle_t *handle`, `spi_master_callback_t callback`, `void *user_data`)  
Инициализация дескриптора SPI Master.
- `enum spi_status SPI_MasterTransferBlocking` (`SPI_Type *base`, `spi_transfer_t *xfer`)  
Блокирующая дуплексная передача данных (с ожиданием завершения операции)
- `enum spi_status SPI_MasterTransferNonBlocking` (`SPI_Type *base`, `spi_master_handle_t *handle`, `spi_transfer_t *xfer`)  
Неблокирующий SPI обмен по прерыванию
- `enum spi_status SPI_MasterHalfDuplexTransferBlocking` (`SPI_Type *base`, `spi_half_duplex_transfer_t *xfer`)  
Блокирующая полудуплексная передача данных (с ожиданием завершения операции)
- `status_t SPI_MasterHalfDuplexTransferNonBlocking` (`SPI_Type *base`, `spi_master_handle_t *handle`, `spi_half_duplex_transfer_t *xfer`)  
Выполнение неблокирующего SPI обмена по прерыванию
- `status_t SPI_MasterTransferGetByte` (`spi_master_handle_t *handle`, `size_t *count`)  
Получение количества переданных и принятых байт
- `status_t SPI_MasterTransferGetRemainingByte` (`spi_master_handle_t *handle`, `size_t *count`)  
Получение количества оставшихся байт для передачи и приема
- `status_t SPI_MasterTransferGetTotalByte` (`spi_master_handle_t *handle`, `size_t *count`)  
Получение общего количества байт на передачу и прием
- `void SPI_MasterTransferAbort` (`SPI_Type *base`, `spi_master_handle_t *handle`)  
Останов передачи для мастер режима
- `void SPI_MasterTransferHandleIRQ` (`SPI_Type *base`, `spi_master_handle_t *handle`)
- `static status_t SPI_SlaveTransferCreateHandle` (`SPI_Type *base`, `spi_slave_handle_t *handle`, `spi_slave_callback_t callback`, `void *userData`)  
Инициализация slave SPI дескриптор
- `static status_t SPI_SlaveTransferNonBlocking` (`SPI_Type *base`, `spi_slave_handle_t *handle`, `spi_transfer_t *xfer`)  
Неблокирующая дуплексная передача данных (без ожидания завершения операции)
- `static status_t SPI_SlaveHalfDuplexTransferNonBlocking` (`SPI_Type *base`, `spi_slave_handle_t *handle`, `spi_half_duplex_transfer_t *xfer`)
- `static status_t SPI_SlaveTransferGetByte` (`spi_slave_handle_t *handle`, `size_t *count`)  
Получение количества байт для обмена
- `static void SPI_SlaveTransferAbort` (`SPI_Type *base`, `spi_slave_handle_t *handle`)  
Останов передачи для режима Slave по прерыванию.
- `static void SPI_SlaveTransferHandleIRQ` (`SPI_Type *base`, `spi_slave_handle_t *handle`)  
Обработчик прерываний для SPI.

### 4.20.1 Подробное описание

Последовательный синхронный стандарт передачи данных в режиме полного дуплекса, полудуплекса или симплекса

#### Заметки

Драйвер поддерживает обмены по интерфейсу SPI по прерыванию и в режиме опроса, ширину поля данных от 4 до 32 битов, форматы кадров: Motorola SPI, Texas Instruments, Synchronous Serial Protocol (SSP) и NS Microwire.

### 4.20.2 Макросы

#### 4.20.2.1 SPI\_DUMMYDATA

```
#define SPI_DUMMYDATA (0xA5U)
```

SPI фиктивные данные для передачи по умолчанию.

#### Заметки

Фиктивные данные передаются в @SPI\_MasterTransferBlocking, когда буфер Tx пуст, но есть запрос на прием.

#### 4.20.2.2 SPI\_RETRY\_TIMES

```
#define SPI_RETRY_TIMES 0U
```

Время повтора для флага ожидания. Ноль означает продолжать ждать, пока флаг не будет установлен/снят

### 4.20.3 Перечисления

#### 4.20.3.1 microwire\_busy\_ready\_check\_t

```
enum microwire_busy_ready_check_t
```

Включить/отключить проверку busy/ready флага (регистр SR) для формата кадра передачи данных National Semiconductor Microwire.

#### Заметки

В активном состоянии модуль SPI проверяет готовность Slave после передачи последнего бита данных, для снятия busy статуса в регистре SR.

#### Элементы перечислений

SPI_MicrowireBusyReadyCheckDisable	Отключить проверку
SPI_MicrowireBusyReadyCheckEnable	Включить проверку

#### 4.20.3.2 microwire\_ctrlword\_len\_t

```
enum microwire_ctrlword_len_t
```

Выбор длины управляющего слова для формата кадра передачи данных National Semiconductor Microwire.

Элементы перечислений

SPI_MicrowireCtrlWordLen1Bit	Длина - 1 бит
SPI_MicrowireCtrlWordLen2Bit	Длина - 2 бита
SPI_MicrowireCtrlWordLen3Bit	Длина - 3 бита
SPI_MicrowireCtrlWordLen4Bit	Длина - 4 бита
SPI_MicrowireCtrlWordLen5Bit	Длина - 5 бит
SPI_MicrowireCtrlWordLen6Bit	Длина - 6 бит
SPI_MicrowireCtrlWordLen7Bit	Длина - 7 бит
SPI_MicrowireCtrlWordLen8Bit	Длина - 8 бит
SPI_MicrowireCtrlWordLen9Bit	Длина - 9 бит
SPI_MicrowireCtrlWordLen10Bit	Длина - 10 бит
SPI_MicrowireCtrlWordLen11Bit	Длина - 11 бит
SPI_MicrowireCtrlWordLen12Bit	Длина - 12 бит
SPI_MicrowireCtrlWordLen13Bit	Длина - 13 бит
SPI_MicrowireCtrlWordLen14Bit	Длина - 14 бит
SPI_MicrowireCtrlWordLen15Bit	Длина - 15 бит
SPI_MicrowireCtrlWordLen16Bit	Длина - 16 бит

#### 4.20.3.3 microwire\_single\_serial\_t

enum [microwire\\_single\\_serial\\_t](#)

Выбор типа передачи (одиночная или последовательная) для формата кадра передачи данных National Semiconductor Microwire.

Элементы перечислений

SPI_MicrowireSingle	Одиночная передача
SPI_MicrowireSerial	Последовательная передача

#### 4.20.3.4 microwire\_tx\_rx\_t

enum [microwire\\_tx\\_rx\\_t](#)

Направление передачи слова данных для формата кадра передачи данных National Semiconductor Microwire.

Элементы перечислений

SPI_MicrowireTx	SPI передает слово данных
SPI_MicrowireRx	SPI принимает слово данных

## 4.20.3.5 spi\_frame\_format\_t

enum [spi\\_frame\\_format\\_t](#)

Формат кадра передачи данных

Заметки

Для Motorola SPI - режим Slave-Select выставляется на всю продолжительность обмена данными.

Для Texas Instruments Synchronous Serial Protocol (SSP):

- Slave-Select выставляется на 1 такт до начала передачи;
- Установка данных происходит по переднему фронту Clk, а выборка - по заднему;
- Значение DFS должно быть кратно 2.

Для National Semiconductor Microwire:

- Сигнал Slave-Select остается активно-низким на протяжении всей передачи и переходит в высокое состояние через полтакта после окончания передачи данных;
- Данные устанавливаются по заднему фронту линии синхронизации, а выборка по переднему;
- Значение DFS должно быть кратно 4.

Элементы перечислений

SPI_FfMotorola	Motorola SPI
SPI_FfTexas	Texas Instruments SSP
SPI_FfMicrowire	National Semiconductor Microwire

## 4.20.3.6 spi\_frame\_width\_t

enum [spi\\_frame\\_width\\_t](#)

Размер кадра данных в 32-х битном режиме передачи данных (CTRLR0.DFS\_32)

Элементы перечислений

SPI_Data4Bits	4 бита
SPI_Data5Bits	5 бит
SPI_Data6Bits	6 бит
SPI_Data7Bits	7 бит
SPI_Data8Bits	8 бит
SPI_Data9Bits	9 бит
SPI_Data10Bits	10 бит
SPI_Data11Bits	11 бит

## Элементы перечислений

SPI_Data12Bits	12 бит
SPI_Data13Bits	13 бит
SPI_Data14Bits	14 бит
SPI_Data15Bits	15 бит
SPI_Data16Bits	16 бит
SPI_Data17Bits	17 бит
SPI_Data18Bits	18 бит
SPI_Data19Bits	19 бит
SPI_Data20Bits	20 бит
SPI_Data21Bits	21 бит
SPI_Data22Bits	22 бита
SPI_Data23Bits	23 бита
SPI_Data24Bits	24 бита
SPI_Data25Bits	25 бит
SPI_Data26Bits	26 бит
SPI_Data27Bits	27 бит
SPI_Data28Bits	28 бит
SPI_Data29Bits	29 бит
SPI_Data30Bits	30 бит
SPI_Data31Bits	31 бит
SPI_Data32Bits	32 бита

## 4.20.3.7 spi\_interrupt\_enable

enum `spi_interrupt_enable`

Источники прерываний SPI.

## Заметки

Битовые маски подходят для работы с регистрами: IMR - регистр маскирования прерываний; ISR - регистр статуса прерываний после маскирования; RISR - регистр статуса прерываний до маскирования.

## Элементы перечислений

SPI_IRQ_MultiMaster	Бит 5: Мультимастер
SPI_IRQ_RxFifoTrigger	Бит 4: Прерывание по триггеру уровня RxFIFO, если уровень RxFIFO больше или равен регистру RXFLTR
SPI_IRQ_RxFifoOverflow	Бит 3: Переполнение RxFIFO
SPI_IRQ_RxFifoUnderflow	Бит 2: Чтение из пустого RxFIFO
SPI_IRQ_TxFifoOverflow	Бит 1: Переполнение TxFIFO
SPI_IRQ_TxFifoTrigger	Бит 0: Прерывание по триггеру уровня TxFIFO, если уровень TxFIFO меньше или равен установленному значению регистра TXFTLR

SPI_IRQ_TxOnly	Только передача
SPI_IRQ_RxOnly	Только прием
SPI_IRQ_All	Все прерывания включены

#### 4.20.3.8 spi\_mode\_t

enum [spi\\_mode\\_t](#)

Режим передачи (CTRLR0.TMOD)

Заметки

- Дуплекс (Duplex) - передача продолжится до последнего слова в FIFO передатчика;
- Симплекс, только передача (SimplexTx) - принимаемые данные не поступают в RxFIFO; при использовании этого режима необходимо маскировать прерывания от приемника;
- Симплекс, только прием (SimplexRx) - передаваемые данные не валидны; при использовании этого режима необходимо маскировать прерывания от передатчика;
- Полудуплекс (Halfduplex) - режим чтения EEPROM. Сначала передаются все данные из TxFIFO, принимаемые в этот момент данные игнорируются; когда все данные были отправлены, модуль принимает заданное в CTRLR1.NDF+1 количество кадров. Режим недоступен для протокола SSP.

Элементы перечислений

SPI_ModeDuplex	Дуплекс (передача и прием идут одновременно)
SPI_ModeSimplexTx	Симплекс (только передача)
SPI_ModeSimplexRx	Симплекс (только прием)
SPI_ModeHalfDuplex	Полудуплекс (сначала передача потом прием). Режим - чтение EEPROM

#### 4.20.3.9 spi\_motorola\_cap\_data\_t

enum [spi\\_motorola\\_cap\\_data\\_t](#)

Выбор фронта для захвата данных для формата кадра передачи данных Motorola SPI.

Элементы перечислений

SPI_MotorolaCapDataRising	Захват данных происходит по переднему фронту тактового сигнала
SPI_MotorolaCapDataFalling	Происходит пропуск одного периода тактового сигнала после установки Slave-Select, захват данных происходит по заднему фронту тактового сигнала

#### 4.20.3.10 spi\_motorola\_clk\_pol\_t

enum [spi\\_motorola\\_clk\\_pol\\_t](#)

Выбор полярности тактового сигнала при отсутствия передаваемых данных в режиме Master для формата кадра передачи данных Motorola SPI.

Элементы перечислений

SPI_MotorolaClkPolLow	Линия синхронизации до начала цикла передачи и после его окончания имеет низкий уровень
SPI_MotorolaClkPolHi	Линия синхронизации до начала цикла передачи и после его окончания имеет высокий уровень

#### 4.20.3.11 spi\_rxfifo\_watermark\_t

enum [spi\\_rxfifo\\_watermark\\_t](#)

Триггер уровня заполнения RxFIFO.

Элементы перечислений

SPI_RxFifoWatermark1	1 элемент в RxFIFO
SPI_RxFifoWatermark2	2 элемента в RxFIFO
SPI_RxFifoWatermark3	3 элемента в RxFIFO
SPI_RxFifoWatermark4	4 элемента в RxFIFO
SPI_RxFifoWatermark5	5 элементов в RxFIFO
SPI_RxFifoWatermark6	6 элементов в RxFIFO
SPI_RxFifoWatermark7	7 элементов в RxFIFO
SPI_RxFifoWatermark8	8 элементов в RxFIFO

#### 4.20.3.12 spi\_shift\_direction\_t

enum [spi\\_shift\\_direction\\_t](#)

Формат передачи данных (MSB или LSB)

Элементы перечислений

SPI_ShiftDirMsbFirst	Передача данных начинается со старшего бита
SPI_ShiftDirLsbFirst	Передача данных начинается с младшего бита

#### 4.20.3.13 spi\_status

enum [spi\\_status](#)

Статусы возврата из функций для драйвера SPI.

Элементы перечислений

SPI_Status_Ok	Успешно
SPI_Status_Fail	Провал
SPI_Status_ReadOnly	Только чтение

Элементы перечислений

SPI_Status_InvalidArgument	Неверный аргумент
SPI_Status_Timeout	Отказ по таймауту
SPI_Status_BaudrateNotSupport	Частота не поддерживается
SPI_Status_Busy	SPI модуль занят
SPI_Status_Idle	SPI модуль простаивает
SPI_Status_TxError	Ошибка в TxFIFO
SPI_Status_RxError	Ошибка в RxFIFO
SPI_Status_RxRingBufferOverrun	Ошибка в кольцевом буфере Rx
SPI_Status_RxFifoBufferOverrun	Ошибка переполнения hw RxFIFO буфера
SPI_Status_NoTransferInProgress	Нет текущего обмена
SPI_Status_IncorrectCall	Некорректный вызов с текущими настройками модуля
SPI_Status_UnexpectedState	Неожиданное состояние

#### 4.20.3.14 spi\_status\_flags

enum [spi\\_status\\_flags](#)

SPI флаги статусов

Элементы перечислений

SPI_TxNotFullFlag	TxFIFO не полон
SPI_TxEmptyFlag	TxFIFO пуст
SPI_RxNotEmptyFlag	RxFIFO не пуст
SPI_RxFullFlag	RxFIFO полон

#### 4.20.3.15 spi\_trans\_status

enum [spi\\_trans\\_status](#)

Состояния приемо-передачи SPI.

Элементы перечислений

SPI_TransStatus_Busy	Модуль занят
SPI_TransStatus_Idle	Модуль простаивает
SPI_TransStatus_Error	Ошибка
SPI_TransStatus_Error↵ _1	Multi Master Конфликт
SPI_TransStatus_Error↵ _2	Внутренняя ошибка, TxFIFO Overflow
SPI_TransStatus_Error↵ _3	Внутренняя ошибка, RxFIFO Overflow
SPI_TransStatus_Error↵ _4	Внутренняя ошибка, RxFIFO Underflow



## 4.20.3.16 spi\_txfifo\_watermark\_t

enum [spi\\_txfifo\\_watermark\\_t](#)

Триггер уровня заполнения TxFIFO.

Элементы перечислений

SPI_TxFifoWatermark0	TxFIFO пуст
SPI_TxFifoWatermark1	1 элемент в TxFIFO
SPI_TxFifoWatermark2	2 элемента в TxFIFO
SPI_TxFifoWatermark3	3 элемента в TxFIFO
SPI_TxFifoWatermark4	4 элемента в TxFIFO
SPI_TxFifoWatermark5	5 элементов в TxFIFO
SPI_TxFifoWatermark6	6 элементов в TxFIFO
SPI_TxFifoWatermark7	7 элементов в TxFIFO

## 4.20.4 Функции

## 4.20.4.1 SPI\_CurrentStatusInterrupts()

```
static uint32_t SPI_CurrentStatusInterrupts (
    SPI_Type * base)  [inline], [static]
```

Получение статуса прерываний

Аргументы

base	Базовый адрес SPI
------	-------------------

Возвращает

Статус прерываний

## 4.20.4.2 SPI\_Deinit()

```
void SPI_Deinit (
    SPI_Type * base)
```

Деинициализация SPI.

Заметки

Вызов этого API сбрасывает модуль SPI. Модуль SPI не может работать без вызова функции [SPI\\_MasterInit](#) / [SPI\\_SlaveInit](#) для инициализации.

## Аргументы

base	Базовый адрес SPI
------	-------------------

## 4.20.4.3 SPI\_DisableInterrupts()

```
static void SPI_DisableInterrupts (
    SPI_Type * base,
    uint32_t irqs) [inline], [static]
```

Отключение прерываний SPI.

## Заметки

В качестве источника прерывания может быть комбинация следующих значений:  
[SPI\\_IRQ\\_MultiMaster](#) [SPI\\_IRQ\\_RxFifoTrigger](#) [SPI\\_IRQ\\_RxFifoOverflow](#) [SPI\\_IRQ\\_RxFifoUnderflow](#)  
[SPI\\_IRQ\\_TxFifoOverflow](#) [SPI\\_IRQ\\_TxFifoTrigger](#) [SPI\\_IRQ\\_All](#)

## Аргументы

base	Базовый адрес SPI
irqs	Источники прерываний

## 4.20.4.4 SPI\_DMADescriptorInitRX()

```
static void SPI_DMADescriptorInitRX (
    SPI_Type * base,
    dma_descriptor_t * desc,
    uint32_t count,
    uint32_t data_size,
    uint8_t dst_incr,
    uint32_t data_width,
    void * dst_addr) [inline], [static]
```

Инициализация дескрипторов DMA для многоблочной передачи RX.

## Аргументы

base	Базовый адрес SPI
desc	Указатель на массив дескрипторов
count	Количество DMA дескрипторов
data_size	Общее число передаваемых данных
data_width	Ширина 1ого передаваемого слова
dst_addr	Адрес Приемника

## 4.20.4.5 SPI\_DMADescriptorInitTX()

```
static void SPI_DMADescriptorInitTX (
    SPI_Type * base,
    dma_descriptor_t * desc,
    uint32_t count,
    uint32_t data_size,
    uint8_t src_incr,
    uint32_t data_width,
    void * src_addr) [inline], [static]
```

Инициализация дескрипторов DMA для многоблочной передачи TX.

Аргументы

base	Базовый адрес SPI
desc	Указатель на массив дескрипторов
count	Количество DMA дескрипторов
data_size	Общее число передаваемых данных
data_width	Ширина одного передаваемого слова
src_addr	Адрес Источника

## 4.20.4.6 SPI\_Enable()

```
static void SPI_Enable (
    SPI_Type * base,
    bool enable) [inline], [static]
```

Включение или отключение модуль SPI Master или Slave.

Аргументы

base	Базовый адрес SPI
enable	Включить (1) или выключить (0)

## 4.20.4.7 SPI\_EnableInterrupts()

```
static void SPI_EnableInterrupts (
    SPI_Type * base,
    uint32_t irqs) [inline], [static]
```

Включение прерываний SPI.

Заметки

В качестве источника прерывания может быть комбинация следующих значений:  
[SPI\\_IRQ\\_MultiMaster](#) [SPI\\_IRQ\\_RxFifoTrigger](#) [SPI\\_IRQ\\_RxFifoOverflow](#) [SPI\\_IRQ\\_RxFifoUnderflow](#)  
[SPI\\_IRQ\\_TxFifoOverflow](#) [SPI\\_IRQ\\_TxFifoTrigger](#) [SPI\\_IRQ\\_All](#)

## Аргументы

base	Базовый адрес SPI
irqs	Источники прерываний

## 4.20.4.8 SPI\_EnableRxDMA()

```
void SPI_EnableRxDMA (
    SPI_Type * base,
    bool enable)
```

Включение или отключение запроса DMA от SPI RxFIFO.

## Аргументы

base	Базовый адрес SPI
enable	Включить (1) или выключить (0)

## 4.20.4.9 SPI\_EnableTxDMA()

```
void SPI_EnableTxDMA (
    SPI_Type * base,
    bool enable)
```

Включение или отключение запроса DMA от SPI TxFIFO.

## Аргументы

base	Базовый адрес SPI
enable	Включить (1) или выключить (0)

## 4.20.4.10 SPI\_GetConfig()

```
spi_config_internal_t * SPI_GetConfig (
    SPI_Type * base)
```

Получение внутренней конфигурации

## Аргументы

base	Базовый адрес SPI
------	-------------------

## Возвращает

Внутренняя структура конфигурации модуля SPI

## 4.20.4.11 SPI\_GetInstance()

```
uint8_t SPI_GetInstance (
    SPI_Type * base)
```

Получение индекса модуля SPI.

## Аргументы

base	Базовый адрес SPI
------	-------------------

## Возвращает

Индекс модуля

## 4.20.4.12 SPI\_GetStatusFlags()

```
static uint32_t SPI_GetStatusFlags (  
    SPI_Type * base)  [inline], [static]
```

Получение флага состояния SPI.

## Заметки

Для получения статуса, соответствующего состоянию SPI, необходимо использовать полученный флаг состояния и [spi\\_status\\_flags](#).

## Аргументы

base	Базовый адрес SPI
------	-------------------

## Возвращает

Состояние SPI

## 4.20.4.13 SPI\_IsEnable()

```
static bool SPI_IsEnable (  
    SPI_Type * base)  [inline], [static]
```

Получение статуса модуля SPI включено/выключено

## Аргументы

base	Базовый адрес SPI
------	-------------------

## Возвращаемые значения

1	Включено
0	Выключено

## 4.20.4.14 SPI\_MasterGetDefaultConfig()

```
void SPI_MasterGetDefaultConfig (
    spi_config_t * config)
```

Инициализация конфигурации SPI значениями по умолчанию

Заметки

Инициализация структуры конфигурации SPI для использования в [SPI\\_MasterInit](#). Инициализированную структуру можно использовать как без изменений, так и изменив какие-либо из полей структуры перед вызовом [SPI\\_MasterInit](#).

Пример:

```
spi_config_t config;
SPI_MasterGetDefaultConfig(&config);
```

Аргументы

config	Конфигурация SPI
--------	------------------

## 4.20.4.15 SPI\_MasterHalfDuplexTransferBlocking()

```
enum spi_status SPI_MasterHalfDuplexTransferBlocking (
    SPI_Type * base,
    spi_half_duplex_transfer_t * xfer)
```

Блокирующая полудуплексная передача данных (с ожиданием завершения операции)

Заметки

Функция не возвращает управление, пока все данные не будут переданы. Данные передаются в полудуплексном режиме. Пользователь может выбрать, что будет выполняться в первую очередь - передача или прием.

Аргументы

base	Базовый адрес SPI
xfer	Структура для полудуплексной приемо-передачи

Возвращаемые значения

<a href="#">SPI_Status_Ok</a>	
<a href="#">SPI_Status_InvalidArgument</a>	
<a href="#">SPI_Status_Timeout</a>	

## 4.20.4.16 SPI\_MasterHalfDuplexTransferDMA()

```
enum spi_status SPI_MasterHalfDuplexTransferDMA (
    SPI_Type * base,
    spi_dma_handle_t * handle,
    spi_half_duplex_transfer_t * xfer)
```

Функция приема/передачи данных в полудуплексном режиме через DMA каналы в порт SPI.

## Аргументы

base	Базовый адрес SPI master
handle	Дескриптор передачи
xfer	Параметры полудуплексной передачи

## Возвращаемые значения

#SPI_DMA_Status_InvalidArgument	
#SPI_DMA_Status_Busy	
#SPI_DMA_Status_SPINotExist	
#SPI_DMA_Status_Fail	
#SPI_DMA_Status_Success	

## 4.20.4.17 SPI\_MasterHalfDuplexTransferNonBlocking()

```
status_t SPI_MasterHalfDuplexTransferNonBlocking (
    SPI_Type * base,
    spi_master_handle_t * handle,
    spi_half_duplex_transfer_t * xfer)
```

Выполнение неблокирующего SPI обмена по прерыванию

## Заметки

Эта функция использует режим опроса для первой части обмена и использует прерывания для выполнения второй части обмена. Режим обмена полудуплексный. Когда завершится первая часть обмена, сразу произойдет выход из функции, по завершении второй части обмена произойдет вызов callback-функции.

## Аргументы

base	Базовый адрес SPI.
handle	SPI структура дескриптора для работы по прерыванию.
xfer	Структура для полудуплексной приемо-передачи.

## Возвращаемые значения

SPI_Status_Ok	
SPI_Status_InvalidArgument	

## 4.20.4.18 SPI\_MasterInit()

```
enum spi_status SPI_MasterInit (
    SPI_Type * base,
    const spi_config_t * config,
    uint32_t src_clock_hz)
```

Инициализация SPI модуля как Master с заданной конфигурацией

Структура конфигурации может быть заполнена пользователем с нуля или установлена по умолчанию значения с помощью [SPI\\_MasterGetDefaultConfig](#). После вызова этого API ведомое устройство готово к передаче.

Пример:

```
spi_config_t config = {
    .frame_width_bits    = SPI_Data8Bits;
    .master.loopback_enable = false;
    .enable              = false;
    .master.baud_rate_bps = 50000;
    ...
};
SPI_MasterInit(SPI0, &config);
```

Аргументы

base	Базовый адрес SPI
config	Основная конфигурация SPI
src_clock_hz	Исходная тактовая частота

Возвращаемые значения

<a href="#">SPI_Status_Ok</a>	
<a href="#">SPI_Status_InvalidArgument</a>	

## 4.20.4.19 SPI\_MasterSetBaud()

```
enum spi_status SPI_MasterSetBaud (
    SPI_Type * base,
    uint32_t baudrate_bps,
    uint32_t src_clock_hz)
```

Установка скорости передачи для SPI Master.

Аргументы

base	Базовый адрес SPI
baudrate_bps	Скорость передачи в Hz
src_clock_hz	SPI частота синхронизации в Hz

Возвращаемые значения

<a href="#">SPI_Status_Ok</a>	
<a href="#">SPI_Status_BaudrateNotSupport</a>	
<a href="#">SPI_Status_InvalidArgument</a>	



## 4.20.4.20 SPI\_MasterTransferAbort()

```
void SPI_MasterTransferAbort (
    SPI_Type * base,
    spi_master_handle_t * handle)
```

Останов передачи для мастер режима

Заметки

Функция останавливает передачу, происходящую по прерываниям.

Аргументы

base	Базовый адрес SPI.
handle	SPI структура дескриптора для работы по прерыванию.

## 4.20.4.21 SPI\_MasterTransferAbortDMA()

```
void SPI_MasterTransferAbortDMA (
    spi_dma_handle_t * handle)
```

Прекращение передачи SPI.

Аргументы

handle	Дескриптор SPI-DMA
--------	--------------------

## 4.20.4.22 SPI\_MasterTransferBlocking()

```
enum spi_status SPI_MasterTransferBlocking (
    SPI_Type * base,
    spi_transfer_t * xfer)
```

Блокирующая дуплексная передача данных (с ожиданием завершения операции)

Заметки

Функция не поддерживает симплексный режим обмена. Это значит, что в xfer должны быть переданы указатели на данные для передачи и буфер на прием с указанием длины, неравной нулю. Если требуется симплексный обмен, используйте @SPI\_MasterHalfDuplexTransfer↔Blocking

Аргументы

base	Базовый адрес SPI
xfer	Структура для приемо-передачи

Возвращаемые значения

<a href="#">SPI_Status_Ok</a>	
<a href="#">SPI_Status_InvalidArgument</a>	
<a href="#">SPI_Status_Timeout</a>	

#### 4.20.4.23 SPI\_MasterTransferCreateHandle()

```
enum spi\_status SPI_MasterTransferCreateHandle (
    SPI_Type * base,
    spi\_master\_handle\_t * handle,
    spi\_master\_callback\_t callback,
    void * user_data)
```

Инициализация дескриптора SPI Master.

Аргументы

base	Базовый адрес SPI
handle	SPI дескриптор
callback	Callback-функция
user_data	Данные пользователя для callback-функции

Возвращаемые значения

<a href="#">SPI_Status_Ok</a>	
<a href="#">SPI_Status_InvalidArgument</a>	

#### 4.20.4.24 SPI\_MasterTransferCreateHandleDMA()

```
enum spi\_status SPI_MasterTransferCreateHandleDMA (
    SPI_Type * base,
    spi\_dma\_callback\_t callback,
    void * user_data,
    spi\_dma\_handle\_t * handle,
    dma\_handle\_t * tx_handle,
    dma\_handle\_t * rx_handle)
```

Функция инициализации дескриптора SPI-DMA.

Аргументы

base	Базовый адрес SPI master
handle	Дескриптор SPI-DMA
callback	Функция обратного вызова
user_data	Пользовательские данные
tx_handle	DMA дескриптор отправки
rx_handle	DMA дескриптор приема

Возвращаемые значения

#SPI_DMA_Status_InvalidArgument	
#SPI_DMA_Status_Success	

#### 4.20.4.25 SPI\_MasterTransferDMA()

```
enum spi_status SPI_MasterTransferDMA (
    SPI_Type * base,
    spi_dma_handle_t * handle,
    spi_transfer_t * xfer)
```

Функция для SPI master приема/передачи данных в полнодуплексном режиме через DMA каналы в порт SPI.

Аргументы

base	Базовый адрес SPI master
handle	Дескриптор SPI-DMA
xfer	Параметры передачи SPI

Возвращаемые значения

#SPI_DMA_Status_InvalidArgument	
#SPI_DMA_Status_Busy	
#SPI_DMA_Status_SPINotExist	
#SPI_DMA_Status_Success	

#### 4.20.4.26 SPI\_MasterTransferGetByte()

```
status_t SPI_MasterTransferGetByte (
    spi_master_handle_t * handle,
    size_t * count)
```

Получение количества переданных и принятых байт

Аргументы

handle	SPI структура дескриптора для работы по прерыванию.
count	Количество байтов, переданных с помощью неблокирующей транзакции.

Возвращаемые значения

SPI_Status_Ok	
SPI_Status_InvalidArgument	

#### 4.20.4.27 SPI\_MasterTransferGetRemainingByte()

```
status_t SPI_MasterTransferGetRemainingByte (  
    spi_master_handle_t * handle,  
    size_t * count)
```

Получение количества оставшихся байт для передачи и приема

## Аргументы

handle	SPI структура дескриптора для работы по прерыванию.
count	Количество байтов, оставшихся для передачи и приема с помощью неблокирующей транзакции.

## Возвращаемые значения

<a href="#">SPI_Status_Ok</a>	
<a href="#">SPI_Status_InvalidArgument</a>	

## 4.20.4.28 SPI\_MasterTransferGetTotalByte()

```
status_t SPI_MasterTransferGetTotalByte (
    spi_master_handle_t * handle,
    size_t * count)
```

Получение общего количества байт на передачу и прием

## Аргументы

handle	SPI структура дескриптора для работы по прерыванию.
count	Общее количество байт на передачу и прием

## Возвращаемые значения

<a href="#">SPI_Status_Ok</a>	
<a href="#">SPI_Status_InvalidArgument</a>	

## 4.20.4.29 SPI\_MasterTransferHandleIRQ()

```
void SPI_MasterTransferHandleIRQ (
    SPI_Type * base,
    spi_master_handle_t * handle)
```

brief Обработчик прерываний для SPI.

## Аргументы

base	Базовый адрес SPI.
handle	SPI структура дескриптора для работы по прерыванию.

## 4.20.4.30 SPI\_MasterTransferNonBlocking()

```
enum spi_status SPI_MasterTransferNonBlocking (
    SPI_Type * base,
    spi_master_handle_t * handle,
    spi_transfer_t * xfer)
```

Неблокирующий SPI обмен по прерыванию

## Заметки

Функция используется для разнесенной по времени передачи и приема, первым этапом происходит передача данных заданной ширины, вторым - прием данных. Весь обмен происходит в асинхронном режиме, т.е в прерываниях. По завершении второй части обмена произойдет вызов callback функции.

Функция не поддерживает симплексный режим обмена. Это значит, что в xfer должны быть переданы указатели на данные для передачи и буфер на прием с указанием длины, неравной нулю. Если требуется симплексный обмен, используйте @SPI\_MasterHalfDuplexTransferNonBlocking

## Аргументы

base	Базовый адрес SPI.
handle	SPI структура дескриптора для работы по прерыванию.
xfer	Структура для полудуплексной приемо-передачи.

## Возвращаемые значения

<a href="#">SPI_Status_Ok</a>	
<a href="#">SPI_Status_InvalidArgument</a>	
<a href="#">SPI_Status_IncorrectCall</a>	

## 4.20.4.31 SPI\_ReadData()

```
uint32_t SPI_ReadData (
    SPI_Type * base)
```

Получение данных из регистра данных SPI.

## Аргументы

base	Базовый адрес SPI
------	-------------------

## Возвращает

Данные регистра

## 4.20.4.32 SPI\_Reset()

```
static void SPI_Reset (  
    SPI_Type * base)  [inline], [static]
```

Останов всех операций SPI.

## Заметки

Любые операции передачи данных прерываются немедленно. FIFO буферы приемника и передатчика очищаются.

## Аргументы

base	Базовый адрес SPI
------	-------------------

## 4.20.4.33 SPI\_SetDummyData()

```
void SPI_SetDummyData (
    SPI_Type * base,
    uint8_t dummy_data)
```

Запись фиктивных данных для передачи

## Заметки

Фиктивные данные передаются в @SPI\_MasterTransferBlocking, когда буфер Tx пуст, но есть запрос на прием.

## Аргументы

base	Базовый адрес SPI
dummy_data	Фиктивные данные

## 4.20.4.34 SPI\_SlaveGetDefaultConfig()

```
void SPI_SlaveGetDefaultConfig (
    spi_config_t * config)
```

Заполнение структуры конфигурации SPI Slave-устройства значениями по умолчанию

## Заметки

Функция инициализирует структуру конфигурации для использования в [SPI\\_SlaveInit](#). Пользователь может изменить некоторые поля структуры перед вызовом [SPI\\_SlaveInit](#).

## Пример:

```
spi_config_t config;
SPI_SlaveGetDefaultConfig(&config);
```

## Аргументы

config	Структура конфигурации Slave
--------	------------------------------



## 4.20.4.35 SPI\_SlaveInit()

```
enum spi_status SPI_SlaveInit (
    SPI_Type * base,
    const spi_config_t * config)
```

Инициализация SPI заданной конфигурацией

Заметки

Структура конфигурации может быть заполнена пользователем или установлена функцией `SPI_SlaveGetDefaultConfig()` в значения по умолчанию. После вызова этой функции устройство готово к работе в режиме Slave.

Пример:

```
spi_config_t config = {
    .direction = SPIMsbFirst;
    ...
};
SPI_SlaveInit(SPI0, &config);
```

Аргументы

base	Базовый адрес SPI
config	Структура конфигурации Slave

Возвращаемые значения

SPI_Status_Ok	
SPI_Status_InvalidArgument	

## 4.20.4.36 SPI\_SlaveTransferAbort()

```
static void SPI_SlaveTransferAbort (
    SPI_Type * base,
    spi_slave_handle_t * handle) [inline], [static]
```

Останов передачи для режима Slave по прерыванию.

Аргументы

base	Базовый адрес SPI.
handle	Структура дескриптора для работы по прерыванию.

## 4.20.4.37 SPI\_SlaveTransferAbortDMA()

```
static void SPI_SlaveTransferAbortDMA (
    spi_dma_handle_t * handle) [inline], [static]
```

Прекращение передачи SPI.

## Аргументы

handle	Дескриптор SPI-DMA
--------	--------------------

## 4.20.4.38 SPI\_SlaveTransferCreateHandle()

```
static status_t SPI_SlaveTransferCreateHandle (
    SPI_Type * base,
    spi_slave_handle_t * handle,
    spi_slave_callback_t callback,
    void * userData) [inline], [static]
```

Инициализация slave SPI дескриптор

## Заметки

Функция инициализирует дескриптор slave SPI. Для указанного модуля SPI достаточно вызывать один раз эту функцию, чтобы получить инициализированный дескриптор.

## Аргументы

base	Базовый адрес SPI.
handle	SPI структура дескриптора для работы по прерыванию.
callback	Callback-функция.
userData	Данные пользователя.

## Возвращаемые значения

SPI_Status_UnexpectedState	
SPI_Status_Ok	

## 4.20.4.39 SPI\_SlaveTransferCreateHandleDMA()

```
static enum spi_status SPI_SlaveTransferCreateHandleDMA (
    SPI_Type * base,
    spi_dma_callback_t callback,
    void * user_data,
    spi_dma_handle_t * handle,
    dma_handle_t * tx_handle,
    dma_handle_t * rx_handle) [inline], [static]
```

Функция инициализации дескриптора SPI-DMA.

## Аргументы

base	Базовый адрес SPI slave
handle	Дескриптор SPI-DMA
callback	Функция обратного вызова
user_data	Пользовательские данные
tx_handle	DMA дескриптор отправки
rx_handle	DMA дескриптор приема

Возвращаемые значения

#SPI_DMA_Status_InvalidArgument	
#SPI_DMA_Status_Success	

#### 4.20.4.40 SPI\_SlaveTransferDMA()

```
static enum spi_status SPI_SlaveTransferDMA (
    SPI_Type * base,
    spi_dma_handle_t * handle,
    spi_transfer_t * xfer) [inline], [static]
```

Функция для SPI slave приема/передачи данных в полнодуплексном режиме через DMA каналы в порт SPI.

Аргументы

base	Базовый адрес SPI slave
handle	Дескриптор SPI-DMA
xfer	Параметры передачи SPI

Возвращаемые значения

#SPI_DMA_Status_InvalidArgument	
#SPI_DMA_Status_Busy	
#SPI_DMA_Status_SPINotExist	
#SPI_DMA_Status_Success	

#### 4.20.4.41 SPI\_SlaveTransferGetByte()

```
static status_t SPI_SlaveTransferGetByte (
    spi_slave_handle_t * handle,
    size_t * count) [inline], [static]
```

Получение количества байт для обмена

Аргументы

handle	SPI структура дескриптора для работы по прерыванию.
count	Количество байтов, переданных с помощью неблокирующей транзакции.

Возвращаемые значения

SPI_Status_Ok	
SPI_Status_InvalidArgument	

#### 4.20.4.42 SPI\_SlaveTransferHandleIRQ()

```
static void SPI_SlaveTransferHandleIRQ (  
    SPI_Type * base,  
    spi_slave_handle_t * handle)  [inline], [static]
```

Обработчик прерываний для SPI.

## Аргументы

base	Базовый адрес SPI.
handle	SPI структура дескриптора для работы по прерыванию.

## 4.20.4.43 SPI\_SlaveTransferNonBlocking()

```
static status_t SPI_SlaveTransferNonBlocking (
    SPI_Type * base,
    spi_slave_handle_t * handle,
    spi_transfer_t * xfer)  [inline], [static]
```

Неблокирующая дуплексная передача данных (без ожидания завершения операции)

## Аргументы

base	Базовый адрес SPI.
handle	Структура, сохраняющая состояние приемо-передачи.
xfer	Структура для приемо-передачи.

## Возвращаемые значения

SPI_Status_Ok	
SPI_Status_InvalidArgument	
SPI_Status_Busy	

## 4.20.4.44 SPI\_TakeDownInterrupts()

```
static void SPI_TakeDownInterrupts (
    SPI_Type * base,
    uint32_t irqs)  [inline], [static]
```

Сброс флагов прерывания

## Заметки

Флаги @SPI\_IRQ\_RxFifoTrigger и @SPI\_IRQ\_TxFifoTrigger таким образом сбросить нельзя, по этой причине использование @SPI\_IRQ\_All для сброса всех прерываний недопустимо.

Прерывания @SPI\_IRQ\_RxFifoTrigger и @SPI\_IRQ\_TxFifoTrigger можно сбросить, вычитав RxFifo и TxFifo соответственно.

## Аргументы

base	Базовый адрес SPI
irqs	Источники прерываний

## 4.20.4.45 SPI\_WriteData()

```
void SPI_WriteData (
    SPI_Type * base,
    uint32_t data)
```

Запись данных в регистр данных SPI.

## Аргументы

base	Базовый адрес SPI
data	Данные для записи

## 4.21 Драйвер модуля TIM

Драйвер таймеров общего назначения

## Файлы

- файл [hal\\_timer.h](#)  
Интерфейс драйвера модуля таймеров общего назначения

## Структуры данных

- struct [timer\\_hardware\\_config](#)  
Конфигурация аппаратной части таймера общего назначения

## Макросы

- `#define` [TIMER\\_COUNT](#) 3
- `#define` [TIMER\\_HARDWARE\\_FIELD\\_MAX](#) (0xFFFFFFFFFUL)
- `#define` [TIMER\\_SOFTWARE\\_FIELD\\_MAX](#) (0xFFFFFFFFFFFFFFFFFULL)
- `#define` [TIMER\\_SOFTWARE\\_FIELD\\_HIGH\\_OFFSET](#) (32)

## Определения типов

- `typedef void(* callback_t) (TIM_Type *base)`  
Функция обратного вызова

## Перечисления

- enum [timer\\_status](#)  
Статусы драйвера таймеров общего назначения
- enum [timer\\_type\\_of\\_counting](#)  
Режимы счета импульсов таймером
- enum [timer\\_work\\_mode](#)  
Режим работы таймера общего назначения

## Интерфейс драйвера

- enum `timer_status` `TIMER_Init` (`TIM_Type` \*base, struct `timer_hardware_config` config, enum `timer_work_mode` mode, `callback_t` callback, `uint32_t` ticks\_h)  
Инициализация таймера общего назначения
- enum `timer_status` `TIMER_Deinit` (`TIM_Type` \*base)  
Деинициализация таймера общего назначения
- enum `timer_status` `TIMER_Run` (`TIM_Type` \*base)  
Запуск таймера общего назначения
- enum `timer_status` `TIMER_Stop` (`TIM_Type` \*base)  
Остановка таймера общего назначения
- enum `timer_status` `TIMER_Reset` (`TIM_Type` \*base)  
Сброс таймера общего назначения
- `uint64_t` `TIMER_GetTicks` (`TIM_Type` \*base)  
Получение количества тиков
- enum `timer_status` `TIMER_SetTick` (`TIM_Type` \*base, `uint64_t` ticks)  
Установка количества тиков
- enum `timer_status` `TIMER_GetAPIStatus` ()  
Получение результата выполнения последней функции
- static `uint32_t` `TIMER_GetTimerHardwareValue` (`TIM_Type` \*base)  
Получение значения регистра счетчика таймера
- enum `timer_status` `TIMER_SetConfig` (`TIM_Type` \*base, struct `timer_hardware_config` config, enum `timer_work_mode` mode, `callback_t` callback, `uint32_t` ticks\_h)  
Инициализация структуры таймера общего назначения
- enum `timer_status` `TIMER_IRQEnable` (`TIM_Type` \*base)  
Включение прерывания
- enum `timer_status` `TIMER_IRQDisable` (`TIM_Type` \*base)  
Отключение прерывания
- `uint32_t` `TIMER_IRQGetStatus` (`TIM_Type` \*base)  
Чтение статуса прерывания
- enum `timer_status` `TIMER_IRQClear` (`TIM_Type` \*base)  
Сброс прерывания

## 4.21.1 Подробное описание

## Драйвер таймеров общего назначения

Драйвер модуля таймеров общего назначения управляет таймерами TIM и TIM1 общего назначения и таймера LPTIM, для работы в режимах энергосбережения.

## 4.21.2 Макросы

4.21.2.1 `TIMER_COUNT`

```
#define TIMER_COUNT 3
```

Количество таймеров общего назначения

#### 4.21.2.2 TIMER\_HARDWARE\_FIELD\_MAX

```
#define TIMER_HARDWARE_FIELD_MAX (0xFFFFFFFFFUL)
```

Максимально возможное аппаратное значение поля счетчика

#### 4.21.2.3 TIMER\_SOFTWARE\_FIELD\_HIGH\_OFFSET

```
#define TIMER_SOFTWARE_FIELD_HIGH_OFFSET (32)
```

Смещение старшей части программного значения поля счетчика

#### 4.21.2.4 TIMER\_SOFTWARE\_FIELD\_MAX

```
#define TIMER_SOFTWARE_FIELD_MAX (0xFFFFFFFFFFFFFFFFFULL)
```

Максимально возможное программное значение поля счетчика

### 4.21.3 Перечисления

#### 4.21.3.1 timer\_status

```
enum timer_status
```

Статусы драйвера таймеров общего назначения

Элементы перечислений

TIMER_Status_Ok	Нет ошибок
TIMER_Status_InvalidArgument	Недопустимый аргумент
TIMER_Status_TimerBusy	Таймер уже занят
TIMER_Status_BadConfigure	Недопустимая конфигурация
TIMER_Status_NotIni	Работа с неинициализированным таймером
TIMER_Status_NotSupport	Функция не поддерживается

#### 4.21.3.2 timer\_type\_of\_counting

```
enum timer_type_of_counting
```

Режимы счета импульсов таймером

Элементы перечислений

TIMER_Work	Стандартный счет частоты
TIMER_Debug	Счет частоты с учетом отладки (СТИ)

#### 4.21.3.3 timer\_work\_mode

```
enum timer_work_mode
```

Режим работы таймера общего назначения



Элементы перечислений

TIMER_Hardware	Работа в 32-битном режиме по аппаратным настройкам
TIMER_Software	Работа в режиме эмуляции 64-битного таймера

#### 4.21.4 Функции

##### 4.21.4.1 TIMER\_Deinit()

```
enum timer_status TIMER_Deinit (
    TIM_Type * base)
```

Деинициализация таймера общего назначения

Аргументы

base	Таймер
------	--------

Возвращаемые значения

TIMER_Status_Ok	
TIMER_Status_InvalidArgument	

##### 4.21.4.2 TIMER\_GetAPIStatus()

```
enum timer_status TIMER_GetAPIStatus ()
```

Получение результата выполнения последней функции

Возвращает статус выполнения последней функции, у которой тип возвращаемого значения отличен от `timer_status`.

Возвращаемые значения

TIMER_Status_Ok	
TIMER_Status_InvalidArgument	
TIMER_Status_TimerBusy	
TIMER_Status_BadConfigure	
TIMER_Status_NotIni	
TIMER_Status_NotSupport	

##### 4.21.4.3 TIMER\_GetTicks()

```
uint64_t TIMER_GetTicks (
    TIM_Type * base)
```

Получение количества тиков

## Аргументы

base	Таймер
------	--------

## Возвращает

Количество подсчитанных тактов

4.21.4.4 `TIMER_GetTimerHardwareValue()`

```
static uint32_t TIMER_GetTimerHardwareValue (
    TIM_Type * base)  [inline], [static]
```

Получение значения регистра счетчика таймера

## Аргументы

base	Таймер
------	--------

## Возвращает

Количество подсчитанных тактов

4.21.4.5 `TIMER_Init()`

```
enum timer_status TIMER_Init (
    TIM_Type * base,
    struct timer_hardware_config config,
    enum timer_work_mode mode,
    callback_t callback,
    uint32_t ticks_h)
```

Инициализация таймера общего назначения

## Заметки

Конфигурация аппаратуры таймера используется полностью при режиме работы [TIMER\\_Hardware](#), а при [TIMER\\_Software](#) - только поля `work_type` и `start_enable`.

## Аргументы

base	Таймер
config	Конфигурация аппаратуры таймера
mode	Режим работы
callback	Функция обратного вызова
ticks↔ _h	Начальное значение для старшей части счетчика обратного счета при режиме работы <a href="#">TIMER_Software</a>

Возвращаемые значения

<a href="#">TIMER_Status_Ok</a>	
<a href="#">TIMER_Status_InvalidArgument</a>	
<a href="#">TIMER_Status_TimerBusy</a>	

#### 4.21.4.6 TIMER\_IRQClear()

```
enum timer\_status TIMER_IRQClear (
    TIM_Type * base)
```

Сброс прерывания

Аргументы

base	Базовый адрес
------	---------------

Возвращаемые значения

<a href="#">TIMER_Status_Ok</a>	
<a href="#">TIMER_Status_InvalidArgument</a>	

#### 4.21.4.7 TIMER\_IRQDisable()

```
enum timer\_status TIMER_IRQDisable (
    TIM_Type * base)
```

Отключение прерывания

Выключает прерывание в таймере; NVIC не конфигурирует.

Аргументы

base	Базовый адрес
------	---------------

Возвращаемые значения

<a href="#">TIMER_Status_Ok</a>	
<a href="#">TIMER_Status_InvalidArgument</a>	

#### 4.21.4.8 TIMER\_IRQEnable()

```
enum timer\_status TIMER_IRQEnable (
    TIM_Type * base)
```

Включение прерывания

Включает прерывание в таймере; NVIC не конфигурирует.

## Аргументы

base	Базовый адрес
------	---------------

## Возвращаемые значения

<a href="#">TIMER_Status_Ok</a>	
<a href="#">TIMER_Status_InvalidArgument</a>	

## 4.21.4.9 TIMER\_IRQGetStatus()

```
uint32_t TIMER_IRQGetStatus (
    TIM_Type * base)
```

## Чтение статуса прерывания

## Аргументы

base	Базовый адрес
------	---------------

## Возвращаемые значения

0	- прерывание отсутствует
1	- прерывание присутствует

## 4.21.4.10 TIMER\_Reset()

```
enum timer\_status TIMER_Reset (
    TIM_Type * base)
```

## Сброс таймера общего назначения

## Аргументы

base	Таймер
------	--------

## Возвращаемые значения

<a href="#">TIMER_Status_NotSupport</a>	
<a href="#">TIMER_Status_InvalidArgument</a>	

## 4.21.4.11 TIMER\_Run()

```
enum timer\_status TIMER_Run (
    TIM_Type * base)
```

## Запуск таймера общего назначения

## Аргументы

base	Таймер
------	--------

## Возвращаемые значения

<a href="#">TIMER_Status_Ok</a>	
<a href="#">TIMER_Status_InvalidArgument</a>	

## 4.21.4.12 TIMER\_SetConfig()

```
enum timer\_status TIMER_SetConfig (
    TIM_Type * base,
    struct timer\_hardware\_config config,
    enum timer\_work\_mode mode,
    callback\_t callback,
    uint32_t ticks_h)
```

Инициализация структуры таймера общего назначения

## Заметки

Конфигурация аппаратуры таймера используется полностью при режиме работы [TIMER\\_Hardware](#), а при [TIMER\\_Software](#) - только поля work\_type и start\_enable.

## Аргументы

base	Таймер
config	Конфигурация аппаратуры таймера
mode	Режим работы
callback	Функция обратного вызова
ticks↔ _h	Начальное значение для старшей части счетчика обратного счета при режиме работы <a href="#">TIMER_Software</a>

## Возвращаемые значения

<a href="#">TIMER_Status_Ok</a>	
<a href="#">TIMER_Status_InvalidArgument</a>	
<a href="#">TIMER_Status_TimerBusy</a>	

## 4.21.4.13 TIMER\_SetTick()

```
enum timer\_status TIMER_SetTick (
    TIM_Type * base,
    uint64_t ticks)
```

Установка количества тиков

## Аргументы

base	Таймер
ticks	Количество тиков

## Возвращаемые значения

<a href="#">TIMER_Status_NotSupport</a>	
---	--

## 4.21.4.14 TIMER\_Stop()

```
enum timer\_status TIMER_Stop (
    TIM_Type * base)
```

Остановка таймера общего назначения

## Аргументы

base	Таймер
------	--------

## Возвращаемые значения

<a href="#">TIMER_Status_Ok</a>	
<a href="#">TIMER_Status_InvalidArgument</a>	

## 4.22 Драйвер модуля UART

Интерфейс предназначен для организации связи с другими цифровыми устройствами

## Файлы

- файл [hal\\_uart.h](#)  
Интерфейс драйвера UART.
- файл [hal\\_uart\\_dma.h](#)  
Дополнение драйвера UART с пересылкой данных через DMA.

## Структуры данных

- struct [uart\\_config](#)  
Конфигурация UART.
- struct [uart\\_transfer](#)  
Указатель на буфер приема или передачи
- struct [uart\\_handle](#)  
Дескриптор состояния приема/передачи для неблокирующих функций обмена
- struct [\\_uart\\_dma\\_handle](#)  
Дескриптор UART-DMA передачи

## Макросы

- `#define HAL_UART_DRIVER_VERSION (MAKE_VERSION(0, 1, 0))`  
Версия драйвера UART.
- `#define UART_RETRY_TIMES 0U /* 0 - ожидание до получения значения */`  
Количество циклов ожидания
- `#define HAL_UART_DMA_DRIVER_VERSION (MAKE_VERSION(1, 0, 0))`  
Версия драйвера

## Определения типов

- `typedef void(* uart_transfer_callback_t) (UART_Type *base, struct uart\_handle *handle, enum uart\_status status, void *user_data)`  
Callback-функция
- `typedef struct uart\_dma\_handle uart_dma_handle_t`  
Дескриптор UART-DMA передачи
- `typedef void(* uart_dma_transfer_callback_t) (UART_Type *base, uart\_dma\_handle\_t *handle, enum uart\_status status, void *user_data)`  
Функция обратного вызова

## Перечисления

- `enum uart\_status`  
Статусы драйвера UART.
- `enum uart\_interrupt\_enable`  
Конфигурация прерываний для UART.
- `enum uart\_lsr\_flags`  
Флаги состояния UART LSR.
- `enum uart\_parity\_mode`  
Режимы четности UART.
- `enum uart\_stop\_bit\_count`  
Количество стоп-битов для UART.
- `enum uart\_data\_len`  
Количество бит данных в передаваемом символе
- `enum uart\_txfifo\_watermark`  
Триггер уровня заполненности TxFIFO.
- `enum uart\_rxfifo\_watermark`  
Триггер уровня заполненности Rx FIFO.
- `enum uart\_rs485\_mode`  
Режим работы RS485.
- `enum uart\_rs485\_active\_state`  
Активное состояние линии для RS485.

## Функции

- void `UART_TransferCreateHandleDMA` (`UART_Type *base`, `uart_dma_handle_t *handle`, `uart_dma_transfer_callback_t callback`, `void *user_data`, `dma_handle_t *tx_dma_handle`, `dma_handle_t *rx_dma_handle`)  
Функция для инициализации полей дескриптора UART-DMA.
- enum `uart_status` `UART_TransferSendDMA` (`UART_Type *base`, `uart_dma_handle_t *handle`, `struct uart_transfer *xfer`)  
Функция осуществляющая передачу данных по UART, данные в буфер попадают через DMA канал
- enum `uart_status` `UART_TransferReceiveDMA` (`UART_Type *base`, `uart_dma_handle_t *handle`, `struct uart_transfer *xfer`)  
Функция осуществляющая передачу данных по UART, данные в буфер попадают через DMA канал
- void `UART_TransferAbortSendDMA` (`UART_Type *base`, `uart_dma_handle_t *handle`)  
Функция прерывающая передачу данных между UART(TX) и DMA.
- void `UART_TransferAbortReceiveDMA` (`UART_Type *base`, `uart_dma_handle_t *handle`)  
Функция прерывающая передачу данных между UART(RX) и DMA.
- static void `UART_WaitWhileActive` (`UART_Type *base`)  
Ожидание завершения передачи. Выход из функции будет осуществлен по окончании UART транзакций
- static void `UART_DMADescriptorInitTX` (`UART_Type *base`, `dma_descriptor_t *desc`, `uint32_t count`, `uint32_t data_size`, `uint32_t data_width`, `void *src_addr`)  
Инициализация дескрипторов DMA для многоблочной передачи TX.
- static void `UART_DMADescriptorInitRX` (`UART_Type *base`, `dma_descriptor_t *desc`, `uint32_t count`, `uint32_t data_size`, `uint32_t data_width`, `void *dst_addr`)  
Инициализация дескрипторов DMA для многоблочной передачи TX.

## Инициализация и деинициализация

- enum `uart_status` `UART_Init` (`UART_Type *base`, `const struct uart_config *config`, `uint32_t src_clock_hz`)  
Инициализирует модуль UART структурой конфигурации пользователя и частотой периферии
- enum `uart_status` `UART_Deinit` (`UART_Type *base`)  
Деинициализирует модуль UART.
- enum `uart_status` `UART_GetDefaultConfig` (`struct uart_config *config`)  
Получает структуру конфигурации по умолчанию
- enum `uart_status` `UART_SetBaudRate` (`UART_Type *base`, `uint32_t baudrate_bps`, `uint32_t src_clock_hz`)  
Устанавливает скорость модуля UART.

## Состояние

- static `uint32_t` `UART_GetStatusFlags` (`UART_Type *base`)  
Извлекает флаги состояния UART.



## Включение/выключение и настройка прерываний

- static void [UART\\_EnableInterrupts](#) (UART\_Type \*base, uint32\_t mask)  
Разрешает прерывания UART в соответствии с предоставленной маской
- static void [UART\\_DisableInterrupts](#) (UART\_Type \*base, uint32\_t mask)  
Отключает прерывания UART в соответствии с предоставленной маской
- static uint32\_t [UART\\_GetEnabledInterrupts](#) (UART\_Type \*base)  
Запрос маски включенных прерываний в UART.
- static void [UART\\_SetRxFifoWatermark](#) (UART\_Type \*base, enum [uart\\_rxfifo\\_watermark](#) water)  
Устанавливает триггер уровня заполненности RxFIFO.
- static void [UART\\_SetTxFifoWatermark](#) (UART\_Type \*base, enum [uart\\_txfifo\\_watermark](#) water)  
Устанавливает триггер уровня заполненности TxFIFO.

## Включение/выключение и настройка расширенных режимов работы UART

- static void [UART\\_SetLoopback](#) (UART\_Type \*base, bool enable)  
Включение/выключение режима петли
- static void [UART\\_SetIr](#) (UART\_Type \*base, bool enable)  
Включение/выключение инфракрасного режима работы
- static void [UART\\_SetRs485](#) (UART\_Type \*base, bool enable)  
Включение/выключение RS485 режима работы
- static void [UART\\_Rs485Mode](#) (UART\_Type \*base, enum [uart\\_rs485\\_mode](#) mode, enum [uart\\_rs485\\_active\\_state](#) de, enum [uart\\_rs485\\_active\\_state](#) re)  
Установка режима работы RS485.

## Прием и передача без использования прерываний

- static void [UART\\_WriteByte](#) (UART\_Type \*base, uint8\_t data)  
Записывает данные на передачу в регистр передачи
- static void [UART\\_WriteByteWait](#) (UART\_Type \*base, uint8\_t data)  
Записывает данные на передачу в регистр передачи с ожиданием освобождения места
- static uint8\_t [UART\\_ReadByte](#) (UART\_Type \*base)  
Вычитывает байт из регистра приема
- static uint8\_t [UART\\_ReadByteWait](#) (UART\_Type \*base)  
Вычитывает байт из регистра приема с ожиданием получения
- static uint8\_t [UART\\_GetRxFifoCount](#) (UART\_Type \*base)  
Получить количество байтов в RxFIFO.
- static uint8\_t [UART\\_GetTxFifoCount](#) (UART\_Type \*base)  
Получить количество байтов в TxFIFO.
- enum [uart\\_status](#) [UART\\_WriteBlocking](#) (UART\_Type \*base, const uint8\_t \*data, size\_t length)  
Записывает в регистр TX с использованием метода блокировки
- enum [uart\\_status](#) [UART\\_ReadBlocking](#) (UART\_Type \*base, uint8\_t \*data, size\_t length)  
Чтение регистра данных RX с использованием метода блокировки

### Прием данных через прерывания с использованием буферов

- enum `uart_status` `UART_TransferStartRingBuffer` (`UART_Type *base`, struct `uart_handle` \*handle, `uint8_t *ring_buffer`, `size_t ring_buffer_size`)  
Инициализация кольцевого буфера на прием
- enum `uart_status` `UART_TransferStopRingBuffer` (`UART_Type *base`, struct `uart_handle` \*handle)  
Прерывает фоновую передачу и удаляет кольцевой буфер
- `size_t` `UART_TransferGetRxRingBufferLength` (struct `uart_handle` \*handle)  
Получить длину данных, принятых в кольцевом RX буфере
- enum `uart_status` `UART_TransferReceiveNonBlocking` (`UART_Type *base`, struct `uart_handle` \*handle, struct `uart_transfer` \*xfer, `size_t *received_bytes`)  
Прием данных в асинхронном режиме (без ожидания) по прерыванию
- enum `uart_status` `UART_TransferAbortReceive` (`UART_Type *base`, struct `uart_handle` \*handle)  
Отмена приема данных по прерыванию через линейный буфер в дескрипторе
- enum `uart_status` `UART_TransferGetReceiveCount` (`UART_Type *base`, struct `uart_handle` \*handle, `uint32_t *count`)  
Возвращает количество принятых байтов

### Отправка данных через прерывания с использованием буферов

- enum `uart_status` `UART_TransferSendNonBlocking` (`UART_Type *base`, struct `uart_handle` \*handle, struct `uart_transfer` \*xfer)  
Передает буфер данных по прерыванию
- enum `uart_status` `UART_TransferAbortSend` (`UART_Type *base`, struct `uart_handle` \*handle)  
Останавливает передачу данных, управляемую прерыванием
- enum `uart_status` `UART_TransferGetSendCount` (`UART_Type *base`, struct `uart_handle` \*handle, `uint32_t *count`)  
Возвращает количество байтов, отправленных в шину
- enum `uart_status` `UART_TransferStartTxRingBuffer` (`UART_Type *base`, struct `uart_handle` \*handle, `uint8_t *tx_ring_buffer`, `size_t ring_buffer_size`)  
Инициализация кольцевого буфера на передачу
- `size_t` `UART_TransferGetTxRingBufferLength` (struct `uart_handle` \*handle)  
Получить количество байт данных для отправки в кольцевом Tx буфере
- enum `uart_status` `UART_WriteTxRing` (`UART_Type *base`, struct `uart_handle` \*handle, const `uint8_t *data`, `size_t *length`)  
Передать линейный буфер на передачу через кольцевой буфер
- enum `uart_status` `UART_TransferStopTxRingBuffer` (`UART_Type *base`, struct `uart_handle` \*handle)  
Отключение кольцевого буфера передатчика

### Прием и передача через прерывания с использованием буферов

- enum `uart_status` `UART_TransferCreateHandle` (`UART_Type *base`, struct `uart_handle` \*handle, `uart_transfer_callback_t` callback, void \*user\_data)  
Инициализирует дескриптор UART.
- void `UART_TransferHandleIRQ` (`UART_Type *base`, struct `uart_handle` \*handle)  
Функция-обработчик UART IRQ.

### 4.22.1 Подробное описание

Интерфейс предназначен для организации связи с другими цифровыми устройствами

Драйвер позволяет производить прием/передачу по последовательному асинхронному интерфейсу в режиме ожидания (polling) и режиме без ожидания (interrupt). Поддерживает некоторые расширенные возможности: IR-режим, режим петли, режим RS485. Использует аппаратный Rx и Tx FIFO.

### 4.22.2 Перечисления

#### 4.22.2.1 uart\_data\_len

enum `uart_data_len`

Количество бит данных в передаваемом символе

Элементы перечислений

UART_5BitsPerChar	5 бит на символ
UART_6BitsPerChar	6 бит на символ
UART_7BitsPerChar	7 бит на символ
UART_8BitsPerChar	8 бит на символ

#### 4.22.2.2 uart\_interrupt\_enable

enum `uart_interrupt_enable`

Конфигурация прерываний для UART.

Элементы перечислений

UART_ThresoldInterruptEnable	0x80 По порогу
UART_ModemInterruptEnable	0x08 По статусу модема
UART_RxLineInterruptEnable	0x04 По состоянию линии приема
UART_TxInterruptEnable	0x02 По опустошении регистра передатчика
UART_RxInterruptEnable	0x01 По доступности полученных данных или при включенном FIFO, прерывания по таймауту входных данных
UART_AllInterruptsEnable	0x8f Все прерывания

#### 4.22.2.3 uart\_lsr\_flags

enum `uart_lsr_flags`

Флаги состояния UART LSR.

Элементы перечислений

UART_LSR_FlagRxError	Суммарный бит ошибки приемника, сбрасывается при чтении LSR
UART_LSR_FlagTxHwEmpty	Бит отсутствия передаваемых данных в FIFO буфере и сдвиговом регистре передатчика
UART_LSR_FlagTxSwEmpty	Бит отсутствия данных в буфере передатчика
UART_LSR_FlagRxLinebreakError	Ошибка обрыв линии, сбрасывается при чтении LSR
UART_LSR_FlagRxFrameError	Ошибка кадрирования LSR
UART_LSR_FlagRxParityError	Ошибка четности
UART_LSR_FlagRxOverflowError	Ошибка переполнения, бит сбрасывается при чтении содержимого регистра LSR
UART_LSR_FlagRxReady	Есть данные в приемнике, которые еще не были прочитаны

#### 4.22.2.4 uart\_parity\_mode

enum [uart\\_parity\\_mode](#)

Режимы четности UART.

Элементы перечислений

UART_ParityOdd	Тип - нечетная
UART_ParityEven	Тип - четная

#### 4.22.2.5 uart\_rs485\_active\_state

enum [uart\\_rs485\\_active\\_state](#)

Активное состояние линии для RS485.

Элементы перечислений

UART_RS485_ActiveStateHigh	Активный уровень для линии высокий
UART_RS485_ActiveStateLow	Активный уровень для линии низкий

#### 4.22.2.6 uart\_rs485\_mode

enum [uart\\_rs485\\_mode](#)

Режим работы RS485.

Элементы перечислений

UART_RS485_ModeFullDuplex	Дуплекс
UART_RS485_ModeHalfDuplexManual	Полудуплекс: переключение направления передачи вручную
UART_RS485_ModeHalfDuplexAuto	Полудуплекс: автопереключение направления передачи

## 4.22.2.7 uart\_rxfifo\_watermark

enum [uart\\_rxfifo\\_watermark](#)

Триггер уровня заполненности RxFIFO.

Элементы перечислений

UART_RxFifoOneChar	В RxFIFO - 1 символ
UART_RxFifoQuarterFull	RxFIFO заполнен на четверть, 1/4
UART_RxFifoHalfFull	RxFIFO заполнен на половину, 1/2
UART_RxFifoTwoToFull	RxFIFO на 2 меньше чем полный

## 4.22.2.8 uart\_status

enum [uart\\_status](#)

Статусы драйвера UART.

Элементы перечислений

UART_Status_Ok	Успешно
UART_Status_Fail	Провал
UART_Status_ReadOnly	Только чтение
UART_Status_InvalidArgument	Неверный аргумент
UART_Status_Timeout	Отказ по таймауту
UART_Status_NoTransferInProgress	Нет текущей передачи данных
UART_Status_TxBusy	Передатчик занят
UART_Status_RxBusy	Ресивер занят
UART_Status_TxIdle	Передатчик простаивает
UART_Status_RxIdle	Приемник простаивает
UART_Status_TxError	Ошибка в TxFIFO
UART_Status_RxError	Ошибка в RxFIFO
UART_Status_RxRingBufferOverrun	Ошибка в кольцевом буфере Rx
UART_Status_RxFifoBufferOverrun	Ошибка переполнения hw RxFIFO буфера
UART_Status_BreakLineError	Ошибка обрыва линии
UART_Status_FramingError	Ошибка кадра
UART_Status_ParityError	Ошибка четности
UART_Status_BaudrateNotSupport	Скорость передачи не поддерживается для текущего источника синхронизации
UART_Status_TxRingBufferNull	Кольцевой буфер передатчика не инициализирован

## 4.22.2.9 uart\_stop\_bit\_count

enum [uart\\_stop\\_bit\\_count](#)

Количество стоп-битов для UART.

Элементы перечислений

UART_OneStopBit	1 стоп бит
UART_TwoOrOneAndHalfStopBit	1.5 или 2 стоп бит, зависит от количества бит данных в передаваемом символе

#### 4.22.2.10 uart\_txfifo\_watermark

enum [uart\\_txfifo\\_watermark](#)

Триггер уровня заполненности TxFIFO.

Элементы перечислений

UART_TxFifoEmpty	TxFIFO пуст
UART_TxFifoTwoChars	В TxFIFO - 2 символа
UART_TxFifoQuarterFull	TxFIFO заполнен на четверть, 1/4
UART_TxFifoHalfFull	TxFIFO заполнен на половину, 1/2

### 4.22.3 Функции

#### 4.22.3.1 UART\_Deinit()

enum [uart\\_status](#) UART\_Deinit (  
UART\_Type \* base)

Деинициализирует модуль UART.

Функция ожидает завершения передачи, отключает передачу и прием и отключает синхронизацию модуля UART.

Возвращаемые значения

<a href="#">UART_Status_Ok</a>	
<a href="#">UART_Status_InvalidArgument</a>	

#### 4.22.3.2 UART\_DisableInterrupts()

```
static void UART_DisableInterrupts (  
    UART_Type * base,  
    uint32_t mask) [inline], [static]
```

Отключает прерывания UART в соответствии с предоставленной маской

Эта функция отключает прерывания UART в соответствии с предоставленной маской. Маска - это логическое ИЛИ членов перечисления ([uart\\_interrupt\\_enable](#)). В этом примере показано, как отключить пустое прерывание TX и полное прерывание RX:

```
UART_DisableInterrupts(UART1, UART_TxInterruptEnable | UART_RxInterruptEnable);
```

## Аргументы

base	Указатель на базовый адрес UART
mask	Маска запрещаемых прерываний

## 4.22.3.3 UART\_DMADescriptorInitRX()

```
static void UART_DMADescriptorInitRX (
    UART_Type * base,
    dma_descriptor_t * desc,
    uint32_t count,
    uint32_t data_size,
    uint32_t data_width,
    void * dst_addr) [inline], [static]
```

Инициализация дескрипторов DMA для многоблочной передачи TX.

## Аргументы

base	Базовый адрес SPI
desc	Указатель на массив дескрипторов
count	Количество DMA дескрипторов
data_size	Общее число передаваемых данных
data_width	Ширина одного передаваемого слова
dst_addr	Адрес Приемника

## 4.22.3.4 UART\_DMADescriptorInitTX()

```
static void UART_DMADescriptorInitTX (
    UART_Type * base,
    dma_descriptor_t * desc,
    uint32_t count,
    uint32_t data_size,
    uint32_t data_width,
    void * src_addr) [inline], [static]
```

Инициализация дескрипторов DMA для многоблочной передачи TX.

## Аргументы

base	Базовый адрес SPI
desc	Указатель на массив дескрипторов
count	Количество DMA дескрипторов
data_size	Общее число передаваемых данных
data_width	Ширина одного передаваемого слова
src_addr	Адрес Источника

#### 4.22.3.5 UART\_EnableInterrupts()

```
static void UART_EnableInterrupts (
    UART_Type * base,
    uint32_t mask) [inline], [static]
```

Разрешает прерывания UART в соответствии с предоставленной маской

Эта функция разрешает прерывания UART в соответствии с предоставленной маской. Маска - это логическое ИЛИ членов перечисления ([uart\\_interrupt\\_enable](#)). Например, чтобы разрешить прерывание TX и прерывание RX:

```
UART_EnableInterrupts(UART1, UART_TxInterruptEnable | UART_RxInterruptEnable);
```

Аргументы

base	Указатель на базовый адрес UART
mask	Маска разрешаемых прерываний

#### 4.22.3.6 UART\_GetDefaultConfig()

```
enum uart\_status UART_GetDefaultConfig (
    struct uart\_config * config)
```

Получает структуру конфигурации по умолчанию

Эта функция инициализирует структуру конфигурации UART значением по умолчанию:

```
config->baudrate_bps      = 115200U;
config->enable_parity     = false;
config->stop_bit_count    = UART_OneStopBit;
config->bit_count_per_char = UART_8BitPerChar;
config->enable_rxfifo      = true;
config->enable_txfifo      = true;
config->enable_loopback    = false;
config->enable_infrared     = false;
config->enable_hardware_flow_control = false;
config->break_line         = false;
UART\_Init(UART1, &uart\_config, 20000000U);
```

Аргументы

config	Указатель на структуру конфигурации
--------	-------------------------------------

Возвращаемые значения

<a href="#">UART_Status_Ok</a>	
<a href="#">UART_Status_InvalidArgument</a>	

#### 4.22.3.7 UART\_GetEnabledInterrupts()

```
static uint32_t UART_GetEnabledInterrupts (
    UART_Type * base) [inline], [static]
```

Запрос маски включенных прерываний в UART.

Запрос маски включенных прерываний в UART; единицы в соответствующих разрядах соответствуют включенным прерываниям.



Аргументы

base	Указатель на базовый адрес UART
------	---------------------------------

Возвращает

Маска включенных прерываний

#### 4.22.3.8 UART\_GetRxFifoCount()

```
static uint8_t UART_GetRxFifoCount (
    UART_Type * base)  [inline], [static]
```

Получить количество байтов в RxFIFO.

Аргументы

base	Указатель на базовый адрес UART
------	---------------------------------

Возвращает

Количество байт в RxFIFO

#### 4.22.3.9 UART\_GetStatusFlags()

```
static uint32_t UART_GetStatusFlags (
    UART_Type * base)  [inline], [static]
```

Извлекает флаги состояния UART.

Функция получает все флаги состояния UART, флаги возвращаются как логические ИЛИ значение `uart_lsr_flags`. Чтобы проверить конкретный статус, сравните возвращаемое значение с перечислителями в `uart_lsr_flags`. Например, чтобы проверить, пуст ли TX:

```
if (UART_LSR_FlagTxHwEmpty & UART_GetStatusFlags(UART1))
{
    ...
}
```

Аргументы

base	Указатель на базовый адрес UART
------	---------------------------------

Возвращает

Маска флагов состояния UART

#### 4.22.3.10 UART\_GetTxFifoCount()

```
static uint8_t UART_GetTxFifoCount (
    UART_Type * base)  [inline], [static]
```

Получить количество байтов в TxFIFO.

## Аргументы

base	Указатель на базовый адрес UART
------	---------------------------------

## Возвращает

Количество байтов в TxFIFO

## 4.22.3.11 UART\_Init()

```
enum uart_status UART_Init (
    UART_Type * base,
    const struct uart_config * config,
    uint32_t src_clock_hz)
```

Инициализирует модуль UART структурой конфигурации пользователя и частотой периферии

Эта функция конфигурирует модуль UART с пользовательскими настройками. Пользователь может настроить конфигурацию структуры, а также получить конфигурацию по умолчанию с помощью функции [UART\\_GetDefaultConfig](#).

Пример ниже показывает, как использовать эту функцию для настройки UART:

```
config->baudrate_bps      = 115200U;
config->enable_parity     = false;
config->stop_bit_count    = UART_OneStopBit;
config->bit_count_per_char = UART_8BitPerChar;
config->enable_rxfifo     = true;
config->enable_txfifo     = true;
config->enable_loopback   = false;
config->enable_infrared   = false;
config->enable_hardware_flow_control = false;
config->break_line        = false;
UART_Init(UART1, &uart_config, 20000000U);
```

## Аргументы

base	Указатель на базовый адрес UART
config	Указатель на определяемую пользователем структуру конфигурации
src_clock_hz	Тактовая частота источника в Гц

## Возвращаемые значения

<a href="#">UART_Status_Ok</a>	
<a href="#">UART_Status_InvalidArgument</a>	

## 4.22.3.12 UART\_ReadBlocking()

```
enum uart_status UART_ReadBlocking (
    UART_Type * base,
    uint8_t * data,
    size_t length)
```

Чтение регистра данных RX с использованием метода блокировки

Эта функция опрашивает регистр RX, ожидает, пока регистр RX будет заполнен или пока RX FIFO не будет иметь данные и читать данные из регистра TX.

## Аргументы

base	Указатель на базовый адрес UART
data	Указатель на начальный адрес буфера для хранения полученных данных
length	Размер буфера

## Возвращаемые значения

<a href="#">UART_Status_Ok</a>	
<a href="#">UART_Status_Timeout</a>	
<a href="#">UART_Status_RxError</a>	
<a href="#">UART_Status_FramingError</a>	
<a href="#">UART_Status_ParityError</a>	

## 4.22.3.13 UART\_ReadByte()

```
static uint8_t UART_ReadByte (
    UART_Type * base)  [inline], [static]
```

Вычитывает байт из регистра приема

Читает байт из регистра приема RBR. Верхний уровень должен проверить, что в регистре RBR что-то есть перед вызовом.

## Аргументы

base	Указатель на базовый адрес UART
------	---------------------------------

## Возвращает

Байт, считанный из регистра

## 4.22.3.14 UART\_ReadByteWait()

```
static uint8_t UART_ReadByteWait (
    UART_Type * base)  [inline], [static]
```

Вычитывает байт из регистра приема с ожиданием получения

Читает из регистра приема RBR. Если в регистре нет данных, функция будет ждать получения хотя бы одного байта.

## Аргументы

base	Указатель на базовый адрес UART
------	---------------------------------

## Возвращает

Байт, считанный из регистра

## 4.22.3.15 UART\_Rs485Mode()

```
static void UART_Rs485Mode (
    UART_Type * base,
    enum uart\_rs485\_mode mode,
    enum uart\_rs485\_active\_state de,
    enum uart\_rs485\_active\_state re)  [inline], [static]
```

Установка режима работы RS485.

Аргументы

base	Указатель на базовый адрес UART
mode	Режим работы RS485
de	<a href="#">uart_rs485_active_state</a>
re	<a href="#">uart_rs485_active_state</a>

## 4.22.3.16 UART\_SetBaudRate()

```
enum uart\_status UART_SetBaudRate (
    UART_Type * base,
    uint32_t baudrate_bps,
    uint32_t src_clock_hz)
```

Устанавливает скорость модуля UART.

Эта функция настраивает скорость передачи модуля UART. Используется для обновления скорости модуля после его инициализации с помощью [UART\\_Init](#) :

```
UART\_SetBaudRate(UART1, 115200U, 20000000U);
```

Аргументы

base	Указатель на базовый адрес UART
baudrate_bps	Устанавливаемая скорость передачи данных
src_clock_hz	Тактовая частота источника в Гц

Возвращаемые значения

<a href="#">UART_Status_Ok</a>	
<a href="#">UART_Status_InvalidArgument</a>	

## 4.22.3.17 UART\_SetIr()

```
static void UART_SetIr (
    UART_Type * base,
    bool enable)  [inline], [static]
```

Включение/выключение инфракрасного режима работы

## Аргументы

base	Указатель на базовый адрес UART
enable	Включение/выключение режима

## 4.22.3.18 UART\_SetLoopback()

```
static void UART_SetLoopback (
    UART_Type * base,
    bool enable) [inline], [static]
```

Включение/выключение режима петли

## Аргументы

base	Указатель на базовый адрес UART
enable	Включение/выключение режима

## 4.22.3.19 UART\_SetRs485()

```
static void UART_SetRs485 (
    UART_Type * base,
    bool enable) [inline], [static]
```

Включение/выключение RS485 режима работы

## Аргументы

base	Указатель на базовый адрес UART
enable	Включение/выключение режима

## 4.22.3.20 UART\_SetRxFifoWatermark()

```
static void UART_SetRxFifoWatermark (
    UART_Type * base,
    enum uart\_rxfifo\_watermark water) [inline], [static]
```

Устанавливает триггер уровня заполненности RxFIFO.

## Аргументы

base	Указатель на базовый адрес UART
water	Триггер уровня заполненности RxFIFO

## 4.22.3.21 UART\_SetTxFifoWatermark()

```
static void UART_SetTxFifoWatermark (
    UART_Type * base,
    enum uart\_txfifo\_watermark water) [inline], [static]
```

Устанавливает триггер уровня заполненности TxFIFO.

## Аргументы

base	Указатель на базовый адрес UART
water	Триггер уровня заполненности TxFIFO

## 4.22.3.22 UART\_TransferAbortReceive()

```
enum uart\_status UART_TransferAbortReceive (
    UART_Type * base,
    struct uart\_handle * handle)
```

Отмена приема данных по прерыванию через линейный буфер в дескрипторе

Не оказывает никакого влияния на работу кольцевого буфера. Для отмены приема через кольцевой буфер используется [UART\\_TransferStopRingBuffer](#). Пользователь может взять `handle->rx_data←_size`, чтобы узнать сколько еще не принято.

## Аргументы

base	Указатель на базовый адрес UART
handle	Указатель на дескриптор

## Возвращаемые значения

<a href="#">UART_Status_Ok</a>	
<a href="#">UART_Status_InvalidArgument</a>	

## 4.22.3.23 UART\_TransferAbortReceiveDMA()

```
void UART_TransferAbortReceiveDMA (
    UART_Type * base,
    uart\_dma\_handle\_t * handle)
```

Функция прерывающая передачу данных между UART(RX) и DMA.

## Аргументы

base	Базовый адрес UART
handle	Дескриптор UART-DMA

## 4.22.3.24 UART\_TransferAbortSend()

```
enum uart\_status UART_TransferAbortSend (
    UART_Type * base,
    struct uart\_handle * handle)
```

Останавливает передачу данных, управляемую прерыванием

Эта функция останавливает отправку данных, управляемую прерыванием. Пользователь может получить остаток, чтобы узнать сколько байтов еще не отправлено.

## Аргументы

base	Указатель на базовый адрес UART
handle	Указатель на дескриптор

## Возвращаемые значения

UART_Status_Ok	
UART_Status_InvalidArgument	

## 4.22.3.25 UART\_TransferAbortSendDMA()

```
void UART_TransferAbortSendDMA (
    UART_Type * base,
    uart_dma_handle_t * handle)
```

Функция прерывающая передачу данных между UART(TX) и DMA.

## Аргументы

base	Базовый адрес UART
handle	Дескриптор UART-DMA

## 4.22.3.26 UART\_TransferCreateHandle()

```
enum uart_status UART_TransferCreateHandle (
    UART_Type * base,
    struct uart_handle * handle,
    uart_transfer_callback_t callback,
    void * user_data)
```

Инициализирует дескриптор UART.

Для указанного экземпляра UART требуется вызвать эту функцию единожды, чтобы получить инициализированный дескриптор.

## Аргументы

base	Указатель на базовый адрес UART
handle	Указатель на дескриптор
callback	Указатель на функцию обратного вызова
user_data	Указатель на параметр функции обратного вызова

## Возвращаемые значения

UART_Status_Ok	
UART_Status_InvalidArgument	

## 4.22.3.27 UART\_TransferCreateHandleDMA()

```
void UART_TransferCreateHandleDMA (
    UART_Type * base,
    uart_dma_handle_t * handle,
    uart_dma_transfer_callback_t callback,
    void * user_data,
    dma_handle_t * tx_dma_handle,
    dma_handle_t * rx_dma_handle)
```

Функция для инициализации полей дескриптора UART-DMA.

Аргументы

base	Базовый адрес UART
handle	Дескриптор UART-DMA передачи
callback	Функция обратного вызова
user_data	Пользовательские данные
tx_dma_handle	Дескриптор DMA для передачи данных
rx_dma_handle	Дескриптор DMA для приема данных

## 4.22.3.28 UART\_TransferGetReceiveCount()

```
enum uart_status UART_TransferGetReceiveCount (
    UART_Type * base,
    struct uart_handle * handle,
    uint32_t * count)
```

Возвращает количество принятых байтов

Аргументы

base	Указатель на базовый адрес UART
handle	Указатель на дескриптор
count	Указатель, возвращает количество принятых байтов

Возвращаемые значения

UART_Status_Ok	
UART_Status_InvalidArgument	
UART_Status_NoTransferInProgress	

## 4.22.3.29 UART\_TransferGetRxRingBufferLength()

```
size_t UART_TransferGetRxRingBufferLength (
    struct uart_handle * handle)
```

Получить длину данных, принятых в кольцевом RX буфере



## Аргументы

handle	Указатель на дескриптор
--------	-------------------------

## Возвращает

Длина данных, принятых в кольцевом RX буфере

## 4.22.3.30 UART\_TransferGetSendCount()

```
enum uart\_status UART_TransferGetSendCount (
    UART_Type * base,
    struct uart\_handle * handle,
    uint32_t * count)
```

Возвращает количество байтов, отправленных в шину

Эта функция возвращает количество байтов, отправленных в шину по прерыванию.

## Аргументы

base	Указатель на базовый адрес UART
handle	Указатель на дескриптор
count	Указатель, возвращает количество байтов отправленных в шину

## Возвращаемые значения

<a href="#">UART_Status_Ok</a>	
<a href="#">UART_Status_InvalidArgument</a>	
<a href="#">UART_Status_NoTransferInProgress</a>	

## 4.22.3.31 UART\_TransferGetTxRingBufferLength()

```
size_t UART_TransferGetTxRingBufferLength (
    struct uart\_handle * handle)
```

Получить количество байт данных для отправки в кольцевом Tx буфере

## Аргументы

handle	Указатель на дескриптор
--------	-------------------------

## Возвращает

Количество байт

## 4.22.3.32 UART\_TransferHandleIRQ()

```
void UART_TransferHandleIRQ (
    UART_Type * base,
    struct uart\_handle * handle)
```

Функция-обработчик UART IRQ.

Эта функция обрабатывает запросы на передачу и прием UART IRQ.

## Аргументы

base	Указатель на базовый адрес UART
handle	Указатель на дескриптор

## 4.22.3.33 UART\_TransferReceiveDMA()

```
enum uart\_status UART_TransferReceiveDMA (
    UART_Type * base,
    uart\_dma\_handle\_t * handle,
    struct uart\_transfer * xfer)
```

Функция осуществляющая передачу данных по UART, данные в буфер попадают через DMA канал

## Аргументы

base	Базовый адрес UART
handle	Дескриптор UART-DMA
xfer	Структура, содержащая указатель на буфер приема и размер принимаемых данных

## Возвращаемые значения

#UART_DMA_Status_InvalidArgument	
#UART_DMA_Status_RxBusy	
#UART_DMA_Status_DMA_Busy	
#UART_DMA_Status_Fail	
#UART_DMA_Status_Success	

## 4.22.3.34 UART\_TransferReceiveNonBlocking()

```
enum uart\_status UART_TransferReceiveNonBlocking (
    UART_Type * base,
    struct uart\_handle * handle,
    struct uart\_transfer * xfer,
    size_t * received_bytes)
```

Прием данных в асинхронном режиме (без ожидания) по прерыванию

Эта функция получает данные по прерыванию. Неблокирующая функция, которая возвращается, не дожидаясь получения всех ожидаемых данных.

Если кольцевой буфер RX используется и не пуст, данные из кольцевого буфера копируются в `xfer->rx_data`, а параметр `received_bytes` показывает, сколько байтов скопировано из кольцевого буфера. После копирования, если данных в кольцевом буфере недостаточно для чтения, прием запрос сохраняется драйвером UART. Когда поступают новые данные, запрос на получение обслуживается в первую очередь. Когда все данные получены, драйвер UART уведомляет верхний уровень через функцию обратного вызова с параметром состояния [UART\\_Status\\_RxIdle](#).

Например: верхнему уровню требуется 10 байтов, но в кольцевом буфере всего 5 байтов. 5 байтов копируются в данные `xfer->rx_data`, и эта функция возвращается с параметром `received_bytes` установлен в 5. Для оставшихся 5 байтов вновь поступившие данные сохраняются в `xfer->data[5..10]`. При получении 5 байтов драйвер UART уведомляет верхний уровень. Если кольцевой буфер RX не включен, эта функция разрешает прерывание и RX для приема данные в `xfer->data[]`. Когда все данные получены, уведомляется верхний уровень.

## Аргументы

base	Указатель на базовый адрес UART
handle	Указатель на дескриптор
xfer	Указатель на структуру с указателем на линейный буфер приема или передачи
received_bytes	Указатель на количество байтов полученных напрямую из кольцевого буфера

## Возвращаемые значения

<a href="#">UART_Status_Ok</a>	
<a href="#">UART_Status_InvalidArgument</a>	
<a href="#">UART_Status_RxBusy</a>	

## 4.22.3.35 UART\_TransferSendDMA()

```
enum uart\_status UART_TransferSendDMA (
    UART_Type * base,
    uart\_dma\_handle\_t * handle,
    struct uart\_transfer * xfer)
```

Функция осуществляющая передачу данных по UART, данные в буфер попадают через DMA канал

## Аргументы

base	Базовый адрес UART
handle	Дескриптор UART-DMA
xfer	Структура, содержащая указатель на буфер передачи и размер передаваемых данных

## Возвращаемые значения

<a href="#">#UART_DMA_Status_InvalidArgument</a>	
<a href="#">#UART_DMA_Status_TxBusy</a>	
<a href="#">#UART_DMA_Status_DMA_Busy</a>	
<a href="#">#UART_DMA_Status_Fail</a>	
<a href="#">#UART_DMA_Status_Success</a>	

## 4.22.3.36 UART\_TransferSendNonBlocking()

```
enum uart\_status UART_TransferSendNonBlocking (
    UART_Type * base,
    struct uart\_handle * handle,
    struct uart\_transfer * xfer)
```

Передает буфер данных по прерыванию

Отправляет данные по прерыванию. Это неблокирующая функция, которая возвращается напрямую, не дожидаясь записи всех данных в регистр TX. Когда все данные записываются в регистр TX в обработке IRQ, в прерывании происходит callback с передачей в него [UART\\_Status\\_TxIdle](#) в качестве параметра статуса.

## Заметки

[UART\\_Status\\_TxIdle](#) передается на верхний уровень, когда все данные записаны в регистр TX. Однако это не гарантирует, что все данные будут отправлены. Перед отключением TX проверьте:

```
if(UART_LSR_FlagTxHwEmpty & UART_GetStatusFlags(UART1))
```

чтобы убедиться, что передача завершена.

## Аргументы

base	Указатель на базовый адрес UART
handle	Указатель на дескриптор
xfer	Указатель на структуру с указателем на линейный буфер приема или передачи

## Возвращаемые значения

<a href="#">UART_Status_Ok</a>	
<a href="#">UART_Status_InvalidArgument</a>	
<a href="#">UART_Status_TxBusy</a>	

## 4.22.3.37 UART\_TransferStartRingBuffer()

```
enum uart\_status UART_TransferStartRingBuffer (
    UART_Type * base,
    struct uart\_handle * handle,
    uint8_t * ring_buffer,
    size_t ring_buffer_size)
```

## Инициализация кольцевого буфера на прием

Эта функция устанавливает кольцевой буфер для заданного дескриптора UART.

Когда RX кольцевой буфер используется, полученные данные сохраняются в кольцевом буфере даже если пользователь не вызывает [UART\\_TransferReceiveNonBlocking](#) API. Если в кольцевом буфере уже есть данные, пользователь может получить полученные данные из кольцевого буфера напрямую.

При использовании кольцевого буфера RX один байт зарезервирован для внутреннего использования, т.е. если `ring_buffer_size` равно 32, то для сохранения данных используется только 31 байт.

## Аргументы

base	Указатель на базовый адрес UART
handle	Указатель на дескриптор
ring_buffer	Указатель на начальный адрес кольцевого буфера для фоновго приема (NULL - отключение буфера)
ring_buffer_size	Размер кольцевого буфера

## Возвращаемые значения

<a href="#">UART_Status_Ok</a>	
<a href="#">UART_Status_InvalidArgument</a>	

## 4.22.3.38 UART\_TransferStartTxRingBuffer()

```
enum uart\_status UART_TransferStartTxRingBuffer (
    UART_Type * base,
    struct uart\_handle * handle,
    uint8_t * tx_ring_buffer,
    size_t ring_buffer_size)
```

Инициализация кольцевого буфера на передачу

Эта функция устанавливает кольцевой буфер Tx для заданного дескриптора UART.

Аргументы

base	Указатель на базовый адрес UART
handle	Указатель на дескриптор
tx_ring_buffer	Указатель на начальный адрес кольцевого буфера
ring_buffer_size	Размер кольцевого буфера

Возвращаемые значения

<a href="#">UART_Status_Ok</a>	
<a href="#">UART_Status_InvalidArgument</a>	

## 4.22.3.39 UART\_TransferStopRingBuffer()

```
enum uart\_status UART_TransferStopRingBuffer (
    UART_Type * base,
    struct uart\_handle * handle)
```

Прерывает фоновую передачу и удаляет кольцевой буфер

Эта функция прерывает фоновую передачу и удаляет кольцевой буфер.

Аргументы

base	Указатель на базовый адрес UART
handle	Указатель на дескриптор

Возвращаемые значения

<a href="#">UART_Status_Ok</a>	
<a href="#">UART_Status_InvalidArgument</a>	

## 4.22.3.40 UART\_TransferStopTxRingBuffer()

```
enum uart\_status UART_TransferStopTxRingBuffer (
    UART_Type * base,
    struct uart\_handle * handle)
```

Отключение кольцевого буфера передатчика

## Аргументы

base	Указатель на базовый адрес UART
handle	Указатель на дескриптор

## Возвращаемые значения

<a href="#">UART_Status_Ok</a>	
<a href="#">UART_Status_InvalidArgument</a>	

## 4.22.3.41 UART\_WaitWhileActive()

```
static void UART_WaitWhileActive (
    UART_Type * base)  [inline], [static]
```

Ожидание завершения передачи. Выход из функции будет осуществлен по окончании UART транзакций

## Аргументы

base	Базовый адрес UART
------	--------------------

## 4.22.3.42 UART\_WriteBlocking()

```
enum uart\_status UART_WriteBlocking (
    UART_Type * base,
    const uint8_t * data,
    size_t length)
```

Записывает в регистр TX с использованием метода блокировки

Эта функция опрашивает регистр TX, ожидает, пока регистр TX не станет пустым, или пока не появится место в TX FIFO.

## Аргументы

base	Указатель на базовый адрес UART
data	Указатель на начальный адрес данных для записи
length	Размер записываемых данных

## Возвращаемые значения

<a href="#">UART_Status_Ok</a>	
<a href="#">UART_Status_InvalidArgument</a>	
<a href="#">UART_Status_Timeout</a>	

## 4.22.3.43 UART\_WriteByte()

```
static void UART_WriteByte (
    UART_Type * base,
    uint8_t data)  [inline], [static]
```

Записывает данные на передачу в регистр передачи

Эта функция записывает данные на передачу в регистр передачи THR. Верхний уровень должен убедиться, что в нем есть место для записи байта.

Аргументы

base	Указатель на базовый адрес UART
data	Байт для записи

## 4.22.3.44 UART\_WriteByteWait()

```
static void UART_WriteByteWait (
    UART_Type * base,
    uint8_t data)  [inline], [static]
```

Записывает данные на передачу в регистр передачи с ожиданием освобождения места

Эта функция пишет в регистр передачи THR. Будет ожидать освобождения места для записи байта.

Аргументы

base	Указатель на базовый адрес UART
data	Байт для записи

## 4.22.3.45 UART\_WriteTxRing()

```
enum uart\_status UART_WriteTxRing (
    UART_Type * base,
    struct uart\_handle * handle,
    const uint8_t * data,
    size_t * length)
```

Передать линейный буфер на передачу через кольцевой буфер

Аргументы

base	Указатель на базовый адрес UART
handle	Указатель на дескриптор
data	Указатель на буфер, который требуется передать
length	Объем байт на передачу , если не удалось поместить в кольцевой буфер весь объем данных length будет содержать непереданное кол-во байт

Возвращаемые значения

<a href="#">UART_Status_Ok</a>	
<a href="#">UART_Status_InvalidArgument</a>	
<a href="#">UART_Status_TxRingBufferNull</a>	

## 4.23 Драйвер модуля VTU

Драйвер универсального блока таймеров

Файлы

- файл [hal\\_vtu.h](#)  
Интерфейс драйвера универсального блока таймеров

Структуры данных

- struct [vtu\\_config](#)  
Структура для конфигурации VTU.

Определения типов

- typedef void(\* [vtu\\_callback](#)) (VTU\_Type \*base, uint32\_t timer, enum [vtu\\_interrupt\\_control](#) value)  
Функция обратного вызова

Перечисления

- enum [vtu\\_status](#)  
Статусы драйвера универсального блока таймеров
- enum [vtu\\_mode](#)  
Режимы работы тамеров универсального блока таймеров
- enum [vtu\\_capture\\_edge\\_control](#)  
Управление фронтами при режиме захвата
- enum [vtu\\_pwm\\_polarity](#)  
Управление полярностью ШИМ
- enum [vtu\\_interrupt\\_control](#)  
Управление прерываниями
- enum [timer\\_num\\_mode](#)  
Номер таймера и его режим работы для прерываний

Функции

- enum [vtu\\_status](#) [VTU\\_GetLastAPIStatus](#) (void)  
Получение статуса выполнения функции, тип результата которой отличен от enum [vtu\\_status](#).



## Инициализация и деинициализации таймера

- enum `vtu_status VTU_GetDefaultConfig` (struct `vtu_config` \*config)  
Создание конфигурации по умолчанию
- enum `vtu_status VTU_Init` (VTU\_Type \*base, uint32\_t timer, struct `vtu_config` \*config)  
Инициализация таймера
- enum `vtu_status VTU_Deinit` (VTU\_Type \*base, uint32\_t timer)  
Деинициализация таймера

## Функции управления VTU

- enum `vtu_status VTU_EnableTimer` (VTU\_Type \*base, uint32\_t timer, bool enable)  
Разрешение работы таймера
- enum `vtu_status VTU_SetCounter` (VTU\_Type \*base, uint32\_t timer, uint16\_t value, bool extended)  
Установка значения счетчика
- uint16\_t `VTU_GetCounter` (VTU\_Type \*base, uint32\_t timer, bool extended)  
Получение значения счетчика
- enum `vtu_status VTU_SetPrescaler` (VTU\_Type \*base, uint32\_t timer, uint8\_t value)  
Установка значения предделителя
- uint8\_t `VTU_GetPrescaler` (VTU\_Type \*base, uint32\_t timer)  
Получение значения предделителя
- enum `vtu_status VTU_SetPeriodCapture` (VTU\_Type \*base, uint32\_t timer, uint16\_t value, bool extended)  
Установка значения периода генерации шим без учета предделителя
- uint16\_t `VTU_GetPeriodCapture` (VTU\_Type \*base, uint32\_t timer, bool extended)  
Получение значения периода генерации шим без учета предделителя
- enum `vtu_status VTU_SetDutyCycleCapture` (VTU\_Type \*base, uint32\_t timer, uint16\_t value, bool extended)  
Установка периода импульса шим без учета предделителя
- uint16\_t `VTU_GetDutyCycleCapture` (VTU\_Type \*base, uint32\_t timer, bool extended)  
Получение периода импульса шим без учета предделителя
- void `VTU_SetCallback` (uint32\_t timer, `vtu_callback` callback)  
Установка функции обратного вызова
- enum `vtu_status VTU_EnableTimerIRQ` (VTU\_Type \*base, uint32\_t timer, enum `vtu_interrupt_control` value, bool enable, enum `vtu_mode` mode)  
Разрешение работы прерывания
- enum `vtu_interrupt_control VTU_GetTimerIRQ` (VTU\_Type \*base, uint32\_t timer, enum `vtu_mode` mode)  
Получение прерываний
- enum `vtu_status VTU_ClearTimerIRQ` (VTU\_Type \*base, uint32\_t timer, enum `vtu_interrupt_control` values, enum `vtu_mode` mode)  
Очистка прерываний
- enum `vtu_status VTU_SetPWMPolarity` (VTU\_Type \*base, uint32\_t timer, enum `vtu_pwm_polarity` value1, enum `vtu_pwm_polarity` value2, bool use\_value2)  
Установка полярности ШИМ
- enum `vtu_status VTU_GetPWMPolarity` (VTU\_Type \*base, uint32\_t timer, enum `vtu_pwm_polarity` \*value1, enum `vtu_pwm_polarity` \*value2, bool use\_value2)  
Получение полярности ШИМ
- enum `vtu_status VTU_SetCaptureEdgeCtrl` (VTU\_Type \*base, uint32\_t timer, enum `vtu_capture_edge_control` value1, enum `vtu_capture_edge_control` value2, bool use\_value2)  
Установка типа захвата
- enum `vtu_status VTU_GetCaptureEdgeCtrl` (VTU\_Type \*base, uint32\_t timer, enum `vtu_capture_edge_control` \*value1, enum `vtu_capture_edge_control` \*value2, bool use\_value2)  
Получение типа захвата

### 4.23.1 Подробное описание

Драйвер универсального блока таймеров

Драйвер модуля универсального блока таймеров.

### 4.23.2 Перечисления

#### 4.23.2.1 timer\_num\_mode

enum [timer\\_num\\_mode](#)

Номер таймера и его режим работы для прерываний

Элементы перечислений

VTU_Timer0Mode8bit	Таймер 0, режим 8 бит
VTU_Timer0Mode16bit	Таймер 0, режим 16 бит
VTU_Timer1Mode8bit	Таймер 1, режим 8 бит
VTU_Timer2Mode8bit	Таймер 2, режим 8 бит
VTU_Timer2Mode16bit	Таймер 2, режим 16 бит
VTU_Timer3Mode8bit	Таймер 3, режим 8 бит

#### 4.23.2.2 vtu\_capture\_edge\_control

enum [vtu\\_capture\\_edge\\_control](#)

Управление фронтами при режиме захвата

Элементы перечислений

VTU_CaptureRisingEdgeResetNo	Захват по возрастающему фронту, сброса счетчика нет
VTU_CaptureFallingEdgeResetNo	Захват по ниспадающему фронту, сброса счетчика нет
VTU_CaptureRisingEdgeResetYes	Захват по возрастающему фронту, сброс счетчика есть
VTU_CaptureFallingEdgeResetYes	Захват по ниспадающему фронту, сброс счетчика есть
VTU_CaptureBothEdgeResetNo	Захват по возрастающему фронту, сброса счетчика нет
VTU_CaptureBothEdgeResetRisingEdge	Захват по возрастающему фронту, сброс счетчика по возрастающему фронту
VTU_CaptureBothEdgeResetFallingEdge	Захват по возрастающему фронту, сброс счетчика ниспадающему фронту
VTU_CaptureBothEdgeResetBothEdge	Захват по возрастающему фронту, сброс счетчика обоим фронтам

#### 4.23.2.3 vtu\_interrupt\_control

enum [vtu\\_interrupt\\_control](#)

Управление прерываниями

Элементы перечислений

VTU_NoInterrupt	Отключение всех прерываний
VTU_LowByteDutyCycleMatch	По совпадению цикла для первого сдвоенного таймера
VTU_LowBytePeriodMatch	По совпадению периода для первого сдвоенного таймера
VTU_HighByteDutyCycleMatch	По совпадению цикла для второго сдвоенного таймера
VTU_HighBytePeriodMatch	По совпадению периода для второго сдвоенного таймера
VTU_DutyCycleMatch	По совпадению цикла для таймера
VTU_PeriodMatch	По совпадению периода для таймера
VTU_CaptureToPERCAPx	Захват по началу импульса
VTU_CaptureToDTYCAPx	Захват по концу импульса
VTU_CounterOverflow	По переполнению счетчика

#### 4.23.2.4 vtu\_mode

enum [vtu\\_mode](#)

Режимы работы тамеров универсального блока таймеров

Элементы перечислений

VTU_LowPower	Режим остановки таймера
VTU_PWMDual8Bit	Режим сдвоенного 8-битного таймера
VTU_PWM16Bit	Режим 16-битного таймера
VTU_Capture	Режим захвата

#### 4.23.2.5 vtu\_pwm\_polarity

enum [vtu\\_pwm\\_polarity](#)

Управление полярностью ШИМ

Элементы перечислений

VTU_One	Высокий уровень импульса
VTU_Zero	Низкий уровень импульса

#### 4.23.2.6 vtu\_status

enum [vtu\\_status](#)

Статусы драйвера универсального блока таймеров

Элементы перечислений

VTU_Status_Ok	Нет ошибок
VTU_Status_InvalidArgument	Недопустимый аргумент
VTU_Status_TimerBusy	Таймер уже занят
VTU_Status_BadConfigure	Недопустимая конфигурация
VTU_Status_DriverError	Ошибка драйвера
VTU_Status_DualTimerNotCanRun	Сдвоенный таймера не может быть запущен, так как второй таймер уже работает
VTU_Status_TimerNotInit	Таймер не инициализирован

### 4.23.3 Функции

#### 4.23.3.1 VTU\_ClearTimerIRQ()

```
enum vtu_status VTU_ClearTimerIRQ (
    VTU_Type * base,
    uint32_t timer,
    enum vtu_interrupt_control values,
    enum vtu_mode mode)
```

Очистка прерываний

Аргументы

base	Подсистема VTU
timer	Таймер
values	Прерывания
mode	Режим таймера для интерпретации прерываний

Возвращает

Прерывания

#### 4.23.3.2 VTU\_Deinit()

```
enum vtu_status VTU_Deinit (
    VTU_Type * base,
    uint32_t timer)
```

Деинициализация таймера

Функция деинициализации таймера

Аргументы

base	Подсистема VTU
timer	Таймер

Возвращаемые значения

VTU_Status_Ok	
VTU_Status_InvalidArgument	

#### 4.23.3.3 VTU\_EnableTimer()

```
enum vtu_status VTU_EnableTimer (
    VTU_Type * base,
    uint32_t timer,
    bool enable)
```

Разрешение работы таймера

Аргументы

base	Подсистема VTU
timer	Таймер
enable	Разрешение работы (1) или запрещение работы (0)

Возвращает

Статус

#### 4.23.3.4 VTU\_EnableTimerIRQ()

```
enum vtu_status VTU_EnableTimerIRQ (
    VTU_Type * base,
    uint32_t timer,
    enum vtu_interrupt_control value,
    bool enable,
    enum vtu_mode mode)
```

Разрешение работы прерывания

Аргументы

base	Подсистема VTU
timer	Таймер
value	Прерывания
enable	Разрешены прерывания по маске (1) или запрещены прерывания по маске (0)
mode	Режим таймера для интерпретации прерываний

Возвращает

Статус

#### 4.23.3.5 VTU\_GetCaptureEdgeCtrl()

```
enum vtu_status VTU_GetCaptureEdgeCtrl (  
    VTU_Type * base,  
    uint32_t timer,  
    enum vtu_capture_edge_control * value1,  
    enum vtu_capture_edge_control * value2,  
    bool use_value2)
```

Получение типа захвата

Аргументы

base	Подсистема VTU
timer	Таймер
value1	Полярность первого сигнала ШИМ
value2	Полярность второго сигнала ШИМ
use_value2	Использовать (1) или не использовать (0) второй сигнал ШИМ

Возвращает

Статус

#### 4.23.3.6 VTU\_GetCounter()

```
uint16_t VTU_GetCounter (  
    VTU_Type * base,  
    uint32_t timer,  
    bool extended)
```

Получение значения счетчика

Аргументы

base	Подсистема VTU
timer	Таймер
extended	Вернуть двухбайтовое (1) или однобайтное (0) значение value

Возвращает

Значение счетчика

#### 4.23.3.7 VTU\_GetDefaultConfig()

```
enum vtu_status VTU_GetDefaultConfig (  
    struct vtu_config * config)
```

Создание конфигурации по умолчанию

Функция инициализации структуры с настройками таймера "по умолчанию"

## Аргументы

config	Конфигурация таймера
--------	----------------------

## Возвращаемые значения

VTU_Status_Ok	
VTU_Status_InvalidArgument	

## 4.23.3.8 VTU\_GetDutyCycleCapture()

```
uint16_t VTU_GetDutyCycleCapture (
    VTU_Type * base,
    uint32_t timer,
    bool extended)
```

Получение периода импульса шим без учета предделителя

## Аргументы

base	Подсистема VTU
timer	Таймер
extended	Вернуть двухбайтное (1) или однобайтное (0) значение value

## Возвращает

Значение счетчика

## 4.23.3.9 VTU\_GetLastAPIStatus()

```
enum vtu_status VTU_GetLastAPIStatus (
    void )
```

Получение статуса выполнения функции, тип результата которой отличен от enum vtu\_status.

## Возвращает

Статус

## 4.23.3.10 VTU\_GetPeriodCapture()

```
uint16_t VTU_GetPeriodCapture (
    VTU_Type * base,
    uint32_t timer,
    bool extended)
```

Получение значения периода генерации шим без учета предделителя

## Аргументы

base	Подсистема VTU
timer	Таймер
extended	Вернуть двухбайтное (1) или однобайтное (0) значение value

## Возвращает

Значение счетчика

## 4.23.3.11 VTU\_GetPrescaler()

```
uint8_t VTU_GetPrescaler (  
    VTU_Type * base,  
    uint32_t timer)
```

## Получение значения предделителя

## Аргументы

base	Подсистема VTU
timer	Таймер

## Возвращает

Значение предделителя

## 4.23.3.12 VTU\_GetPWMPolarity()

```
enum vtu_status VTU_GetPWMPolarity (  
    VTU_Type * base,  
    uint32_t timer,  
    enum vtu_pwm_polarity * value1,  
    enum vtu_pwm_polarity * value2,  
    bool use_value2)
```

## Получение полярности ШИМ

## Аргументы

base	Подсистема VTU
timer	Таймер
value1	Полярность первого сигнала ШИМ
value2	Полярность второго сигнала ШИМ
use_value2	Использовать (1) или не использовать (0) второй сигнал ШИМ

## Возвращает

Статус



## 4.23.3.13 VTU\_GetTimerIRQ()

```
enum vtu_interrupt_control VTU_GetTimerIRQ (
    VTU_Type * base,
    uint32_t timer,
    enum vtu_mode mode)
```

Получение прерываний

Аргументы

base	Подсистема VTU
timer	Таймер
mode	Режим таймера для интерпретации прерываний

Возвращает

Прерывания

## 4.23.3.14 VTU\_Init()

```
enum vtu_status VTU_Init (
    VTU_Type * base,
    uint32_t timer,
    struct vtu_config * config)
```

Инициализация таймера

Функция инициализации таймера с указанными настройками

Аргументы

base	Система VTU
timer	Таймер в системе VTU
config	Конфигурация таймера

Возвращаемые значения

VTU_Status_Ok	
VTU_Status_InvalidArgument	

## 4.23.3.15 VTU\_SetCallback()

```
void VTU_SetCallback (
    uint32_t timer,
    vtu_callback callback)
```

Установка функции обратного вызова

## Аргументы

callback	Указатель на пользовательскую функцию обратного вызова
----------	--

## 4.23.3.16 VTU\_SetCaptureEdgeCtrl()

```
enum vtu_status VTU_SetCaptureEdgeCtrl (  
    VTU_Type * base,  
    uint32_t timer,  
    enum vtu_capture_edge_control value1,  
    enum vtu_capture_edge_control value2,  
    bool use_value2)
```

## Установка типа захвата

## Аргументы

base	Подсистема VTU
timer	Таймер
value1	Полярность первого сигнала ШИМ
value2	Полярность второго сигнала ШИМ
use_value2	Использовать (1) или не использовать (0) второй сигнал ШИМ

## Возвращает

Статус

## 4.23.3.17 VTU\_SetCounter()

```
enum vtu_status VTU_SetCounter (  
    VTU_Type * base,  
    uint32_t timer,  
    uint16_t value,  
    bool extended)
```

## Установка значения счетчика

## Аргументы

base	Подсистема VTU
timer	Таймер
value	Значение
extended	Использовать (1) или не использовать (0) двухбайтное значение value

## Возвращает

Статус

## 4.23.3.18 VTU\_SetDutyCycleCapture()

```
enum vtu_status VTU_SetDutyCycleCapture (
    VTU_Type * base,
    uint32_t timer,
    uint16_t value,
    bool extended)
```

Установка периода импульса шим без учета предделителя

Аргументы

base	Подсистема VTU
timer	Таймер
value	Значение
extended	Использовать (1) или не использовать (0) двухбайтное значение value

Возвращает

Статус

## 4.23.3.19 VTU\_SetPeriodCapture()

```
enum vtu_status VTU_SetPeriodCapture (
    VTU_Type * base,
    uint32_t timer,
    uint16_t value,
    bool extended)
```

Установка значения периода генерации шим без учета предделителя

Аргументы

base	Подсистема VTU
timer	Таймер
value	Значение
extended	Использовать (1) или не использовать (0) двухбайтное значение value

Возвращает

Статус

## 4.23.3.20 VTU\_SetPrescaler()

```
enum vtu_status VTU_SetPrescaler (
    VTU_Type * base,
    uint32_t timer,
    uint8_t value)
```

Установка значения предделителя

## Аргументы

base	Подсистема VTU
timer	Таймер
value	Значение

Возвращает

Статус

## 4.23.3.21 VTU\_SetPWMPolarity()

```
enum vtu_status VTU_SetPWMPolarity (
    VTU_Type * base,
    uint32_t timer,
    enum vtu_pwm_polarity value1,
    enum vtu_pwm_polarity value2,
    bool use_value2)
```

Установка полярности ШИМ

## Аргументы

base	Подсистема VTU
timer	Таймер
value1	Полярность первого сигнала ШИМ
value2	Полярность второго сигнала ШИМ
use_value2	Использовать (1) или не использовать (0) второй сигнал ШИМ

Возвращает

Статус

## 4.24 Драйвер модуля WDT

Драйвер сторожевого таймера

## Файлы

- файл [hal\\_wdt.h](#)  
Интерфейс драйвера сторожевого таймера

## Структуры данных

- struct [wdt\\_config](#)  
Структура инициализации сторожевого таймера

## Макросы

- `#define WDT_NUMBER_OF_TIMERS 3`

## Перечисления

- enum `wdt_status`  
Статусы драйвера сторожевого таймера
- enum `wdt_resen_type`  
Управление сбросом при таймауте сторожевого таймера
- enum `wdt_inten_type`  
Управление прерыванием предупреждения от сторожевого таймера\ и разрешением работы таймера

## Инициализация и деинициализации таймера

- enum `wdt_status WDT_GetDefaultConfig` (struct `wdt_config *config`)  
Создание конфигурации по умолчанию
- enum `wdt_status WDT_Init` (WDT\_Type \*base, const struct `wdt_config *config`)  
Инициализация таймера
- enum `wdt_status WDT_Deinit` (WDT\_Type \*base)  
Деинициализация таймера

## Функции управления WDT

- enum `wdt_status WDT_Enable` (WDT\_Type \*base)  
Разрешение работы таймера
- enum `wdt_status WDT_Disable` (WDT\_Type \*base)  
Запрещение работы таймера
- uint32\_t `WDT_GetStatusFlagsRaw` (WDT\_Type \*base)  
Получение немаскированных статусов таймера
- uint32\_t `WDT_GetStatusFlagsMsk` (WDT\_Type \*base)  
Получение маскированных статусов таймера
- enum `wdt_status WDT_ClearStatusFlags` (WDT\_Type \*base, uint32\_t mask)  
Очищение статусов таймера
- enum `wdt_status WDT_SetWarningValue` (WDT\_Type \*base, uint32\_t warning\_value)  
Установка времени срабатывания предупреждения
- enum `wdt_status WDT_SetTimeoutValue` (WDT\_Type \*base, uint32\_t timeout\_count)  
Установка времени таймаута таймера
- uint32\_t `WDT_GetWarningValue` (WDT\_Type \*base)  
Получение значения счетчика
- enum `wdt_status WDT_Refresh` (WDT\_Type \*base)  
Обновление времени сторожевого таймера
- enum `wdt_status WDT_GetLastAPIStatus` ()  
Получение статуса выполнения функции, тип результата которой отличен от enum `wdt_status`.

## 4.24.1 Подробное описание

## Драйвер сторожевого таймера

Драйвер модуля сторожевого таймера управляет сторожевым таймером.

## 4.24.2 Макросы

### 4.24.2.1 WDT\_NUMBER\_OF\_TIMERS

```
#define WDT_NUMBER_OF_TIMERS 3
```

Количество сдвоенных таймеров

## 4.24.3 Перечисления

### 4.24.3.1 wdt\_inten\_type

```
enum wdt_inten_type
```

Управление прерыванием предупреждения от сторожевого таймера\ и разрешением работы таймера

Элементы перечислений

WDT_IntenDisable	Прерывание запрещено, таймер не работает
WDT_IntenEnable	Прерывание разрешено, таймер работает

### 4.24.3.2 wdt\_resen\_type

```
enum wdt_resen_type
```

Управление сбросом при таймауте сторожевого таймера

Элементы перечислений

WDT_ResenDisable	Сброс запрещён
WDT_ResenEnable	Сброс разрешен

### 4.24.3.3 wdt\_status

```
enum wdt_status
```

Статусы драйвера сторожевого таймера

Элементы перечислений

WDT_Status_Ok	Нет ошибок
WDT_Status_InvalidArgument	Недопустимый аргумент
WDT_Status_TimerBusy	Таймер уже занят
WDT_Status_BadConfigure	Недопустимая конфигурация

## 4.24.4 Функции

### 4.24.4.1 WDT\_ClearStatusFlags()

```
enum wdt_status WDT_ClearStatusFlags (
    WDT_Type * base,
    uint32_t mask)
```

Очищение статусов таймера

Аргументы

base	Таймер
mask	Маска статусов

Возвращаемые значения

WDT_Status_Ok	
WDT_Status_InvalidArgument	

#### 4.24.4.2 WDT\_Deinit()

```
enum wdt_status WDT_Deinit (
    WDT_Type * base)
```

Деинициализация таймера

Функция деинициализации таймера

Аргументы

base	Таймер
------	--------

Возвращаемые значения

WDT_Status_Ok	
WDT_Status_InvalidArgument	

#### 4.24.4.3 WDT\_Disable()

```
enum wdt_status WDT_Disable (
    WDT_Type * base)
```

Запрещение работы таймера

Аргументы

base	Таймер
------	--------

Возвращаемые значения

WDT_Status_Ok	
WDT_Status_InvalidArgument	

#### 4.24.4.4 WDT\_Enable()

```
enum wdt_status WDT_Enable (
    WDT_Type * base)
```

Разрешение работы таймера

## Аргументы

base	Таймер
------	--------

## Возвращаемые значения

WDT_Status_Ok	
WDT_Status_InvalidArgument	

## 4.24.4.5 WDT\_GetDefaultConfig()

```
enum wdt_status WDT_GetDefaultConfig (
    struct wdt_config * config)
```

## Создание конфигурации по умолчанию

Функция инициализации структуры с настройками таймера "по умолчанию":

```
config->enableWwdt = true; ...
```

## Аргументы

config	Конфигурация таймера
--------	----------------------

## Возвращаемые значения

WDT_Status_Ok	
WDT_Status_InvalidArgument	

## 4.24.4.6 WDT\_GetLastAPIStatus()

```
enum wdt_status WDT_GetLastAPIStatus ()
```

Получение статуса выполнения функции, тип результата которой отличен от enum wdt\_status.

## Возвращаемые значения

WDT_Status_Ok	
WDT_Status_InvalidArgument	
WDT_Status_TimerBusy	
WDT_Status_BadConfigure	

## 4.24.4.7 WDT\_GetStatusFlagsMsk()

```
uint32_t WDT_GetStatusFlagsMsk (
    WDT_Type * base)
```

Получение маскированных статусов таймера



Аргументы

base	Таймер
------	--------

#### 4.24.4.8 WDT\_GetStatusFlagsRaw()

```
uint32_t WDT_GetStatusFlagsRaw (  
    WDT_Type * base)
```

Получение немаскированных статусов таймера

Аргументы

base	Таймер
------	--------

#### 4.24.4.9 WDT\_GetWarningValue()

```
uint32_t WDT_GetWarningValue (  
    WDT_Type * base)
```

Получение значения счетчика

Аргументы

base	Таймер
------	--------

Возвращает

Значение счетчика

#### 4.24.4.10 WDT\_Init()

```
enum wdt_status WDT_Init (  
    WDT_Type * base,  
    const struct wdt_config * config)
```

Инициализация таймера

Функция инициализации таймера с указанными настройками

Аргументы

base	Таймер
config	Конфигурация таймера

Возвращаемые значения

<a href="#">WDT_Status_Ok</a>	
<a href="#">WDT_Status_InvalidArgument</a>	

#### 4.24.4.11 WDT\_Refresh()

```
enum wdt\_status WDT_Refresh (
    WDT_Type * base)
```

Обновление времени сторожевого таймера

Аргументы

base	Таймер
------	--------

Возвращаемые значения

<a href="#">WDT_Status_Ok</a>	
<a href="#">WDT_Status_InvalidArgument</a>	

#### 4.24.4.12 WDT\_SetTimeoutValue()

```
enum wdt\_status WDT_SetTimeoutValue (
    WDT_Type * base,
    uint32_t timeout_count)
```

Установка времени таймаута таймера

Установка времени таймаута таймера в значениях периода частоты тактирования. Значение времени срабатывания предупреждения равно половине значения времени таймаута таймера

Аргументы

base	Таймер
timeout_count	Время таймаута таймера

Заметки

timeout\_count должно быть четное, минимальное значение равно 2

Возвращаемые значения

<a href="#">WDT_Status_Ok</a>	
<a href="#">WDT_Status_InvalidArgument</a>	

## 4.24.4.13 WDT\_SetWarningValue()

```
enum wdt\_status WDT_SetWarningValue (  
    WDT_Type * base,  
    uint32_t warning_value)
```

Установка времени срабатывания предупреждения

Установка времени срабатывания предупреждения в значениях периода частоты тактирования. Значение таймаута таймера равно удвоенному значению времени срабатывания предупреждения

Аргументы

base	Таймер
warning_value	Время срабатывания,

Заметки

минимальное значение warning\_value равно 1

Возвращаемые значения

<a href="#">WDT_Status_Ok</a>	
<a href="#">WDT_Status_InvalidArgument</a>	



## Глава 5

# Структуры данных

### 5.1 Структура `_can_config`

Структура конфигурации контроллера CAN.

```
#include <hal_can.h>
```

Поля данных

- bool `enable_listen_only`
- bool `enable_loopback_int`
- bool `enable_loopback_ext`
- `can_ptb_config_t` `ptb_config`
- `can_stb_config_t` `stb_config`
- `can_rxb_config_t` `rxb_config`
- `can_timing_config_t` `timing_config`
- `canfd_mode_t` `can_fd_mode`
- `can_frame_filter_config_t` `filter_config`

#### 5.1.1 Подробное описание

Структура конфигурации контроллера CAN.

#### 5.1.2 Поля

##### 5.1.2.1 `can_fd_mode`

```
canfd_mode_t _can_config::can_fd_mode
```

Режим работы по CAN FD

#### 5.1.2.2 enable\_listen\_only

`bool _can_config::enable_listen_only`

Работа только в режиме прослушки шины данных

#### 5.1.2.3 enable\_loopback\_ext

`bool _can_config::enable_loopback_ext`

Включение внешней петли контроллера CAN

#### 5.1.2.4 enable\_loopback\_int

`bool _can_config::enable_loopback_int`

Включение внутренней петли контроллера CAN

#### 5.1.2.5 filter\_config

`can_frame_filter_config_t _can_config::filter_config`

Установки входных фильтров кадров CAN

#### 5.1.2.6 ptb\_config

`can_ptb_config_t _can_config::ptb_config`

Параметры высокоприоритетного буфера

#### 5.1.2.7 rxb\_config

`can_rxb_config_t _can_config::rxb_config`

Параметры буфера приема

#### 5.1.2.8 stb\_config

`can_stb_config_t _can_config::stb_config`

Параметры низкоприоритетного буфера

5.1.2.9 `timing_config`

`can_timing_config_t _can_config::timing_config`

Битовая времянка

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_can.h`

5.2 Структура `_can_frame_filter`

Фильтр принятых кадров CAN.

```
#include <hal_can.h>
```

Поля данных

- `uint32_t id`: 29
- `uint32_t` : 3
- `uint32_t mask`: 29
- `uint32_t accepted_id`: 1
- `uint32_t enable_id_check`: 1
- `uint32_t` : 1

## 5.2.1 Подробное описание

Фильтр принятых кадров CAN.

## 5.2.2 Поля

5.2.2.1 `__pad0__`

`uint32_t _can_frame_filter::__pad0__`

Выравнивание

5.2.2.2 `__pad1__`

`uint32_t _can_frame_filter::__pad1__`

Выравнивание

5.2.2.3 `accepted_id`

`uint32_t _can_frame_filter::accepted_id`

Значение признака расширенного кадра, если его проверка включена (`enable_id_check`)

#### 5.2.2.4 enable\_ide\_check

`uint32_t _can_frame_filter::enable_ide_check`

Проверять ли при фильтрации признак расширенного кадра

#### 5.2.2.5 id

`uint32_t _can_frame_filter::id`

Идентификатор принятого кадра

#### 5.2.2.6 mask

`uint32_t _can_frame_filter::mask`

Маска битов проверки принятого кадра. Для каждого бита идентификатора принятого кадра он проверяется на равенство с битом, заданном в фильтре, если соответствующий бит маски установлен; иначе не проверяется (значение бита идентификатора принятого кадра может быть любым)

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_can.h`

### 5.3 Структура `_can_frame_filter_config`

Конфигурация фильтрации принятых кадров CAN.

```
#include <hal_can.h>
```

Поля данных

- `can_frame_filter_t filter [CAN_NB_OF_FILTERS]`
- `uint8_t nb_filters_used`

#### 5.3.1 Подробное описание

Конфигурация фильтрации принятых кадров CAN.

#### 5.3.2 Поля

##### 5.3.2.1 filter

`can_frame_filter_t _can_frame_filter_config::filter [CAN_NB_OF_FILTERS]`

Массив настроек фильтров CAN



5.3.2.2 `nb_filters_used`

```
uint8_t _can_frame_filter_config::nb_filters_used
```

Количество задействованных фильтров

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_can.h`

5.4 Структура `_can_handle`

Структура дескриптора драйвера CAN.

```
#include <hal_can.h>
```

Поля данных

- `volatile const can_tx_buffer_frame_t * tx_frames_prim`
- `volatile size_t tx_nb_frames_rest_prim`
- `size_t tx_nb_frames_all_prim`
- `volatile const can_tx_buffer_frame_t * tx_frames_sec`
- `volatile size_t tx_nb_frames_rest_sec`
- `size_t tx_nb_frames_all_sec`
- `volatile can_rx_buffer_frame_t * rx_frames`
- `volatile size_t rx_nb_frames_rest`
- `size_t rx_nb_frames_all`
- `can_transfer_callback_t callback`
- `void * user_data`

## 5.4.1 Подробное описание

Структура дескриптора драйвера CAN.

## 5.4.2 Поля

5.4.2.1 `callback`

```
can_transfer_callback_t _can_handle::callback
```

Функция обратного вызова

5.4.2.2 `rx_frames`

```
volatile can_rx_buffer_frame_t* _can_handle::rx_frames
```

Адрес оставшихся кадров для приема

#### 5.4.2.3 rx\_nb\_frames\_all

size\_t \_can\_handle::rx\_nb\_frames\_all

Количество кадров для приема

#### 5.4.2.4 rx\_nb\_frames\_rest

volatile size\_t \_can\_handle::rx\_nb\_frames\_rest

Количество оставшихся кадров для приема

#### 5.4.2.5 tx\_frames\_prim

volatile const [can\\_tx\\_buffer\\_frame\\_t](#)\* \_can\_handle::tx\_frames\_prim

Адрес оставшихся кадров для отправки через высокоприоритетный буфер

#### 5.4.2.6 tx\_frames\_sec

volatile const [can\\_tx\\_buffer\\_frame\\_t](#)\* \_can\_handle::tx\_frames\_sec

Адрес оставшихся кадров для отправки через низкоприоритетный буфер

#### 5.4.2.7 tx\_nb\_frames\_all\_prim

size\_t \_can\_handle::tx\_nb\_frames\_all\_prim

Количество кадров для отправки через высокоприоритетный буфер

#### 5.4.2.8 tx\_nb\_frames\_all\_sec

size\_t \_can\_handle::tx\_nb\_frames\_all\_sec

Количество кадров для отправки через низкоприоритетный буфер

#### 5.4.2.9 tx\_nb\_frames\_rest\_prim

volatile size\_t \_can\_handle::tx\_nb\_frames\_rest\_prim

Количество оставшихся кадров для отправки через высокоприоритетный буфер

#### 5.4.2.10 tx\_nb\_frames\_rest\_sec

volatile size\_t \_can\_handle::tx\_nb\_frames\_rest\_sec

Количество оставшихся кадров для отправки через низкоприоритетный буфер

5.4.2.11 `user_data`

```
void* _can_handle::user_data
```

Параметр функции обратного вызова

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_can.h`

5.5 Структура `_can_ptb_config`

Параметры высокоприоритетного буфера выдачи

```
#include <hal_can.h>
```

Поля данных

- `bool tx_single_shot`

## 5.5.1 Подробное описание

Параметры высокоприоритетного буфера выдачи

## 5.5.2 Поля

5.5.2.1 `tx_single_shot`

```
bool _can_ptb_config::tx_single_shot
```

Включение режима единичной выдачи

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_can.h`

5.6 Структура `_can_rx_buffer_frame`

Структура буфера приема кадра CAN.

```
#include <hal_can.h>
```

Поля данных

- struct {
  - uint32\_t id: 29
  - uint32\_t : 2
  - bool esi: 1
 };
- struct {
  - can\_bytes\_in\_datafield\_t dlc: 4
  - bool brs: 1
  - bool fdf: 1
  - bool rtr: 1
  - bool ide: 1
  - uint32\_t : 4
  - bool tx: 1
  - can\_kind\_of\_error\_t koer: 3
  - uint32\_t cycle\_time: 16
 };
- uint8\_t data [64]
- uint64\_t rts

### 5.6.1 Подробное описание

Структура буфера приема кадра CAN.

### 5.6.2 Поля

#### 5.6.2.1 \_\_pad0\_\_

uint32\_t \_can\_rx\_buffer\_frame::\_\_pad0\_\_

Выравнивание

#### 5.6.2.2 brs

bool \_can\_rx\_buffer\_frame::brs

Разрешение переключения скорости передачи (для CAN FD)

#### 5.6.2.3 cycle\_time

uint32\_t \_can\_rx\_buffer\_frame::cycle\_time

Время приема кадра по таймеру TTCAN

5.6.2.4 `data`

```
uint8_t _can_rx_buffer_frame::data[64]
```

Поле данных кадра CAN

5.6.2.5 `dlc`

```
can_bytes_in_datafield_t _can_rx_buffer_frame::dlc
```

Длина поля данных

5.6.2.6 `esi`

```
bool _can_rx_buffer_frame::esi
```

Признак состояния ошибки узла, передавшего кадр

5.6.2.7 `fdf`

```
bool _can_rx_buffer_frame::fdf
```

Признак формата CAN FD

5.6.2.8 `id`

```
uint32_t _can_rx_buffer_frame::id
```

Идентификатор кадра CAN

5.6.2.9 `ide`

```
bool _can_rx_buffer_frame::ide
```

Признак расширенного идентификатора

5.6.2.10 `koer`

```
can_kind_of_error_t _can_rx_buffer_frame::koer
```

Вид ошибки

5.6.2.11 `rtr`

```
bool _can_rx_buffer_frame::rtr
```

Признак кадра удаленного запроса

#### 5.6.2.12 rts

`uint64_t _can_rx_buffer_frame::rts`

Отметка времени приема кадра (CiA 603)

#### 5.6.2.13 tx

`bool _can_rx_buffer_frame::tx`

Буфер активен

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_can.h`

### 5.7 Структура `_can_rx_transfer`

Структура для приема кадра CAN в неблокирующем режиме (по прерыванию)

```
#include <hal_can.h>
```

Поля данных

- `can_rx_buffer_frame_t * frames`
- `size_t nb_frames`

#### 5.7.1 Подробное описание

Структура для приема кадра CAN в неблокирующем режиме (по прерыванию)

#### 5.7.2 Поля

##### 5.7.2.1 frames

`can_rx_buffer_frame_t* _can_rx_transfer::frames`

Указатель на буфер для приема кадра CAN

##### 5.7.2.2 nb\_frames

`size_t _can_rx_transfer::nb_frames`

Количество кадров для приема

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_can.h`

## 5.8 Структура `_can_rxb_config`

Параметры буфера приема

```
#include <hal_can.h>
```

Поля данных

- `uint32_t almost_full_level`
- `bool self_acknowledge`
- `bool prohibit_overflow`

### 5.8.1 Подробное описание

Параметры буфера приема

#### 5.8.2 Поля

##### 5.8.2.1 `almost_full_level`

```
uint32_t _can_rxb_config::almost_full_level
```

Количество кадров в приемном буфере, при котором он считает почти полным

##### 5.8.2.2 `prohibit_overflow`

```
bool _can_rxb_config::prohibit_overflow
```

При заполненной очереди новый кадр не принимается

##### 5.8.2.3 `self_acknowledge`

```
bool _can_rxb_config::self_acknowledge
```

Режим подтверждения приема своих же кадров

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_can.h`

## 5.9 Структура `_can_stb_config`

Параметры низкоприоритетного буфера выдачи

```
#include <hal_can.h>
```

Поля данных

- `bool tx_single_shot`
- `can_stb_discipline_t tx_discipline`

### 5.9.1 Подробное описание

Параметры низкоприоритетного буфера выдачи

### 5.9.2 Поля

#### 5.9.2.1 tx\_discipline

`can_stb_discipline_t _can_stb_config::tx_discipline`

Дисциплина выдачи кадров из низкоприоритетного буфера

#### 5.9.2.2 tx\_single\_shot

`bool _can_stb_config::tx_single_shot`

Включение режима единичной выдачи

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_can.h`

## 5.10 Структура \_can\_timing\_config

Временные параметры передачи битов CAN.

```
#include <hal_can.h>
```

Поля данных

- `uint8_t prescaler`
- `uint8_t sjw`
- `uint8_t seg1`
- `uint8_t seg2`
- `uint8_t data_prescaler`
- `uint8_t data_sjw`
- `uint8_t data_seg1`
- `uint8_t data_seg2`
- `uint8_t delay_compensation_enable`
- `uint8_t secondary_sample_point_offset`



### 5.10.1 Подробное описание

Временные параметры передачи битов CAN.

### 5.10.2 Поля

#### 5.10.2.1 `data_prescaler`

`uint8_t _can_timing_config::data_prescaler`

Делитель системной частоты для сегмента данных (CAN FD)

#### 5.10.2.2 `data_seg1`

`uint8_t _can_timing_config::data_seg1`

Время сегмента 1 (CAN FD)

#### 5.10.2.3 `data_seg2`

`uint8_t _can_timing_config::data_seg2`

Время сегмента 2 (CAN FD)

#### 5.10.2.4 `data_sjw`

`uint8_t _can_timing_config::data_sjw`

Максимально допустимый скачок при синхронизации сегмента данных (CAN FD)

#### 5.10.2.5 `delay_compensation_enable`

`uint8_t _can_timing_config::delay_compensation_enable`

Включение компенсации задержки передатчика (CAN FD)

#### 5.10.2.6 `prescaler`

`uint8_t _can_timing_config::prescaler`

Делитель системной частоты

#### 5.10.2.7 `secondary_sample_point_offset`

`uint8_t _can_timing_config::secondary_sample_point_offset`

Смещение второй точки чтения для компенсации задержки (CAN FD)

## 5.10.2.8 seg1

uint8\_t \_can\_timing\_config::seg1

Время сегмента 1

## 5.10.2.9 seg2

uint8\_t \_can\_timing\_config::seg2

Время сегмента 2

## 5.10.2.10 sjw

uint8\_t \_can\_timing\_config::sjw

Максимально допустимый скачок при синхронизации

Объявления и описания членов структуры находятся в файле:

- devices/eliot1/drivers/[hal\\_can.h](#)

## 5.11 Структура \_can\_tx\_buffer\_frame

Структура буфера передачи кадра CAN.

```
#include <hal_can.h>
```

Поля данных

- struct {  
    uint32\_t id: 29  
    uint32\_t : 2  
    bool ttsen: 1  
};
- struct {  
    [can\\_bytes\\_in\\_datafield\\_t](#) dlc: 4  
    bool brs: 1  
    bool fdf: 1  
    bool rtr: 1  
    bool ide: 1  
    uint32\_t : 24  
};
- uint8\_t [data](#) [64]

## 5.11.1 Подробное описание

Структура буфера передачи кадра CAN.

### 5.11.2 Поля

#### 5.11.2.1 `__pad0__`

`uint32_t _can_tx_buffer_frame::__pad0__`

Выравнивание

#### 5.11.2.2 `brs`

`bool _can_tx_buffer_frame::brs`

Разрешение переключения скорости передачи (для CAN FD)

#### 5.11.2.3 `data`

`uint8_t _can_tx_buffer_frame::data[64]`

Поле данных кадра CAN

#### 5.11.2.4 `dlc`

`can_bytes_in_datafield_t _can_tx_buffer_frame::dlc`

Длина поля данных

#### 5.11.2.5 `fdf`

`bool _can_tx_buffer_frame::fdf`

Признак формата CAN FD

#### 5.11.2.6 `id`

`uint32_t _can_tx_buffer_frame::id`

Идентификатор кадра CAN

#### 5.11.2.7 `ide`

`bool _can_tx_buffer_frame::ide`

Признак расширенного идентификатора

#### 5.11.2.8 rtr

`bool _can_tx_buffer_frame::rtr`

Признак кадра удаленного запроса

#### 5.11.2.9 ttsen

`bool _can_tx_buffer_frame::ttsen`

Включение отметок времени передачи (CiA 603)

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_can.h`

### 5.12 Структура `_can_tx_transfer`

Структура для передачи кадра CAN в неблокирующем режиме (по прерыванию)

```
#include <hal_can.h>
```

Поля данных

- `can_tx_buffer_frame_t * frames`
- `size_t nb_frames`

#### 5.12.1 Подробное описание

Структура для передачи кадра CAN в неблокирующем режиме (по прерыванию)

#### 5.12.2 Поля

##### 5.12.2.1 frames

`can_tx_buffer_frame_t* _can_tx_transfer::frames`

Указатель на буфер с кадрами для передачи

##### 5.12.2.2 nb\_frames

`size_t _can_tx_transfer::nb_frames`

Количество кадров для передачи

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_can.h`

## 5.13 Структура `_dma_channel_config`

Конфигурация канала DMA.

```
#include <hal_dma.h>
```

Поля данных

- `void * src_addr`
- `void * dst_addr`
- `void * next_desc`
- `uint64_t ctlx_cfg`
- `bool is_src_periph`
- `bool is_dst_periph`

### 5.13.1 Подробное описание

Конфигурация канала DMA.

### 5.13.2 Поля

#### 5.13.2.1 `ctlx_cfg`

```
uint64_t _dma_channel_config::ctlx_cfg
```

Конфигурация регистра CTLx

#### 5.13.2.2 `dst_addr`

```
void* _dma_channel_config::dst_addr
```

Адрес приемника

#### 5.13.2.3 `is_dst_periph`

```
bool _dma_channel_config::is_dst_periph
```

Тип запроса приемника

#### 5.13.2.4 `is_src_periph`

```
bool _dma_channel_config::is_src_periph
```

Тип запроса источника

## 5.13.2.5 next\_desc

```
void* _dma_channel_config::next_desc
```

Указатель на дескриптор

## 5.13.2.6 src\_addr

```
void* _dma_channel_config::src_addr
```

Адрес источника

Объявления и описания членов структуры находятся в файле:

- [devices/eliot1/drivers/hal\\_dma.h](#)

## 5.14 Структура \_dma\_channel\_reg

Дескриптор на регистры канала DMA.

```
#include <hal_dma.h>
```

Поля данных

- \_\_IOM uint32\_t [SAR\\_LO](#)
- \_\_IOM uint32\_t [SAR\\_HI](#)
- \_\_IOM uint32\_t [DAR\\_LO](#)
- \_\_IOM uint32\_t [DAR\\_HI](#)
- \_\_IOM uint32\_t [LLP\\_LO](#)
- \_\_IOM uint32\_t [LLP\\_HI](#)
- \_\_IOM uint32\_t [CTL\\_LO](#)
- \_\_IOM uint32\_t [CTL\\_HI](#)
- \_\_IOM uint32\_t [SSTAT\\_LO](#)
- \_\_IOM uint32\_t [SSTAT\\_HI](#)
- \_\_IOM uint32\_t [DSTAT\\_LO](#)
- \_\_IOM uint32\_t [DSTAT\\_HI](#)
- \_\_IOM uint32\_t [SSTATAR\\_LO](#)
- \_\_IOM uint32\_t [SSTATAR\\_HI](#)
- \_\_IOM uint32\_t [DSTATAR\\_LO](#)
- \_\_IOM uint32\_t [DSTATAR\\_HI](#)
- \_\_IOM uint32\_t [CFG\\_LO](#)
- \_\_IOM uint32\_t [CFG\\_HI](#)
- \_\_IOM uint32\_t [SGR\\_LO](#)
- \_\_IOM uint32\_t [SGR\\_HI](#)
- \_\_IOM uint32\_t [DSR\\_LO](#)
- \_\_IOM uint32\_t [DSR\\_HI](#)

## 5.14.1 Подробное описание

Дескриптор на регистры канала DMA.

## 5.14.2 Поля

### 5.14.2.1 CFG\_HI

\_\_IOM uint32\_t \_dma\_channel\_reg::CFG\_HI

Регистр конфигурации, старшие биты

### 5.14.2.2 CFG\_LO

\_\_IOM uint32\_t \_dma\_channel\_reg::CFG\_LO

Регистр конфигурации, младшие биты

### 5.14.2.3 CTL\_HI

\_\_IOM uint32\_t \_dma\_channel\_reg::CTL\_HI

Регистр управления, старшие биты

### 5.14.2.4 CTL\_LO

\_\_IOM uint32\_t \_dma\_channel\_reg::CTL\_LO

Регистр управления, младшие биты

### 5.14.2.5 DAR\_HI

\_\_IOM uint32\_t \_dma\_channel\_reg::DAR\_HI

Адрес Приемника, старшие биты

### 5.14.2.6 DAR\_LO

\_\_IOM uint32\_t \_dma\_channel\_reg::DAR\_LO

Адрес Приемника, младшие биты

### 5.14.2.7 DSR\_HI

\_\_IOM uint32\_t \_dma\_channel\_reg::DSR\_HI

Регистр разброса Приемника, старшие биты

## 5.14.2.8 DSR\_LO

`__IOM uint32_t _dma_channel_reg::DSR_LO`

Регистр разброса Приемника, младшие биты

## 5.14.2.9 DSTAT\_HI

`__IOM uint32_t _dma_channel_reg::DSTAT_HI`

Статус Приемника, старшие биты

## 5.14.2.10 DSTAT\_LO

`__IOM uint32_t _dma_channel_reg::DSTAT_LO`

Статус Приемника, младшие биты

## 5.14.2.11 DSTATAR\_HI

`__IOM uint32_t _dma_channel_reg::DSTATAR_HI`

Адрес Статуса Приемника, старшие биты

## 5.14.2.12 DSTATAR\_LO

`__IOM uint32_t _dma_channel_reg::DSTATAR_LO`

Адрес Статуса Приемника, младшие биты

## 5.14.2.13 LLP\_HI

`__IOM uint32_t _dma_channel_reg::LLP_HI`

Адрес Описателя следующего блока, старшие биты

## 5.14.2.14 LLP\_LO

`__IOM uint32_t _dma_channel_reg::LLP_LO`

Адрес Описателя следующего блока, младшие биты

## 5.14.2.15 SAR\_HI

`__IOM uint32_t _dma_channel_reg::SAR_HI`

Адрес Источника, старшие биты



5.14.2.16 `SAR_LO`

```
__IOM uint32_t _dma_channel_reg::SAR_LO
```

Адрес Источника, младшие биты

5.14.2.17 `SGR_HI`

```
__IOM uint32_t _dma_channel_reg::SGR_HI
```

Регистр сбора Источника, старшие биты

5.14.2.18 `SGR_LO`

```
__IOM uint32_t _dma_channel_reg::SGR_LO
```

Регистр сбора Источника, младшие биты

5.14.2.19 `SSTAT_HI`

```
__IOM uint32_t _dma_channel_reg::SSTAT_HI
```

Статус Источника, старшие биты

5.14.2.20 `SSTAT_LO`

```
__IOM uint32_t _dma_channel_reg::SSTAT_LO
```

Статус Источника, младшие биты

5.14.2.21 `SSTATAR_HI`

```
__IOM uint32_t _dma_channel_reg::SSTATAR_HI
```

Адрес Статуса Источника, старшие биты

5.14.2.22 `SSTATAR_LO`

```
__IOM uint32_t _dma_channel_reg::SSTATAR_LO
```

Адрес Статуса Источника, младшие биты

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_dma.h`

## 5.15 Структура `_dma_descriptor`

Описатель следующего блока DMA (LLI)

```
#include <hal_dma.h>
```

Поля данных

- `__IOM uint32_t SAR`
- `__IOM uint32_t DAR`
- `__IOM uint32_t LLP`
- `__IOM uint32_t CTL_LO`
- `__IOM uint32_t CTL_HI`
- `__IOM uint32_t SSTAT`
- `__IOM uint32_t DSTAT`

### 5.15.1 Подробное описание

Описатель следующего блока DMA (LLI)

### 5.15.2 Поля

#### 5.15.2.1 CTL\_HI

```
__IOM uint32_t _dma_descriptor::CTL_HI
```

Конфигурация передачи канала

#### 5.15.2.2 CTL\_LO

```
__IOM uint32_t _dma_descriptor::CTL_LO
```

Конфигурация передачи канала

#### 5.15.2.3 DAR

```
__IOM uint32_t _dma_descriptor::DAR
```

Адрес приемника

#### 5.15.2.4 DSTAT

```
__IOM uint32_t _dma_descriptor::DSTAT
```

Состояние приемника

## 5.15.2.5 LLP

`__IOM uint32_t _dma_descriptor::LLP`

Адрес следующего блока

## 5.15.2.6 SAR

`__IOM uint32_t _dma_descriptor::SAR`

Адрес источника

## 5.15.2.7 SSTAT

`__IOM uint32_t _dma_descriptor::SSTAT`

Состояние источника

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_dma.h`

5.16 Структура `_dma_handle`

Управляющая структура передачи

```
#include <hal_dma.h>
```

Поля данных

- `dma_callback callback`
- `void * user_data`
- `DMA_Type * base`
- `uint32_t channel`

## 5.16.1 Подробное описание

Управляющая структура передачи

## 5.16.2 Поля

## 5.16.2.1 base

`DMA_Type* _dma_handle::base`

Базовый адрес DMA

### 5.16.2.2 callback

`dma_callback _dma_handle::callback`

Функция обратного вызова во время прерывания

### 5.16.2.3 channel

`uint32_t _dma_handle::channel`

Номер канала DMA

### 5.16.2.4 user\_data

`void* _dma_handle::user_data`

Параметр функции обратного вызова

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_dma.h`

## 5.17 Структура `_dma_multiblock_config`

Конфигурация многоблочной передачи

```
#include <hal_dma.h>
```

Поля данных

- `uint32_t count`
- `uint32_t data_size`
- `dma_transfer_type_t transfer_type`
- `bool scatter_en`
- `bool gather_en`
- `uint32_t src_burst_size`
- `uint32_t dst_burst_size`
- `uint32_t src_incr`
- `uint32_t dst_incr`
- `uint32_t src_data_width`
- `uint32_t dst_data_width`
- `void * src_addr`
- `void * dst_addr`
- `bool int_en`

### 5.17.1 Подробное описание

Конфигурация многоблочной передачи

## 5.17.2 Поля

### 5.17.2.1 `count`

`uint32_t _dma_multiblock_config::count`

Количество дескрипторов многоблочной передачи

### 5.17.2.2 `data_size`

`uint32_t _dma_multiblock_config::data_size`

Общее число данных

### 5.17.2.3 `dst_addr`

`void* _dma_multiblock_config::dst_addr`

Адрес Приемника

### 5.17.2.4 `dst_burst_size`

`uint32_t _dma_multiblock_config::dst_burst_size`

Размер пакета Приемника

### 5.17.2.5 `dst_data_width`

`uint32_t _dma_multiblock_config::dst_data_width`

Ширина передаваемых данных Приемника

### 5.17.2.6 `dst_incr`

`uint32_t _dma_multiblock_config::dst_incr`

Инкремент Приемника

### 5.17.2.7 `gather_en`

`bool _dma_multiblock_config::gather_en`

Разрешение режима сбора Источника

## 5.17.2.8 int\_en

`bool _dma_multiblock_config::int_en`

Разрешение прерываний

## 5.17.2.9 scatter\_en

`bool _dma_multiblock_config::scatter_en`

Разрешение режима разброса Приемника

## 5.17.2.10 src\_addr

`void* _dma_multiblock_config::src_addr`

Адрес Источника

## 5.17.2.11 src\_burst\_size

`uint32_t _dma_multiblock_config::src_burst_size`

Размер пакета Источника

## 5.17.2.12 src\_data\_width

`uint32_t _dma_multiblock_config::src_data_width`

Ширина передаваемых данных Источника

## 5.17.2.13 src\_incr

`uint32_t _dma_multiblock_config::src_incr`

Инкремент Источника

## 5.17.2.14 transfer\_type

`dma\_transfer\_type\_t _dma_multiblock_config::transfer_type`

Тип передачи

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal\_dma.h`

## 5.18 Структура `_i2c_master_dma_handle`

Контекст данных прерывания I2C-DMA.

```
#include <hal_i2c_dma.h>
```

Поля данных

- `i2c_master_transfer_states` state
- `uint32_t` `remaining_bytes_DMA`
- `uint8_t *` `buf`
- `uint32_t *` `dummy_data`
- `dma_handle_t *` `tx_dma`
- `dma_handle_t *` `rx_dma`
- `dma_descriptor_t *` `tx_desc`
- `dma_descriptor_t *` `rx_desc`
- `i2c_master_dma_transfer_callback_t` `completion_callback`
- `void *` `user_data`

### 5.18.1 Подробное описание

Контекст данных прерывания I2C-DMA.

### 5.18.2 Поля

#### 5.18.2.1 `buf`

```
uint8_t* _i2c_master_dma_handle::buf
```

Указатель на буфер приема/передачи данных

#### 5.18.2.2 `completion_callback`

```
i2c_master_dma_transfer_callback_t _i2c_master_dma_handle::completion_callback
```

Функция обратного вызова

#### 5.18.2.3 `dummy_data`

```
uint32_t* _i2c_master_dma_handle::dummy_data
```

Указатель на фиктивные данные для приема

#### 5.18.2.4 `remaining_bytes_DMA`

```
uint32_t _i2c_master_dma_handle::remaining_bytes_DMA
```

Число байт, участвующее в передаче между I2C и DMA

#### 5.18.2.5 rx\_desc

`dma_descriptor_t* _i2c_master_dma_handle::rx_desc`

Дескриптор многоблочной передачи DMA

#### 5.18.2.6 rx\_dma

`dma_handle_t* _i2c_master_dma_handle::rx_dma`

Контекст драйвера DMA на прием

#### 5.18.2.7 state

`i2c_master_transfer_states _i2c_master_dma_handle::state`

Состояние передачи

#### 5.18.2.8 tx\_desc

`dma_descriptor_t* _i2c_master_dma_handle::tx_desc`

Дескриптор многоблочной передачи DMA

#### 5.18.2.9 tx\_dma

`dma_handle_t* _i2c_master_dma_handle::tx_dma`

Контекст драйвера DMA на передачу

#### 5.18.2.10 user\_data

`void* _i2c_master_dma_handle::user_data`

Пользовательские данные

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_i2c_dma.h`

### 5.19 Структура `_i2c_master_handle`

Дескриптор для работы по прерыванию.

```
#include <hal_i2c.h>
```



Поля данных

- `uint8_t state`
- `uint32_t transfer_count`
- `uint32_t remaining_bytes`
- `uint8_t * buf`
- `uint32_t remaining_subaddr`
- `uint8_t subaddr_buf [4]`
- `bool check_addr_nack`
- `i2c_master_transfer_t transfer`
- `i2c_master_transfer_callback_t completion_callback`
- `void * user_data`

### 5.19.1 Подробное описание

Дескриптор для работы по прерыванию.

### 5.19.2 Поля

#### 5.19.2.1 `buf`

`uint8_t* _i2c_master_handle::buf`

Буфера для текущего состояния

#### 5.19.2.2 `check_addr_nack`

`bool _i2c_master_handle::check_addr_nack`

Проверять сигнал NACK после передачи адреса

#### 5.19.2.3 `completion_callback`

`i2c_master_transfer_callback_t _i2c_master_handle::completion_callback`

Функция обратного вызова

#### 5.19.2.4 `remaining_bytes`

`uint32_t _i2c_master_handle::remaining_bytes`

Количество оставшихся байтов в текущем состоянии

#### 5.19.2.5 `remaining_subaddr`

`uint32_t _i2c_master_handle::remaining_subaddr`

Оставшийся дополнительный адрес

## 5.19.2.6 state

```
uint8_t _i2c_master_handle::state
```

Текущее состояние конечного автомата

## 5.19.2.7 subaddr\_buf

```
uint8_t _i2c_master_handle::subaddr_buf[4]
```

TODO

## 5.19.2.8 transfer

```
i2c_master_transfer_t _i2c_master_handle::transfer
```

Копия текущего статуса передачи

## 5.19.2.9 transfer\_count

```
uint32_t _i2c_master_handle::transfer_count
```

Указывает на ход передачи

## 5.19.2.10 user\_data

```
void* _i2c_master_handle::user_data
```

Данные пользователя для передачи в функцию обратного вызова

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_i2c.h`

5.20 Структура `_i2c_slave_handle`

I2C Slave-дескриптор

```
#include <hal_i2c.h>
```

Поля данных

- `int32_t irq_num`
- `volatile i2c_slave_transfer_t transfer`
- `volatile i2c_slave_fsm_t slave_fsm`
- `i2c_slave_transfer_callback_t callback`
- `void * user_data`

### 5.20.1 Подробное описание

I2C Slave-дескриптор

### 5.20.2 Поля

#### 5.20.2.1 `callback`

`i2c_slave_transfer_callback_t _i2c_slave_handle::callback`

Функция обратного вызова, вызываемая при обмене

#### 5.20.2.2 `irq_num`

`int32_t _i2c_slave_handle::irq_num`

Номер IRQ

#### 5.20.2.3 `slave_fsm`

`volatile i2c_slave_fsm_t _i2c_slave_handle::slave_fsm`

Slave конечный автомат передачи

#### 5.20.2.4 `transfer`

`volatile i2c_slave_transfer_t _i2c_slave_handle::transfer`

Структура обмена данными

#### 5.20.2.5 `user_data`

`void* _i2c_slave_handle::user_data`

Параметр функции обратного вызова

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_i2c.h`

## 5.21 Структура `_i2s_config`

Структура конфигурации контроллера I2S.

```
#include <hal_i2s.h>
```

Поля данных

- [uint32\\_t sample\\_rate](#)
- [i2s\\_sclk\\_per\\_sample\\_t sclk\\_per\\_sample](#)
- [i2s\\_sclk\\_gating\\_t sclk\\_gating](#)
- [i2s\\_resolution\\_t resolution](#)
- [i2s\\_interrupt\\_level\\_t interrupt\\_level](#)

### 5.21.1 Подробное описание

Структура конфигурации контроллера I2S.

### 5.21.2 Поля

#### 5.21.2.1 interrupt\_level

[i2s\\_interrupt\\_level\\_t](#) `_i2s_config::interrupt_level`

Уровень опустошения очереди, при котором происходит прерывание

#### 5.21.2.2 resolution

[i2s\\_resolution\\_t](#) `_i2s_config::resolution`

Разрядность данных

#### 5.21.2.3 sample\_rate

[uint32\\_t](#) `_i2s_config::sample_rate`

Частота аудиоданных

#### 5.21.2.4 sclk\_gating

[i2s\\_sclk\\_gating\\_t](#) `_i2s_config::sclk_gating`

Обрезание синхроимпульсов sclk

#### 5.21.2.5 sclk\_per\_sample

[i2s\\_sclk\\_per\\_sample\\_t](#) `_i2s_config::sclk_per_sample`

Количество синхроимпульсов sclk на левое/правое значение

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_i2s.h`

## 5.22 Структура `_i2s_handle`

Структура обработчика драйвера I2S.

```
#include <hal_i2s.h>
```

Поля данных

- `i2s_transfer_callback_t` `callback`
- `void *` `user_data`
- `uint32_t *` `left_samples`
- `uint32_t *` `right_samples`
- `size_t` `nb_samples`

### 5.22.1 Подробное описание

Структура обработчика драйвера I2S.

### 5.22.2 Поля

#### 5.22.2.1 `callback`

`i2s_transfer_callback_t` `_i2s_handle::callback`

Функция обратного вызова

#### 5.22.2.2 `left_samples`

`uint32_t *` `_i2s_handle::left_samples`

Буфер с данными для левого потока

#### 5.22.2.3 `nb_samples`

`size_t` `_i2s_handle::nb_samples`

Количество сэмплов

#### 5.22.2.4 `right_samples`

`uint32_t *` `_i2s_handle::right_samples`

Буфер с данными для правого потока

#### 5.22.2.5 user\_data

```
void* _i2s_handle::user_data
```

Данные пользователя

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal\_i2s.h`

### 5.23 Структура `_i2s_transfer`

Структура для передачи данных I2S в неблокирующем режиме (по прерыванию)

```
#include <hal_i2s.h>
```

Поля данных

- `uint32_t * left\_samples`
- `uint32_t * right\_samples`
- `size_t nb\_samples`

#### 5.23.1 Подробное описание

Структура для передачи данных I2S в неблокирующем режиме (по прерыванию)

#### 5.23.2 Поля

##### 5.23.2.1 left\_samples

```
uint32_t* _i2s_transfer::left_samples
```

Буфер с данными для передачи в левый поток

##### 5.23.2.2 nb\_samples

```
size_t _i2s_transfer::nb_samples
```

Количество значений в буфере

##### 5.23.2.3 right\_samples

```
uint32_t* _i2s_transfer::right_samples
```

Буфер с данными для передачи в правый поток

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal\_i2s.h`

## 5.24 Структура \_jtm\_handle

Структура обработчика событий JTM.

```
#include <hal_jtm.h>
```

Поля данных

- [jtm\\_callback\\_t](#) callback
- [jtm\\_parameter\\_t](#) parameter
- void \* [user\\_data](#)

### 5.24.1 Подробное описание

Структура обработчика событий JTM.

#### 5.24.2 Поля

##### 5.24.2.1 callback

[jtm\\_callback\\_t](#) \_jtm\_handle::callback

Функция обратного вызова

##### 5.24.2.2 parameter

[jtm\\_parameter\\_t](#) \_jtm\_handle::parameter

Запрашиваемый параметр JTM

##### 5.24.2.3 user\_data

void\* \_jtm\_handle::user\_data

Указатель на пользовательские данные

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_jtm.h`

## 5.25 Структура \_nor\_command\_set

Основной набор команд для флеш-памяти NOR.

```
#include <hal_nor_flash.h>
```

Поля данных

- `uint8_t write_status_cmd`
- `uint8_t page_write_memory_cmd`
- `uint8_t read_memory_command`
- `uint8_t write_disable_cmd`
- `uint8_t read_status_cmd`
- `uint8_t write_enable_cmd`
- `uint8_t erase_sector_cmd`
- `uint8_t erase_chip_cmd`

### 5.25.1 Подробное описание

Основной набор команд для флеш-памяти NOR.

### 5.25.2 Поля

#### 5.25.2.1 `erase_chip_cmd`

`uint8_t _nor_command_set::erase_chip_cmd`

Очистка всей памяти

#### 5.25.2.2 `erase_sector_cmd`

`uint8_t _nor_command_set::erase_sector_cmd`

Очистка сектора

#### 5.25.2.3 `page_write_memory_cmd`

`uint8_t _nor_command_set::page_write_memory_cmd`

Программирование страницы

#### 5.25.2.4 `read_memory_command`

`uint8_t _nor_command_set::read_memory_command`

Чтение памяти

#### 5.25.2.5 `read_status_cmd`

`uint8_t _nor_command_set::read_status_cmd`

Включение записи



5.25.2.6 `write_disable_cmd`

```
uint8_t _nor_command_set::write_disable_cmd
```

Запрет записи

5.25.2.7 `write_enable_cmd`

```
uint8_t _nor_command_set::write_enable_cmd
```

Разрешение записи

5.25.2.8 `write_status_cmd`

```
uint8_t _nor_command_set::write_status_cmd
```

Запись в статусный регистр

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal\_nor\_flash.h`

## 5.26 Структура `_nor_config`

Структура первичной конфигурации флеш-памяти NOR.

```
#include <hal_nor_flash.h>
```

Поля данных

- `void * mem\_control\_config`
- `void * quad\_control\_config`
- `QSPI_Type * driver\_base\_addr`

### 5.26.1 Подробное описание

Структура первичной конфигурации флеш-памяти NOR.

### 5.26.2 Поля

#### 5.26.2.1 `driver_base_addr`

```
QSPI_Type* _nor_config::driver_base_addr
```

Базовый адрес контроллера

### 5.26.2.2 mem\_control\_config

```
void* _nor_config::mem_control_config
```

Конфигурация контроллера памяти, должен быть присвоен указатель на заполненную структуру [qspi\\_nor\\_init\\_config\\_t](#)

### 5.26.2.3 quad\_control\_config

```
void* _nor_config::quad_control_config
```

Конфигурация режима QUAD, должен быть присвоен указатель на заполненную структуру [qspi\\_nor\\_config\\_t](#)

Объявления и описания членов структуры находятся в файле:

- [devices/eliot1/drivers/hal\\_nor\\_flash.h](#)

## 5.27 Структура \_nor\_handle

Контекст драйвера флеш-памяти NOR.

```
#include <hal_nor_flash.h>
```

Поля данных

- [QSPI\\_Type](#) \* [driver\\_base\\_addr](#)
- [uint32\\_t](#) [page\\_size\\_bytes](#)
- [uint32\\_t](#) [sector\\_size\\_bytes](#)
- [uint32\\_t](#) [memory\\_size\\_bytes](#)
- [uint32\\_t](#) [max\\_sector\\_erase\\_time](#)
- [uint32\\_t](#) [max\\_page\\_program\\_time](#)
- [uint32\\_t](#) [max\\_chip\\_erase\\_time](#)
- [void](#) \* [device\\_specific](#)
- [qspi\\_xip\\_config\\_t](#) [xip\\_config](#)
- [qspi\\_config\\_t](#) [qspi\\_config](#)

### 5.27.1 Подробное описание

Контекст драйвера флеш-памяти NOR.

### 5.27.2 Поля

#### 5.27.2.1 device\_specific

```
void* _nor_handle::device_specific
```

Указатель на конкретное устройство, должен быть присвоен указатель на структуру [qspi\\_nor\\_handle\\_t](#)

5.27.2.2 `driver_base_addr``QSPI_Type* _nor_handle::driver_base_addr`

Базовый адрес контроллера

5.27.2.3 `max_chip_erase_time``uint32_t _nor_handle::max_chip_erase_time`

Максимальное время очистки всей флеш-памяти в мс

5.27.2.4 `max_page_program_time``uint32_t _nor_handle::max_page_program_time`

Максимальное время записи страницы в мкс

5.27.2.5 `max_sector_erase_time``uint32_t _nor_handle::max_sector_erase_time`

Максимальное время очистки сектора в мс

5.27.2.6 `memory_size_bytes``uint32_t _nor_handle::memory_size_bytes`

Общий размер памяти микросхемы флеш-памяти

5.27.2.7 `page_size_bytes``uint32_t _nor_handle::page_size_bytes`

Размер страницы в байтах

5.27.2.8 `qspi_config``qspi_config_t _nor_handle::qspi_config`

Конфигурация контроллера QSPI

5.27.2.9 `sector_size_bytes``uint32_t _nor_handle::sector_size_bytes`

Размер сектора в байтах

### 5.27.2.10 xip\_config

`qspi_xip_config_t _nor_handle::xip_config`

Структура с настройками режима XIP

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_nor_flash.h`

## 5.28 Структура \_pwm\_chopper

Конфигурация дробления сигнала ШИМ

```
#include <hal_pwm_newgen.h>
```

Поля данных

- `bool chopper_en`
- `enum pwm_chopper_duty chopper_duty`
- `enum pwm_chopper_freq chopper_freq`
- `enum pwm_chopper_first_width chopper_first_width`

### 5.28.1 Подробное описание

Конфигурация дробления сигнала ШИМ

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_pwm_newgen.h`

## 5.29 Структура \_pwm\_dz\_cfg

Конфигурация мертвой зоны ШИМ

```
#include <hal_pwm_newgen.h>
```

Поля данных

- `bool dz_en`
- `enum pwm_dz_outx_inv inv`

### 5.29.1 Подробное описание

Конфигурация мертвой зоны ШИМ

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_pwm_newgen.h`

## 5.30 Структура `_pwm_handle`

Контекст драйвера ШИМ

```
#include <hal_pwm_newgen.h>
```

Поля данных

- `uint32_t channel`
- `uint32_t out`
- `uint32_t ref_clk`
- `float duty_cycle`
- `uint32_t period_us`
- `bool int_en`
- `enum pwm_int_source int_source`
- `enum pwm_eventprd int_freq`
- `void(* pwm_callback)(struct _pwm_handle *hpwm)`
- `pwm_chopper_cfg_t chopper_cfg`
- `pwm_dz_cfg_t deadzone_cfg`
- `pwm_trip_unit_cfg_t trip_unit_cfg`

### 5.30.1 Подробное описание

Контекст драйвера ШИМ

### 5.30.2 Поля

#### 5.30.2.1 `channel`

```
uint32_t _pwm_handle::channel
```

Номер канала

#### 5.30.2.2 `duty_cycle`

```
float _pwm_handle::duty_cycle
```

Ширина импульса от 0.0 до 1.0

### 5.30.2.3 int\_en

bool \_pwm\_handle::int\_en

Разрешение прерываний

### 5.30.2.4 int\_freq

enum [pwm\\_eventprd](#) \_pwm\_handle::int\_freq

Частота возникновения прерываний

### 5.30.2.5 int\_source

enum [pwm\\_int\\_source](#) \_pwm\_handle::int\_source

Источник прерывания

### 5.30.2.6 out

uint32\_t \_pwm\_handle::out

Выход канала

### 5.30.2.7 period\_us

uint32\_t \_pwm\_handle::period\_us

Период в микросекундах

### 5.30.2.8 pwm\_callback

void(\* \_pwm\_handle::pwm\_callback) (struct [\\_pwm\\_handle](#) \*hpwm)

Функция обратного вызова при возникновении прерываний

### 5.30.2.9 ref\_clk

uint32\_t \_pwm\_handle::ref\_clk

Опорная частота контроллера ШИМ

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_pwm_newgen.h`

## 5.31 Структура `_pwm_trip_unit_cfg`

Поля данных

- `bool trip_unit_en`
- `enum pwm_trip_unit_action action`

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_pwm_newgen.h`

## 5.32 Структура `_qspi_config`

Количество бит во фрейме

```
#include <hal_qspi.h>
```

Поля данных

- `uint32_t delay_en`
- `uint32_t cpol`
- `uint32_t cpha`
- `uint32_t msb`
- `uint32_t cont_trans_en`
- `uint16_t cont_transfer_ext`
- `qspi_qmode_t spi_mode`
- `uint32_t slave_select`
- `uint32_t slave_pol`
- `qspi_bit_size_t bit_size`
- `uint32_t mode`
- `uint32_t dma_en`
- `uint32_t inhibit_din`
- `uint32_t inhibit_dout`

### 5.32.1 Подробное описание

Количество бит во фрейме

Структура, определяющая параметры конфигурации контроллера QSPI

### 5.32.2 Поля

#### 5.32.2.1 `bit_size`

`qspi_bit_size_t _qspi_config::bit_size`

Количество битов в передаче

#### 5.32.2.2 cont\_trans\_en

uint32\_t \_qspi\_config::cont\_trans\_en

Бит непрерывной передачи

#### 5.32.2.3 cont\_transfer\_ext

uint16\_t \_qspi\_config::cont\_transfer\_ext

Бит продление непрерывной передачи

#### 5.32.2.4 cpha

uint32\_t \_qspi\_config::cpha

Фаза тактового сигнала

#### 5.32.2.5 cpol

uint32\_t \_qspi\_config::cpol

Полярность тактового сигнала

#### 5.32.2.6 delay\_en

uint32\_t \_qspi\_config::delay\_en

Включение задержки между передачами для работы в режиме Master

#### 5.32.2.7 dma\_en

uint32\_t \_qspi\_config::dma\_en

Включение режима DMA

#### 5.32.2.8 inhibit\_din

uint32\_t \_qspi\_config::inhibit\_din

Запрет записи в RX FIFO

#### 5.32.2.9 inhibit\_dout

uint32\_t \_qspi\_config::inhibit\_dout

Запрет чтение из TX FIFO



5.32.2.10 `mode``uint32_t _qspi_config::mode`

Режим работы контроллера (Master/Slave)

5.32.2.11 `msb``uint32_t _qspi_config::msb`

Порядок передачи битов

5.32.2.12 `slave_pol``uint32_t _qspi_config::slave_pol`

Полярность сигнала SS

5.32.2.13 `slave_select``uint32_t _qspi_config::slave_select`

Выбор slave-устройства

5.32.2.14 `spi_mode``qspi_qmode_t _qspi_config::spi_mode`

Режим работы SPI

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_qspi.h`

5.33 Структура `_qspi_nor_config`

Конфигурационный блок для режима Quad.

```
#include <hal_qspi_nor_flash.h>
```

Поля данных

- `bool is_quad_need_enable`
- `bool write_two_status_bytes`
- `uint8_t quad_enable_command`
- `uint8_t quad_enable_bit_shift`
- `uint8_t quad_read_command`

### 5.33.1 Подробное описание

Конфигурационный блок для режима Quad.

### 5.33.2 Поля

#### 5.33.2.1 is\_quad\_need\_enable

`bool _qspi_nor_config::is_quad_need_enable`

Установка Quad Enable: 1 - Установка, 0 - Сброс

#### 5.33.2.2 quad\_enable\_bit\_shift

`uint8_t _qspi_nor_config::quad_enable_bit_shift`

Смещение бита Quad Enable

#### 5.33.2.3 quad\_enable\_command

`uint8_t _qspi_nor_config::quad_enable_command`

Код операции для установки Quad Enable

#### 5.33.2.4 quad\_read\_command

`uint8_t _qspi_nor_config::quad_read_command`

Команда чтения в режиме Quad

#### 5.33.2.5 write\_two\_status\_bytes

`bool _qspi_nor_config::write_two_status_bytes`

Необходимо ли запись двух статусных регистров при установке бита Quad Enable

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal\_qspi\_nor\_flash.h`

## 5.34 Структура `_qspi_nor_handle`

Контекст драйвера NOR Flash.

```
#include <hal_qspi_nor_flash.h>
```

Поля данных

- `qspi_command_type_t` `command_type`
- `qspi_command_format_t` `read_cmd_format`
- `uint8_t` `intermediate_len`
- `nor_command_set_t` `command_set`

### 5.34.1 Подробное описание

Контекст драйвера NOR Flash.

### 5.34.2 Поля

#### 5.34.2.1 `command_set`

`nor_command_set_t` `_qspi_nor_handle::command_set`

Набор базовых команд

#### 5.34.2.2 `command_type`

`qspi_command_type_t` `_qspi_nor_handle::command_type`

Тип кода операции и адреса

#### 5.34.2.3 `intermediate_len`

`uint8_t` `_qspi_nor_handle::intermediate_len`

Количество промежуточных байтов

#### 5.34.2.4 `read_cmd_format`

`qspi_command_format_t` `_qspi_nor_handle::read_cmd_format`

Формат команды для чтения

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_qspi_nor_flash.h`

## 5.35 Структура `_qspi_nor_init_config`

Первоначальная конфигурация QSPI.

```
#include <hal_qspi_nor_flash.h>
```

Поля данных

- [qspi\\_command\\_format\\_t cmd\\_format](#)
- [uint32\\_t quad\\_mode\\_setting](#)

### 5.35.1 Подробное описание

Первоначальная конфигурация QSPI.

### 5.35.2 Поля

#### 5.35.2.1 cmd\_format

[qspi\\_command\\_format\\_t \\_qspi\\_nor\\_init\\_config::cmd\\_format](#)

Формат команды для операции чтения

#### 5.35.2.2 quad\_mode\_setting

[uint32\\_t \\_qspi\\_nor\\_init\\_config::quad\\_mode\\_setting](#)

Требования для включения режима Quad

Объявления и описания членов структуры находятся в файле:

- [devices/eliot1/drivers/hal\\_qspi\\_nor\\_flash.h](#)

## 5.36 Структура \_qspi\_xip\_config

Структура параметров конфигурации XIP контроллера QSPI.

```
#include <hal_qspi.h>
```

Поля данных

- [uint32\\_t cmd](#)
- [uint32\\_t hpen](#)
- [uint32\\_t cpha](#)
- [uint32\\_t cpol](#)
- [uint32\\_t addr4](#)
- [uint32\\_t le32](#)
- [uint32\\_t hp\\_mode](#)
- [uint32\\_t dummy\\_cycles](#)
- [uint32\\_t hp\\_end\\_dummy](#)

### 5.36.1 Подробное описание

Структура параметров конфигурации XIP контроллера QSPI.

### 5.36.2 Поля

#### 5.36.2.1 `addr4`

`uint32_t _qspi_xip_config::addr4`

4-байтовый режим адресов

#### 5.36.2.2 `cmd`

`uint32_t _qspi_xip_config::cmd`

Тип команды чтения режима XIP

#### 5.36.2.3 `cpha`

`uint32_t _qspi_xip_config::cpha`

Фаза тактового сигнала

#### 5.36.2.4 `cpol`

`uint32_t _qspi_xip_config::cpol`

Полярность тактового сигнала

#### 5.36.2.5 `dummy_cycles`

`uint32_t _qspi_xip_config::dummy_cycles`

Количество dummy тактов

#### 5.36.2.6 `hp_end_dummy`

`uint32_t _qspi_xip_config::hp_end_dummy`

Количество dummy тактов для выхода из HP режима

#### 5.36.2.7 `hp_mode`

`uint32_t _qspi_xip_config::hp_mode`

Командный байт режима XIP

#### 5.36.2.8 hpen

`uint32_t _qspi_xip_config::hpen`

Включение режима высокой производительности

#### 5.36.2.9 le32

`uint32_t _qspi_xip_config::le32`

32-битная организация данных

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_qspi.h`

### 5.37 Структура `_rtc_datetime`

Структура используемая для хранения даты и времени

```
#include <hal_rwc.h>
```

Поля данных

- `uint16_t year`
- `uint8_t month`
- `uint8_t day`
- `uint8_t hour`
- `uint8_t minute`
- `uint8_t second`

#### 5.37.1 Подробное описание

Структура используемая для хранения даты и времени

#### 5.37.2 Поля

##### 5.37.2.1 day

`uint8_t _rtc_datetime::day`

Диапазон от 1 до 31 (зависит от месяца).

##### 5.37.2.2 hour

`uint8_t _rtc_datetime::hour`

Диапазон от 0 до 23.

## 5.37.2.3 minute

`uint8_t _rtc_datetime::minute`

Диапазон от 0 до 59.

## 5.37.2.4 month

`uint8_t _rtc_datetime::month`

Диапазон от 1 до 12.

## 5.37.2.5 second

`uint8_t _rtc_datetime::second`

Диапазон от 0 до 59.

## 5.37.2.6 year

`uint16_t _rtc_datetime::year`

Диапазон от 1970 до 2099.

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_rwc.h`

5.38 Структура `_spi_dma_handle`

Дескриптор SPI-DMA.

```
#include <hal_spi_dma.h>
```

Поля данных

- `volatile bool tx_in_progress`
- `volatile bool rx_in_progress`
- `dma_handle_t * tx_handle`
- `dma_descriptor_t * tx_desc`
- `dma_handle_t * rx_handle`
- `dma_descriptor_t * rx_desc`
- `uint8_t bytes_per_frame`
- `spi_dma_callback_t callback`
- `void * user_data`
- `enum spi_trans_status state`
- `size_t transfer_size`
- `void * dummy_data`

### 5.38.1 Подробное описание

Дескриптор SPI-DMA.

### 5.38.2 Поля

#### 5.38.2.1 bytes\_per\_frame

`uint8_t _spi_dma_handle::bytes_per_frame`

Количество байт во фрейме SPI

#### 5.38.2.2 callback

`spi_dma_callback_t _spi_dma_handle::callback`

Функция обратного вызова

#### 5.38.2.3 dummy\_data

`void* _spi_dma_handle::dummy_data`

Указатель на фиктивные данные

#### 5.38.2.4 rx\_desc

`dma_descriptor_t* _spi_dma_handle::rx_desc`

Указатель на дескриптор многоблочной передачи для приема

#### 5.38.2.5 rx\_handle

`dma_handle_t* _spi_dma_handle::rx_handle`

Контекст драйвера DMA для приема

#### 5.38.2.6 rx\_in\_progress

`volatile bool _spi_dma_handle::rx_in_progress`

Статус приема DMA: 1 - в процессе, 0 - завершена

#### 5.38.2.7 state

`enum spi_trans_status _spi_dma_handle::state`

Текущее состояние SPI DMA передачи



5.38.2.8 `transfer_size`

`size_t _spi_dma_handle::transfer_size`

Количество байт для передачи

5.38.2.9 `tx_desc`

`dma_descriptor_t* _spi_dma_handle::tx_desc`

Указатель на дескриптор многоблочной передачи для отправки

5.38.2.10 `tx_handle`

`dma_handle_t* _spi_dma_handle::tx_handle`

Контекст драйвера DMA для отправки

5.38.2.11 `tx_in_progress`

`volatile bool _spi_dma_handle::tx_in_progress`

Статус передачи DMA: 1 - в процессе, 0 - завершена

5.38.2.12 `user_data`

`void* _spi_dma_handle::user_data`

Пользовательские данные

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_spi_dma.h`

5.39 Структура `_ttcan_config`

Временные параметры передачи битов CAN.

`#include <hal_can.h>`

Поля данных

- `ttcan_timer_prescaler_t` `prescaler`
- `uint8_t` `transmit_trigger_pointer`
- `ttcan_trigger_type_t` `trigger_type`
- `uint8_t` `transmit_enable_window`
- `uint8_t` `trigger_time0`
- `uint8_t` `trigger_time1`
- `uint8_t` `watch_trigger_time0`
- `uint8_t` `watch_trigger_time1`
- `uint32_t` `reference_id`
- `bool` `reference_ide`

### 5.39.1 Подробное описание

Временные параметры передачи битов CAN.

### 5.39.2 Поля

#### 5.39.2.1 prescaler

`ttcan_timer_prescaler_t _ttcan_config::prescaler`

Предделитель счетчика TTCAN

#### 5.39.2.2 reference\_id

`uint32_t _ttcan_config::reference_id`

Идентификатор референсного кадра

#### 5.39.2.3 reference\_ide

`bool _ttcan_config::reference_ide`

Признак расширенного идентификатора у референсного кадра

#### 5.39.2.4 transmit\_enable\_window

`uint8_t _ttcan_config::transmit_enable_window`

Ширина окна разрешения передачи

#### 5.39.2.5 transmit\_trigger\_pointer

`uint8_t _ttcan_config::transmit_trigger_pointer`

Указатель на слот передачи

#### 5.39.2.6 trigger\_time0

`uint8_t _ttcan_config::trigger_time0`

Время по циклическому таймеру TTCAN для триггера 0

#### 5.39.2.7 trigger\_time1

`uint8_t _ttcan_config::trigger_time1`

Время по циклическому таймеру TTCAN для триггера 1

5.39.2.8 `trigger_type`

```
ttcan_trigger_type_t _ttcan_config::trigger_type
```

Тип триггера

5.39.2.9 `watch_trigger_time0`

```
uint8_t _ttcan_config::watch_trigger_time0
```

Время по циклическому таймеру TTCAN для следящего триггера 0

5.39.2.10 `watch_trigger_time1`

```
uint8_t _ttcan_config::watch_trigger_time1
```

Время по циклическому таймеру TTCAN для следящего триггера 1

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_can.h`

5.40 Структура `_uart_dma_handle`

Дескриптор UART-DMA передачи

```
#include <hal_uart_dma.h>
```

Поля данных

- `UART_Type * base`
- `uart_dma_transfer_callback_t callback`
- `void * user_data`
- `uint32_t rx_data_size_all`
- `uint32_t tx_data_size_all`
- `dma_handle_t * tx_dma_handle`
- `dma_handle_t * rx_dma_handle`
- `dma_descriptor_t * tx_desc`
- `dma_descriptor_t * rx_desc`
- `volatile uint8_t tx_state`
- `volatile uint8_t rx_state`

## 5.40.1 Подробное описание

Дескриптор UART-DMA передачи

### 5.40.2 Поля

#### 5.40.2.1 base

`UART_Type* _uart_dma_handle::base`

Базовый адрес UART

#### 5.40.2.2 callback

`uart_dma_transfer_callback_t _uart_dma_handle::callback`

Функция обратного вызова

#### 5.40.2.3 rx\_data\_size\_all

`uint32_t _uart_dma_handle::rx_data_size_all`

Число данных на прием

#### 5.40.2.4 rx\_dma\_handle

`dma_handle_t* _uart_dma_handle::rx_dma_handle`

Дескриптор DMA на прием

#### 5.40.2.5 rx\_state

`volatile uint8_t _uart_dma_handle::rx_state`

Состояние шины RX

#### 5.40.2.6 tx\_data\_size\_all

`uint32_t _uart_dma_handle::tx_data_size_all`

Число данных на передачу

#### 5.40.2.7 tx\_dma\_handle

`dma_handle_t* _uart_dma_handle::tx_dma_handle`

Дескриптор DMA на передачу

5.40.2.8 `tx_state`

```
volatile uint8_t _uart_dma_handle::tx_state
```

Состояние шины TX

5.40.2.9 `user_data`

```
void* _uart_dma_handle::user_data
```

Пользовательские данные

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_uart_dma.h`

5.41 Структура `clkctr_div`

Делители блока

```
#include <hal_clkctr.h>
```

Поля данных

- `uint32_t clkctr_fclk_div`
- `uint32_t clkctr_sysclk_div`
- `uint32_t clkctr_gnssclk_div`
- `uint32_t clkctr_qspiclk_div`
- `uint32_t clkctr_i2sclk_div`
- `uint32_t clkctr_mco_div`

## 5.41.1 Подробное описание

Делители блока

## 5.41.2 Поля

5.41.2.1 `clkctr_fclk_div`

```
uint32_t clkctr_div::clkctr_fclk_div
```

Делитель частоты FCLK

5.41.2.2 `clkctr_gnssclk_div`

```
uint32_t clkctr_div::clkctr_gnssclk_div
```

Делитель частоты GNSSCLK

#### 5.41.2.3 clkctr\_i2sclk\_div

uint32\_t clkctr\_div::clkctr\_i2sclk\_div

Делитель частоты I2SCLK

#### 5.41.2.4 clkctr\_mco\_div

uint32\_t clkctr\_div::clkctr\_mco\_div

Делитель частоты MCO

#### 5.41.2.5 clkctr\_qspiclk\_div

uint32\_t clkctr\_div::clkctr\_qspiclk\_div

Делитель частоты QSPICLK

#### 5.41.2.6 clkctr\_sysclk\_div

uint32\_t clkctr\_div::clkctr\_sysclk\_div

Делитель частоты SYSCLK

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal\_clkctr.h`

### 5.42 Структура clkctr\_pll\_cfg

Коэффициенты PLL.

```
#include <hal_clkctr.h>
```

Поля данных

- uint32\_t [lock](#)
- uint32\_t [nr\\_man](#)
- uint32\_t [nf\\_man](#)
- uint32\_t [od\\_man](#)
- uint32\_t [man](#)
- uint32\_t [sel](#)

#### 5.42.1 Подробное описание

Коэффициенты PLL.

### 5.42.2 Поля

#### 5.42.2.1 lock

uint32\_t clkctr\_pll\_cfg::lock

Признак готовности PLL, при записи игнорируется

#### 5.42.2.2 man

uint32\_t clkctr\_pll\_cfg::man

Тип установки множителей: 0 - используется поле sel, 1 - используются поля nr\_man, nf\_man, od\_man

#### 5.42.2.3 nf\_man

uint32\_t clkctr\_pll\_cfg::nf\_man

Значение множителя - 1; может быть [0:8191]

#### 5.42.2.4 nr\_man

uint32\_t clkctr\_pll\_cfg::nr\_man

Значение делителя - 1; может быть [0:15]

#### 5.42.2.5 od\_man

uint32\_t clkctr\_pll\_cfg::od\_man

Значение делителя - 1; может быть [0:15]

#### 5.42.2.6 sel

uint32\_t clkctr\_pll\_cfg::sel

Значение предустановленного множителя - 1; может быть [0:511], реально используется [0:374]

Объявления и описания членов структуры находятся в файле:

- [devices/eliot1/drivers/hal\\_clkctr.h](#)

## 5.43 Структура dma\_channel\_ctl\_cfg

Описание конфигурации пересылки

```
#include <hal_dma.h>
```

## Поля данных

- uint16\_t block\_size
- bool llp\_src\_en
- bool llp\_dst\_en
- dma\_transfer\_type\_t transfer\_type
- bool scatter\_en
- bool gather\_en
- uint8\_t src\_burst\_size
- uint8\_t dst\_burst\_size
- uint8\_t src\_incr
- uint8\_t dst\_incr
- uint8\_t src\_tr\_width
- uint8\_t dst\_tr\_width
- bool int\_en

## 5.43.1 Подробное описание

Описание конфигурации пересылки

## 5.43.2 Поля

## 5.43.2.1 block\_size

uint16\_t dma\_channel\_ctl\_cfg::block\_size

Размер блока передачи

## 5.43.2.2 dst\_burst\_size

uint8\_t dma\_channel\_ctl\_cfg::dst\_burst\_size

Размер пакета Приемника

## 5.43.2.3 dst\_incr

uint8\_t dma\_channel\_ctl\_cfg::dst\_incr

Инкремент адреса Приемника

## 5.43.2.4 dst\_tr\_width

uint8\_t dma\_channel\_ctl\_cfg::dst\_tr\_width

Ширина одного слова при передаче в Приемник



## 5.43.2.5 gather\_en

bool dma\_channel\_ctl\_cfg::gather\_en

Разрешение режима Сбора

## 5.43.2.6 int\_en

bool dma\_channel\_ctl\_cfg::int\_en

Разрешение прерываний

## 5.43.2.7 llp\_dst\_en

bool dma\_channel\_ctl\_cfg::llp\_dst\_en

Разрешение многоблочной передачи для Приемника

## 5.43.2.8 llp\_src\_en

bool dma\_channel\_ctl\_cfg::llp\_src\_en

Разрешение многоблочной передачи для Источника

## 5.43.2.9 scatter\_en

bool dma\_channel\_ctl\_cfg::scatter\_en

Разрешение режима Разброса

## 5.43.2.10 src\_burst\_size

uint8\_t dma\_channel\_ctl\_cfg::src\_burst\_size

Размер пакета Источника

## 5.43.2.11 src\_incr

uint8\_t dma\_channel\_ctl\_cfg::src\_incr

Инкремент адреса Источника

## 5.43.2.12 src\_tr\_width

uint8\_t dma\_channel\_ctl\_cfg::src\_tr\_width

Ширина одного слова при передаче от Источника

### 5.43.2.13 transfer\_type

`dma_transfer_type_t dma_channel_ctl_cfg::transfer_type`

Тип пересылки

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_dma.h`

## 5.44 Структура dualtimer\_hardware\_config

Конфигурация аппаратной части сдвоенного таймера

```
#include <hal_dualtimer.h>
```

Поля данных

- `uint32_t load`
- `uint32_t bg_load`
- `enum dualtimer_work_enable enable`
- `enum dualtimer_mode mode`
- `enum dualtimer_interrupt_control int_ctrl`
- `enum dualtimer_prescale prescale`
- `enum dualtimer_timer_size size`
- `enum dualtimer_number_of_repetitions cyclicity`

### 5.44.1 Подробное описание

Конфигурация аппаратной части сдвоенного таймера

### 5.44.2 Поля

#### 5.44.2.1 bg\_load

`uint32_t dualtimer_hardware_config::bg_load`

Тип работы

#### 5.44.2.2 cyclicity

`enum dualtimer_number_of_repetitions dualtimer_hardware_config::cyclicity`

Количество повторений запусков

5.44.2.3 `enable`

enum `dualtimer_work_enable` `dualtimer_hardware_config::enable`

Разрешение работы

5.44.2.4 `int_ctrl`

enum `dualtimer_interrupt_control` `dualtimer_hardware_config::int_ctrl`

Управление прерыванием

5.44.2.5 `load`

uint32\_t `dualtimer_hardware_config::load`

Стартовое значение счетчика

5.44.2.6 `mode`

enum `dualtimer_mode` `dualtimer_hardware_config::mode`

Режим работы

5.44.2.7 `prescale`

enum `dualtimer_prescale` `dualtimer_hardware_config::prescale`

Делитель частоты

5.44.2.8 `size`

enum `dualtimer_timer_size` `dualtimer_hardware_config::size`

Разрядность счетчика

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_dualtimer.h`

5.45 Структура `i2c_master_config_t`

Структура с настройками для инициализации Master-модуля I2C.

```
#include <hal_i2c.h>
```

Поля данных

- bool [enable\\_master](#)
- uint32\_t [baudrate\\_bps](#)
- uint8\_t [sda\\_setup](#)
- uint16\_t [sda\\_hold](#)

### 5.45.1 Подробное описание

Структура с настройками для инициализации Master-модуля I2C.

Заметки

Структура содержит настройки конфигурации для периферийного модуля I2C. Чтобы инициализировать структуру с значениями по умолчанию, необходимо вызвать функцию [I2C\\_MasterGetDefaultConfig](#) и передать ей указатель на экземпляр структуры конфигурации.

### 5.45.2 Поля

#### 5.45.2.1 baudrate\_bps

uint32\_t i2c\_master\_config\_t::baudrate\_bps

Скорость передачи в битах в секунду

#### 5.45.2.2 enable\_master

bool i2c\_master\_config\_t::enable\_master

Включить ли модуль при инициализации

#### 5.45.2.3 sda\_hold

uint16\_t i2c\_master\_config\_t::sda\_hold

Количество тактов удерживания SDA после заднего фронта SCL

#### 5.45.2.4 sda\_setup

uint8\_t i2c\_master\_config\_t::sda\_setup

Количество тактов удерживания SDA после переднего фронта SCL

Объявления и описания членов структуры находятся в файле:

- [devices/eliot1/drivers/hal\\_i2c.h](#)

## 5.46 Структура `i2c_master_transfer_t`

Структура дескриптора для неблокирующего обмена.

```
#include <hal_i2c.h>
```

Поля данных

- `uint32_t flags`
- `uint16_t slave_address`
- `i2c_addr_size_t addr_size`
- `i2c_direction_t direction`
- `uint32_t subaddress`
- `size_t subaddress_size`
- `void * data`
- `size_t data_size`

### 5.46.1 Подробное описание

Структура дескриптора для неблокирующего обмена.

Используется для передачи параметров обмена в `I2C_MasterTransferNonBlocking`.

Описание поля `flags`:

- `I2C_TransferStartFlag | I2C_TransferStopFlag`
  - Стандартная транзакция начинающаяся со `Start` и заканчивающаяся `Stop` условием.
- `I2C_TransferStartFlag`
  - Начальный пакет транзакции начинающаяся со `Start` и подразумевающий продолжение.
- `I2C_TransferDataFlag`
  - Пакет передачи данных без `Start` и `Stop` условия.
- `I2C_TransferStopFlag`
  - Завершающий пакет транзакции, заканчивающейся `Stop` условием.
- `I2C_TransferReStartFlag`
  - Продолжение транзакции с выдачей повторного старта и подразумевающей продолжение. Используется для смены направления обмена.
- `I2C_TransferReStartFlag | I2C_TransferStopFlag`
  - Продолжение транзакции с выдачей повторного старта и не допускающей продолжение. Используется для смены направления обмена.

### 5.46.2 Поля

#### 5.46.2.1 `addr_size`

```
i2c_addr_size_t i2c_master_transfer_t::addr_size
```

Разрядность Slave-адреса

#### 5.46.2.2 data

```
void* i2c_master_transfer_t::data
```

Данные для передачи

#### 5.46.2.3 data\_size

```
size_t i2c_master_transfer_t::data_size
```

Количество байтов для передачи

#### 5.46.2.4 direction

```
i2c_direction_t i2c_master_transfer_t::direction
```

Направление передачи Master -> Slave или Master <- Slave

#### 5.46.2.5 flags

```
uint32_t i2c_master_transfer_t::flags
```

Флаги управления передачей для управления специальным поведением, см. описание

#### 5.46.2.6 slave\_address

```
uint16_t i2c_master_transfer_t::slave_address
```

Slave-адрес

#### 5.46.2.7 subaddress

```
uint32_t i2c_master_transfer_t::subaddress
```

Дополнительный адрес. Сначала передан MSB

#### 5.46.2.8 subaddress\_size

```
size_t i2c_master_transfer_t::subaddress_size
```

Длина дополнительного адрес для отправки в байтах. Максимальный размер 4 байта

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_i2c.h`

## 5.47 Структура `i2c_slave_config_t`

Структура с настройками для инициализации Slave-модуля I2C.

```
#include <hal_i2c.h>
```

Поля данных

- `uint16_t address`
- `bool ack_gen_call`
- `i2c_addr_size_t i2c_addr_size`
- `bool enable_slave`

### 5.47.1 Подробное описание

Структура с настройками для инициализации Slave-модуля I2C.

Эта структура содержит параметры конфигурации для Slave-устройства I2C. Чтобы инициализировать ее значения по умолчанию, необходимо вызывать `I2C_SlaveGetDefaultConfig` и передать указатель на экземпляр структуры конфигурации.

### 5.47.2 Поля

#### 5.47.2.1 `ack_gen_call`

```
bool i2c_slave_config_t::ack_gen_call
```

Отмечать на General Call адрес

#### 5.47.2.2 `address`

```
uint16_t i2c_slave_config_t::address
```

Slave-адрес

#### 5.47.2.3 `enable_slave`

```
bool i2c_slave_config_t::enable_slave
```

Включить Slave-режим

#### 5.47.2.4 `i2c_addr_size`

```
i2c_addr_size_t i2c_slave_config_t::i2c_addr_size
```

Разрядность адреса: 7 или 10 бит

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_i2c.h`

## 5.48 Структура `i2c_slave_transfer_t`

I2C Slave структура обмена данными

```
#include <hal_i2c.h>
```

Поля данных

- `i2c_slave_handle_t * handle`
- `i2c_slave_event_transfer_t event`
- `uint32_t event_mask`
- `uint8_t * rx_data`
- `size_t rx_size`
- `const uint8_t * tx_data`
- `size_t tx_size`
- `size_t transferred_count`
- `i2c_status_t completion_status`

### 5.48.1 Подробное описание

I2C Slave структура обмена данными

### 5.48.2 Поля

#### 5.48.2.1 `completion_status`

`i2c_status_t i2c_slave_transfer_t::completion_status`

Код успеха или ошибки, описывающий завершение передачи. Применимо только для `I2C_SlaveEvent_Completion`

#### 5.48.2.2 `event`

`i2c_slave_event_transfer_t i2c_slave_transfer_t::event`

Причина, по которой вызывается функция обратного вызова

#### 5.48.2.3 `event_mask`

`uint32_t i2c_slave_transfer_t::event_mask`

Маска событий для вызова функции обратного вызова

#### 5.48.2.4 `handle`

`i2c_slave_handle_t* i2c_slave_transfer_t::handle`

Дескриптор, содержащий эту передачу



5.48.2.5 `rx_data`

```
uint8_t* i2c_slave_transfer_t::rx_data
```

Буфер обмена для приема данных

5.48.2.6 `rx_size`

```
size_t i2c_slave_transfer_t::rx_size
```

Количество данных на прием

5.48.2.7 `transferred_count`

```
size_t i2c_slave_transfer_t::transferred_count
```

Количество байтов переданных во время этого обмена

5.48.2.8 `tx_data`

```
const uint8_t* i2c_slave_transfer_t::tx_data
```

Буфер обмена для передачи данных

5.48.2.9 `tx_size`

```
size_t i2c_slave_transfer_t::tx_size
```

Количество данных на передачу

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_i2c.h`

## 5.49 Структура `jtm_config_t`

Структура конфигурации JTM.

```
#include <hal_jtm.h>
```

Поля данных

- `int16_t tcal`
- `int16_t wcal`
- `int16_t wtcalconf`
- `int16_t wtconf`

### 5.49.1 Подробное описание

Структура конфигурации JTM.

### 5.49.2 Поля

#### 5.49.2.1 tcal

```
int16_t jtm_config_t::tcal
```

Калибровочный параметр TCAL

#### 5.49.2.2 wcal

```
int16_t jtm_config_t::wcal
```

Калибровочный параметр WCAL

#### 5.49.2.3 wtcalconf

```
int16_t jtm_config_t::wtcalconf
```

Калибровочный параметр WTCALCONF

#### 5.49.2.4 wtconf

```
int16_t jtm_config_t::wtconf
```

Калибровочный параметр WTCONF

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_jtm.h`

## 5.50 Структура power\_config

Структура конфигурации блока POWER.

```
#include <hal_power.h>
```

Поля данных

- struct `power_mode_config` `run_configuration`
- struct `power_mode_config` `standby_configuration`
- struct `power_trim_config` `trim_configuration`
- bool `flash_low_voltage_read_enable`
- bool `dcdc_enable`
- enum `power_dcdc_vlevel_value` `vlevel0`
- enum `power_dcdc_vlevel_value` `vlevel1`
- enum `power_dcdc_vlevel_value` `vlevel2`

### 5.50.1 Подробное описание

Структура конфигурации блока POWER.

### 5.50.2 Поля

#### 5.50.2.1 dcdc\_enable

bool power\_config::dcdc\_enable

Включение встроенного DC-DC

#### 5.50.2.2 flash\_low\_voltage\_read\_enable

bool power\_config::flash\_low\_voltage\_read\_enable

Включение режима низковольтного чтения флеш-памяти

#### 5.50.2.3 run\_configuration

struct [power\\_mode\\_config](#) power\_config::run\_configuration

Конфигурация для режима работы RUN

#### 5.50.2.4 standby\_configuration

struct [power\\_mode\\_config](#) power\_config::standby\_configuration

Конфигурация для режима работы STANDBY

#### 5.50.2.5 trim\_configuration

struct [power\\_trim\\_config](#) power\_config::trim\_configuration

Подстроечные параметры APC и DC-DC

#### 5.50.2.6 vlevel0

enum [power\\_dcdc\\_vlevel\\_value](#) power\_config::vlevel0

Уровень напряжения VLEVEL0

#### 5.50.2.7 vlevel1

enum [power\\_dcdc\\_vlevel\\_value](#) power\_config::vlevel1

Уровень напряжения VLEVEL1

#### 5.50.2.8 vlevel2

```
enum power\_dcdc\_vlevel\_value power_config::vlevel2
```

Уровень напряжения VLEVEL2

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal\_power.h`

### 5.51 Структура power\_handle

Структура обработчика драйвера I2S.

```
#include <hal\_power.h>
```

Поля данных

- [power\\_callback\\_t](#) callback
- void \* [user\\_data](#)

#### 5.51.1 Подробное описание

Структура обработчика драйвера I2S.

#### 5.51.2 Поля

##### 5.51.2.1 callback

```
power\_callback\_t power_handle::callback
```

Функция обратного вызова

##### 5.51.2.2 user\_data

```
void* power_handle::user_data
```

Данные пользователя

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal\_power.h`

## 5.52 Структура power\_mode\_config

Структура параметров режима питания

```
#include <hal_power.h>
```

Поля данных

- enum power\_dcdc\_vlevel dcdc\_level
- enum power\_dcdc\_mode dcdc\_mode
- uint8\_t dcdc\_swdrv
- bool dcdc\_sink\_enable
- bool dcdc\_ccm\_enable
- bool dcdc\_low\_consumption\_enable
- enum power\_eco\_mode eco\_mode
- bool apc\_low\_clk\_enable
- enum power\_dcdc\_threshold apc\_eco\_threshold
- enum power\_flash\_mode flash\_power\_mode

### 5.52.1 Подробное описание

Структура параметров режима питания

### 5.52.2 Поля

#### 5.52.2.1 apc\_eco\_threshold

```
enum power_dcdc_threshold power_mode_config::apc_eco_threshold
```

Порог для монитора выходного напряжения DC-DC

#### 5.52.2.2 apc\_low\_clk\_enable

```
bool power_mode_config::apc_low_clk_enable
```

Включение семплирования монитора питания низкочастотным тактом

#### 5.52.2.3 dcdc\_ccm\_enable

```
bool power_mode_config::dcdc_ccm_enable
```

Включение генерации 16 ССМ-импульсов при снижении нагрузки

#### 5.52.2.4 dcdc\_level

```
enum power_dcdc_vlevel power_mode_config::dcdc_level
```

Выходное напряжение встроенного DC-DC

#### 5.52.2.5 dcdc\_low\_consumption\_enable

bool power\_mode\_config::dcdc\_low\_consumption\_enable

Включение низкопотребляющего режима конвертера

#### 5.52.2.6 dcdc\_mode

enum [power\\_dcdc\\_mode](#) power\_mode\_config::dcdc\_mode

Режим работы встроенного DC-DC

#### 5.52.2.7 dcdc\_sink\_enable

bool power\_mode\_config::dcdc\_sink\_enable

Включение резистора подтяжки выхода DC-DC к земле

#### 5.52.2.8 dcdc\_swdrv

uint8\_t power\_mode\_config::dcdc\_swdrv

Управление мощностью DC-DC

#### 5.52.2.9 eco\_mode

enum [power\\_eco\\_mode](#) power\_mode\_config::eco\_mode

Выбор режима ECO DC-DC и APC

#### 5.52.2.10 flash\_power\_mode

enum [power\\_flash\\_mode](#) power\_mode\_config::flash\_power\_mode

Режим работы флеш-памяти

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal\_power.h`

### 5.53 Структура power\_state

Структура параметров состояния блока POWER.

```
#include <hal_power.h>
```

Поля данных

- bool [vdda\\_is\\_lower\\_threshold](#)
- bool [dcdc\\_is\\_ready](#)
- enum [power\\_flash\\_mode](#) [flash\\_power\\_mode](#)
- bool [flash\\_low\\_voltage\\_read\\_enabled](#)

### 5.53.1 Подробное описание

Структура параметров состояния блока POWER.

### 5.53.2 Поля

#### 5.53.2.1 dcdc\_is\_ready

bool power\_state::dcdc\_is\_ready

Признак готовности DC-DC

#### 5.53.2.2 flash\_low\_voltage\_read\_enabled

bool power\_state::flash\_low\_voltage\_read\_enabled

Признак включения режима низковольтного чтения флеш-памяти

#### 5.53.2.3 flash\_power\_mode

enum [power\\_flash\\_mode](#) power\_state::flash\_power\_mode

Текущий режим работы флеш-памяти

#### 5.53.2.4 vdda\_is\_lower\_threshold

bool power\_state::vdda\_is\_lower\_threshold

Признак, VDDA ниже порогового уровня

Объявления и описания членов структуры находятся в файле:

- [devices/eliot1/drivers/hal\\_power.h](#)

## 5.54 Структура power\_trim\_config

Структура подстроечных параметров APC и DC-DC.

```
#include <hal_power.h>
```

Поля данных

- uint8\_t [apc\\_vref\\_it](#)
- uint8\_t [apc\\_vref\\_tt](#)
- uint8\_t [apc\\_vref\\_vt](#)
- bool [apc\\_force\\_trim](#)
- uint8\_t [dcdc\\_imax](#)
- uint8\_t [dcdc\\_imin](#)
- uint8\_t [dcdc\\_trimlc](#)

#### 5.54.1 Подробное описание

Структура подстроечных параметров APC и DC-DC.

#### 5.54.2 Поля

##### 5.54.2.1 [apc\\_force\\_trim](#)

bool power\_trim\_config::apc\_force\_trim

Включение подстройки vref в режиме сверхнизкого потребления

##### 5.54.2.2 [apc\\_vref\\_it](#)

uint8\_t power\_trim\_config::apc\_vref\_it

Подстройка APC по току

##### 5.54.2.3 [apc\\_vref\\_tt](#)

uint8\_t power\_trim\_config::apc\_vref\_tt

Подстройка APC по температуре

##### 5.54.2.4 [apc\\_vref\\_vt](#)

uint8\_t power\_trim\_config::apc\_vref\_vt

Подстройка APC по напряжению

##### 5.54.2.5 [dcdc\\_imax](#)

uint8\_t power\_trim\_config::dcdc\_imax

Подстройка DC-DC по пику тока в катушке



## 5.54.2.6 dcdc\_imin

```
uint8_t power_trim_config::dcdc_imin
```

Подстройка синхронной коррекции DC-DC

## 5.54.2.7 dcdc\_trimlc

```
uint8_t power_trim_config::dcdc_trimlc
```

Подстройка выходного фильтра DC-DC

Объявления и описания членов структуры находятся в файле:

- [devices/eliot1/drivers/hal\\_power.h](#)

## 5.55 Структура ppu\_config

Структура для инициализации

```
#include <hal_ppu.h>
```

Поля данных

- [ReqOffForCPU reqForCPU](#)

## 5.55.1 Подробное описание

Структура для инициализации

## 5.55.2 Поля

## 5.55.2.1 reqForCPU

```
ReqOffForCPU ppu_config::reqForCPU
```

Функция запроса остановки от CPU0 к CPU1

Объявления и описания членов структуры находятся в файле:

- [devices/eliot1/drivers/hal\\_ppu.h](#)

## 5.56 Структура pwm\_channel\_config

Конфигурация канала широтно-импульсного модулятора

```
#include <hal_pwm.h>
```

Поля данных

- uint32\_t channel
- enum pwm\_prescaler\_mode prescaler\_mode
- enum pwm\_prescaler\_cmd prescaler\_cmd
- enum pwm\_prescaler\_syncrst prescaler\_syncrst
- uint8\_t prescaler
- enum pwm\_prescaler\_divmux prescaler\_divmux
- enum pwm\_cntmode cntmode
- uint32\_t counter
- uint32\_t period
- enum pwm\_loadprd loadprd
- uint32\_t ctrphs
- enum pwm\_synchpsen synchpsen
- uint32\_t cmpa
- uint32\_t cmpb
- enum pwm\_scmpxmode scmpamode
- enum pwm\_scmpxmode scmpbmode
- enum pwm\_ldxmode ldamode
- enum pwm\_ldxmode ldbmode
- enum pwm\_outx\_cmd cnt\_eq\_prd\_outa
- enum pwm\_outx\_cmd cnt\_eq\_prd\_outb
- enum pwm\_outx\_cmd cnt\_eq\_cmpa\_dec\_outa
- enum pwm\_outx\_cmd cnt\_eq\_cmpa\_inc\_outa
- enum pwm\_outx\_cmd cnt\_eq\_cmpa\_dec\_outb
- enum pwm\_outx\_cmd cnt\_eq\_cmpa\_inc\_outb
- enum pwm\_outx\_cmd cnt\_eq\_cmpb\_dec\_outa
- enum pwm\_outx\_cmd cnt\_eq\_cmpb\_inc\_outa
- enum pwm\_outx\_cmd cnt\_eq\_cmpb\_dec\_outb
- enum pwm\_outx\_cmd cnt\_eq\_cmpb\_inc\_outb
- enum pwm\_outx\_cmd cnt\_eq\_zero\_outa
- enum pwm\_outx\_cmd cnt\_eq\_zero\_outb
- enum pwm\_outx\_cmd sw\_forced\_outa
- enum pwm\_outx\_cmd sw\_forced\_outb
- enum pwm\_outx\_cmd sw\_forced\_long\_outa
- enum pwm\_outx\_cmd sw\_forced\_long\_outb
- enum pwm\_ldcswrf ldcswrf
- enum pwm\_int\_en pwm\_int\_enable
- enum pwm\_int\_source pwm\_int\_source
- enum pwm\_eventprd eventprd
- uint16\_t dz\_rising\_edge\_delay\_clk
- uint16\_t dz\_falling\_edge\_delay\_clk
- enum pwm\_dz\_signal dz\_rising\_edge\_source
- enum pwm\_dz\_signal dz\_falling\_edge\_source
- enum pwm\_dz\_outx\_inv dz\_rising\_edge\_outa\_inv
- enum pwm\_dz\_outx\_inv dz\_falling\_edge\_outb\_inv
- enum pwm\_dz\_mode dz\_outa\_enable

- enum `pwm_dz_mode` `dz_outb_enable`
- enum `pwm_chopper_duty` `chopper_duty`
- enum `pwm_chopper_freq` `chopper_freq`
- enum `pwm_chopper_first_width` `chopper_first_width`
- enum `pwm_chopper_work` `chopper_work`
- `uint8_t` `inputs_mask_one`
- `uint8_t` `inputs_mask_mult`
- enum `pwm_trip_unit_action` `trip_unit_action_outa`
- enum `pwm_trip_unit_action` `trip_unit_action_outb`
- enum `pwm_int_en` `pwmtu_int_one`
- enum `pwm_int_en` `pwmtu_int_mult`
- enum `pwm_run_command` `cmd`

### 5.56.1 Подробное описание

Конфигурация канала широтно-импульсного модулятора

### 5.56.2 Поля

#### 5.56.2.1 channel

`uint32_t` `pwm_channel_config::channel`

Конфигурируемый канал

#### 5.56.2.2 chopper\_duty

enum `pwm_chopper_duty` `pwm_channel_config::chopper_duty`

Скважность дробящего сигнала

#### 5.56.2.3 chopper\_first\_width

enum `pwm_chopper_first_width` `pwm_channel_config::chopper_first_width`

Ширина первого импульса

#### 5.56.2.4 chopper\_freq

enum `pwm_chopper_freq` `pwm_channel_config::chopper_freq`

Частота дробящего сигнала

#### 5.56.2.5 chopper\_work

enum `pwm_chopper_work` `pwm_channel_config::chopper_work`

Работа блока дробления

## 5.56.2.6 cmd

enum [pwm\\_run\\_command](#) pwm\_channel\_config::cmd

Разрешение работы

## 5.56.2.7 cmpa

uint32\_t pwm\_channel\_config::cmpa

Регистр сравнения СМРА

## 5.56.2.8 cmpb

uint32\_t pwm\_channel\_config::cmpb

Регистр сравнения СМРВ

## 5.56.2.9 cnt\_eq\_cmpa\_dec\_outa

enum [pwm\\_outx\\_cmd](#) pwm\_channel\_config::cnt\_eq\_cmpa\_dec\_outa

Событие канала OUTA по совпадению счетчика и регистра СМРА при декременте счетчика

## 5.56.2.10 cnt\_eq\_cmpa\_dec\_outb

enum [pwm\\_outx\\_cmd](#) pwm\_channel\_config::cnt\_eq\_cmpa\_dec\_outb

Событие канала OUTB по совпадению счетчика и регистра СМРА при декременте счетчика

## 5.56.2.11 cnt\_eq\_cmpa\_inc\_outa

enum [pwm\\_outx\\_cmd](#) pwm\_channel\_config::cnt\_eq\_cmpa\_inc\_outa

Событие канала OUTA по совпадению счетчика и регистра СМРА при инкременте счетчика

## 5.56.2.12 cnt\_eq\_cmpa\_inc\_outb

enum [pwm\\_outx\\_cmd](#) pwm\_channel\_config::cnt\_eq\_cmpa\_inc\_outb

Событие канала OUTB по совпадению счетчика и регистра СМРА при инкременте счетчика

## 5.56.2.13 cnt\_eq\_cmpb\_dec\_outa

enum [pwm\\_outx\\_cmd](#) pwm\_channel\_config::cnt\_eq\_cmpb\_dec\_outa

Событие канала OUTA по совпадению счетчика и регистра СМРВ при декременте счетчика

## 5.56.2.14 cnt\_eq\_cmpb\_dec\_outb

enum [pwm\\_outx\\_cmd](#) pwm\_channel\_config::cnt\_eq\_cmpb\_dec\_outb

Событие канала OUTB по совпадению счетчика и регистра CMPB при декременте счетчика

## 5.56.2.15 cnt\_eq\_cmpb\_inc\_outa

enum [pwm\\_outx\\_cmd](#) pwm\_channel\_config::cnt\_eq\_cmpb\_inc\_outa

Событие канала OUTA по совпадению счетчика и регистра CMPB при инкременте счетчика

## 5.56.2.16 cnt\_eq\_cmpb\_inc\_outb

enum [pwm\\_outx\\_cmd](#) pwm\_channel\_config::cnt\_eq\_cmpb\_inc\_outb

Событие канала OUTB по совпадению счетчика и регистра CMPB при инкременте счетчика

## 5.56.2.17 cnt\_eq\_prd\_outa

enum [pwm\\_outx\\_cmd](#) pwm\_channel\_config::cnt\_eq\_prd\_outa

Событие канала OUTA по совпадению счетчика и периода

## 5.56.2.18 cnt\_eq\_prd\_outb

enum [pwm\\_outx\\_cmd](#) pwm\_channel\_config::cnt\_eq\_prd\_outb

Событие канала OUTB по совпадению счетчика и периода

## 5.56.2.19 cnt\_eq\_zero\_outa

enum [pwm\\_outx\\_cmd](#) pwm\_channel\_config::cnt\_eq\_zero\_outa

Событие канала OUTA по достижению счетчиком 0

## 5.56.2.20 cnt\_eq\_zero\_outb

enum [pwm\\_outx\\_cmd](#) pwm\_channel\_config::cnt\_eq\_zero\_outb

Событие канала OUTB по достижению счетчиком 0

## 5.56.2.21 cntmode

enum [pwm\\_cntmode](#) pwm\_channel\_config::cntmode

Режим работы

## 5.56.2.22 counter

```
uint32_t pwm_channel_config::counter
```

Значение счетчика

## 5.56.2.23 ctrphs

```
uint32_t pwm_channel_config::ctrphs
```

Фазы синхронизации блока

## 5.56.2.24 dz\_falling\_edge\_delay\_clk

```
uint16_t pwm_channel_config::dz_falling_edge_delay_clk
```

Запратная зона в тактах около ниспадающего фронта

## 5.56.2.25 dz\_falling\_edge\_outb\_inv

```
enum pwm_dz_outx_inv pwm_channel_config::dz_falling_edge_outb_inv
```

Полярность OUTB после генерации запрещенной зоны

## 5.56.2.26 dz\_falling\_edge\_source

```
enum pwm_dz_signal pwm_channel_config::dz_falling_edge_source
```

Источник сигнала для генерации запрещенной зоны около ниспадающего фронта

## 5.56.2.27 dz\_outa\_enable

```
enum pwm_dz_mode pwm_channel_config::dz_outa_enable
```

Выбор режима работы блока запрещенной зоны при формировании OUTA

## 5.56.2.28 dz\_outb\_enable

```
enum pwm_dz_mode pwm_channel_config::dz_outb_enable
```

Выбор режима работы блока запрещенной зоны при формировании OUTB

## 5.56.2.29 dz\_rising\_edge\_delay\_clk

```
uint16_t pwm_channel_config::dz_rising_edge_delay_clk
```

Запратная зона в тактах около возрастающего фронта

## 5.56.2.30 dz\_rising\_edge\_outa\_inv

enum [pwm\\_dz\\_outx\\_inv](#) pwm\_channel\_config::dz\_rising\_edge\_outa\_inv

Полярность OUTA после генерации запрещенной зоны

## 5.56.2.31 dz\_rising\_edge\_source

enum [pwm\\_dz\\_signal](#) pwm\_channel\_config::dz\_rising\_edge\_source

Источник сигнала для генерации запрещенной зоны около возрастающего фронта

## 5.56.2.32 eventprd

enum [pwm\\_eventprd](#) pwm\_channel\_config::eventprd

Период прерывания

## 5.56.2.33 inputs\_mask\_mult

uint8\_t pwm\_channel\_config::inputs\_mask\_mult

Выбор используемых TU[7:0] сигналов для канала PWM работающих в режиме многократного срабатывания

## 5.56.2.34 inputs\_mask\_one

uint8\_t pwm\_channel\_config::inputs\_mask\_one

Выбор используемых TU[7:0] сигналов для канала PWM работающих в режиме однократного срабатывания

## 5.56.2.35 ldamode

enum [pwm\\_ldxmode](#) pwm\_channel\_config::ldamode

режима загрузки данных из теневого регистра в активный СМРА

## 5.56.2.36 ldbmode

enum [pwm\\_ldxmode](#) pwm\_channel\_config::ldbmode

режима загрузки данных из теневого регистра в активный СМРВ

## 5.56.2.37 ldcswrf

enum [pwm\\_ldcswrf](#) pwm\_channel\_config::ldcswrf

Загрузка активного регистра из теневого регистра для регистра программного управления выходами

## 5.56.2.38 loadprd

enum [pwm\\_loadprd](#) pwm\_channel\_config::loadprd

Моментом переписи данных

## 5.56.2.39 period

uint32\_t pwm\_channel\_config::period

Период

## 5.56.2.40 prescaler

uint8\_t pwm\_channel\_config::prescaler

Предделитель (PRESPRD)

## 5.56.2.41 prescaler\_cmd

enum [pwm\\_prescaler\\_cmd](#) pwm\_channel\_config::prescaler\_cmd

Режим сброса счетчика предделителя

## 5.56.2.42 prescaler\_divmux

enum [pwm\\_prescaler\\_divmux](#) pwm\_channel\_config::prescaler\_divmux

Мультиплексор частоты

## 5.56.2.43 prescaler\_mode

enum [pwm\\_prescaler\\_mode](#) pwm\_channel\_config::prescaler\_mode

Режим управления



## 5.56.2.44 prescaler\_syncrst

enum [pwm\\_prescaler\\_syncrst](#) pwm\_channel\_config::prescaler\_syncrst

Режим сброса при событиях

## 5.56.2.45 pwm\_int\_enable

enum [pwm\\_int\\_en](#) pwm\_channel\_config::pwm\_int\_enable

Разрешение прерывания

## 5.56.2.46 pwm\_int\_source

enum [pwm\\_int\\_source](#) pwm\_channel\_config::pwm\_int\_source

Источник прерывания

## 5.56.2.47 pwmtu\_int\_mult

enum [pwm\\_int\\_en](#) pwm\_channel\_config::pwmtu\_int\_mult

Разрешение прерывания блока

## 5.56.2.48 pwmtu\_int\_one

enum [pwm\\_int\\_en](#) pwm\_channel\_config::pwmtu\_int\_one

Разрешение прерывания блока

## 5.56.2.49 scmpamode

enum [pwm\\_scmpxmode](#) pwm\_channel\_config::scmpamode

Режим работы регистра СМРА

## 5.56.2.50 scmpbmode

enum [pwm\\_scmpxmode](#) pwm\_channel\_config::scmpbmode

Режим работы регистра СМРВ

## 5.56.2.51 sw\_forced\_long\_outa

enum [pwm\\_outx\\_cmd](#) pwm\_channel\_config::sw\_forced\_long\_outa

Событие канала ОУТА при долговременном программном событии

5.56.2.52 `sw_forced_long_outb`

```
enum pwm\_outx\_cmd pwm_channel_config::sw_forced_long_outb
```

Событие канала OUTB при долговременном программном событии

5.56.2.53 `sw_forced_outa`

```
enum pwm\_outx\_cmd pwm_channel_config::sw_forced_outa
```

Событие канала OUTA при кратковременном программном событии

5.56.2.54 `sw_forced_outb`

```
enum pwm\_outx\_cmd pwm_channel_config::sw_forced_outb
```

Событие канала OUTB при кратковременном программном событии

5.56.2.55 `syncphsen`

```
enum pwm\_syncphsen pwm_channel_config::syncphsen
```

Загрузки счетчика из регистра фазы

5.56.2.56 `trip_unit_action_outa`

```
enum pwm\_trip\_unit\_action pwm_channel_config::trip_unit_action_outa
```

Реакции на событие блока trip unit для OUTA

5.56.2.57 `trip_unit_action_outb`

```
enum pwm\_trip\_unit\_action pwm_channel_config::trip_unit_action_outb
```

Реакции на событие блока trip unit для OUTB

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal\_pwm.h`

5.57 Структура `qspi_dma_handle_t`

Дескриптор QSPI-DMA передачи

```
#include <hal_qspi_dma.h>
```

Поля данных

- `QSPI_Type * base`
- `dma_handle_t * tx_handle`
- `dma_handle_t * rx_handle`
- `void * dummy_data`
- `dma_descriptor_t * tx_desc`
- `dma_descriptor_t * rx_desc`

### 5.57.1 Подробное описание

Дескриптор QSPI-DMA передачи

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_qspi_dma.h`

## 5.58 Структура `rcw_alarm_reg`

Регистр времени пробуждения

```
#include <hal_rcw.h>
```

Поля данных

- `uint32_t alarm`: ((31) - (0) + 1)

### 5.58.1 Подробное описание

Регистр времени пробуждения

Значение времени пробуждения, сравниваемое с регистром `TIME`.

### 5.58.2 Поля

#### 5.58.2.1 `alarm`

```
uint32_t rcw_alarm_reg::alarm
```

Значение времени в тиках

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_rcw.h`

## 5.59 Структура `rcw_config`

Структура используемая для конфигурирования RWC.

```
#include <hal_rcw.h>
```

Поля данных

- bool `trimload`
- uint32\_t `time`
- uint32\_t `alarm_time`
- enum `rcw_lfe_bypass` `lfe_bypass`
- uint32\_t `trim_lfi`
- uint32\_t `trim_lfe`
- uint32\_t `wake_stat1`
- enum `rcw_shutdown_force` `shutdown_force`
- enum `rcw_wkup_enable` `wake_in_en`
- uint32\_t `pl`
- uint32\_t `pz`
- enum `rcw_alarm_enable` `alarm_en`
- enum `rcw_reset_type` `reset_en`
- enum `rcw_rtclk_divisor` `clkdiv`
- enum `rcw_rtclk_type` `osc_sel`
- enum `rcw_time_clk_sel` `time_clk_sel`
- uint32\_t `general`
- enum `rcw_wake_up_polarity` `wake_pol`
- enum `rcw_wake_up_irq_enable` `wake_en`
- enum `rcw_freq_serial` `presc`

### 5.59.1 Подробное описание

Структура используемая для конфигурирования RWC.

### 5.59.2 Поля

#### 5.59.2.1 `alarm_en`

```
enum rcw_alarm_enable rcw_config::alarm_en
```

Разрешение прерывания RWC\_ALARM по совпадению значений регистров TIME и ALARM.

#### 5.59.2.2 `alarm_time`

```
uint32_t rcw_config::alarm_time
```

Значение времени пробуждения - будильника

## 5.59.2.3 clkdiv

```
enum rwc_rtclk_divisor rwc_config::clkdiv
```

Значение делителя тактового сигнала clk\_32kHz и соответствующего выхода RTCCLK

## 5.59.2.4 general

```
uint32_t rwc_config::general
```

Регистр общего назначения

## 5.59.2.5 lfe\_bypass

```
enum rwc_lfe_bypass rwc_config::lfe_bypass
```

Режим работы осциллятора LFE

## 5.59.2.6 osc\_sel

```
enum rwc_rtclk_type rwc_config::osc_sel
```

Выбор осциллятора

## 5.59.2.7 pl

```
uint32_t rwc_config::pl
```

Бит устанавливается при первом включении питания. Сбрасывать нельзя

## 5.59.2.8 presc

```
enum rwc_freq_serial rwc_config::presc
```

Делитель для формирования тактовой частоты интерфейса. Рекомендуемое значение частоты - RWC\_FS1MHz

## 5.59.2.9 pz

```
uint32_t rwc_config::pz
```

Бит устанавливается при первом включении питания. Сбрасывать нельзя

## 5.59.2.10 reset\_en

```
enum rwc_reset_type rwc_config::reset_en
```

Влияние сброса SRSTn на состояние внутренних регистров RWC

## 5.59.2.11 shutdown\_force

enum [rwc\\_shutdown\\_force](#) rwc\_config::shutdown\_force

Принудительный переход системы в режим SHUTDOWN. Не рекомендуется использовать.

## 5.59.2.12 time

uint32\_t rwc\_config::time

Значение счетчика времени

## 5.59.2.13 time\_clk\_sel

enum [rwc\\_time\\_clk\\_sel](#) rwc\_config::time\_clk\_sel

Выбор сигнала для тактирования счетчика времени

## 5.59.2.14 trim\_lfe

uint32\_t rwc\_config::trim\_lfe

Подстройка частоты 1 Гц для оциллятора LFE

## 5.59.2.15 trim\_lfi

uint32\_t rwc\_config::trim\_lfi

Подстройка частоты 1 Гц для оциллятора LFI

## 5.59.2.16 trimload

bool rwc\_config::trimload

Признак применения значения trim\_lfi и trim\_lfe

## 5.59.2.17 wake\_en

enum [rwc\\_wake\\_up\\_irq\\_enable](#) rwc\_config::wake\_en

Разрешение прерывания RWC\_WKUP

## 5.59.2.18 wake\_in\_en

enum [rwc\\_wkup\\_enable](#) rwc\_config::wake\_in\_en

Разрешение работы входа WKUP

## 5.59.2.19 wake\_pol

```
enum rwc_wake_up_polarity rwc_config::wake_pol
```

Полярность сигнала WKUP для генерирования прерывания

## 5.59.2.20 wake\_stat1

```
uint32_t rwc_config::wake_stat1
```

Бит устанавливается при выходе из режима SHUTDOWN. Бит сбрасывается при переходе в режим SHUTDOWN.

Объявления и описания членов структуры находятся в файле:

- devices/eliot1/drivers/[hal\\_rwc.h](#)

## 5.60 Структура rwc\_config\_reg

Конфигурационный регистр

```
#include <hal_rwc.h>
```

Поля данных

- uint32\_t [time\\_clk\\_sel](#): ((0) - (0) + 1)
- uint32\_t : ((3) - (1) + 1)
- uint32\_t [osc\\_sel](#): ((4) - (4) + 1)
- uint32\_t : ((5) - (5) + 1)
- uint32\_t [clk\\_div](#): ((10) - (6) + 1)
- uint32\_t [reset\\_en](#): ((11) - (11) + 1)
- uint32\_t [alarm\\_en](#): ((12) - (12) + 1)
- uint32\_t [pz](#): ((13) - (13) + 1)
- uint32\_t [pl](#): ((14) - (14) + 1)
- uint32\_t [wake\\_in\\_en](#): ((15) - (15) + 1)
- uint32\_t : ((29) - (16) + 1)
- uint32\_t [shutdown\\_force](#): ((30) - (30) + 1)
- uint32\_t [wake\\_stat1](#): ((31) - (31) + 1)

## 5.60.1 Подробное описание

Конфигурационный регистр

Заметки

Управляет выбором тактирования счетчика времени, выбором используемого осциллятора, установкой делителя частоты RTCCLK, сбросов внутренних регистров, разрешением прерывания ALARM, работой входа WKUP, принудительным переходом в режим SHUTDOWN, а также содержит признак режима SHUTDOWN.

## 5.60.2 Поля

### 5.60.2.1 \_\_pad0\_\_

uint32\_t rwc\_config\_reg::\_\_pad0\_\_

Поле зарезервировано

### 5.60.2.2 \_\_pad1\_\_

uint32\_t rwc\_config\_reg::\_\_pad1\_\_

Поле зарезервировано. Исходное состояние 1

### 5.60.2.3 \_\_pad2\_\_

uint32\_t rwc\_config\_reg::\_\_pad2\_\_

Поле зарезервировано

### 5.60.2.4 alarm\_en

uint32\_t rwc\_config\_reg::alarm\_en

Разрешение прерывания RWC\_ALARM по совпадению значений регистров TIME и ALARM

### 5.60.2.5 clk\_div

uint32\_t rwc\_config\_reg::clk\_div

Поле для деления тактового сигнала ([rwc\\_rtclk\\_divisor](#))

### 5.60.2.6 osc\_sel

uint32\_t rwc\_config\_reg::osc\_sel

Выбор осциллятора ( [rwc\\_rtclk\\_type](#) )

### 5.60.2.7 pl

uint32\_t rwc\_config\_reg::pl

Бит устанавливается при первом включении питания. Сбрасывать нельзя.



## 5.60.2.8 pz

```
uint32_t rwc_config_reg::pz
```

Бит устанавливается при первом включении питания. Сбрасывать нельзя.

## 5.60.2.9 reset\_en

```
uint32_t rwc_config_reg::reset_en
```

Поле влияние сброса SRSTn на состояние внутренних регистров RWC ( [rwc\\_reset\\_type](#) )

## 5.60.2.10 shutdown\_force

```
uint32_t rwc_config_reg::shutdown_force
```

Установка этого бита приводит к принудительному переходу системы в режим SHUTDOWN. Не рекомендуется использовать.

## 5.60.2.11 time\_clk\_sel

```
uint32_t rwc_config_reg::time_clk_sel
```

Выбор сигнала для тактирования счетчика времени ( [rwc\\_time\\_clk\\_sel](#) )

## 5.60.2.12 wake\_in\_en

```
uint32_t rwc_config_reg::wake_in_en
```

Разрешение работы входа WKUP

## 5.60.2.13 wake\_stat1

```
uint32_t rwc_config_reg::wake_stat1
```

Бит устанавливается при выходе из режима SHUTDOWN, сбрасывается при переходе в режим SHUTDOWN.

Объявления и описания членов структуры находятся в файле:

- [devices/eliot1/drivers/hal\\_rwc.h](#)

## 5.61 Структура rwc\_general\_reg

Регистр общего назначения

```
#include <hal_rwc.h>
```

Поля данных

- `uint32_t time`:  $((31) - (0) + 1)$

### 5.61.1 Подробное описание

Регистр общего назначения

Заметки

В документации назван GPR. Сохраняет свое состояние в любом режиме работы микросхемы при наличии питания VBAT. Сброс регистра выполняется только при первичном включении питания VBAT.

### 5.61.2 Поля

#### 5.61.2.1 time

`uint32_t rwc_general_reg::time`

Поле для хранения информации

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_rwc.h`

## 5.62 Структура `rcw_time_reg`

Регистр текущего значения счетчика времени

```
#include <hal_rwc.h>
```

Поля данных

- `uint32_t time`:  $((31) - (0) + 1)$

### 5.62.1 Подробное описание

Регистр текущего значения счетчика времени

Заметки

Запись устанавливает значение счетчика времени. Чтение возвращает текущее значение счетчика.

### 5.62.2 Поля

#### 5.62.2.1 time

```
uint32_t rcw_time_reg::time
```

Значение времени в тиках

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_rcw.h`

## 5.63 Структура `rcw_trim_reg`

Регистр подстройки осцилляторов

```
#include <hal_rcw.h>
```

Поля данных

- `uint32_t trim_lfe`: ((10) - (0) + 1)
- `uint32_t trim_lfi`: ((21) - (11) + 1)
- `uint32_t lfe_bypass`: ((22) - (22) + 1)
- `uint32_t` : ((24) - (23) + 1)
- `uint32_t wake_stat2`: ((25) - (25) + 1)
- `uint32_t wake_stat3`: ((26) - (26) + 1)
- `uint32_t` : ((31) - (27) + 1)

### 5.63.1 Подробное описание

Регистр подстройки осцилляторов

Заметки

Регистр поля подстройки осцилляторов ( `trim_lfe` и `trim_lfi`), поле режима работы LFE ( `lfe_bypass`) и поля-признаки режима SHUTDOWN ( `wake_stat2` и `wake_stat3`). Значения полей `trim_lfe` и `trim_lfi` применяются только после записи регистра `rcw_trimload_reg`.

### 5.63.2 Поля

#### 5.63.2.1 \_\_pad0\_\_

```
uint32_t rcw_trim_reg::__pad0__
```

Поле зарезервировано

## 5.63.2.2 \_\_pad1\_\_

```
uint32_t rwc_trim_reg::__pad1__
```

Поле зарезервировано. Исходное состояние 0x10

## 5.63.2.3 lfe\_bypass

```
uint32_t rwc_trim_reg::lfe_bypass
```

Режим работы осциллятора LFE ( [rwc\\_lfe\\_bypass](#) )

## 5.63.2.4 trim\_lfe

```
uint32_t rwc_trim_reg::trim_lfe
```

Подстройка частоты 1 Гц для осциллятора LFE

## 5.63.2.5 trim\_lfi

```
uint32_t rwc_trim_reg::trim_lfi
```

Подстройка частоты 1 Гц для осциллятора LFI

## 5.63.2.6 wake\_stat2

```
uint32_t rwc_trim_reg::wake_stat2
```

Бит устанавливается при выходе из режима SHUTDOWN по внешнему событию WKUP, либо по сбросу SRSTn. Бит сбрасывается при переходе в режим SHUTDOWN.

## 5.63.2.7 wake\_stat3

```
uint32_t rwc_trim_reg::wake_stat3
```

Бит устанавливается после первичной подачи питания на RWC. Бит сбрасывается при переходе в режим SHUTDOWN.

Объявления и описания членов структуры находятся в файле:

- [devices/eliot1/drivers/hal\\_rwc.h](#)

## 5.64 Структура rwc\_trimload\_reg

Регистр записи значения подстройки из регистра TRIM.

```
#include <hal_rwc.h>
```

Поля данных

- uint32\_t [trimload](#): ((0) - (0) + 1)
- uint32\_t : ((31) - (1) + 1)

### 5.64.1 Подробное описание

Регистр записи значения подстройки из регистра TRIM.

Заметки

Запись в этот регистр применяет значение подстройки в регистре TRIM

### 5.64.2 Поля

#### 5.64.2.1 \_\_pad0\_\_

uint32\_t rwc\_trimload\_reg::\_\_pad0\_\_

Резерв

#### 5.64.2.2 trimload

uint32\_t rwc\_trimload\_reg::trimload

Поле для инициации записи

Объявления и описания членов структуры находятся в файле:

- [devices/eliot1/drivers/hal\\_rwc.h](#)

## 5.65 Объединение rwc\_union\_reg

Объединение для доступа к регистрам

```
#include <hal_rwc.h>
```

Поля данных

- uint32\_t [reg\\_value](#)
- struct [rwc\\_trimload\\_reg](#) trimload
- struct [rwc\\_time\\_reg](#) time
- struct [rwc\\_alarm\\_reg](#) alarm
- struct [rwc\\_trim\\_reg](#) trim
- struct [rwc\\_config\\_reg](#) config
- struct [rwc\\_general\\_reg](#) general
- struct [rwc\\_wake\\_config\\_reg](#) wake\_config

### 5.65.1 Подробное описание

Объединение для доступа к регистрам

Заметки

Объединение для доступа к регистрам через функции [RWC\\_GetInternalRegister](#) и [RWC\\_SetInternalRegister](#).

### 5.65.2 Поля

#### 5.65.2.1 alarm

```
struct rwc\_alarm\_reg rwc_union_reg::alarm
```

Регистр времени пробуждения

#### 5.65.2.2 config

```
struct rwc\_config\_reg rwc_union_reg::config
```

Конфигурационный регистр

#### 5.65.2.3 general

```
struct rwc\_general\_reg rwc_union_reg::general
```

Регистр общего назначения

#### 5.65.2.4 reg\_value

```
uint32_t rwc_union_reg::reg_value
```

Представление регистра как `uint32_t`

#### 5.65.2.5 time

```
struct rwc\_time\_reg rwc_union_reg::time
```

Регистр текущего значения счетчика времени

#### 5.65.2.6 trim

```
struct rwc\_trim\_reg rwc_union_reg::trim
```

Регистр подстройки осцилляторов

## 5.65.2.7 trimload

```
struct rcw_trimload_reg rcw_union_reg::trimload
```

Регистр записи значения подстройки из регистра TRIM

## 5.65.2.8 wake\_config

```
struct rcw_wake_config_reg rcw_union_reg::wake_config
```

Регистр настройки контроллера пробуждения

Объявления и описания членов объединения находятся в файле:

- `devices/eliot1/drivers/hal_rcw.h`

5.66 Структура `rcw_wake_config_reg`

Регистр настройки контроллера пробуждения

```
#include <hal_rcw.h>
```

Поля данных

- `uint32_t wake_en`: ((0) - (0) + 1)
- `uint32_t` : ((15) - (1) + 1)
- `uint32_t wake_pol`: ((16) - (16) + 1)
- `uint32_t` : ((31) - (17) + 1)

## 5.66.1 Подробное описание

Регистр настройки контроллера пробуждения

Заметки

В документации назван `wakecfg`. Позволяет задать полярность сигнала и управлять разрешением прерывания `RWC_WKUP`.

## 5.66.2 Поля

5.66.2.1 `__pad0__`

```
uint32_t rcw_wake_config_reg::__pad0__
```

Резерв

### 5.66.2.2 \_\_pad1\_\_

uint32\_t rwc\_wake\_config\_reg::\_\_pad1\_\_

Резерв

### 5.66.2.3 wake\_en

uint32\_t rwc\_wake\_config\_reg::wake\_en

Разрешение прерывания RWC\_WKUP.

### 5.66.2.4 wake\_pol

uint32\_t rwc\_wake\_config\_reg::wake\_pol

Полярность сигнала WKUP для генерирования прерывания ( [rwc\\_wake\\_up\\_polarity](#) )

Объявления и описания членов структуры находятся в файле:

- [devices/eliot1/drivers/hal\\_rwc.h](#)

## 5.67 Структура sdmmc\_card\_t

Контекст драйвера контроллера SDMMC.

```
#include <hal_sdmmc.h>
```

Поля данных

- int32\_t [dev\\_num](#)
- const [sdmmc\\_port\\_cfg\\_t](#) \* [cfg](#)
- SDMMC\_Type \* [regs](#)
- uint32\_t [rca](#)
- int32\_t [type](#)
- int32\_t [sdhc\\_mode](#)
- int32\_t [hs\\_mode](#)
- int32\_t [ddr\\_mode](#)
- int32\_t [version](#)
- [sdmmc\\_voltage\\_t](#) [gpio\\_vol](#)
- int32\_t [need\\_1V8en](#)
- uint32\_t [freq\\_input](#)
- int32\_t [freq\\_divider](#)
- int32\_t [lock](#)
- uint64\_t [total\\_size](#)
- uint32\_t [cid](#) [4]
- uint32\_t [csd](#) [4]



### 5.67.1 Подробное описание

Контекст драйвера контроллера SDMMC.

### 5.67.2 Поля

#### 5.67.2.1 cfg

```
const sdmmc_port_cfg_t* sdmmc_card_t::cfg
```

Конфигурация контроллера SDMMC

#### 5.67.2.2 cid

```
uint32_t sdmmc_card_t::cid[4]
```

Значение регистра CID

#### 5.67.2.3 csd

```
uint32_t sdmmc_card_t::csd[4]
```

Значение регистра CSD

#### 5.67.2.4 ddr\_mode

```
int32_t sdmmc_card_t::ddr_mode
```

Доступность режима Double Data Rate: 0 - SDR, 1 - DDR

#### 5.67.2.5 dev\_num

```
int32_t sdmmc_card_t::dev_num
```

Номер порта SDMMC контроллера

#### 5.67.2.6 freq\_divider

```
int32_t sdmmc_card_t::freq_divider
```

Делитель выходной тактовой частоты контроллера SDMMC

#### 5.67.2.7 freq\_input

```
uint32_t sdmmc_card_t::freq_input
```

Входная тактовая частота контроллера SDMMC

## 5.67.2.8 gpio\_vol

`sdmmc_voltage_t sdmmc_card_t::gpio_vol`

Напряжение внешних выводов GPIO и карты SD или MMC

## 5.67.2.9 hs\_mode

`int32_t sdmmc_card_t::hs_mode`

Доступность режима High Speed: 0 - Default, 1 - High

## 5.67.2.10 lock

`int32_t sdmmc_card_t::lock`

Флаг блокировки контроллера SDMMC

## 5.67.2.11 need\_1V8en

`int32_t sdmmc_card_t::need_1V8en`

Необходимость переключения напряжения питания с 3,3 В на 1,8 В у SD карты

## 5.67.2.12 rca

`uint32_t sdmmc_card_t::rca`

Относительный адрес карты

## 5.67.2.13 regs

`SDMMC_Type* sdmmc_card_t::regs`

Указатель на структуру регистров контроллера SDMMC

## 5.67.2.14 sdhc\_mode

`int32_t sdmmc_card_t::sdhc_mode`

Доступность режима SDHC: 0 - SDSC, 1 - SDHC/SDXC

## 5.67.2.15 total\_size

`uint64_t sdmmc_card_t::total_size`

Общий размер пространства памяти карты

## 5.67.2.16 type

```
int32_t sdmmc_card_t::type
```

Тип карты: 0 - MMC, 1 - SD

## 5.67.2.17 version

```
int32_t sdmmc_card_t::version
```

Версия SD карты: 1 - версия до 1.1, 2 - версия 2.2 и выше

Объявления и описания членов структуры находятся в файле:

- [devices/eliot1/drivers/hal\\_sdmmc.h](#)

## 5.68 Структура sdmmc\_cfg\_pin\_map

Конфигурация выводов контроллера SDMMC.

```
#include <hal_sdmmc_cfg.h>
```

Поля данных

- uint8\_t [pull\\_en](#)
- uint8\_t [max\\_current](#)
- uint8\_t [CK](#)
- uint8\_t [CMD](#)
- uint8\_t [D0](#)
- uint8\_t [D1](#)
- uint8\_t [D2](#)
- uint8\_t [D3](#)
- uint8\_t [D4](#)
- uint8\_t [D5](#)
- uint8\_t [D6](#)
- uint8\_t [D7](#)
- uint8\_t [CD](#)

## 5.68.1 Подробное описание

Конфигурация выводов контроллера SDMMC.

## 5.68.2 Поля

## 5.68.2.1 CD

```
uint8_t sdmmc_cfg_pin_map::CD
```

Вывод обнаружения карты SD

#### 5.68.2.2 CK

uint8\_t sdmmc\_cfg\_pin\_map::CK

Вывод выходной частоты

#### 5.68.2.3 CMD

uint8\_t sdmmc\_cfg\_pin\_map::CMD

Вывод управления

#### 5.68.2.4 D0

uint8\_t sdmmc\_cfg\_pin\_map::D0

Вывод данных бит 0

#### 5.68.2.5 D1

uint8\_t sdmmc\_cfg\_pin\_map::D1

Вывод данных бит 1

#### 5.68.2.6 D2

uint8\_t sdmmc\_cfg\_pin\_map::D2

Вывод данных бит 2

#### 5.68.2.7 D3

uint8\_t sdmmc\_cfg\_pin\_map::D3

Вывод данных бит 3

#### 5.68.2.8 D4

uint8\_t sdmmc\_cfg\_pin\_map::D4

Вывод данных бит 4

#### 5.68.2.9 D5

uint8\_t sdmmc\_cfg\_pin\_map::D5

Вывод данных бит 5

## 5.68.2.10 D6

```
uint8_t sdmme_cfg_pin_map::D6
```

Вывод данных бит 6

## 5.68.2.11 D7

```
uint8_t sdmme_cfg_pin_map::D7
```

Вывод данных бит 7

## 5.68.2.12 max\_current

```
uint8_t sdmme_cfg_pin_map::max_current
```

Максимальный ток выводов

## 5.68.2.13 pull\_en

```
uint8_t sdmme_cfg_pin_map::pull_en
```

Включение программных резистивных подтяжек на сигнальных выводах

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_sdmme_cfg.h`

5.69 Структура `sdmme_port_cfg_t`

Конфигурация контроллера SDMMC.

```
#include <hal_sdmme_cfg.h>
```

Поля данных

- `uint32_t reg_base [2]`
- `uint32_t slot_count`
- `uint8_t slot_type`
- `uint8_t emmc_8bit_en`
- `uint8_t hs_en`
- `uint8_t sd_uhs_mode`
- `uint32_t freq_out_init`
- `uint32_t freq_out`
- `uint32_t timeout_cd`
- `uint32_t timeout_init`
- `void(* delay_us )(uint32_t)`
- `sdmme_cfg_pin_map pin_map`

### 5.69.1 Подробное описание

Конфигурация контроллера SDMMC.

### 5.69.2 Поля

#### 5.69.2.1 delay\_us

```
void(* sdmmc_port_cfg_t::delay_us) (uint32_t)
```

Указатель на функцию задержки в мкс

#### 5.69.2.2 emmc\_8bit\_en

```
uint8_t sdmmc_port_cfg_t::emmc_8bit_en
```

Включение 8-битного режима (только для MMC)

#### 5.69.2.3 freq\_out

```
uint32_t sdmmc_port_cfg_t::freq_out
```

Рабочая выходная частота карты SD/MMC

#### 5.69.2.4 freq\_out\_init

```
uint32_t sdmmc_port_cfg_t::freq_out_init
```

Выходная частота карты SD/MMC во время инициализации

#### 5.69.2.5 hs\_en

```
uint8_t sdmmc_port_cfg_t::hs_en
```

Включение режима High Speed

#### 5.69.2.6 pin\_map

```
sdmmc\_cfg\_pin\_map sdmmc_port_cfg_t::pin_map
```

Карта выводов SDMMC

#### 5.69.2.7 reg\_base

```
uint32_t sdmmc_port_cfg_t::reg_base[2]
```

Базовые адреса модулей SDMMC

5.69.2.8 `sd_uhs_mode``uint8_t sdmmc_port_cfg_t::sd_uhs_mode`

Выбор режима работы для UHS-I

5.69.2.9 `slot_count``uint32_t sdmmc_port_cfg_t::slot_count`

Количество слотов под карту

5.69.2.10 `slot_type``uint8_t sdmmc_port_cfg_t::slot_type`

Тип слота карты карты

5.69.2.11 `timeout_cd``uint32_t sdmmc_port_cfg_t::timeout_cd`

Максимальное время ожидания определения наличия карты

5.69.2.12 `timeout_init``uint32_t sdmmc_port_cfg_t::timeout_init`

Максимальное время ожидания инициализации карты

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_sdmmc_cfg.h`

5.70 Структура `spi_config_internal_t`

Внутренняя структура конфигурации модуля SPI.

```
#include <hal_spi.h>
```

Поля данных

- `spi_shift_direction_t shift_dir`
- `uint8_t frame_width_bits`
- `uint8_t frame_width_bytes`

### 5.70.1 Подробное описание

Внутренняя структура конфигурации модуля SPI.

### 5.70.2 Поля

#### 5.70.2.1 frame\_width\_bits

`uint8_t spi_config_internal_t::frame_width_bits`

Размер кадра данных в битах (допустимые значения: 4 - 32)

#### 5.70.2.2 frame\_width\_bytes

`uint8_t spi_config_internal_t::frame_width_bytes`

Размер кадра данных в байтах (допустимые значения: 1, 2 и 4)

#### 5.70.2.3 shift\_dir

`spi_shift_direction_t spi_config_internal_t::shift_dir`

Направление сдвига данных при выдаче в шину

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_spi.h`

## 5.71 Структура spi\_config\_t

Структура конфигурации для Master SPI.

```
#include <hal_spi.h>
```

Поля данных

- `struct {`  
     `bool loopback_enable`  
     `uint32_t baud_rate_bps`  
     `} master`
- `spi_frame_width_t data_width_bits`
- `spi_shift_direction_t direction`
- `spi_frame_format_t frame_format`
- `spi_microwire_cfg_t microwire_cfg`
- `spi_motorola_cfg_t motorola_cfg`



### 5.71.1 Подробное описание

Структура конфигурации для Master SPI.

### 5.71.2 Поля

#### 5.71.2.1 `baud_rate_bps`

`uint32_t spi_config_t::baud_rate_bps`

Скорость обмена SPI в Hz

#### 5.71.2.2 `data_width_bits`

`spi_frame_width_t spi_config_t::data_width_bits`

Размер кадра данных

#### 5.71.2.3 `direction`

`spi_shift_direction_t spi_config_t::direction`

Формат передачи данных (MSB или LSB)

#### 5.71.2.4 `frame_format`

`spi_frame_format_t spi_config_t::frame_format`

Формат кадра (протокла) передачи данных

#### 5.71.2.5 `loopback_enable`

`bool spi_config_t::loopback_enable`

Включить закольцовывание (в целях тестирования)

#### 5.71.2.6 `[struct]`

`struct { ... } spi_config_t::master`

Конфигурация, актуальная только для master режима

#### 5.71.2.7 `microwire_cfg`

`spi_microwire_cfg_t spi_config_t::microwire_cfg`

Конфигурация для протокола National Semiconductor Microwire

### 5.71.2.8 motorola\_cfg

`spi_motorola_cfg_t` `spi_config_t::motorola_cfg`

Конфигурация для протокола Motorola

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_spi.h`

## 5.72 Структура `spi_half_duplex_transfer_t`

Структура SPI для полудуплексной приемо-передачи в режиме Master.

```
#include <hal_spi.h>
```

Поля данных

- `uint8_t * tx_data`
- `uint8_t * rx_data`
- `size_t tx_data_size`
- `size_t rx_data_size`

### 5.72.1 Подробное описание

Структура SPI для полудуплексной приемо-передачи в режиме Master.

### 5.72.2 Поля

#### 5.72.2.1 `rx_data`

`uint8_t* spi_half_duplex_transfer_t::rx_data`

Приемный буфер

#### 5.72.2.2 `rx_data_size`

`size_t spi_half_duplex_transfer_t::rx_data_size`

Количество принятых байт

#### 5.72.2.3 `tx_data`

`uint8_t* spi_half_duplex_transfer_t::tx_data`

Буфер на отправку

## 5.72.2.4 tx\_data\_size

```
size_t spi_half_duplex_transfer_t::tx_data_size
```

Количество байт для передачи

Объявления и описания членов структуры находятся в файле:

- [devices/eliot1/drivers/hal\\_spi.h](#)

## 5.73 Структура spi\_handle

SPI структура дескриптора для работы по прерыванию

```
#include <hal_spi.h>
```

Поля данных

- volatile uint8\_t \* [tx\\_data](#)
- volatile uint8\_t \* [rx\\_data](#)
- volatile size\_t [tx\\_remaining\\_bytes](#)
- volatile size\_t [rx\\_remaining\\_bytes](#)
- int8\_t [instance](#)
- size\_t [total\\_byte\\_to\\_transfer](#)
- int32\_t [state](#)
- [spi\\_master\\_callback\\_t](#) [callback](#)
- void \* [user\\_data](#)
- uint8\_t [frame\\_width\\_bits](#)
- uint8\_t [frame\\_width\\_bytes](#)
- [spi\\_mode\\_t](#) [mode](#)

## 5.73.1 Подробное описание

SPI структура дескриптора для работы по прерыванию

Заметки

Поскольку количество полученных и отправленных кадров должно совпадать для завершения передачи, значит, если отправленное количество равно x, а полученное количество равно y, то `#to_receive_frame` равно x-y.

## 5.73.2 Поля

## 5.73.2.1 callback

```
spi\_master\_callback\_t spi_handle::callback
```

Указатель на пользовательскую функцию обратного вызова

#### 5.73.2.2 frame\_width\_bits

uint8\_t spi\_handle::frame\_width\_bits

Размер кадра данных в битах (допустимые значения: 4 - 32)

#### 5.73.2.3 frame\_width\_bytes

uint8\_t spi\_handle::frame\_width\_bytes

Размер кадра данных в байтах (допустимые значения: 1, 2 и 4)

#### 5.73.2.4 instance

int8\_t spi\_handle::instance

Индекс модуля SPI

#### 5.73.2.5 mode

[spi\\_mode\\_t](#) spi\_handle::mode

Режим обмена

#### 5.73.2.6 rx\_data

volatile uint8\_t\* spi\_handle::rx\_data

Rx буфер

#### 5.73.2.7 rx\_remaining\_bytes

volatile size\_t spi\_handle::rx\_remaining\_bytes

Количество байт, которые осталось принять

#### 5.73.2.8 state

int32\_t spi\_handle::state

Текущее состояние модуля SPI, может быть комбинацией состояний [spi\\_trans\\_status](#) по ИЛИ

#### 5.73.2.9 total\_byte\_to\_transfer

size\_t spi\_handle::total\_byte\_to\_transfer

Общее количество байтов для обмена, если полудуплекс, то передача + прием

## 5.73.2.10 tx\_data

```
volatile uint8_t* spi_handle::tx_data
```

Тх буфер

## 5.73.2.11 tx\_remaining\_bytes

```
volatile size_t spi_handle::tx_remaining_bytes
```

Количество байт, которые осталось передать

## 5.73.2.12 user\_data

```
void* spi_handle::user_data
```

Параметр пользовательской функции обратного вызова

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_spi.h`

## 5.74 Структура spi\_microwire\_cfg\_t

Конфигурация для протокола Microwire National Semiconductor.

```
#include <hal_spi.h>
```

Поля данных

- `microwire_ctrlword_len_t ctrl_word_len`
- `microwire_busy_ready_check_t busy_ready_check`
- `microwire_tx_rx_t tx_rx`
- `microwire_single_serial_t single_serial`

## 5.74.1 Подробное описание

Конфигурация для протокола Microwire National Semiconductor.

## 5.74.2 Поля

## 5.74.2.1 busy\_ready\_check

```
microwire_busy_ready_check_t spi_microwire_cfg_t::busy_ready_check
```

Включить/отключить проверку busy/ready флаг (регистр SR)

#### 5.74.2.2 ctrl\_word\_len

[microwire\\_ctrlword\\_len\\_t](#) spi\_microwire\_cfg\_t::ctrl\_word\_len

Выбор длины управляющего слова для протокола Microwire

#### 5.74.2.3 single\_serial

[microwire\\_single\\_serial\\_t](#) spi\_microwire\_cfg\_t::single\_serial

Одиночная или последовательная передача

#### 5.74.2.4 tx\_rx

[microwire\\_tx\\_rx\\_t](#) spi\_microwire\_cfg\_t::tx\_rx

Направление передачи слова данных

Объявления и описания членов структуры находятся в файле:

- [devices/eliot1/drivers/hal\\_spi.h](#)

### 5.75 Структура spi\_motorola\_cfg\_t

Конфигурация для протокола Motorola SPI.

```
#include <hal_spi.h>
```

Поля данных

- [spi\\_motorola\\_clk\\_pol\\_t](#) clk\_pol
- [spi\\_motorola\\_cap\\_data\\_t](#) cap\_data

#### 5.75.1 Подробное описание

Конфигурация для протокола Motorola SPI.

#### 5.75.2 Поля

##### 5.75.2.1 cap\_data

[spi\\_motorola\\_cap\\_data\\_t](#) spi\_motorola\_cfg\_t::cap\_data

Захват данных происходит по переднему фронту или по заднему фронту

## 5.75.2.2 clk\_pol

```
spi_motorola_clk_pol_t spi_motorola_cfg_t::clk_pol
```

Полярность тактового сигнала, при отсутствия передаваемых данных в режиме Master

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_spi.h`

## 5.76 Структура spi\_transfer\_t

Структура SPI для приемо-передачи

```
#include <hal_spi.h>
```

Поля данных

- `uint8_t * tx_data`
- `uint8_t * rx_data`
- `size_t data_size`

## 5.76.1 Подробное описание

Структура SPI для приемо-передачи

Заметки

Если `tx_data` равна `NULL`, то осуществляется только прием, если `rx_data` равна `NULL` - только передача. Если `tx_data` и `rx_data` одновременно не равны `NULL`, то осуществляются и прием, и передача.

## 5.76.2 Поля

## 5.76.2.1 data\_size

```
size_t spi_transfer_t::data_size
```

Количество байт

## 5.76.2.2 rx\_data

```
uint8_t* spi_transfer_t::rx_data
```

Приемный буфер

### 5.76.2.3 tx\_data

```
uint8_t* spi_transfer_t::tx_data
```

Буфер на отправку

Объявления и описания членов структуры находятся в файле:

- [devices/eliot1/drivers/hal\\_spi.h](#)

## 5.77 Структура timer\_hardware\_config

Конфигурация аппаратной части таймера общего назначения

```
#include <hal_timer.h>
```

Поля данных

- uint32\_t [start\\_value](#)
- uint32\_t [reload\\_value](#)
- uint32\_t [interrupt\\_enable](#)
- enum [timer\\_type\\_of\\_counting\\_work\\_type](#)
- uint32\_t [start\\_enable](#)

### 5.77.1 Подробное описание

Конфигурация аппаратной части таймера общего назначения

#### 5.77.2 Поля

##### 5.77.2.1 interrupt\_enable

```
uint32_t timer_hardware_config::interrupt_enable
```

Разрешение прерывания

##### 5.77.2.2 reload\_value

```
uint32_t timer_hardware_config::reload_value
```

Загружаемое значение счетчика

##### 5.77.2.3 start\_enable

```
uint32_t timer_hardware_config::start_enable
```

Разрешение работы



5.77.2.4 `start_value`

```
uint32_t timer_hardware_config::start_value
```

Стартовое значение счетчика

5.77.2.5 `work_type`

```
enum timer_type_of_counting timer_hardware_config::work_type
```

Тип работы

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal_timer.h`

5.78 Структура `uart_config`

Конфигурация UART.

```
#include <hal_uart.h>
```

Поля данных

- `uint32_t baudrate_bps`
- `bool enable_parity`
- `enum uart_parity_mode parity_mode`
- `bool parity_manual`
- `enum uart_stop_bit_count stop_bit_count`
- `enum uart_data_len bit_count_per_char`
- `bool enable_rxfifo`
- `bool enable_txfifo`
- `bool enable_loopback`
- `bool enable_infrared`
- `bool enable_hardware_flow_control`
- `bool break_line`

## 5.78.1 Подробное описание

Конфигурация UART.

## 5.78.2 Поля

5.78.2.1 `baudrate_bps`

```
uint32_t uart_config::baudrate_bps
```

Скорость интерфейса UART

#### 5.78.2.2 bit\_count\_per\_char

enum [uart\\_data\\_len](#) uart\_config::bit\_count\_per\_char

Количество бит данных в передаваемом символе: от 5 до 8 бит

#### 5.78.2.3 break\_line

bool uart\_config::break\_line

Бит обрыва линии

#### 5.78.2.4 enable\_hardware\_flow\_control

bool uart\_config::enable\_hardware\_flow\_control

Включено ли аппаратное управление потоком RTS/CTS

#### 5.78.2.5 enable\_infrared

bool uart\_config::enable\_infrared

Включен ли инфракрасный режим интерфейса

#### 5.78.2.6 enable\_loopback

bool uart\_config::enable\_loopback

Включена ли петля

#### 5.78.2.7 enable\_parity

bool uart\_config::enable\_parity

Включена ли четность (по умолчанию - выключена)

#### 5.78.2.8 enable\_rxfifo

bool uart\_config::enable\_rxfifo

Включена ли RxFIFO

#### 5.78.2.9 enable\_txfifo

bool uart\_config::enable\_txfifo

Включена ли TxFIFO

5.78.2.10 `parity_manual`

```
bool uart_config::parity_manual
```

Ручное управление битом четности

5.78.2.11 `parity_mode`

```
enum uart\_parity\_mode uart_config::parity_mode
```

Режим четности - чет или нечет

5.78.2.12 `stop_bit_count`

```
enum uart\_stop\_bit\_count uart_config::stop_bit_count
```

Количество стоп-битов

Объявления и описания членов структуры находятся в файле:

- `devices/eliot1/drivers/hal\_uart.h`

5.79 Структура `uart_handle`

Дескриптор состояния приема/передачи для неблокирующих функций обмена

```
#include <hal_uart.h>
```

Поля данных

- `volatile const uint8_t * tx\_data`
- `volatile size_t tx\_data\_size`
- `size_t tx\_data\_size\_all`
- `uint8_t * tx\_ring\_buffer`
- `size_t tx\_ring\_buffer\_size`
- `volatile uint16_t tx\_ring\_buffer\_head`
- `volatile uint16_t tx\_ring\_buffer\_tail`
- `volatile uint8_t * rx\_data`
- `volatile size_t rx\_data\_size`
- `size_t rx\_data\_size\_all`
- `uint8_t * rx\_ring\_buffer`
- `size_t rx\_ring\_buffer\_size`
- `volatile uint16_t rx\_ring\_buffer\_head`
- `volatile uint16_t rx\_ring\_buffer\_tail`
- `uart\_transfer\_callback\_t callback`
- `void * user\_data`
- `volatile uint8_t tx\_state`
- `volatile uint8_t rx\_state`

### 5.79.1 Подробное описание

Дескриптор состояния приема/передачи для неблокирующих функций обмена

### 5.79.2 Поля

#### 5.79.2.1 callback

```
uart_transfer_callback_t uart_handle::callback
```

Функция обратного вызова

#### 5.79.2.2 rx\_data

```
volatile uint8_t* uart_handle::rx_data
```

Адрес оставшихся данных для получения

#### 5.79.2.3 rx\_data\_size

```
volatile size_t uart_handle::rx_data_size
```

Размер оставшихся данных для получения

#### 5.79.2.4 rx\_data\_size\_all

```
size_t uart_handle::rx_data_size_all
```

Размер получаемых данных

#### 5.79.2.5 rx\_ring\_buffer

```
uint8_t* uart_handle::rx_ring_buffer
```

Начальный адрес кольцевого буфера приемника

#### 5.79.2.6 rx\_ring\_buffer\_head

```
volatile uint16_t uart_handle::rx_ring_buffer_head
```

Индекс для драйвера для сохранения полученных данных в кольцевом буфере

#### 5.79.2.7 rx\_ring\_buffer\_size

```
size_t uart_handle::rx_ring_buffer_size
```

Размер кольцевого буфера

5.79.2.8 `rx_ring_buffer_tail`

```
volatile uint16_t uart_handle::rx_ring_buffer_tail
```

Индекс, позволяющий пользователю получать данные из кольцевого буфера

5.79.2.9 `rx_state`

```
volatile uint8_t uart_handle::rx_state
```

Состояние приема

5.79.2.10 `tx_data`

```
volatile const uint8_t* uart_handle::tx_data
```

Адрес оставшихся данных для отправки

5.79.2.11 `tx_data_size`

```
volatile size_t uart_handle::tx_data_size
```

Размер оставшихся данных для отправки

5.79.2.12 `tx_data_size_all`

```
size_t uart_handle::tx_data_size_all
```

Размер данных для отправки

5.79.2.13 `tx_ring_buffer`

```
uint8_t* uart_handle::tx_ring_buffer
```

Начальный адрес кольцевого буфера передатчика

5.79.2.14 `tx_ring_buffer_head`

```
volatile uint16_t uart_handle::tx_ring_buffer_head
```

Индекс, позволяющий пользователю записывать данные в кольцевой буфер Tx

5.79.2.15 `tx_ring_buffer_size`

```
size_t uart_handle::tx_ring_buffer_size
```

Размер кольцевого буфера передатчика

### 5.79.2.16 tx\_ring\_buffer\_tail

volatile uint16\_t uart\_handle::tx\_ring\_buffer\_tail

Индекс для драйвера для отправки данных из кольцевого буфера Tx

### 5.79.2.17 tx\_state

volatile uint8\_t uart\_handle::tx\_state

Состояние передачи

### 5.79.2.18 user\_data

void\* uart\_handle::user\_data

UART-параметр функции обратного вызова

Объявления и описания членов структуры находятся в файле:

- [devices/eliot1/drivers/hal\\_uart.h](#)

## 5.80 Структура uart\_transfer

Указатель на буфер приема или передачи

```
#include <hal_uart.h>
```

Поля данных

- union {  
    uint8\_t \* [rx\\_data](#)  
    uint8\_t const \* [tx\\_data](#)  
};
- size\_t [data\\_size](#)

### 5.80.1 Подробное описание

Указатель на буфер приема или передачи

Раздельные указатели rx\_data и tx\_data, потому что tx\_data const.

## 5.80.2 Поля

### 5.80.2.1 data\_size

size\_t uart\_transfer::data\_size

Счетчик байтов

### 5.80.2.2 rx\_data

uint8\_t\* uart\_transfer::rx\_data

Буфер для приема

### 5.80.2.3 tx\_data

uint8\_t const\* uart\_transfer::tx\_data

Буфер на передачу

Объявления и описания членов структуры находятся в файле:

- devices/eliot1/drivers/[hal\\_uart.h](#)

## 5.81 Структура vtu\_config

Структура для конфигурации VTU.

```
#include <hal_vtu.h>
```

Поля данных

- enum [vtu\\_mode](#) mode
- enum [vtu\\_capture\\_edge\\_control](#) capture\_edge\_control1
- enum [vtu\\_capture\\_edge\\_control](#) capture\_edge\_control2
- enum [vtu\\_pwm\\_polarity](#) pwm\_polarity
- enum [vtu\\_pwm\\_polarity](#) pwm\_polarity2
- enum [vtu\\_interrupt\\_control](#) interrupt\_control
- uint8\_t [prescaler](#)
- uint16\_t [counter](#)
- uint16\_t [period](#)
- uint16\_t [duty\\_cycle\\_capture](#)

### 5.81.1 Подробное описание

Структура для конфигурации VTU.

## 5.81.2 Поля

### 5.81.2.1 capture\_edge\_control1

enum [vtu\\_capture\\_edge\\_control](#) vtu\_config::capture\_edge\_control1

Управление фронтами захвата для режима захвата для ТЮ1

### 5.81.2.2 capture\_edge\_control2

enum [vtu\\_capture\\_edge\\_control](#) vtu\_config::capture\_edge\_control2

Управление фронтами захвата для режима захвата для ТЮ2

### 5.81.2.3 counter

uint16\_t vtu\_config::counter

Начальное значение счетчика

### 5.81.2.4 duty\_cycle\_capture

uint16\_t vtu\_config::duty\_cycle\_capture

Ширина импульса

### 5.81.2.5 interrupt\_control

enum [vtu\\_interrupt\\_control](#) vtu\_config::interrupt\_control

Разрешение прерываний

### 5.81.2.6 mode

enum [vtu\\_mode](#) vtu\_config::mode

Режимы работы тамера, кроме VTU\_LowPower

### 5.81.2.7 period

uint16\_t vtu\_config::period

Период ШИМ



## 5.81.2.8 prescaler

uint8\_t vtu\_config::prescaler

Значение предделителя

## 5.81.2.9 pwm\_polarity

enum vtu\_pwm\_polarity vtu\_config::pwm\_polarity

Полярность ШИМ

## 5.81.2.10 pwm\_polarity2

enum vtu\_pwm\_polarity vtu\_config::pwm\_polarity2

Полярность второго вывода ШИМ для 16-битного режима

Объявления и описания членов структуры находятся в файле:

- devices/eliot1/drivers/[hal\\_vtu.h](#)

## 5.82 Структура wdt\_config

Структура инициализации сторожевого таймера

#include <hal\_wdt.h>

Поля данных

- uint32\_t load
- enum wdt\_resen\_type resen
- enum wdt\_inten\_type inten

## 5.82.1 Подробное описание

Структура инициализации сторожевого таймера

## 5.82.2 Поля

## 5.82.2.1 inten

enum wdt\_inten\_type wdt\_config::inten

Разрешение прерывания и работы сторожевого таймера

## 5.82.2.2 load

uint32\_t wdt\_config::load

Время срабатывания предупреждения или половина времени таймаута

## 5.82.2.3 resen

enum wdt\_resen\_type wdt\_config::resen

Разрешение сброса по таймауту

Объявления и описания членов структуры находятся в файле:

- devices/eliot1/drivers/[hal\\_wdt.h](#)



## Глава 6

# Файлы

### 6.1 Файл devices/eliot1/drivers/hal\_can.h

Интерфейс драйвера модуля ввода-вывода по интерфейсу CAN.

```
#include "hal_common.h"
```

Структуры данных

- [struct \\_can\\_tx\\_buffer\\_frame](#)  
Структура буфера передачи кадра CAN.
- [struct \\_can\\_rx\\_buffer\\_frame](#)  
Структура буфера приема кадра CAN.
- [struct \\_can\\_frame\\_filter](#)  
Фильтр принятых кадров CAN.
- [struct \\_can\\_frame\\_filter\\_config](#)  
Конфигурация фильтрации принятых кадров CAN.
- [struct \\_ttcan\\_config](#)  
Временные параметры передачи битов CAN.
- [struct \\_can\\_timing\\_config](#)  
Временные параметры передачи битов CAN.
- [struct \\_can\\_ptb\\_config](#)  
Параметры высокоприоритетного буфера выдачи
- [struct \\_can\\_stb\\_config](#)  
Параметры низкоприоритетного буфера выдачи
- [struct \\_can\\_rxb\\_config](#)  
Параметры буфера приема
- [struct \\_can\\_config](#)  
Структура конфигурации контроллера CAN.
- [struct \\_can\\_tx\\_transfer](#)  
Структура для передачи кадра CAN в неблокирующем режиме (по прерыванию)
- [struct \\_can\\_rx\\_transfer](#)  
Структура для приема кадра CAN в неблокирующем режиме (по прерыванию)
- [struct \\_can\\_handle](#)  
Структура дескриптора драйвера CAN.

## Макросы

- `#define HAL_CAN_DRIVER_VERSION (MAKE_VERSION(1, 1, 0))`  
Версия драйвера CAN.
- `#define CAN_NB_OF_FILTERS 4`  
Количество входных фильтров CAN.

## Определения типов

- `typedef enum __can_status can_status_t`  
Коды возврата функций драйвера CAN.
- `typedef enum __can_kind_of_error can_kind_of_error_t`  
Виды ошибок на линии CAN.
- `typedef enum __can_flag can_flag_t`  
Флаги прерываний и состояний CAN.
- `typedef enum __canfd_mode canfd_mode_t`  
Режимы работы по протоколу CAN FD (есть отличия в расчете контрольной суммы и формировании битов стаффинга)
- `typedef enum __can_stb_discipline can_stb_discipline_t`  
Дисциплина выдачи из низкоприоритетного буфера
- `typedef enum __can_bytes_in_datafield can_bytes_in_datafield_t`  
Размер данных кадра CAN, указываемый в поле DLC.
- `typedef struct __can_tx_buffer_frame can_tx_buffer_frame_t`  
Структура буфера передачи кадра CAN.
- `typedef struct __can_rx_buffer_frame can_rx_buffer_frame_t`  
Структура буфера приема кадра CAN.
- `typedef struct __can_frame_filter can_frame_filter_t`  
Фильтр принятых кадров CAN.
- `typedef struct __can_frame_filter_config can_frame_filter_config_t`  
Конфигурация фильтрации принятых кадров CAN.
- `typedef enum __ttcan_timer_prescaler ttcan_timer_prescaler_t`  
Символьные константы значений делителя блока TTCAN.
- `typedef enum __ttcan_trigger_type ttcan_trigger_type_t`  
Тип триггера TTCAN.
- `typedef struct __ttcan_config ttcan_config_t`  
Временные параметры передачи битов CAN.
- `typedef struct __can_timing_config can_timing_config_t`  
Временные параметры передачи битов CAN.
- `typedef struct __can_ptb_config can_ptb_config_t`  
Параметры высокоприоритетного буфера выдачи
- `typedef struct __can_stb_config can_stb_config_t`  
Параметры низкоприоритетного буфера выдачи
- `typedef struct __can_rxb_config can_rxb_config_t`  
Параметры буфера приема
- `typedef struct __can_config can_config_t`  
Структура конфигурации контроллера CAN.
- `typedef struct __can_tx_transfer can_tx_transfer_t`  
Структура для передачи кадра CAN в неблокирующем режиме (по прерыванию)
- `typedef struct __can_rx_transfer can_rx_transfer_t`  
Структура для приема кадра CAN в неблокирующем режиме (по прерыванию)
- `typedef struct __can_handle can_handle_t`  
Декларация типа дескриптора драйвера CAN.
- `typedef void(* can_transfer_callback_t)(CAN_Type *base, can_handle_t *handle, can_status_t status, can_flag_t interrupt_flag, void *user_data)`  
Функция обратного вызова CAN.

## Перечисления

- enum [\\_can\\_status](#)  
Коды возврата функций драйвера CAN.
- enum [\\_can\\_kind\\_of\\_error](#)  
Виды ошибок на линии CAN.
- enum [\\_can\\_flag](#)  
Флаги прерываний и состояний CAN.
- enum [\\_canfd\\_mode](#)  
Режимы работы по протоколу CAN FD (есть отличия в расчете контрольной суммы и формировании битов стаффинга)
- enum [\\_can\\_stb\\_discipline](#)  
Дисциплина выдачи из низкоприоритетного буфера
- enum [\\_can\\_bytes\\_in\\_datafield](#)  
Размер данных кадра CAN, указываемый в поле DLC.
- enum [\\_ttcan\\_timer\\_prescaler](#)  
Символьные константы значений делителя блока TTCAN.
- enum [\\_ttcan\\_trigger\\_type](#)  
Тип триггера TTCAN.

## Функции

## Инициализация и деинициализация

- [can\\_status\\_t CAN\\_Init](#) (CAN\_Type \*base, const [can\\_config\\_t](#) \*config)  
Инициализация драйвера CAN.
- void [CAN\\_Deinit](#) (CAN\_Type \*base)  
Деинициализация драйвера CAN.
- void [CAN\\_GetDefaultConfig](#) ([can\\_config\\_t](#) \*config, uint32\_t source\_clock\_hz)  
Получение параметров драйвера CAN по умолчанию
- void [CAN\\_EnterNormalMode](#) (CAN\_Type \*base)  
Переключение контроллера CAN в рабочий режим
- void [CAN\\_EnterStandbyMode](#) (CAN\_Type \*base)  
Переключение контроллера CAN в режим ожидания

## Конфигурирование настроек приемопередачи

- bool [CAN\\_CalculateImprovedTimingValues](#) (uint32\_t baudrate, uint32\_t baudrate\_data, uint32\_t source\_clock\_hz, [can\\_timing\\_config\\_t](#) \*pconfig)  
Расчет рекомендуемых временных параметров (битовых таймингов) для указанных скоростей обмена в сегменте управления и данных
- void [CAN\\_SetArbitrationTimingConfig](#) (CAN\_Type \*base, const [can\\_timing\\_config\\_t](#) \*config)  
Установка временных параметров (битовых таймингов) CAN.
- void [CAN\\_SetPrimaryTxBufferConfig](#) (CAN\_Type \*base, const [can\\_ptb\\_config\\_t](#) \*config)  
Установка параметров высокоприоритетного буфера выдачи
- void [CAN\\_SetSecondaryTxBufferConfig](#) (CAN\_Type \*base, const [can\\_stb\\_config\\_t](#) \*config)  
Установка параметров низкоприоритетного буфера выдачи
- void [CAN\\_SetRxBufferConfig](#) (CAN\_Type \*base, const [can\\_rxb\\_config\\_t](#) \*config)  
Установка параметров буфера приема
- void [CAN\\_SetFilterConfig](#) (CAN\_Type \*base, const [can\\_frame\\_filter\\_config\\_t](#) \*config)  
Установка параметров фильтрации кадров при приеме
- void [CAN\\_SetTTCANConfig](#) (CAN\_Type \*base, const [ttcan\\_config\\_t](#) \*config)  
Установка параметров работы контроллера в режиме TTCAN.

### Флаги статусов и прерываний

- bool `CAN_GetStatusFlag` (CAN\_Type \*base, can\_flag\_t idx, can\_status\_t \*status)  
Получение состояния флага состояния/прерывания
- uint32\_t `CAN_GetStatusFlagMask` (CAN\_Type \*base)  
Получение маски активных прерываний
- can\_status\_t `CAN_ClearStatusFlag` (CAN\_Type \*base, can\_flag\_t idx)  
Сброс флага состояния/прерывания
- void `CAN_ClearStatusFlagMask` (CAN\_Type \*base, uint32\_t mask)  
Сброс флагов прерываний по маске

### Управление прерываниями

- can\_status\_t `CAN_EnableInterrupt` (CAN\_Type \*base, can\_flag\_t idx)  
Разрешение прерывания
- void `CAN_EnableInterruptMask` (CAN\_Type \*base, uint32\_t mask)  
Разрешение прерываний по маске
- bool `CAN_IsInterruptEnabled` (CAN\_Type \*base, can\_flag\_t idx, can\_status\_t \*status)  
Запрос - разрешено ли прерывание CAN.
- uint32\_t `CAN_GetEnabledInterruptMask` (CAN\_Type \*base)  
Запрос маски разрешенных прерываний CAN.
- can\_status\_t `CAN_DisableInterrupt` (CAN\_Type \*base, can\_flag\_t idx)  
Запрет прерывания
- void `CAN_DisableInterruptMask` (CAN\_Type \*base, uint32\_t mask)  
Запрет прерываний по маске

### Прямое управление выдачей и приемом

- bool `CAN_IsPrimaryTransmitRequestPending` (CAN\_Type \*base)  
Получение признака требования выдачи для высокоприоритетного буфера
- bool `CAN_IsSecondaryTransmitRequestPending` (CAN\_Type \*base)  
Получение признака требования выдачи для низкоприоритетного буфера
- bool `CAN_IsSecondaryTxBufferEmpty` (CAN\_Type \*base)  
Получение признака опустошения низкоприоритетного буфера
- bool `CAN_IsSecondaryTxBufferMoreThanHalfFull` (CAN\_Type \*base)  
Получение признака заполнения низкоприоритетного буфера выше середины
- bool `CAN_IsSecondaryTxBufferFull` (CAN\_Type \*base)  
Получение признака заполнения низкоприоритетного буфера.
- can\_status\_t `CAN_WritePrimaryTxBuffer` (CAN\_Type \*base, const can\_tx\_buffer\_frame\_t \*ptxframe)  
Запись кадра в высокоприоритетный буфер передачи.
- can\_status\_t `CAN_WriteSecondaryTxBuffer` (CAN\_Type \*base, const can\_tx\_buffer\_frame\_t \*ptxframe)  
Запись кадра в низкоприоритетный буфер передачи
- void `CAN_AbortPrimaryTxBuffer` (CAN\_Type \*base)  
Отмена выдачи кадра из высокоприоритетного буфера
- void `CAN_AbortSecondaryTxBuffer` (CAN\_Type \*base)  
Отмена выдачи кадров из низкоприоритетного буфера
- bool `CAN_IsRxBufferEmpty` (CAN\_Type \*base)  
Получение признака опустошения буфера приема
- bool `CAN_IsRxBufferAlmostFull` (CAN\_Type \*base)  
Получение признака заполнения буфера приема до границы "почти полный".
- bool `CAN_IsRxBufferFull` (CAN\_Type \*base)  
Получение признака заполнения буфера приема
- can\_status\_t `CAN_ReadRxBuffer` (CAN\_Type \*base, can\_rx\_buffer\_frame\_t \*prxframe)  
Чтение принятого кадра из приемной очереди

## Транзакционные передача и прием

- `can_status_t CAN_TransferSendPrimaryBlocking` (CAN\_Type \*base, can\_tx\_buffer\_frame\_t \*ptxframe, size\_t nb\_frames)  
Блокирующая выдача кадра через высокоприоритетный буфер выдачи
- `can_status_t CAN_TransferSendSecondaryBlocking` (CAN\_Type \*base, can\_tx\_buffer\_frame\_t \*ptxframe, size\_t nb\_frames)  
Блокирующая выдача кадра через низкоприоритетный буфер выдачи
- `can_status_t CAN_TransferReceiveFifoBlocking` (CAN\_Type \*base, can\_rx\_buffer\_frame\_t \*prxframe, size\_t nb\_frames)  
Блокирующий прием кадра
- `can_status_t CAN_TransferCreateHandle` (CAN\_Type \*base, can\_handle\_t \*handle, can\_transfer\_callback\_t callback, void \*user\_data)  
Инициализация обработчика событий CAN.
- `can_status_t CAN_TransferSendPrimaryNonBlocking` (CAN\_Type \*base, can\_handle\_t \*handle, can\_tx\_transfer\_t \*xfer)  
Неблокирующая выдача через высокоприоритетный буфер
- `can_status_t CAN_TransferGetSentPrimaryCount` (CAN\_Type \*base, can\_handle\_t \*handle, uint32\_t \*nb\_frames)  
Возвращает количество кадров, отправленных в шину данных через высокоприоритетный буфер
- `void CAN_TransferAbortSendPrimary` (CAN\_Type \*base, can\_handle\_t \*handle)  
Отмена выдачи из высокоприоритетного буфера
- `can_status_t CAN_TransferSendSecondaryNonBlocking` (CAN\_Type \*base, can\_handle\_t \*handle, can\_tx\_transfer\_t \*xfer)  
Неблокирующая выдача через низкоприоритетный буфер
- `can_status_t CAN_TransferGetSentSecondaryCount` (CAN\_Type \*base, can\_handle\_t \*handle, uint32\_t \*nb\_frames)  
Возвращает количество кадров, отправленных в шину данных через низкоприоритетный буфер
- `void CAN_TransferAbortSendSecondary` (CAN\_Type \*base, can\_handle\_t \*handle)  
Отмена выдачи из низкоприоритетного буфера
- `can_status_t CAN_TransferReceiveFifoNonBlocking` (CAN\_Type \*base, can\_handle\_t \*handle, can\_rx\_transfer\_t \*xfer)  
Неблокирующий прием
- `can_status_t CAN_TransferGetReceivedCount` (CAN\_Type \*base, can\_handle\_t \*handle, uint32\_t \*nb\_frames)  
Возвращает количество принятых кадров по прерыванию
- `void CAN_TransferAbortReceive` (CAN\_Type \*base, can\_handle\_t \*handle)  
Отмена приема
- `void CAN_TransferHandleIRQ` (CAN\_Type \*base, can\_handle\_t \*handle)  
Установка обработчика на прерывания от CAN, не связанные с приемом/выдачей

## 6.1.1 Подробное описание

Интерфейс драйвера модуля ввода-вывода по интерфейсу CAN.

## 6.2 hal\_can.h

См. документацию.

```
00001
00020 #ifndef HAL_CAN_H
00021 #define HAL_CAN_H
00022
00023 #ifdef __cplusplus
00024 extern "C" {
00025 #endif
00026
00027 #include "hal_common.h"
```

```

00028
00032 #define HAL_CAN_DRIVER_VERSION (MAKE_VERSION(1, 1, 0))
00033
00037 typedef enum _can_status {
00038     CAN_Status_Ok           = 0U,
00039     CAN_Status_Fail         = 1U,
00040     CAN_Status_InvalidArgument = 2U,
00041     CAN_Status_TxBusy       = 3U,
00042     CAN_Status_RxEmpty      = 4U,
00043     CAN_Status_TxIdle       = 5U,
00044     CAN_Status_RxIdle       = 6U,
00045     CAN_Status_RxBusy       = 7U,
00046 } can_status_t;
00047
00055 typedef enum _can_kind_of_error {
00056     CAN_ErrorNone           = 0U,
00057     CAN_ErrorBitError       = 1U,
00058     CAN_ErrorFormError      = 2U,
00059     CAN_ErrorStuffError     = 3U,
00060     CAN_ErrorAckError       = 4U,
00061     CAN_ErrorCrcError       = 5U,
00062     CAN_ErrorOtherError     = 6U,
00063     CAN_ErrorReserved       = 7U,
00064 } can_kind_of_error_t;
00065
00075 typedef enum _can_flag {
00076     CAN_FlagAbortState      = 0U,
00077     CAN_FlagError           = 1U,
00078     CAN_FlagTransmissionSecondary = 2U,
00079     CAN_FlagTransmissionPrimary  = 3U,
00080     CAN_FlagTransmitBufferFull   = 4U,
00081     CAN_FlagRBAlmostFull        = 5U,
00082     CAN_FlagRBFfull            = 6U,
00083     CAN_FlagRBOverrun          = 7U,
00084     CAN_FlagReceive            = 8U,
00085     CAN_FlagBusError           = 9U,
00086     CAN_FlagArbitrationLost    = 10U,
00087     CAN_FlagErrorPassiveInterrupt = 11U,
00088     CAN_FlagErrorPassiveState   = 12U,
00089     CAN_FlagErrorWarningState   = 13U,
00090     CAN_FlagTimeTriggered       = 14U,
00091     CAN_FlagTriggerError        = 15U,
00092     CAN_FlagWatchTriggerError   = 16U,
00093     CAN_FlagsNumber
00094 } can_flag_t;
00095
00100 typedef enum _canfd_mode {
00101     CAN_BoschFd = 0U,
00102     CAN_IsoFd   = 1U,
00103 } canfd_mode_t;
00104
00108 typedef enum _can_stb_discipline {
00109     CAN_Fifo = 0U,
00110     CAN_Priority = 1U,
00111 } can_stb_discipline_t;
00112
00116 typedef enum _can_bytes_in_datafield {
00117     CAN_0ByteDatafield = 0U,
00118     CAN_1ByteDatafield = 1U,
00119     CAN_2ByteDatafield = 2U,
00120     CAN_3ByteDatafield = 3U,
00121     CAN_4ByteDatafield = 4U,
00122     CAN_5ByteDatafield = 5U,
00123     CAN_6ByteDatafield = 6U,
00124     CAN_7ByteDatafield = 7U,
00125     CAN_8ByteDatafield = 8U,
00126     CAN_12ByteDatafield = 9U,
00127     CAN_16ByteDatafield = 10U,
00128     CAN_20ByteDatafield = 11U,
00129     CAN_24ByteDatafield = 12U,
00130     CAN_32ByteDatafield = 13U,
00131     CAN_48ByteDatafield = 14U,
00132     CAN_64ByteDatafield = 15U,
00133 } can_bytes_in_datafield_t;
00134
00135 #if defined(__CC_ARM)
00136 #pragma anon_unions
00137 #endif
00138
00142 typedef struct _can_tx_buffer_frame {
00143     struct {
00144         uint32_t id : 29;
00145         uint32_t : 2;
00146         bool ttsen : 1;
00147     };
00148     struct {
00149         can_bytes_in_datafield_t dlc : 4;

```



```

00150     bool                brs   : 1;
00151     bool                fdf   : 1;
00152     bool                rtr   : 1;
00153     bool                ide   : 1;
00154     uint32_t            : 24;
00155 };
00156 uint8_t                data[64];
00157 } can_tx_buffer_frame_t;
00158
00162 typedef struct _can_rx_buffer_frame {
00163     struct {
00164         uint32_t            id       : 29;
00165         uint32_t            : 2;
00166         bool                esi      : 1;
00167     };
00168     struct {
00169         can_bytes_in_datafield_t dlc : 4;
00170         bool                brs      : 1;
00171         bool                fdf      : 1;
00172         bool                rtr      : 1;
00173         bool                ide      : 1;
00174         uint32_t            : 4;
00175         bool                tx       : 1;
00176         can_kind_of_error_t koer     : 3;
00177         uint32_t            cycle_time : 16;
00178     };
00179     uint8_t                data[64];
00180     uint64_t               rts;
00181 } can_rx_buffer_frame_t;
00182
00186 typedef struct _can_frame_filter {
00187     uint32_t id           : 29;
00188     uint32_t            : 3;
00189     uint32_t mask        : 29;
00190     uint32_t accepted_ide : 1;
00191     uint32_t enable_ide_check : 1;
00192     uint32_t            : 1;
00193 } can_frame_filter_t;
00194
00198 #define CAN_NB_OF_FILTERS 4
00199
00203 typedef struct _can_frame_filter_config {
00204     can_frame_filter_t filter[CAN_NB_OF_FILTERS];
00205     uint8_t            nb_filters_used;
00206 } can_frame_filter_config_t;
00207
00211 typedef enum _ttcan_timer_prescaler {
00212     CAN_TTCANDiv1 = 0U,
00213     CAN_TTCANDiv2 = 1U,
00214     CAN_TTCANDiv4 = 2U,
00215     CAN_TTCANDiv8 = 3U,
00216 } ttcan_timer_prescaler_t;
00217
00221 typedef enum _ttcan_trigger_type {
00222     CAN_TriggerImmediate = 0U,
00223     CAN_TriggerTime      = 1U,
00224     CAN_TriggerSingleShotTransmit = 2U,
00225     CAN_TriggerTransmitStart = 3U,
00226     CAN_TriggerTransmitStop  = 4U,
00227 } ttcan_trigger_type_t;
00228
00232 typedef struct _ttcan_config {
00233     ttcan_timer_prescaler_t prescaler;
00234     uint8_t                transmit_trigger_pointer;
00235     ttcan_trigger_type_t   trigger_type;
00236     uint8_t                transmit_enable_window;
00237     uint8_t                trigger_time0;
00238     uint8_t                trigger_time1;
00239     uint8_t                watch_trigger_time0;
00240     uint8_t                watch_trigger_time1;
00241     uint32_t               reference_id;
00242     bool                   reference_ide;
00243 } ttcan_config_t;
00244
00248 typedef struct _can_timing_config {
00249     uint8_t prescaler;
00250     uint8_t sjw;
00251     uint8_t seg1;
00252     uint8_t seg2;
00253     uint8_t data_prescaler;
00254     uint8_t data_sjw;
00255     uint8_t data_seg1;
00256     uint8_t data_seg2;
00257     uint8_t delay_compensation_enable;
00258     uint8_t secondary_sample_point_offset;
00259 } can_timing_config_t;
00260

```

```

00264 typedef struct _can_ptb_config {
00265     bool tx_single_shot;
00266 } can_ptb_config_t;
00267
00271 typedef struct _can_stb_config {
00272     bool tx_single_shot;
00273     can_stb_discipline_t tx_discipline;
00274 } can_stb_config_t;
00275
00279 typedef struct _can_rxb_config {
00280     uint32_t almost_full_level;
00281     bool self_acknowledge;
00282     bool prohibit_overflow;
00283 } can_rxb_config_t;
00284
00288 typedef struct _can_config {
00289     bool enable_listen_only;
00290     bool enable_loopback_int;
00291     bool enable_loopback_ext;
00292     can_ptb_config_t ptb_config;
00293     can_stb_config_t stb_config;
00294     can_rxb_config_t rxb_config;
00295     can_timing_config_t timing_config;
00296     canfd_mode_t can_fd_mode;
00297     can_frame_filter_config_t filter_config;
00298 } can_config_t;
00299
00304 typedef struct _can_tx_transfer {
00305     can_tx_buffer_frame_t *frames;
00306     size_t nb_frames;
00307 } can_tx_transfer_t;
00308
00312 typedef struct _can_rx_transfer {
00313     can_rx_buffer_frame_t *frames;
00314     size_t nb_frames;
00315 } can_rx_transfer_t;
00316
00320 typedef struct _can_handle can_handle_t;
00321
00325 typedef void (*can_transfer_callback_t)(CAN_Type *base, can_handle_t *handle,
00326     can_status_t status, can_flag_t interrupt_flag, void *user_data);
00327
00331 struct _can_handle {
00332     volatile const can_tx_buffer_frame_t *tx_frames_prim;
00333     volatile size_t tx_nb_frames_rest_prim;
00334     size_t tx_nb_frames_all_prim;
00336     volatile const can_tx_buffer_frame_t *tx_frames_sec;
00337     volatile size_t tx_nb_frames_rest_sec;
00338     size_t tx_nb_frames_all_sec;
00340     volatile can_rx_buffer_frame_t *rx_frames;
00341     volatile size_t rx_nb_frames_rest;
00342     size_t rx_nb_frames_all;
00344     can_transfer_callback_t callback;
00345     void *user_data;
00346 };
00347
00362 can_status_t CAN_Init(CAN_Type *base, const can_config_t *config);
00363
00371 void CAN_Deinit(CAN_Type *base);
00372
00379 void CAN_GetDefaultConfig(can_config_t *config, uint32_t source_clock_hz);
00380
00389 void CAN_EnterNormalMode(CAN_Type *base);
00390
00396 void CAN_EnterStandbyMode(CAN_Type *base);
00397
00420 bool CAN_CalculateImprovedTimingValues(uint32_t baudrate,
00421     uint32_t baudrate_data, uint32_t source_clock_hz,
00422     can_timing_config_t *pconfig);
00423
00430 void CAN_SetArbitrationTimingConfig(CAN_Type *base,
00431     const can_timing_config_t *config);
00432
00440 void CAN_SetPrimaryTxBufferConfig(CAN_Type *base,
00441     const can_ptb_config_t *config);
00442
00449 void CAN_SetSecondaryTxBufferConfig(CAN_Type *base,
00450     const can_stb_config_t *config);
00451
00458 void CAN_SetRxBufferConfig(CAN_Type *base, const can_rxb_config_t *config);
00459
00466 void CAN_SetFilterConfig(CAN_Type *base,
00467     const can_frame_filter_config_t *config);
00468
00475 void CAN_SetTTCANConfig(CAN_Type *base, const ttcan_config_t *config);
00476
00498 bool CAN_GetStatusFlag(CAN_Type *base, can_flag_t idx, can_status_t *status);

```

```

00499
00507 uint32_t CAN_GetStatusFlagMask(CAN_Type *base);
00508
00518 can_status_t CAN_ClearStatusFlag(CAN_Type *base, can_flag_t idx);
00519
00529 void CAN_ClearStatusFlagMask(CAN_Type *base, uint32_t mask);
00530
00549 can_status_t CAN_EnableInterrupt(CAN_Type *base, can_flag_t idx);
00550
00557 void CAN_EnableInterruptMask(CAN_Type *base, uint32_t mask);
00558
00570 bool CAN_IsInterruptEnabled(CAN_Type *base, can_flag_t idx,
00571     can_status_t *status);
00572
00584 uint32_t CAN_GetEnabledInterruptMask(CAN_Type *base);
00585
00595 can_status_t CAN_DisableInterrupt(CAN_Type *base, can_flag_t idx);
00596
00603 void CAN_DisableInterruptMask(CAN_Type *base, uint32_t mask);
00604
00625 bool CAN_IsPrimaryTransmitRequestPending(CAN_Type *base);
00626
00639 bool CAN_IsSecondaryTransmitRequestPending(CAN_Type *base);
00640
00651 bool CAN_IsSecondaryTxBufferEmpty(CAN_Type *base);
00652
00664 bool CAN_IsSecondaryTxBufferMoreThanHalfFull(CAN_Type *base);
00665
00676 bool CAN_IsSecondaryTxBufferFull(CAN_Type *base);
00677
00693 can_status_t CAN_WritePrimaryTxBuffer(CAN_Type *base,
00694     const can_tx_buffer_frame_t *ptxframe);
00695
00709 can_status_t CAN_WriteSecondaryTxBuffer(CAN_Type *base,
00710     const can_tx_buffer_frame_t *ptxframe);
00711
00717 void CAN_AbortPrimaryTxBuffer(CAN_Type *base);
00718
00731 void CAN_AbortSecondaryTxBuffer(CAN_Type *base);
00732
00743 bool CAN_IsRxBufferEmpty(CAN_Type *base);
00744
00756 bool CAN_IsRxBufferAlmostFull(CAN_Type *base);
00757
00768 bool CAN_IsRxBufferFull(CAN_Type *base);
00769
00782 can_status_t CAN_ReadRxBuffer(CAN_Type *base, can_rx_buffer_frame_t *prxframe);
00783
00806 can_status_t CAN_TransferSendPrimaryBlocking(CAN_Type *base,
00807     can_tx_buffer_frame_t *ptxframe, size_t nb_frames);
00808
00822 can_status_t CAN_TransferSendSecondaryBlocking(CAN_Type *base,
00823     can_tx_buffer_frame_t *ptxframe, size_t nb_frames);
00824
00838 can_status_t CAN_TransferReceiveFifoBlocking(CAN_Type *base,
00839     can_rx_buffer_frame_t *prxframe, size_t nb_frames);
00840
00852 can_status_t CAN_TransferCreateHandle(CAN_Type *base, can_handle_t *handle,
00853     can_transfer_callback_t callback, void *user_data);
00854
00870 can_status_t CAN_TransferSendPrimaryNonBlocking(CAN_Type *base,
00871     can_handle_t *handle, can_tx_transfer_t *xfer);
00872
00887 can_status_t CAN_TransferGetSentPrimaryCount(CAN_Type *base,
00888     can_handle_t *handle, uint32_t *nb_frames);
00889
00896 void CAN_TransferAbortSendPrimary(CAN_Type *base, can_handle_t *handle);
00897
00913 can_status_t CAN_TransferSendSecondaryNonBlocking(CAN_Type *base,
00914     can_handle_t *handle, can_tx_transfer_t *xfer);
00915
00930 can_status_t CAN_TransferGetSentSecondaryCount(CAN_Type *base,
00931     can_handle_t *handle, uint32_t *nb_frames);
00932
00939 void CAN_TransferAbortSendSecondary(CAN_Type *base, can_handle_t *handle);
00940
00956 can_status_t CAN_TransferReceiveFifoNonBlocking(CAN_Type *base,
00957     can_handle_t *handle, can_rx_transfer_t *xfer);
00958
00969 can_status_t CAN_TransferGetReceivedCount(CAN_Type *base,
00970     can_handle_t *handle, uint32_t *nb_frames);
00971
00978 void CAN_TransferAbortReceive(CAN_Type *base, can_handle_t *handle);
00979
00987 void CAN_TransferHandleIRQ(CAN_Type *base, can_handle_t *handle);
00988
00993 #ifdef __cplusplus

```

```

00994 }
00995 #endif
00996
00997 #endif /* HAL_CAN_H */
00998

```

### 6.3 Файл devices/eliot1/drivers/hal\_clkctr.h

Интерфейс драйвера модуля CLKCTR.

```
#include "hal_rwc.h"
```

Структуры данных

- struct `clkctr_div`  
Делители блока
- struct `clkctr_pll_cfg`  
Коэффициенты PLL.

Макросы

Экстремальные значения коэффициентов делителей частот параметров PLL (делитель частоты = коэффициент деления + 1)

- #define `CLKCTR_MAX_SYSCLK_DIV` 31
- #define `CLKCTR_MAX_FCLK_DIV` 31
- #define `CLKCTR_MAX_GNSSCLK_DIV` 7
- #define `CLKCTR_MAX_QSPICLK_DIV` 31
- #define `CLKCTR_MAX_MCOCLK_DIV` 31
- #define `CLKCTR_MAX_I2SCLK_DIV` 31
- #define `CLKCTR_MIN_SYSCLK_DIV` 0
- #define `CLKCTR_MIN_FCLK_DIV` 0
- #define `CLKCTR_MIN_GNSSCLK_DIV` 0
- #define `CLKCTR_MIN_QSPICLK_DIV` 0
- #define `CLKCTR_MIN_MCOCLK_DIV` 0
- #define `CLKCTR_MIN_I2SCLK_DIV` 0
- #define `PLL_MAX_MULTIPLIER` 0x176
- #define `PLL_MIN_MULTIPLIER` 0x0
- #define `CLKCTR_NR_MAN_MAX`
- #define `CLKCTR_NF_MAN_MAX`
- #define `CLKCTR_OD_MAN_MAX`
- #define `CLKCTR_MAN_MAX`
- #define `CLKCTR_SEL_MAX`

Минимально и максимально допустимые внешние частоты блока CLKCTR

- #define `CLKCTR_XTI_MIN` 1
- #define `CLKCTR_XTI_MAX` 50000000
- #define `CLKCTR_XTI32_MIN` 30000
- #define `CLKCTR_XTI32_MAX` 34000
- #define `CLKCTR_HFI_MIN` 4000000
- #define `CLKCTR_HFI_MAX` 25000000
- #define `CLKCTR_LFI_MIN` 32112
- #define `CLKCTR_LFI_MAX` 33423
- #define `CLKCTR_I2S_EXTCLK_MIN` 1
- #define `CLKCTR_I2S_EXTCLK_MAX` 50000000

Минимально и максимально допустимые внутренние частоты блока CLKCTR

- #define CLKCTR\_PLLREF\_MIN 30000
- #define CLKCTR\_PLLREF\_MAX 50000000
- #define CLKCTR\_PLLCLK\_MIN 1880000
- #define CLKCTR\_PLLCLK\_MAX 375000000
- #define CLKCTR\_PLLCLK\_OD\_MIN 30000000
- #define CLKCTR\_PLLCLK\_OD\_MAX 375000000
- #define CLKCTR\_FCLK\_MIN 1
- #define CLKCTR\_FCLK\_MAX 150000000
- #define CLKCTR\_SYSCLK\_MIN 1
- #define CLKCTR\_SYSCLK\_MAX 50000000
- #define CLKCTR\_QSPICLK\_MIN 1
- #define CLKCTR\_QSPICLK\_MAX 96000000
- #define CLKCTR\_GNSSCLK\_MIN 1
- #define CLKCTR\_GNSSCLK\_MAX 80000000
- #define CLKCTR\_I2SCLK\_MIN 1
- #define CLKCTR\_I2SCLK\_MAX 25000000

Служебные определения

- #define HFI\_FREQUENCY 15100000
- #define CLKCTR\_FREQ\_NOT\_SET 0

Перечисления

- enum clkctr\_mccclk  
Подаваемая на MCO частота
- enum clkctr\_lpcclk  
Подаваемая на LPCLK частота
- enum clkctr\_rtccclk  
Подаваемая на RTCCLK частота
- enum clkctr\_mainclk  
Подаваемая на MAINCLK частота
- enum clkctr\_pllref  
Опорная частота PLL.
- enum clkctr\_usbcclk  
Подаваемая на USBCLK частота
- enum clkctr\_i2sclk  
Подаваемая на I2SCLK частота
- enum clkctr\_hfi\_for\_main\_type  
Частота, определенная как HFICLK для MAINCLK (CLKCTR\_MainClkTypeHFICLK)
- enum clkctr\_extern\_freq  
Генератор/осциллятор/вход, для которого групповой функцией устанавливается частота
- enum clkctr\_int\_freq  
Имя внутренней частоты тактирования микросхемы, для которой производится действие групповой функцией
- enum clkctr\_clk\_force\_type  
Тип тактирования
- enum clkctr\_device\_clk\_force  
Домены, к которым может быть применено динамическое тактирование
- enum clkctr\_status  
Статусы драйвера CLKCTR.

## Функции

Устаревшие функции для поддержки ранних примеров

- void [CLKCTR\\_SetPll](#) (CLKCTR\_Type \*base, uint32\_t xti\_hz, uint32\_t pll\_mul)  
Установка целочисленного множителя PLL.
- enum [clkctr\\_status](#) [CLKCTR\\_SetPllMan](#) (CLKCTR\_Type \*base, uint32\_t xti\_hz, uint32\_t nr, uint32\_t nf, uint32\_t od)  
Установка множителей PLL в ручном режиме
- void [CLKCTR\\_SetSysDiv](#) (CLKCTR\_Type \*base, uint16\_t fclk\_div, uint16\_t sysclk\_div, uint16\_t gnssclk\_div, uint16\_t qspiclk\_div)  
Установка делителей

Функции установки и получения частот генераторов/осцилляторов

- enum [clkctr\\_status](#) [CLKCTR\\_SetXTI](#) (CLKCTR\_Type \*base, uint32\_t frequency)  
Установка значения частоты, подаваемой на вход XTI.
- uint32\_t [CLKCTR\\_GetXTI](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты, подаваемой на вход XTI.
- enum [clkctr\\_status](#) [CLKCTR\\_SetXTI32](#) (CLKCTR\_Type \*base, uint32\_t frequency)  
Установка значения частоты, подаваемой на вход XTI32.
- uint32\_t [CLKCTR\\_GetXTI32](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты, подаваемой на вход XTI32.
- enum [clkctr\\_status](#) [CLKCTR\\_SetHFI](#) (CLKCTR\_Type \*base, uint32\_t frequency)  
Установка значения частоты внутреннего генератора APC.
- uint32\_t [CLKCTR\\_GetHFI](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты внутреннего генератора APC.
- enum [clkctr\\_status](#) [CLKCTR\\_SetLFI](#) (CLKCTR\_Type \*base, uint32\_t frequency)  
Установка значения частоты внутреннего генератора RWC.
- uint32\_t [CLKCTR\\_GetLFI](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты внутреннего генератора RWC.
- enum [clkctr\\_status](#) [CLKCTR\\_SetI2SExtClk](#) (CLKCTR\_Type \*base, uint32\_t frequency)  
Установка значения частоты, поступающей на PA15.
- uint32\_t [CLKCTR\\_GetI2SExtClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты, поступающей на PA15.

Функции установки и получения делителей, кроме PLL

- enum [clkctr\\_status](#) [CLKCTR\\_GetAllDiv](#) (CLKCTR\_Type \*base, struct [clkctr\\_div](#) \*divisors)  
Извлечение всех делителей блока CLKCTR.
- uint32\_t [CLKCTR\\_GetMCOdiv](#) (CLKCTR\_Type \*base)  
Извлечение значения делителя MCOdiv.
- enum [clkctr\\_status](#) [CLKCTR\\_SetMCOdiv](#) (CLKCTR\_Type \*base, uint32\_t value)  
Установка значения делителя MCOdiv.
- uint32\_t [CLKCTR\\_GetFCLKdiv](#) (CLKCTR\_Type \*base)  
Извлечение значения делителя FCLKdiv.
- enum [clkctr\\_status](#) [CLKCTR\\_SetFCLKdiv](#) (CLKCTR\_Type \*base, uint32\_t value)  
Установка значения делителя FCLKdiv.
- uint32\_t [CLKCTR\\_GetSysClkDiv](#) (CLKCTR\_Type \*base)  
Извлечение значения делителя SYSCLKdiv.
- enum [clkctr\\_status](#) [CLKCTR\\_SetSysClkDiv](#) (CLKCTR\_Type \*base, uint32\_t value)  
Установка значения делителя SYSCLKdiv.
- uint32\_t [CLKCTR\\_GetGNSSClkDiv](#) (CLKCTR\_Type \*base)  
Извлечение значения делителя GNSSCLKdiv.
- enum [clkctr\\_status](#) [CLKCTR\\_SetGNSSClkDiv](#) (CLKCTR\_Type \*base, uint32\_t value)  
Установка значения делителя GNSSCLKdiv.
- uint32\_t [CLKCTR\\_GetQSPIClkDiv](#) (CLKCTR\_Type \*base)  
Извлечение значения делителя QSPICLKdiv.

- enum [clkctr\\_status](#) [CLKCTR\\_SetQSPIClkDiv](#) (CLKCTR\_Type \*base, uint32\_t value)  
Установка значения делителя QSPICLKDiv.
- uint32\_t [CLKCTR\\_GetI2SCLKDiv](#) (CLKCTR\_Type \*base)  
Извлечение значения делителя I2SCLKDiv.
- enum [clkctr\\_status](#) [CLKCTR\\_SetI2SCLKDiv](#) (CLKCTR\_Type \*base, uint32\_t value)  
Установка значения делителя I2SCLKDiv.

#### Функции установки и получения коэффициентов PLL

- enum [clkctr\\_status](#) [CLKCTR\\_GetPLLConfig](#) (CLKCTR\_Type \*base, struct [clkctr\\_pll\\_cfg](#) \*config)  
Извлечение коэффициентов PLL.
- enum [clkctr\\_status](#) [CLKCTR\\_SetPLLConfig](#) (CLKCTR\_Type \*base, struct [clkctr\\_pll\\_cfg](#) config)  
Установка коэффициентов PLL.

#### Функции извлечения частот тактирования

- uint32\_t [CLKCTR\\_GetXTIClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты XTICLK.
- uint32\_t [CLKCTR\\_GetHFIClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты HFICLK.
- uint32\_t [CLKCTR\\_GetRTCClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты RTCCLK.
- uint32\_t [CLKCTR\\_GetLPClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты LPCLK.
- uint32\_t [CLKCTR\\_GetPLLClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты PLLCLK.
- uint32\_t [CLKCTR\\_GetPLLRef](#) (CLKCTR\_Type \*base)  
Извлечение значения опорной частоты PLL.
- uint32\_t [CLKCTR\\_GetMainClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты MAINCLK.
- uint32\_t [CLKCTR\\_GetUSBClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты USBCLK, подаваемого на USB PHY.
- uint32\_t [CLKCTR\\_GetFCLKInt](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты FCLK\_INT.
- uint32\_t [CLKCTR\\_GetFCLK](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты FCLK.
- uint32\_t [CLKCTR\\_GetSysClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты SYSCLK.
- uint32\_t [CLKCTR\\_GetGNSSClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты GNSSCLK.
- uint32\_t [CLKCTR\\_GetQSPIClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты QSPICLK.
- uint32\_t [CLKCTR\\_GetI2SCLK](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты I2SCLK.
- uint32\_t [CLKCTR\\_GetMCOClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты MCOCLK, подаваемой на вывод PA15.
- uint32\_t [CLKCTR\\_GetHFIClkForMainClk](#) (CLKCTR\_Type \*base)  
Извлечение значения частоты HFICLK для MAINCLK.

#### Функции выбора и извлечения источников частот

- enum [clkctr\\_status](#) [CLKCTR\\_SetSwitchMCOClk](#) (CLKCTR\_Type \*base, uint32\_t value)  
Выбор источника частоты MCO.
- enum [clkctr\\_status](#) [CLKCTR\\_SetSwitchPLLRef](#) (CLKCTR\_Type \*base, uint32\_t value)  
Выбор источника опорной частоты PLL.



- enum [clkctr\\_status CLKCTR\\_SetSwitchMainClk](#) (CLKCTR\_Type \*base, uint32\_t value)  
Выбор источника частоты MAINCLK.
- enum [clkctr\\_status CLKCTR\\_SetSwitchRTCClk](#) (CLKCTR\_Type \*base, uint32\_t value)  
Выбор источника частоты RTCCLK.
- enum [clkctr\\_status CLKCTR\\_SetSwitchLPCLk](#) (CLKCTR\_Type \*base, uint32\_t value)  
Выбор источника частоты LPCLK.
- enum [clkctr\\_status CLKCTR\\_SetSwitchUSBClk](#) (CLKCTR\_Type \*base, uint32\_t value)  
Выбор источника частоты USBCLK.
- enum [clkctr\\_status CLKCTR\\_SetSwitchI2SClk](#) (CLKCTR\_Type \*base, uint32\_t value)  
Выбор источника частоты I2SCLK.
- enum [clkctr\\_status CLKCTR\\_SetSwitchPMUDIS](#) (CLKCTR\_Type \*base, uint32\_t value)  
Выбор режима работы процессора
- uint32\_t [CLKCTR\\_GetSwitchMCOClk](#) (CLKCTR\_Type \*base)  
Извлечение источника частоты MCO.
- uint32\_t [CLKCTR\\_GetSwitchPLLRef](#) (CLKCTR\_Type \*base)  
Извлечение источника частоты PLL.
- uint32\_t [CLKCTR\\_GetSwitchMainClk](#) (CLKCTR\_Type \*base)  
Извлечение источника частоты MAINCLK.
- uint32\_t [CLKCTR\\_GetSwitchRTCClk](#) (CLKCTR\_Type \*base)  
Извлечение источника частоты RTCCLK.
- uint32\_t [CLKCTR\\_GetSwitchLPCLk](#) (CLKCTR\_Type \*base)  
Извлечение источника частоты LPCLK.
- uint32\_t [CLKCTR\\_GetSwitchUSBClk](#) (CLKCTR\_Type \*base)  
Извлечение источника частоты USBCLK.
- uint32\_t [CLKCTR\\_GetSwitchI2SClk](#) (CLKCTR\_Type \*base)  
Извлечение источника частоты I2SCLK.
- uint32\_t [CLKCTR\\_GetSwitchPMUDIS](#) (CLKCTR\_Type \*base)  
Определение режима работы процессора

#### Прочие функции

- enum [clkctr\\_status CLKCTR\\_SetHFITrim](#) (CLKCTR\_Type \*base, uint32\_t index)  
Установка индекса коэффициента подстройки частоты осциллятора HFI.
- uint32\_t [CLKCTR\\_GetHFITrim](#) (CLKCTR\_Type \*base)  
Извлечение индекса коэффициента подстройки частоты осциллятора HFI.
- uint32\_t [CLKCTR\\_GetMCOEn](#) (CLKCTR\_Type \*base)  
Извлечение разрешения тактового сигнала MCO.
- enum [clkctr\\_status CLKCTR\\_SetMCOEn](#) (CLKCTR\_Type \*base, uint32\_t enable)  
Установка или запрет разрешения тактового сигнала MCO.
- enum [clkctr\\_status CLKCTR\\_GetClkForce](#) (CLKCTR\_Type \*base, enum [clkctr\\_device\\_clk\\_force](#) device, enum [clkctr\\_clk\\_force\\_type](#) \*clk\_type)  
Извлечение типа тактирования для модуля
- enum [clkctr\\_status CLKCTR\\_SetClkForce](#) (CLKCTR\_Type \*base, enum [clkctr\\_device\\_clk\\_force](#) device, enum [clkctr\\_clk\\_force\\_type](#) clk\_type)  
Установка типа тактирования для модуля

#### Функции-обертки

- enum [clkctr\\_status CLKCTR\\_SetFrequency](#) (CLKCTR\_Type \*base, enum [clkctr\\_extern\\_freq](#) input, uint32\_t frequency)  
Установка значения частоты, подаваемой на выбранный вход
- uint32\_t [CLKCTR\\_GetFrequency](#) (CLKCTR\_Type \*base, enum [clkctr\\_extern\\_freq](#) input)  
Извлечение значения частоты, подаваемой на выбранный вход
- enum [clkctr\\_status CLKCTR\\_GetClk](#) (CLKCTR\_Type \*base, enum [clkctr\\_int\\_freq](#) clk, uint32\_t \*value)  
Извлечение значения указанной частоты
- enum [clkctr\\_status CLKCTR\\_SetSwitchClk](#) (CLKCTR\_Type \*base, enum [clkctr\\_int\\_freq](#) clk, uint32\_t value)



- Выбор источника частоты для указанной частоты
  - enum `clkctr_status` `CLKCTR_GetSwitchClk` (`CLKCTR_Type *base`, enum `clkctr_int_freq` `clk`, `uint32_t *value`)
- Извлечение источника частоты для указанной частоты
  - enum `clkctr_status` `CLKCTR_SetDivClk` (`CLKCTR_Type *base`, enum `clkctr_int_freq` `clk`, `uint32_t value`)
- Установка коэффициента деления для указанной частоты
  - enum `clkctr_status` `CLKCTR_GetDivClk` (`CLKCTR_Type *base`, enum `clkctr_int_freq` `clk`, `uint32_t *value`)
- Извлечение коэффициента деления для указанной частоты

### 6.3.1 Подробное описание

Интерфейс драйвера модуля CLKCTR.

## 6.4 hal\_clkctr.h

[См. документацию.](#)

```

00001
00020 #ifndef HAL_CLKCTR_H
00021 #define HAL_CLKCTR_H
00022
00023 #if defined(__cplusplus)
00024 extern "C" {
00025 #endif /* __cplusplus */
00026
00027 #include "hal_rwc.h"
00028
00034 #define CLKCTR_MAX_SYSCLK_DIV 31
00035 #define CLKCTR_MAX_FCLK_DIV 31
00036 #define CLKCTR_MAX_GNSSCLK_DIV 7
00037 #define CLKCTR_MAX_QSPICLK_DIV 31
00038 #define CLKCTR_MAX_MCOCLK_DIV 31
00039 #define CLKCTR_MAX_I2SCLK_DIV 31
00040 #define CLKCTR_MIN_SYSCLK_DIV 0
00041 #define CLKCTR_MIN_FCLK_DIV 0
00042 #define CLKCTR_MIN_GNSSCLK_DIV 0
00043 #define CLKCTR_MIN_QSPICLK_DIV 0
00044 #define CLKCTR_MIN_MCOCLK_DIV 0
00045 #define CLKCTR_MIN_I2SCLK_DIV 0
00047 #define PLL_MAX_MULTIPLIER 0x176
00048 #define PLL_MIN_MULTIPLIER 0x0
00050 #define CLKCTR_NR_MAN_MAX (CLKCTR_PLLCFG_NR_MAN_Msk \
00051  » CLKCTR_PLLCFG_NR_MAN_Pos)
00052 #define CLKCTR_NF_MAN_MAX (CLKCTR_PLLCFG_NF_MAN_Msk \
00053  » CLKCTR_PLLCFG_NF_MAN_Pos)
00054 #define CLKCTR_OD_MAN_MAX (CLKCTR_PLLCFG_OD_MAN_Msk \
00055  » CLKCTR_PLLCFG_OD_MAN_Pos)
00056 #define CLKCTR_MAN_MAX (CLKCTR_PLLCFG_MAN_Msk \
00057  » CLKCTR_PLLCFG_MAN_Pos)
00058 #define CLKCTR_SEL_MAX (CLKCTR_PLLCFG_SEL_Msk \
00059  » CLKCTR_PLLCFG_SEL_Pos)
00067 enum clkctr_mcoclk {
00068     CLKCTR_MCOClkTypeHFIClk = 0,
00069     CLKCTR_MCOClkTypeRTCClk = 1,
00070     CLKCTR_MCOClkTypeLPClk = 2,
00071     CLKCTR_MCOClkTypeMainClk = 3,
00072     CLKCTR_MCOClkTypePLLClk = 4,
00073     CLKCTR_MCOClkTypeSysClk = 5,
00074     CLKCTR_MCOClkTypeFClkInt = 6,
00075     CLKCTR_MCOClkTypeFClk = 7,
00076 };
00077
00081 enum clkctr_lpcclk {
00082     CLKCTR_LPClkTypeRTCClk = 0,
00083     CLKCTR_LPClkType500 = 1,
00084 };
00085
00089 enum clkctr_rtccclk {
00090     CLKCTR_RTCClkTypeLFI = RWC_RTCClkTypeLFI,
00091     CLKCTR_RTCClkTypeLFE = RWC_RTCClkTypeLFE,
00092 };

```

```

00093
00097 enum clkctr_mainclk {
00098     CLKCTR_MainClkTypeHFIClk = 0,
00099     CLKCTR_MainClkTypeXTIClk = 1,
00100     CLKCTR_MainClkTypePLLClk = 2,
00101     CLKCTR_MainClkTypeMax    = CLKCTR_MainClkTypePLLClk,
00102 };
00103
00107 enum clkctr_pllref {
00108     CLKCTR_PLLRefTypeHFIClk = 0,
00109     CLKCTR_PLLRefTypeXTIClk = 1,
00110 };
00111
00115 enum clkctr_usbcclk {
00116     CLKCTR_USBClkTypeHFIClk = 0,
00117     CLKCTR_USBClkTypeXTIClk = 1,
00118 };
00119
00123 enum clkctr_i2sclk {
00124     CLKCTR_I2SClkTypeSysClk = 0,
00125     CLKCTR_I2SClkTypeI2SClk = 1,
00126 };
00127
00136 enum clkctr_hfi_for_main_type {
00137     CLKCTR_HFIForMainTypeHFI = 0,
00138     CLKCTR_HFIForMainTypeXTI = 1,
00139 };
00140
00145 enum clkctr_extern_freq {
00146     CLKCTR_ExternFreqXTI    = 0,
00147     CLKCTR_ExternFreqXTI32  = 1,
00148     CLKCTR_ExternFreqHFI    = 2,
00149     CLKCTR_ExternFreqLFI    = 3,
00150     CLKCTR_ExternFreqI2SExtClk = 4,
00151 };
00152
00157 enum clkctr_int_freq {
00158     CLKCTR_IntFreqXTIClk    = 0,
00159     CLKCTR_IntFreqHFIClk    = 1,
00160     CLKCTR_IntFreqRTCClk    = 2,
00161     CLKCTR_IntFreqLPClk     = 3,
00162     CLKCTR_IntFreqPLLClk    = 4,
00163     CLKCTR_IntFreqPLLRefClk = 5,
00164     CLKCTR_IntFreqMainClk   = 6,
00165     CLKCTR_IntFreqUSBClk    = 7,
00166     CLKCTR_IntFreqFClkInt   = 8,
00167     CLKCTR_IntFreqFClk     = 9,
00168     CLKCTR_IntFreqSysClk    = 10,
00169     CLKCTR_IntFreqGNSSClk   = 11,
00170     CLKCTR_IntFreqQSPIClk   = 12,
00171     CLKCTR_IntFreqI2SClk    = 13,
00172     CLKCTR_IntFreqMCOClk    = 14,
00173     CLKCTR_IntFreqHFIForMain = 15,
00174 };
00175
00179 enum clkctr_clk_force_type {
00180     CLKCTR_ClkForceTypeDynamic = 0,
00181     CLKCTR_ClkForceTypeForce   = 1,
00182 };
00183
00187 enum clkctr_device_clk_force {
00188     CLKCTR_SysSysClkForce   = 1,
00189     CLKCTR_SysFClkForce     = 2,
00190     CLKCTR_SRAMSysClkForce  = 3,
00191     CLKCTR_SRAMFClkForce    = 4,
00192     CLKCTR_CPUSysClkForce   = 5,
00193     CLKCTR_CPUFClkForce     = 6,
00194     CLKCTR_CryptoSysClkForce = 7,
00195     CLKCTR_CPUDBGPIkClkForce = 8,
00196     CLKCTR_BasePikClkForce  = 9,
00197     CLKCTR_SMCClkForce      = 10,
00198     CLKCTR_GMSSSysClkForce  = 11,
00199 };
00200
00204 struct clkctr_div {
00205     uint32_t clkctr_fclk_div;
00206     uint32_t clkctr_sysclk_div;
00207     uint32_t clkctr_gnssclk_div;
00208     uint32_t clkctr_qspiclk_div;
00209     uint32_t clkctr_i2sclk_div;
00210     uint32_t clkctr_mco_div;
00211 };
00212
00216 struct clkctr_pll_cfg {
00217     uint32_t lock;
00218     uint32_t nr_man;
00219     uint32_t nf_man;

```

```

00220     uint32_t od_man;
00221     uint32_t man;
00222     uint32_t sel;
00223 };
00224
00228 enum clkctr_status {
00229     CLKCTR_Status_Ok = 0,
00230     CLKCTR_Status_InvalidArgument = 1,
00231     CLKCTR_Status_CheckError = 2,
00232     CLKCTR_Status_VerifyError = 3,
00233     CLKCTR_Status_ConfigureError = 4,
00234 };
00235
00240 #define CLKCTR_XTI_MIN 1
00241 #define CLKCTR_XTI_MAX 50000000
00242 #define CLKCTR_XTI32_MIN 30000
00243 #define CLKCTR_XTI32_MAX 34000
00244 #define CLKCTR_HFI_MIN 4000000
00245 #define CLKCTR_HFI_MAX 25000000
00246 #define CLKCTR_LFI_MIN 32112
00247 #define CLKCTR_LFI_MAX 33423
00248 #define CLKCTR_I2S_EXTCLK_MIN 1
00249 #define CLKCTR_I2S_EXTCLK_MAX 50000000
00259 #define CLKCTR_PLLREF_MIN 30000
00260 #define CLKCTR_PLLREF_MAX 50000000
00261 #define CLKCTR_PLLCLK_MIN 1880000
00262 #define CLKCTR_PLLCLK_MAX 375000000
00263 #define CLKCTR_PLLCLK_OD_MIN 30000000
00264 #define CLKCTR_PLLCLK_OD_MAX 375000000
00265 #define CLKCTR_FCLK_MIN 1
00266 #define CLKCTR_FCLK_MAX 150000000
00267 #define CLKCTR_SYSCLK_MIN 1
00268 #define CLKCTR_SYSCLK_MAX 50000000
00269 #define CLKCTR_QSPICLK_MIN 1
00270 #define CLKCTR_QSPICLK_MAX 96000000
00271 #define CLKCTR_GNSSCLK_MIN 1
00272 #define CLKCTR_GNSSCLK_MAX 80000000
00273 #define CLKCTR_I2SCLK_MIN 1
00274 #define CLKCTR_I2SCLK_MAX 25000000
00283 #define CLKCTR_FREQ 15100000
00284 #define CLKCTR_FREQ_NOT_SET 0
00307 void CLKCTR_SetPll(CLKCTR_Type * base, uint32_t xti_hz, uint32_t pll_mul);
00308
00334 enum clkctr_status CLKCTR_SetPllMan(CLKCTR_Type * base, uint32_t xti_hz,
00335     uint32_t nr, uint32_t nf, uint32_t od);
00336
00346 void CLKCTR_SetSysDiv(CLKCTR_Type * base, uint16_t fclk_div,
00347     uint16_t sysclk_div, uint16_t gnssclk_div, uint16_t qspiclk_div);
00348
00369 enum clkctr_status CLKCTR_SetXTI(CLKCTR_Type * base, uint32_t frequency);
00370
00380 uint32_t CLKCTR_GetXTI(CLKCTR_Type * base);
00381
00393 enum clkctr_status CLKCTR_SetXTI32(CLKCTR_Type * base, uint32_t frequency);
00394
00404 uint32_t CLKCTR_GetXTI32(CLKCTR_Type * base);
00405
00419 enum clkctr_status CLKCTR_SetHFI(CLKCTR_Type * base, uint32_t frequency);
00420
00430 uint32_t CLKCTR_GetHFI(CLKCTR_Type * base);
00431
00443 enum clkctr_status CLKCTR_SetLFI(CLKCTR_Type * base, uint32_t frequency);
00444
00454 uint32_t CLKCTR_GetLFI(CLKCTR_Type * base);
00455
00469 enum clkctr_status CLKCTR_SetI2SExtClk(CLKCTR_Type * base, uint32_t frequency);
00470
00480 uint32_t CLKCTR_GetI2SExtClk(CLKCTR_Type * base);
00481
00500 enum clkctr_status CLKCTR_GetAllDiv(CLKCTR_Type * base,
00501     struct clkctr_div *divisors);
00502
00508 uint32_t CLKCTR_GetMCOdiv(CLKCTR_Type * base);
00509
00522 enum clkctr_status CLKCTR_SetMCOdiv(CLKCTR_Type * base, uint32_t value);
00523
00531 uint32_t CLKCTR_GetFClkDiv(CLKCTR_Type * base);
00532
00545 enum clkctr_status CLKCTR_SetFClkDiv(CLKCTR_Type * base, uint32_t value);
00546
00554 uint32_t CLKCTR_GetSysClkDiv(CLKCTR_Type * base);
00555
00568 enum clkctr_status CLKCTR_SetSysClkDiv(CLKCTR_Type * base, uint32_t value);
00569
00577 uint32_t CLKCTR_GetGNSSClkDiv(CLKCTR_Type * base);
00578
00591 enum clkctr_status CLKCTR_SetGNSSClkDiv(CLKCTR_Type * base, uint32_t value);

```

```

00592
00600 uint32_t CLKCTR_GetQSPIClkDiv(CLKCTR_Type * base);
00601
00614 enum clkctr_status CLKCTR_SetQSPIClkDiv(CLKCTR_Type * base, uint32_t value);
00615
00623 uint32_t CLKCTR_GetI2SClkDiv(CLKCTR_Type * base);
00624
00637 enum clkctr_status CLKCTR_SetI2SClkDiv(CLKCTR_Type * base, uint32_t value);
00638
00657 enum clkctr_status CLKCTR_GetPLLConfig(CLKCTR_Type * base,
00658     struct clkctr_pll_cfg *config);
00659
00675 enum clkctr_status CLKCTR_SetPLLConfig(CLKCTR_Type * base,
00676     struct clkctr_pll_cfg config);
00677
00694 uint32_t CLKCTR_GetXTIClk(CLKCTR_Type * base);
00695
00703 uint32_t CLKCTR_GetHFIClk(CLKCTR_Type * base);
00704
00712 uint32_t CLKCTR_GetRTCClk(CLKCTR_Type * base);
00713
00721 uint32_t CLKCTR_GetLPClk(CLKCTR_Type * base);
00722
00731 uint32_t CLKCTR_GetPLLClk(CLKCTR_Type * base);
00732
00740 uint32_t CLKCTR_GetPLLRef(CLKCTR_Type * base);
00741
00749 uint32_t CLKCTR_GetMainClk(CLKCTR_Type * base);
00750
00758 uint32_t CLKCTR_GetUSBClk(CLKCTR_Type * base);
00759
00767 uint32_t CLKCTR_GetFCIkInt(CLKCTR_Type * base);
00768
00778 uint32_t CLKCTR_GetFCIk(CLKCTR_Type * base);
00779
00787 uint32_t CLKCTR_GetSysClk(CLKCTR_Type * base);
00788
00796 uint32_t CLKCTR_GetGNSSClk(CLKCTR_Type * base);
00797
00805 uint32_t CLKCTR_GetQSPIClk(CLKCTR_Type * base);
00806
00814 uint32_t CLKCTR_GetI2SClk(CLKCTR_Type * base);
00815
00823 uint32_t CLKCTR_GetMCOClk(CLKCTR_Type * base);
00824
00835 uint32_t CLKCTR_GetHFIClkForMainClk(CLKCTR_Type * base);
00836
00858 enum clkctr_status CLKCTR_SetSwitchMCOClk(CLKCTR_Type * base, uint32_t value);
00859
00875 enum clkctr_status CLKCTR_SetSwitchPLLRef(CLKCTR_Type * base, uint32_t value);
00876
00889 enum clkctr_status CLKCTR_SetSwitchMainClk(CLKCTR_Type * base, uint32_t value);
00890
00903 enum clkctr_status CLKCTR_SetSwitchRTCClk(CLKCTR_Type * base, uint32_t value);
00904
00921 enum clkctr_status CLKCTR_SetSwitchLPClk(CLKCTR_Type * base, uint32_t value);
00922
00935 enum clkctr_status CLKCTR_SetSwitchUSBClk(CLKCTR_Type * base, uint32_t value);
00936
00949 enum clkctr_status CLKCTR_SetSwitchI2SClk(CLKCTR_Type * base, uint32_t value);
00950
00970 enum clkctr_status CLKCTR_SetSwitchPMUDIS(CLKCTR_Type * base, uint32_t value);
00971
00979 uint32_t CLKCTR_GetSwitchMCOClk(CLKCTR_Type * base);
00980
00988 uint32_t CLKCTR_GetSwitchPLLRef(CLKCTR_Type * base);
00989
00997 uint32_t CLKCTR_GetSwitchMainClk(CLKCTR_Type * base);
00998
01006 uint32_t CLKCTR_GetSwitchRTCClk(CLKCTR_Type * base);
01007
01015 uint32_t CLKCTR_GetSwitchLPClk(CLKCTR_Type * base);
01016
01024 uint32_t CLKCTR_GetSwitchUSBClk(CLKCTR_Type * base);
01025
01033 uint32_t CLKCTR_GetSwitchI2SClk(CLKCTR_Type * base);
01034
01047 uint32_t CLKCTR_GetSwitchPMUDIS(CLKCTR_Type * base);
01048
01071 enum clkctr_status CLKCTR_SetHFITrim(CLKCTR_Type * base, uint32_t index);
01072
01081 uint32_t CLKCTR_GetHFITrim(CLKCTR_Type * base);
01082
01091 uint32_t CLKCTR_GetMCOEn(CLKCTR_Type * base);
01092
01101 enum clkctr_status CLKCTR_SetMCOEn(CLKCTR_Type * base, uint32_t enable);
01102

```

```

01113 enum clkctr_status CLKCTR_GetClkForce(CLKCTR_Type * base,
01114     enum clkctr_device_clk_force device,
01115     enum clkctr_clk_force_type *clk_type);
01116
01127 enum clkctr_status CLKCTR_SetClkForce(CLKCTR_Type * base,
01128     enum clkctr_device_clk_force device,
01129     enum clkctr_clk_force_type clk_type);
01130
01151 enum clkctr_status CLKCTR_SetFrequency(CLKCTR_Type * base,
01152     enum clkctr_extern_freq input, uint32_t frequency);
01153
01163 uint32_t CLKCTR_GetFrequency(CLKCTR_Type * base,
01164     enum clkctr_extern_freq input);
01165
01176 enum clkctr_status CLKCTR_GetClk(CLKCTR_Type * base, enum clkctr_int_freq clk,
01177     uint32_t *value);
01178
01190 enum clkctr_status CLKCTR_SetSwitchClk(CLKCTR_Type * base,
01191     enum clkctr_int_freq clk, uint32_t value);
01192
01203 enum clkctr_status CLKCTR_GetSwitchClk(CLKCTR_Type * base,
01204     enum clkctr_int_freq clk, uint32_t *value);
01205
01217 enum clkctr_status CLKCTR_SetDivClk(CLKCTR_Type * base,
01218     enum clkctr_int_freq clk, uint32_t value);
01219
01230 enum clkctr_status CLKCTR_GetDivClk(CLKCTR_Type * base,
01231     enum clkctr_int_freq clk, uint32_t *value);
01232
01237 #if defined(__cplusplus)
01238 }
01239 #endif /* __cplusplus */
01240
01241 #endif /* HAL_CLKCTR_H */
01242

```

## 6.5 hal\_common.h

```

00001
00012 /*
00013  * @file hal_common.h
00014  *
00015  * @brief Интерфейс общих определений, используемых библиотекой HAL
00016  */
00017
00018 #ifndef HAL_COMMON_H
00019 #define HAL_COMMON_H
00020
00021 #include <inttypes.h>
00022 #include <assert.h>
00023 #include <stdbool.h>
00024 #include <stdint.h>
00025 #include <stdlib.h>
00026 #include <string.h>
00027 #include <stdio.h>
00028 #include <math.h>
00029
00030 #include "hal_device.h"
00031
00032 #if defined(__ICCARM__) || (defined(__CC_ARM) || defined(__ARMCC_VERSION)) || defined(__GNUC__)
00033 /* #if __GNUC__ < 10 || (__GNUC__ == 10 && __GNUC_MINOR__ < 3)
00034  * #error "Unsupported GCC version. Download ARM GCC toolchain v10.3.0 or newer."
00035  */
00036 #include <stddef.h>
00037 #ifndef __cplusplus
00038 #pragma GCC diagnostic ignored "-Wstrict-prototypes"
00039 #endif
00040 #endif
00041
00042 #include "hal_device.h"
00043
00044 /* #undef UNUSED */
00045 #define UNUSED(x) ((void)(x));
00050 #define MAKE_VERSION(major, minor, bugfix) \
00051     (((major) << 16) | ((minor) << 8) | (bugfix))
00052
00056 #define HAL_COMMON_DRIVER_VERSION (MAKE_VERSION(0, 1, 0))
00057
00061 typedef int32_t status_t;
00062
00072 void LBBLEndian(uint32_t *val, uint8_t num_bits);
00073
00081 #if !defined(BE_TO_LE32)
00082 #define BE_TO_LE32(x) \

```

```

00083     (((unsigned)(x) » 24) & 0xFF) \
00084     | (((unsigned)(x) « 24) & 0xFF000000) \
00085     | (((unsigned)(x) » 8) & 0xFF00) \
00086     | (((unsigned)(x) « 8) & 0xFF0000))
00087 #endif
00088
00096 #if !defined(BE_TO_LE24)
00097 #define BE_TO_LE24(x) \
00098     (((unsigned)(x) » 16) & 0xFF) \
00099     | (((unsigned)(x) & 0xFF00) \
00100     | (((unsigned)(x) « 16) & 0xFF0000))
00101 #endif
00102
00110 #if !defined(BE_TO_LE16)
00111 #define BE_TO_LE16(x) \
00112     (((unsigned)(x) » 8) & 0xFF) \
00113     | (((unsigned)(x) « 8) & 0xFF00))
00114 #endif
00115
00124 #if !defined(MIN)
00125 #define MIN(a, b) (((a) < (b)) ? (a) : (b))
00126 #endif
00127
00136 #if !defined(MAX)
00137 #define MAX(a, b) (((a) > (b)) ? (a) : (b))
00138 #endif
00139
00147 #if !defined(DIM)
00148 #define DIM(x) (sizeof(x) / sizeof((x)[0]))
00149 #endif
00150
00151 #if !defined(UINT16_MAX)
00152 #define UINT16_MAX ((uint16_t) -1)
00153 #endif
00154
00155 #if !defined(UINT32_MAX)
00156 #define UINT32_MAX ((uint32_t) -1)
00157 #endif
00158
00171 #if defined(__GNUC__) && !defined(__ARMCC_VERSION)
00172 #define SUPPRESS_FALL_THROUGH_WARNING() __attribute__((fallthrough))
00173 #else
00174 #define SUPPRESS_FALL_THROUGH_WARNING()
00175 #endif
00176
00189 #undef FIELD_BIT
00190 #define FIELD_BIT(top, bottom) ((top) - (bottom) + 1)
00191
00192 #endif /* HAL_COMMON_H */
00193

```

## 6.6 Файл devices/eliot1/drivers/hal\_dma.h

Интерфейс драйвера прямого доступа к памяти

```
#include "hal_common.h"
```

Структуры данных

- struct [\\_dma\\_descriptor](#)  
Описатель следующего блока DMA (LLI)
- struct [\\_dma\\_channel\\_reg](#)  
Дескриптор на регистры канала DMA.
- struct [dma\\_channel\\_ctl\\_cfg](#)  
Описание конфигурации пересылки
- struct [\\_dma\\_channel\\_config](#)  
Конфигурация канала DMA.
- struct [\\_dma\\_handle](#)  
Управляющая структура передачи
- struct [\\_dma\\_multiblock\\_config](#)  
Конфигурация многоблочной передачи

## Макросы

- `#define HAL_DMA_DRIVER_VERSION (MAKE_VERSION(1, 0, 0))`  
Версия драйвера
- `#define DMA_AHB_MAX_BLOCK_SIZE (4095U)`  
Максимальный размер транзакции на AHB шине
- `#define HAL_FEATURE_DMA_NUMBER_OF_CHANNELS (8U)`  
Число каналов одного контроллера DMA.
- `#define HAL_FEATURE_DMA_MAX_NUMBER_HW_HANDSHAKES (16U)`  
Максимальное число аппаратных интерфейсов запросов DMA.
- `#define HAL_FEATURE_DMA_ALL_CHANNELS (HAL_FEATURE_DMA_NUMBER_OF_CHANNELS * 2U)`  
Общее число каналов DMA.
- `#define HAL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE (4U)`  
Размер выравнивания дескриптора
- `#define DMA_ALLOCATE_HEAD_DESCRIPTOR(name, number) dma_descriptor_t name[number] __attribute__((aligned(HAL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE)))`  
Объявление дескриптора и выравнивание адреса дескриптора
- `#define REG_OFFSET (0x58)`  
Разница адресов идентичных регистров из соседних каналов
- `#define DMA_HANDSHAKE_UART0_TX (0U)`  
Номер handshake запроса UART0 TX.
- `#define DMA_HANDSHAKE_UART0_RX (1U)`  
Номер handshake запроса UART0 RX.
- `#define DMA_HANDSHAKE_UART1_TX (2U)`  
Номер handshake запроса UART1 TX.
- `#define DMA_HANDSHAKE_UART1_RX (3U)`  
Номер handshake запроса UART1 RX.
- `#define DMA_HANDSHAKE_UART2_TX (0U)`  
Номер handshake запроса UART2 TX.
- `#define DMA_HANDSHAKE_UART2_RX (1U)`  
Номер handshake запроса UART2 RX.
- `#define DMA_HANDSHAKE_UART3_TX (2U)`  
Номер handshake запроса UART3 TX.
- `#define DMA_HANDSHAKE_UART3_RX (3U)`  
Номер handshake запроса UART3 RX.
- `#define DMA_HANDSHAKE_SPI0_TX (4U)`  
Номер handshake запроса SPI0 TX.
- `#define DMA_HANDSHAKE_SPI0_RX (5U)`  
Номер handshake запроса SPI0 RX.
- `#define DMA_HANDSHAKE_SPI1_TX (6U)`  
Номер handshake запроса SPI1 TX.
- `#define DMA_HANDSHAKE_SPI1_RX (7U)`  
Номер handshake запроса SPI1 RX.
- `#define DMA_HANDSHAKE_SPI2_TX (8U)`  
Номер handshake запроса SPI2 TX.
- `#define DMA_HANDSHAKE_SPI2_RX (9U)`  
Номер handshake запроса SPI2 RX.
- `#define DMA_HANDSHAKE_I2C0_TX (10U)`  
Номер handshake запроса I2C0 TX.
- `#define DMA_HANDSHAKE_I2C0_RX (11U)`

- Номер handshake запроса I2C0 RX.
- `#define DMA_HANDSHAKE_I2C1_TX (12U)`  
Номер handshake запроса I2C1 TX.
- `#define DMA_HANDSHAKE_I2C1_RX (13U)`  
Номер handshake запроса I2C1 RX.
- `#define DMA_HANDSHAKE_QSPI_TX (14U)`  
Номер handshake запроса QSPI TX.
- `#define DMA_HANDSHAKE_QSPI_RX (15U)`  
Номер handshake запроса QSPI RX.

## Функции

API HAL для драйвера модуля DMA

- void `DMA_Init` (DMA\_Type \*base)  
Включение DMA.
- void `DMA_Deinit` (DMA\_Type \*base)  
Отключение DMA.
- `dma_status_t` `DMA_CreateHandle` (`dma_handle_t` \*handle, DMA\_Type \*base, uint32\_t channel)  
Инициализация структуры `dma_handle_t`.
- static void `DMA_EnableChannel` (DMA\_Type \*base, uint32\_t channel)  
Активация канала
- void `DMA_DisableChannel` (DMA\_Type \*base, uint32\_t channel)  
Выключение канала
- static void `DMA_EnableChannelPeriphRq` (DMA\_Type \*base, uint32\_t channel)  
Установка запроса на передачу с периферией
- void `DMA_HardwareHandshakeEnable` (`dma_handle_t` \*handle, bool enable, uint32\_t req\_num)  
Установка аппаратного интерфейса запроса DMA.
- void `DMA_SetChannelPriority` (DMA\_Type \*base, uint32\_t channel, `dma_priority_t` priority)  
Установка приоритета канала
- void `DMA_SetCallback` (`dma_handle_t` \*handle, `dma_callback` callback, void \*user\_data)  
Установка функции обратного вызова
- static bool `DMA_ChannelsIsActive` (DMA\_Type \*base, uint32\_t channel)  
Функция для проверки работы DMA канала
- `dma_priority_t` `DMA_GetChannelPriority` (DMA\_Type \*base, uint32\_t channel)  
Получение информации о приоритете канала
- uint64\_t `DMA_GetCTLCfgMask` (struct `dma_channel_ctl_cfg` \*ctlxcfg)  
Получение маски для конфигурации пересылки
- void `DMA_PrepareChannelTransfer` (`dma_channel_config_t` \*config, void \*src\_addr, void \*dst\_addr, uint64\_t xfer\_cfg)  
Подготовка канала к передаче
- `dma_status_t` `DMA_SubmitChannelTransfer` (`dma_handle_t` \*handle, `dma_channel_config_t` \*config)  
Отправление конфигурации передачи
- void `DMA_SubmitChannelDescriptor` (`dma_handle_t` \*handle, `dma_descriptor_t` \*descriptor)  
Отправка дескриптора передачи
- uint32\_t `DMA_GetDescriptorCount` (uint32\_t size\_bytes, uint8\_t transfer\_width)  
Расчёт количества дескрипторов многоблочной передачи
- void `DMA_AbortTransfer` (`dma_handle_t` \*handle)  
Прерывание передачи без потери данных
- `dma_status_t` `DMA_StartTransfer` (`dma_handle_t` \*handle)  
Начать транзакцию
- void `DMA_EnableChannelInterrupt` (DMA\_Type \*base, uint32\_t channel, uint8\_t int\_mask)  
Разрешение источника прерывания
- void `DMA_DisableChannelInterrupt` (DMA\_Type \*base, uint32\_t channel, uint8\_t int\_mask)



- Отключение источника прерывания
- void `DMA_SetupDescriptor` (`dma_descriptor_t` \*desc, uint64\_t xfercfg, void \*src\_addr, void \*dst\_addr, void \*next\_desc)
- Инициализация дескриптора
- `dma_status_t` `DMA_SubmitChannelTransferParameter` (`dma_handle_t` \*handle, uint64\_t xfer\_cfg, void \*src\_addr, void \*dst\_addr)
- Отправка конфигурации передачи в таблицу дескрипторов
- void `DMA_ChannelIRQHandle` (`dma_handle_t` \*handle)
- Обработчик прерываний
- void `DMA_ScatterGatherEnable` (struct `dma_channel_ctl_cfg` \*ctlxcfg, bool scatter\_en, bool gather\_en)
- Разрешение режимов Разброса/Сбора
- void `DMA_SetupSrcGather` (`dma_handle_t` \*handle, uint32\_t count, uint32\_t interval)
- Настройка режима сбора источника
- void `DMA_SetupDstScatter` (`dma_handle_t` \*handle, uint32\_t count, uint32\_t interval)
- Настройка режима разброса приемника
- void `DMA_InitMultiblockDescriptor` (`dma_descriptor_t` \*desc, `dma_multiblock_config_t` \*config)
- Функция, инициализирующая DMA дескрипторы многоблочной передачи.

#### CHANNEL\_CTL - Регистр управления каналом

- #define `DMA_CHANNEL_CTL_BLOCKSIZE_MASK` (0xFFF00000000ULL)
- #define `DMA_CHANNEL_CTL_BLOCKSIZE_SHIFT` (32ULL)
- #define `DMA_CHANNEL_CTL_BLOCKSIZE(x)` (((uint64\_t)((uint64\_t)(x)) << `DMA_CHANNEL_CTL_BLOCKSIZE_SHIFT`) & `DMA_CHANNEL_CTL_BLOCKSIZE_MASK`)
- Размер блока
- #define `DMA_CHANNEL_CTL`(blockSize, srcEnChain, dstEnChain, transferType, scatter, gather, srcMSize, dstMSize, sInc, dInc, srcTransferWidth, dstTransferWidth, int\_en)
- Конфигурация регистра CTL.
- enum
- Разрядность передачи канала DMA.
- enum
- Размер пакета передачи в словах данных разрядностью DMA\_TransferBitWidth.
- enum
- Тип изменения адреса при пересылке
- enum `_dma_status`
- Статусы возврата из функций для драйвера DMA.
- enum `_dma_priority`
- Приоритеты каналов
- enum `_dma_int`
- Источники прерываний DMA.
- enum `_dma_transfer_type`
- Тип пересылок DMA и управление размером блока
- typedef enum `_dma_status` dma\_status\_t
- Статусы возврата из функций для драйвера DMA.
- typedef enum `_dma_priority` dma\_priority\_t
- Приоритеты каналов
- typedef enum `_dma_int` dma\_irq\_t
- Источники прерываний DMA.
- typedef enum `_dma_transfer_type` dma\_transfer\_type\_t
- Тип пересылок DMA и управление размером блока
- typedef struct `_dma_descriptor` dma\_descriptor\_t

- Описатель следующего блока DMA (LLI)
- typedef struct `_dma_channel_reg` dma\_channel\_regs  
Дескриптор на регистры канала DMA.
- typedef struct `_dma_channel_config` dma\_channel\_config\_t  
Конфигурация канала DMA.
- typedef void(\* dma\_callback) (struct `_dma_handle` \*handle, void \*user\_data, dma\_irq\_t intmode)  
Функция обратного вызова
- typedef struct `_dma_handle` dma\_handle\_t  
Управляющая структура передачи
- typedef struct `_dma_multiblock_config` dma\_multiblock\_config\_t  
Конфигурация многоблочной передачи

## 6.6.1 Подробное описание

Интерфейс драйвера прямого доступа к памяти

## 6.7 hal\_dma.h

[См. документацию.](#)

```
00001
00020 #ifndef HAL_DMA_H
00021 #define HAL_DMA_H
00022
00023 #include "hal_common.h"
00024
00028 #define HAL_DMA_DRIVER_VERSION (MAKE_VERSION(1, 0, 0))
00029
00031 #define DMA_AHB_MAX_BLOCK_SIZE (4095U)
00033 #define HAL_FEATURE_DMA_NUMBER_OF_CHANNELS (8U)
00035 #define HAL_FEATURE_DMA_MAX_NUMBER_HW_HANDSHAKES (16U)
00037 #define HAL_FEATURE_DMA_ALL_CHANNELS (HAL_FEATURE_DMA_NUMBER_OF_CHANNELS * 2U)
00039 #define HAL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE (4U)
00041 #define DMA_ALLOCATE_HEAD_DESCRIPTOR(name, number) \
00042     dma_descriptor_t name[number] \
00043     __attribute__((aligned(HAL_FEATURE_DMA_LINK_DESCRIPTOR_ALIGN_SIZE)))
00044 #define REG_OFFSET (0x58)
00045
00047 #define DMA_HANDSHAKE_UART0_TX (0U)
00049 #define DMA_HANDSHAKE_UART0_RX (1U)
00051 #define DMA_HANDSHAKE_UART1_TX (2U)
00053 #define DMA_HANDSHAKE_UART1_RX (3U)
00055 #define DMA_HANDSHAKE_UART2_TX (0U)
00057 #define DMA_HANDSHAKE_UART2_RX (1U)
00059 #define DMA_HANDSHAKE_UART3_TX (2U)
00061 #define DMA_HANDSHAKE_UART3_RX (3U)
00062
00064 #define DMA_HANDSHAKE_SPI0_TX (4U)
00066 #define DMA_HANDSHAKE_SPI0_RX (5U)
00068 #define DMA_HANDSHAKE_SPI1_TX (6U)
00070 #define DMA_HANDSHAKE_SPI1_RX (7U)
00072 #define DMA_HANDSHAKE_SPI2_TX (8U)
00074 #define DMA_HANDSHAKE_SPI2_RX (9U)
00075
00077 #define DMA_HANDSHAKE_I2C0_TX (10U)
00079 #define DMA_HANDSHAKE_I2C0_RX (11U)
00081 #define DMA_HANDSHAKE_I2C1_TX (12U)
00083 #define DMA_HANDSHAKE_I2C1_RX (13U)
00085 #define DMA_HANDSHAKE_QSPI_TX (14U)
00087 #define DMA_HANDSHAKE_QSPI_RX (15U)
00088
00089
00091 #define DMA_CHANNEL_CTL_BLOCKSIZE_MASK (0xFFF00000000ULL)
00092 #define DMA_CHANNEL_CTL_BLOCKSIZE_SHIFT (32ULL)
00097 #define DMA_CHANNEL_CTL_BLOCKSIZE(x) (((uint64_t)((uint64_t)(x)) <
DMA_CHANNEL_CTL_BLOCKSIZE_SHIFT) & DMA_CHANNEL_CTL_BLOCKSIZE_MASK)
00098
00116 #define DMA_CHANNEL_CTL(blockSize, srcEnChain, dstEnChain, transferType, \
00117     scatter, gather, srcMSize, dstMSize, sInc, dInc, \
```

```

00118     srcTransferWidth, dstTransferWidth, int_en) \
00119 0x0ULL | DMA_CHANNEL_CTL_BLOCKSIZE(blockSize) | \
00120 DMA_CTL0_LO_LLPSRC_EN_VAL(srcEnChain) | \
00121 DMA_CTL0_LO_LLPSRCDST_EN_VAL(dstEnChain) | \
00122 DMA_CTL0_LO_TT_FC_VAL(transferType) | \
00123 DMA_CTL0_LO_DST_SCATTER_EN_VAL(scatter) | \
00124 DMA_CTL0_LO_SRC_GATHER_EN_VAL(gather) | \
00125 DMA_CTL0_LO_SRC_MSIZESRC_VAL(srcMSize) | \
00126 DMA_CTL0_LO_DEST_MSIZEDST_VAL(dstMSize) | \
00127 DMA_CTL0_LO_SINC_VAL(sInc) | \
00128 DMA_CTL0_LO_DINC_VAL(dInc) | \
00129 DMA_CTL0_LO_SRC_TR_WIDTH_VAL(srcTransferWidth) | \
00130 DMA_CTL0_LO_DST_TR_WIDTH_VAL(dstTransferWidth) | \
00131 DMA_CTL0_LO_INT_EN_VAL(int_en)
00132
00133 enum {
00134     DMA_Transfer8BitWidth = 0U,
00135     DMA_Transfer16BitWidth = 1U,
00136     DMA_Transfer32BitWidth = 2U,
00137     DMA_Transfer64BitWidth = 3U,
00138     DMA_Transfer128BitWidth = 4U,
00139     DMA_Transfer256BitWidth = 5U,
00140 };
00141
00142 enum {
00143     DMA_BurstSize1 = 0U,
00144     DMA_BurstSize4 = 1U,
00145     DMA_BurstSize8 = 2U,
00146     DMA_BurstSize16 = 3U,
00147     DMA_BurstSize32 = 4U,
00148     DMA_BurstSize64 = 5U,
00149     DMA_BurstSize128 = 6U,
00150     DMA_BurstSize256 = 7U,
00151 };
00152
00153 enum {
00154     DMA_Incr = 0U,
00155     DMA_Decr = 1U,
00156     DMA_NoChange = 2U,
00157 };
00158
00159 typedef enum dma_status {
00160     DMA_Status_Success = 0U,
00161     DMA_Status_Busy = 1U,
00162     DMA_Status_NoBase = 3U,
00163     DMA_Status_Fail = 4U,
00164 } dma_status_t;
00165
00166 typedef enum dma_priority {
00167     DMA_ChannelPriority0 = 0,
00168     DMA_ChannelPriority1 = 1,
00169     DMA_ChannelPriority2 = 2,
00170     DMA_ChannelPriority3 = 3,
00171     DMA_ChannelPriority4 = 4,
00172     DMA_ChannelPriority5 = 5,
00173     DMA_ChannelPriority6 = 6,
00174     DMA_ChannelPriority7 = 7,
00175 } dma_priority_t;
00176
00177 typedef enum dma_irq {
00178     DMA_IntTfr = 1U,
00179     DMA_IntBlock = 2U,
00180     DMA_IntDstTran = 4U,
00181     DMA_IntSrcTran = 8U,
00182     DMA_IntError = 16U,
00183     DMA_AllIRQCount = 5U,
00184     DMA_AllIRQ = DMA_IntTfr
00185         | DMA_IntBlock | DMA_IntDstTran
00186         | DMA_IntSrcTran | DMA_IntError,
00187 } dma_irq_t;
00188
00189 typedef enum dma_transfer_type {
00190     DMA_MemoryToMemory_DMA = 0U,
00191     DMA_MemoryToPeripheral_DMA = 1U,
00192     DMA_PeripheralToMemory_DMA = 2U,
00193     DMA_PeripheralToPeripheral_DMA = 3U,
00194     DMA_PeripheralToMemory_Peripheral = 4U,
00195     DMA_PeripheralToPeripheral_SRC = 5U,
00196     DMA_MemoryToPeripheral_Peripheral = 6U,
00197     DMA_PeripheralToPeripheral_DST = 7U,
00198 } dma_transfer_type_t;
00199
00200 typedef struct dma_descriptor {
00201     __IOM uint32_t SAR;
00202     __IOM uint32_t DAR;
00203     __IOM uint32_t LLP;
00204     __IOM uint32_t CTL_LO;

```

```

00227     __IOM uint32_t CTL_HI;
00228     __IOM uint32_t SSTAT;
00229     __IOM uint32_t DSTAT;
00230 } dma_descriptor_t;
00231
00232 typedef struct dma_channel_reg {
00233     __IOM uint32_t SAR_LO;
00234     __IOM uint32_t SAR_HI;
00235     __IOM uint32_t DAR_LO;
00236     __IOM uint32_t DAR_HI;
00237     __IOM uint32_t LLP_LO;
00238     __IOM uint32_t LLP_HI;
00239     __IOM uint32_t CTL_LO;
00240     __IOM uint32_t CTL_HI;
00241     __IOM uint32_t SSTAT_LO;
00242     __IOM uint32_t SSTAT_HI;
00243     __IOM uint32_t DSTAT_LO;
00244     __IOM uint32_t DSTAT_HI;
00245     __IOM uint32_t SSTATAR_LO;
00246     __IOM uint32_t SSTATAR_HI;
00247     __IOM uint32_t DSTATAR_LO;
00248     __IOM uint32_t DSTATAR_HI;
00249     __IOM uint32_t CFG_LO;
00250     __IOM uint32_t CFG_HI;
00251     __IOM uint32_t SGR_LO;
00252     __IOM uint32_t SGR_HI;
00253     __IOM uint32_t DSR_LO;
00254     __IOM uint32_t DSR_HI;
00255 } dma_channel_regs;
00256
00257 struct dma_channel_ctl_cfg {
00258     uint16_t block_size;
00259     bool llp_src_en;
00260     bool llp_dst_en;
00261     dma_transfer_type_t transfer_type;
00262     bool scatter_en;
00263     bool gather_en;
00264     uint8_t src_burst_size;
00265     uint8_t dst_burst_size;
00266     uint8_t src_incr;
00267     uint8_t dst_incr;
00268     uint8_t src_tr_width;
00269     uint8_t dst_tr_width;
00270     bool int_en;
00271 };
00272
00273 typedef struct dma_channel_config {
00274     void *src_addr;
00275     void *dst_addr;
00276     void *next_desc;
00277     uint64_t ctx_cfg;
00278     bool is_src_periph;
00279     bool is_dst_periph;
00280 } dma_channel_config_t;
00281
00282 struct dma_handle;
00283
00284 typedef void (*dma_callback)(struct dma_handle *handle, void *user_data,
00285     dma_irq_t intmode);
00286
00287 typedef struct dma_handle {
00288     dma_callback callback;
00289     void *user_data;
00290     DMA_Type *base;
00291     uint32_t channel;
00292 } dma_handle_t;
00293
00294 typedef struct dma_multiblock_config {
00295     uint32_t count;
00296     uint32_t data_size;
00297     dma_transfer_type_t transfer_type;
00298     bool scatter_en;
00299     bool gather_en;
00300     uint32_t src_burst_size;
00301     uint32_t dst_burst_size;
00302     uint32_t src_incr;
00303     uint32_t dst_incr;
00304     uint32_t src_data_width;
00305     uint32_t dst_data_width;
00306     void *src_addr;
00307     void *dst_addr;
00308     bool int_en;
00309 } dma_multiblock_config_t;
00310
00311 void DMA_Init(DMA_Type *base);
00312
00313 void DMA_Deinit(DMA_Type *base);

```

```

00341
00352 dma_status_t DMA_CreateHandle(dma_handle_t *handle, DMA_Type *base,
00353     uint32_t channel);
00354
00361 static inline void DMA_EnableChannel(DMA_Type *base, uint32_t channel)
00362 {
00363     base->CHENREG_LO |= (uint32_t)((1 << (channel + 8)) | (1 << channel));
00364 }
00365
00372 void DMA_DisableChannel(DMA_Type *base, uint32_t channel);
00373
00380 static inline void DMA_EnableChannelPeriphRq(DMA_Type *base,
00381     uint32_t channel)
00382 {
00383     base->REQDSTREG_LO = ((1 << (channel + 8)) | (1 << channel));
00384     base->REQSRCREG_LO = ((1 << (channel + 8)) | (1 << channel));
00385 }
00386
00394 void DMA_HardwareHandshakeEnable(dma_handle_t *handle, bool enable,
00395     uint32_t req_num);
00396
00404 void DMA_SetChannelPriority(DMA_Type *base, uint32_t channel,
00405     dma_priority_t priority
00406 );
00407
00415 void DMA_SetCallback(dma_handle_t *handle, dma_callback callback,
00416     void *user_data);
00417
00426 static inline bool DMA_ChannelIsActive(DMA_Type *base, uint32_t channel)
00427 {
00428     return ((base->CHENREG_LO >> channel) & 1);
00429 }
00430
00448 dma_priority_t DMA_GetChannelPriority(DMA_Type *base, uint32_t channel);
00449
00457 uint64_t DMA_GetCTLCfgMask(struct dma_channel_ctl_cfg *ctlxcfg);
00458
00469 void DMA_PrepareChannelTransfer(dma_channel_config_t *config,
00470     void *src_addr,
00471     void *dst_addr,
00472     uint64_t xfer_cfg
00473 );
00474
00503 dma_status_t DMA_SubmitChannelTransfer(dma_handle_t *handle,
00504     dma_channel_config_t *config
00505 );
00506
00533 void DMA_SubmitChannelDescriptor(dma_handle_t *handle,
00534     dma_descriptor_t *descriptor
00535 );
00536
00545 uint32_t DMA_GetDescriptorCount(uint32_t size_bytes, uint8_t transfer_width);
00546
00552 void DMA_AbortTransfer(dma_handle_t *handle);
00553
00567 dma_status_t DMA_StartTransfer(dma_handle_t *handle);
00568
00580 void DMA_EnableChannelInterrupt(DMA_Type *base, uint32_t channel,
00581     uint8_t int_mask);
00582
00590 void DMA_DisableChannelInterrupt(DMA_Type *base, uint32_t channel,
00591     uint8_t int_mask);
00592
00602 void DMA_SetupDescriptor(dma_descriptor_t *desc, uint64_t xfercfg,
00603     void *src_addr, void *dst_addr,
00604     void *next_desc
00605 );
00606
00636 dma_status_t DMA_SubmitChannelTransferParameter(dma_handle_t *handle,
00637     uint64_t xfer_cfg, void *src_addr, void *dst_addr
00638 );
00639
00645 void DMA_ChannelIRQHandle(dma_handle_t *handle);
00646
00654 void DMA_ScatterGatherEnable(struct dma_channel_ctl_cfg *ctlxcfg,
00655     bool scatter_en, bool gather_en);
00656
00664 void DMA_SetupSrcGather(dma_handle_t *handle, uint32_t count, uint32_t interval);
00665
00673 void DMA_SetupDstScatter(dma_handle_t *handle, uint32_t count, uint32_t interval);
00674
00681 void DMA_InitMultiblockDescriptor(dma_descriptor_t *desc,
00682     dma_multiblock_config_t *config);
00683
00688 #endif /* HAL_DMA_H */
00689

```

## 6.8 Файл devices/eliot1/drivers/hal\_dualtimer.h

Интерфейс драйвера сдвоенного таймера

```
#include "hal_common.h"
```

Структуры данных

- struct [dualtimer\\_hardware\\_config](#)  
Конфигурация аппаратной части сдвоенного таймера

Макросы

- #define [DUALTIMER\\_NUMBER\\_OF\\_DUALTIMERS](#) 1
- #define [DUALTIMER\\_MAX\\_INDEX](#) 1

Перечисления

- enum [dualtimer\\_status](#)  
Статусы драйвера сдвоенного таймера
- enum [dualtimer\\_work\\_enable](#)  
Разрешение работы таймера
- enum [dualtimer\\_mode](#)  
Режим работы таймера
- enum [dualtimer\\_interrupt\\_control](#)  
Управление прерываниями
- enum [dualtimer\\_prescale](#)  
Предделители частоты
- enum [dualtimer\\_timer\\_size](#)  
Размер счетчика
- enum [dualtimer\\_number\\_of\\_repetitions](#)  
Количество запусков

Функции

Интерфейс драйвера

- enum [dualtimer\\_status](#) [DUALTIMER\\_GetDefaultConfig](#) (struct [dualtimer\\_hardware\\_config](#) \*config)  
Создание конфигурации по умолчанию
- enum [dualtimer\\_status](#) [DUALTIMER\\_Init](#) (DTIM\_Type \*base, uint32\_t index, struct [dualtimer\\_hardware\\_config](#) config)  
Инициализация сдвоенного таймера
- enum [dualtimer\\_status](#) [DUALTIMER\\_Deinit](#) (DTIM\_Type \*base, uint32\_t index)  
Деинициализация сдвоенного таймера
- enum [dualtimer\\_status](#) [DUALTIMER\\_Run](#) (DTIM\_Type \*base, uint32\_t index)  
Запуск сдвоенного таймера
- enum [dualtimer\\_status](#) [DUALTIMER\\_Stop](#) (DTIM\_Type \*base, uint32\_t index)  
Останов сдвоенного таймера
- uint32\_t [DUALTIMER\\_GetRawStatus](#) (DTIM\_Type \*base, uint32\_t index)

- Получение немаскированного статуса сдвоенного таймера
- uint32\_t `DUALTIMER_GetStatus` (DTIM\_Type \*base, uint32\_t index)
- Получение маскированного статуса сдвоенного таймера
- uint32\_t `DUALTIMER_GetTick` (DTIM\_Type \*base, uint32\_t index)
- Получение количества тиков
- enum `dualtimer_status` `DUALTIMER_GetAPIStatus` ()
- Получение результата последнего выполнения функции
- enum `dualtimer_status` `DUALTIMER_Reload` (DTIM\_Type \*base, uint32\_t index, uint32\_t value)
- Немедленная перезапись значения таймера
- enum `dualtimer_status` `DUALTIMER_IrqClr` (DTIM\_Type \*base, uint32\_t index)
- Сброс прерывания от таймера

### 6.8.1 Подробное описание

Интерфейс драйвера сдвоенного таймера

## 6.9 hal\_dualtimer.h

[См. документацию.](#)

```

00001
00020 #ifndef HAL_DUALTIMER_H
00021 #define HAL_DUALTIMER_H
00022
00023 #if defined( __cplusplus )
00024 extern "C" {
00025 #endif /* __cplusplus */
00026
00027 #include "hal_common.h"
00028
00029 #define DUALTIMER_NUMBER_OF_DUALTIMERS 1
00030 #define DUALTIMER_MAX_INDEX 1
00035 enum dualtimer_status {
00036     DUALTIMER_Status_Ok = 0,
00037     DUALTIMER_Status_InvalidArgument = 1,
00038     DUALTIMER_Status_TimerBusy = 2,
00039     DUALTIMER_Status_BadConfigure = 3,
00040 };
00041
00045 enum dualtimer_work_enable {
00046     DUALTIMER_Disable = 0,
00047     DUALTIMER_Enable = 1,
00048 };
00049
00053 enum dualtimer_mode {
00054     DUALTIMER_FreeRunning = 0,
00055     DUALTIMER_Periodic = 1,
00056 };
00057
00061 enum dualtimer_interrupt_control {
00062     DUALTIMER_InterruptDisable = 0,
00063     DUALTIMER_InterruptEnable = 1,
00064 };
00065
00069 enum dualtimer_prescale {
00070     DUALTIMER_Prescale1 = 0,
00071     DUALTIMER_Prescale16 = 1,
00072     DUALTIMER_Prescale256 = 2,
00073 };
00074
00078 enum dualtimer_timer_size {
00079     DUALTIMER_TimerSize16 = 0,
00080     DUALTIMER_TimerSize32 = 1,
00081 };
00082
00086 enum dualtimer_number_of_repetitions {
00087     DUALTIMER_WrappingMode = 0,
00088     DUALTIMER_OneShot = 1,
00089 };
00090
00094 struct dualtimer_hardware_config {

```

```

00095     uint32_t                load;
00096     uint32_t                bg_load;
00097     enum dualtimer_work_enable enable;
00098     enum dualtimer_mode      mode;
00099     enum dualtimer_interrupt_control int_ctrl;
00100     enum dualtimer_prescale   prescale;
00101     enum dualtimer_timer_size size;
00102     enum dualtimer_number_of_repetitions cyclicity;
00103 };
00104
00123 enum dualtimer_status DUALTIMER_GetDefaultConfig(
00124     struct dualtimer_hardware_config *config);
00125
00137 enum dualtimer_status DUALTIMER_Init(DTIM_Type *base, uint32_t index,
00138     struct dualtimer_hardware_config config);
00139
00151 enum dualtimer_status DUALTIMER_Deinit(DTIM_Type *base, uint32_t index);
00152
00162 enum dualtimer_status DUALTIMER_Run(DTIM_Type *base, uint32_t index);
00163
00175 enum dualtimer_status DUALTIMER_Stop(DTIM_Type *base, uint32_t index);
00176
00190 uint32_t DUALTIMER_GetRawStatus(DTIM_Type *base, uint32_t index);
00191
00204 uint32_t DUALTIMER_GetStatus(DTIM_Type *base, uint32_t index);
00205
00214 uint32_t DUALTIMER_GetTick(DTIM_Type* base, uint32_t index);
00215
00227 enum dualtimer_status DUALTIMER_GetAPIStatus();
00228
00242 enum dualtimer_status DUALTIMER_Reload(DTIM_Type* base,
00243     uint32_t index, uint32_t value);
00244
00256 enum dualtimer_status DUALTIMER_IrqClr(DTIM_Type* base, uint32_t index);
00257
00262 #if defined(__cplusplus)
00263 }
00264 #endif /* __cplusplus */
00265
00266 #endif /* HAL_DUALTIMER_H */
00267

```

## 6.10 Файл devices/eliot1/drivers/hal\_flash.h

Интерфейс драйвера модуля FLASH.

```
#include "hal_common.h"
```

Макросы

- `#define FCTR_IRQ_STS_SET_RESULT_FLAGS`
- `#define FCTR_IRQ_STS_CLR_SUCCESS_FLAGS`

Команды для накристалльной FLASH-памяти.

- `#define FCTR_CMD_READ (0x1)`
- `#define FCTR_CMD_WRITE (0x2)`
- `#define FCTR_CMD_ROW_WRITE (0x3)`
- `#define FCTR_CMD_ERASE (0x4)`
- `#define FCTR_CMD_MASS_ERASE (0x7)`

Перечисления

- `enum flash_status`  
Статусы драйвера модуля FLASH.
- `enum flash_region`  
Регионы накристалльной FLASH-памяти.



API HAL для драйвера модуля FLASH.

- `#define FLASH_TEST_ADDRESSES(address, length)`  
Проверка корректности адреса и количества записываемых/стираемых байтов.
- `enum flash_status FLASH_Init (FCTR_Type *base)`  
Инициализация накристалльной FLASH-памяти.
- `enum flash_status FLASH_WriteWord (FCTR_Type *base, uint32_t addr, uint32_t data)`  
Запись 32-битного слова во встроенную FLASH-память.
- `enum flash_status FLASH_Program (FCTR_Type *base, uint32_t *addr, uint32_t *src, uint32_t *length)`  
Запись данных во встроенную FLASH-память.
- `enum flash_status FLASH_VerifyProgram (uint32_t *addr, uint32_t length, uint32_t *expected_data, uint32_t *failed_address, uint32_t *failed_data)`  
Проверка корректности данных, записанных во внутреннюю FLASH-память.
- `enum flash_status FLASH_Erase (FCTR_Type *base, uint32_t *addr, uint32_t length)`  
Стирание сектора накристалльной FLASH-памяти.
- `enum flash_status FLASH_MassErase (FCTR_Type *base, enum flash_region region)`  
Стирание раздела накристалльной FLASH-памяти.
- `enum flash_status FLASH_VerifyErase (uint32_t *addr, uint32_t length)`  
Проверка корректности стирания данных накристалльной FLASH-памяти.
- `enum flash_status FLASH_Read (uint32_t *addr, uint32_t *dest, uint32_t length)`  
Чтение данных из накристалльной FLASH-памяти.

### 6.10.1 Подробное описание

Интерфейс драйвера модуля FLASH.

## 6.11 hal\_flash.h

[См. документацию.](#)

```
00001
00022 #ifndef HAL_FLASH_H
00023 #define HAL_FLASH_H
00024
00025 #include "hal_common.h"
00026
00027 #if defined(__cplusplus)
00028 extern "C" {
00029 #endif /* __cplusplus */
00030
00031 #define FCTR_IRQ_STS_SET_RESULT_FLAGS \
00032     (FCTR_IRQ_STATUS_SET_READ_OVERFLOW_IRQ_STS_SET_Msk \
00033      | FCTR_IRQ_STATUS_SET_CMD_REJECT_IRQ_STS_SET_Msk \
00034      | FCTR_IRQ_STATUS_SET_CMD_FAIL_IRQ_STS_SET_Msk \
00035      | FCTR_IRQ_STATUS_SET_CMD_SUCCESS_IRQ_STS_SET_Msk)
00037 #define FCTR_IRQ_STS_CLR_SUCCESS_FLAGS \
00038     (FCTR_IRQ_STATUS_CLR_CMD_SUCCESS_IRQ_STS_CLR_Msk \
00039      | FCTR_IRQ_STATUS_CLR_CMD_ACCEPT_IRQ_STS_CLR_Msk)
00045 #define FCTR_CMD_READ (0x1)
00046 #define FCTR_CMD_WRITE (0x2)
00047 #define FCTR_CMD_ROW_WRITE (0x3)
00048 #define FCTR_CMD_ERASE (0x4)
00049 #define FCTR_CMD_MASS_ERASE (0x7)
00057 enum flash_status {
00058     FLASH_Status_Ok = 0,
00059     FLASH_Status_InvalidArgument = 1,
00060     FLASH_Status_CheckError = 2,
00061     FLASH_Status_VerifyError = 3,
00062     FLASH_Status_ConfigureError = 4,
00063     FLASH_Status_AddressAlignmentError = 5,
00064     FLASH_Status_AddressOutOfRange = 6,
00065 };
```

```

00066
00070 enum flash_region {
00071     FLASH_MainRegion = FLASH_BASE_ADDR,
00072     FLASH_SystemRegion = FLASH_BASE_ADDR
00073     + FLASH_SYSTEM_REGION_OFFSET,
00074 };
00075
00093 #define FLASH_TEST_ADDRESSES(address, length) \
00094     ( \
00095         /* Проверка на выравнивание по слову. */ \
00096         ((address & FLASH_WORD_ALIGN_Msk) > 0x0U) \
00097         || ( \
00098             /* Проверка на выравнивание по странице. */ \
00099             ((length & FLASH_PAGE_ALIGN_Msk) == 0x0U) \
00100             && ( \
00101                 ( \
00102                     (((address + length) & \
00103                         FLASH_PAGE_ALIGN_Msk) > 0x0U) \
00104                     && ((address & FLASH_PAGE_ALIGN_Msk) == 0x0U) \
00105                 ) \
00106                 || ((address & FLASH_PAGE_ALIGN_Msk) > 0x0U) \
00107             ) \
00108         ) \
00109     ) ? FLASH_Status_AddressAlignmentError : \
00110     ( \
00111         ( \
00112             /* Проверка попадания в основной раздел. */ \
00113             (address >= FLASH_BASE_ADDR) \
00114             && ((address + length) \
00115                 <= FLASH_BASE_ADDR + FLASH_MAIN_REGION_SIZE_IN_BYTE) \
00116         ) ? FLASH_Status_Ok : \
00117         ( \
00118             ( \
00119                 /* Проверка попадания в системный раздел. */ \
00120                 (address >= FLASH_BASE_ADDR \
00121                     + FLASH_SYSTEM_REGION_OFFSET) \
00122                 && ((address + length) <= FLASH_BASE_ADDR \
00123                     + FLASH_SYSTEM_REGION_OFFSET \
00124                     + FLASH_SYSTEM_REGION_SIZE_IN_BYTE) \
00125                 ) ? FLASH_Status_Ok : FLASH_Status_AddressOutOfRange \
00126             ) \
00127         ) \
00128     )
00137 enum flash_status FLASH_Init(FCTR_Type *base);
00138
00155 enum flash_status FLASH_WriteWord(FCTR_Type *base, uint32_t addr,
00156     uint32_t data);
00157
00178 enum flash_status FLASH_Program(FCTR_Type *base, uint32_t *addr, uint32_t *src,
00179     uint32_t length);
00180
00204 enum flash_status FLASH_VerifyProgram(uint32_t *addr, uint32_t length,
00205     uint32_t *expected_data, uint32_t *failed_address, uint32_t *failed_data);
00206
00224 enum flash_status FLASH_Erase(FCTR_Type *base, uint32_t *addr,
00225     uint32_t length);
00226
00236 enum flash_status FLASH_MassErase(FCTR_Type *base, enum flash_region region);
00237
00256 enum flash_status FLASH_VerifyErase(uint32_t *addr, uint32_t length);
00257
00279 enum flash_status FLASH_Read(uint32_t *addr, uint32_t *dest, uint32_t length);
00280
00285 #if defined(__cplusplus)
00286 }
00287 #endif /* __cplusplus */
00288
00289 #endif /* HAL_FLASH_H */
00290

```

## 6.12 Файл devices/eliot1/drivers/hal\_gpio.h

```
#include "hal_common.h"
```

Макросы

- #define GPIO\_PORTA 0U

- `#define GPIO_PORTB 1U`
- `#define GPIO_PORTC 2U`
- `#define GPIO_PORTD 3U`
- `#define GPIO_PORTPIN(port, pin) (((port) & 0xF) << 4) | ((pin) & 0xF)`
- `#define GPIO_PORTPIN_GET_PIN_NUM(portpin) ((portpin) & 0xF)`
- `#define GPIO_PORTPIN_GET_MASK(portpin) (1 << GPIO_PORTPIN_GET_PIN_NUM(portpin))`
- `#define GPIO_PORTPIN_GET_PORT_NUM(portpin) (((portpin) >> 4) & 0xF)`

Secure прерывания порта GPIO

- `#define GPIO_SEC_INT_SEC_ACC_MASK 0x1`
- `#define GPIO_SEC_INT_PORT_NONSEC_MASK 0x2`

Перечисления

Режимы работы выводов GPIO

- `enum gpio_mode_t`  
Список режимов работы вывода GPIO.

- `enum gpio_pin_function_t`  
Список альтернативных функций IOCTR выводов устройств

Характеристики внешних выводов GPIO

- `enum gpio_pin_direction_t`  
Направление вывода GPIO (вход или выход)
- `enum gpio_pin_pupd_t`  
Настройки внутренних резистивных подтяжек на выводах для режимов Push-Pull и Open-Drain.
- `enum gpio_pin_max_current_t`  
Настройки максимального выходного тока вывода
- `enum gpio_int_type_t`  
Список типов прерываний входных выводов GPIO.

## Функции

HAL функции для работы с конкретным выводом GPIO по Secure адресам

- `gpio_mode_t GPIO_PinMode_Get (uint32_t pin)`  
Чтение режима работы вывода GPIO.
- `void GPIO_PinMode_HiZ (uint32_t pin)`  
Установка вывода в состояние Hi-Z.
- `void GPIO_PinMode_Function (uint32_t pin, gpio_pin_function_t func)`  
Установка вывода в режим альтернативной функции IOCTR.
- `void GPIO_PinMode_GPIO (uint32_t pin, gpio_pin_direction_t direction)`  
Установка вывода в режим GPIO.
- `void GPIO_PinSet_PUPD (uint32_t pin, gpio_pin_pupd_t pupd)`  
Настройка резистивной подтяжки вывода GPIO.
- `void GPIO_PinSet_MaxCurrent (uint32_t pin, gpio_pin_max_current_t current)`  
Настройка максимального выходного тока вывода GPIO.
- `void GPIO_PinSet_Schmitt (uint32_t pin, uint32_t value)`  
Включение триггера Шмидта на входном выводе GPIO.
- `void GPIO_PinSet_SpeedRaise (uint32_t pin, uint32_t value)`  
Настройка скорости нарастания сигнала на выводе GPIO.
- `void GPIO_PinWrite (uint32_t pin, uint32_t bit)`  
Установка логического уровня вывода GPIO.
- `void GPIO_PinToggle (uint32_t pin)`  
Инвертирование логического уровня вывода GPIO.
- `uint32_t GPIO_PinRead (uint32_t pin)`  
Чтение логического уровня вывода GPIO.
- `void GPIO_PinIRQ_Enable (uint32_t pin, gpio_int_type_t int_type)`  
Включение прерывания на входном выводе GPIO.
- `void GPIO_PinIRQ_Disable (uint32_t pin)`  
Отключение прерывания на входном выводе GPIO.
- `uint32_t GPIO_PinIRQ_GetStatus (uint32_t pin)`  
Чтение статуса прерывания на входном выводе GPIO.
- `void GPIO_PinIRQ_Clear (uint32_t pin)`  
Сброс прерывания на входном выводе GPIO.

HAL функции для работы с конкретным портом GPIO по маске выводов

- `int32_t GPIO_GetInstance (GPIO_Type *port)`  
Определение номера порта GPIO.
- `void GPIO_PortMode_HiZ (GPIO_Type *port, uint32_t bit_mask)`  
Установка выводов порта в состояние Hi-Z.
- `void GPIO_PortMode_Function (GPIO_Type *port, uint32_t bit_mask, gpio_pin_function_t func)`  
Установка выводов порта в режим альтернативной функции IOCTR.
- `void GPIO_PortMode_GPIO (GPIO_Type *port, uint32_t bit_mask, gpio_pin_direction_t direction)`  
Установка выводов порта в режим GPIO.
- `void GPIO_PortSet_PUPD (GPIO_Type *port, uint32_t bit_mask, gpio_pin_pupd_t pupd)`  
Настройка резистивной подтяжки выводов порта GPIO.
- `void GPIO_PortSet_MaxCurrent (GPIO_Type *port, uint32_t bit_mask, gpio_pin_max_current_t current)`  
Настройка максимального выходного тока выводов порта GPIO.
- `void GPIO_PortSet_Schmitt (GPIO_Type *port, uint32_t bit_mask, uint32_t bit_value)`  
Включение триггера Шмидта на входных выводах порта GPIO.
- `void GPIO_PortSet_SpeedRaise (GPIO_Type *port, uint32_t bit_mask, uint32_t bit_value)`  
Настройка скорости нарастания сигнала на выводах порта GPIO.
- `void GPIO_PortWrite (GPIO_Type *port, uint32_t bit_mask, uint32_t bit_value)`  
Установка логического уровня выводов порта GPIO.
- `void GPIO_PortToggle (GPIO_Type *port, uint32_t bit_mask)`

- Инвертирование логического уровня выводов порта GPIO.
- uint32\_t [GPIO\\_PortRead](#) (GPIO\_Type \*port, uint32\_t bit\_mask)  
Чтение логического уровня выводов порта GPIO.
- void [GPIO\\_PortIRQ\\_Enable](#) (GPIO\_Type \*port, uint32\_t bit\_mask, uint32\_t int\_type)  
Включение прерываний на входных выводах порта GPIO.
- void [GPIO\\_PortIRQ\\_Disable](#) (GPIO\_Type \*port, uint32\_t bit\_mask)  
Отключение прерываний на входных выводах порта GPIO.
- uint32\_t [GPIO\\_PortIRQ\\_GetStatus](#) (GPIO\_Type \*port, uint32\_t bit\_mask)  
Чтение статусов прерываний на входных выводах порта GPIO.
- void [GPIO\\_PortIRQ\\_Clear](#) (GPIO\_Type \*port, uint32\_t bit\_mask)  
Сброс прерываний на входных выводах порта GPIO.
- uint32\_t [GPIO\\_PortSecureIRQ\\_GetStatus](#) (GPIO\_Type \*port)  
Чтение статуса прерывания по нарушению безопасности порта GPIO.
- void [GPIO\\_PortSecureIRQ\\_Clear](#) (GPIO\_Type \*port)  
Сброс прерывания по нарушению безопасности порта GPIO.
- void [GPIO\\_PortSecureIRQ\\_SetMask](#) (GPIO\_Type \*port, uint32\_t mask)  
Установка разрешений прерываний по нарушению безопасности порта GPIO.
- uint32\_t [GPIO\\_PortSecureIRQ\\_GetInfo1](#) (GPIO\_Type \*port)  
Чтение информации Info1 о прерывании по нарушению безопасности порта GPIO.
- uint32\_t [GPIO\\_PortSecureIRQ\\_GetInfo2](#) (GPIO\_Type \*port)  
Чтение информации Info2 о прерывании по нарушению безопасности порта GPIO.
- void [GPIO\\_PortSet\\_NonsecureMask](#) (GPIO\_Type \*port, uint32\_t mask)  
Установка разрешения для Non-secure обращений к порту GPIO.

## 6.12.1 Макросы

### 6.12.1.1 GPIO\_SEC\_INT\_PORT\_NONSEC\_MASK

```
#define GPIO_SEC_INT_PORT_NONSEC_MASK 0x2
```

Маска прерывания PORT\_NONSEC

### 6.12.1.2 GPIO\_SEC\_INT\_SEC\_ACC\_MASK

```
#define GPIO_SEC_INT_SEC_ACC_MASK 0x1
```

Маска прерывания SEC\_ACC

## 6.12.2 Перечисления

### 6.12.2.1 gpio\_int\_type\_t

```
enum gpio\_int\_type\_t
```

Список типов прерываний входных выводов GPIO.

Элементы перечислений

GPIO_INT_LVL_LO	Прерывание по низкому логическому уровню на входном выводе
GPIO_INT_LVL_HI	Прерывание по высокому логическому уровню на входном выводе
GPIO_INT_EDGE_FALL	Прерывание по спадающему фронту логического уровня на входном выводе
GPIO_INT_EDGE_RISE	Прерывание по восходящему фронту логического уровня на входном выводе

### 6.12.2.2 `gpio_pin_direction_t`

enum `gpio_pin_direction_t`

Направление вывода GPIO (вход или выход)

Элементы перечислений

GPIO_DigitalInput	Направление работы вывода - вход
GPIO_DigitalOutput	Направление работы вывода - выход

### 6.12.2.3 gpio\_pin\_max\_current\_t

enum [gpio\\_pin\\_max\\_current\\_t](#)

Настройки максимального выходного тока вывода

Элементы перечислений

GPIO_MAX_CURRENT_2mA	Максимальный выходной ток вывода 2mA
GPIO_MAX_CURRENT_4mA	Максимальный выходной ток вывода 4mA
GPIO_MAX_CURRENT_8mA	Максимальный выходной ток вывода 8mA
GPIO_MAX_CURRENT_12mA	Максимальный выходной ток вывода 12mA

### 6.12.2.4 gpio\_pin\_pupd\_t

enum [gpio\\_pin\\_pupd\\_t](#)

Настройки внутренних резистивных подтяжек на выводах для режимов Push-Pull и Open-Drain.

Элементы перечислений

GPIO_PULL_NONE	Без резистивных подтяжек на выводе
GPIO_PULL_DOWN	Резистивная подтяжка на минус питания на выводе
GPIO_PULL_UP	Резистивная подтяжка на плюс питания на выводе
GPIO_PULL_NONE_OD	Без резистивных подтяжек и режим Open-Drain на выводе
GPIO_PULL_DOWN_OD	Резистивная подтяжка на минус питания и режим Open-Drain на выводе
GPIO_PULL_UP_OD	Резистивная подтяжка на плюс питания и режим Open-Drain на выводе

## 6.12.3 Функции

### 6.12.3.1 GPIO\_GetInstance()

```
int32_t GPIO_GetInstance (
    GPIO_Type * port)
```

Определение номера порта GPIO.

## Аргументы

port	Базовый адрес порта GPIO
------	--------------------------

## Возвращаемые значения

-1	Порт GPIO не определен
0	Порт GPIOA
1	Порт GPIOB
2	Порт GPIOC
3	Порт GPIOD

## 6.12.3.2 GPIO\_PinIRQ\_Clear()

```
void GPIO_PinIRQ_Clear (  
    uint32_t pin)
```

Сброс прерывания на входном выводе GPIO.

## Аргументы

pin	Номер вывода из карты выводов GPIO
-----	------------------------------------

## 6.12.3.3 GPIO\_PinIRQ\_Disable()

```
void GPIO_PinIRQ_Disable (  
    uint32_t pin)
```

Отключение прерывания на входном выводе GPIO.

## Аргументы

pin	Номер вывода из карты выводов GPIO
-----	------------------------------------

## 6.12.3.4 GPIO\_PinIRQ\_Enable()

```
void GPIO_PinIRQ_Enable (  
    uint32_t pin,  
    gpio_int_type_t int_type)
```

Включение прерывания на входном выводе GPIO.

## Аргументы

pin	Номер вывода из карты выводов GPIO
int_type	Тип прерывания

## 6.12.3.5 GPIO\_PinIRQ\_GetStatus()

```
uint32_t GPIO_PinIRQ_GetStatus (  
    uint32_t pin)
```

Чтение статуса прерывания на входном выводе GPIO.



## Аргументы

pin	Номер вывода из карты выводов GPIO
-----	------------------------------------

## Возвращаемые значения

0	- прерывание отсутствует
1	- прерывание присутствует

## 6.12.3.6 GPIO\_PinMode\_Function()

```
void GPIO_PinMode_Function (
    uint32_t pin,
    gpio_pin_function_t func)
```

Установка вывода в режим альтернативной функции IOCTR.

## Аргументы

pin	Номер вывода из карты выводов GPIO
func	Номер альтернативной функции вывода

## 6.12.3.7 GPIO\_PinMode\_Get()

```
gpio_mode_t GPIO_PinMode_Get (
    uint32_t pin)
```

Чтение режима работы вывода GPIO.

## Аргументы

pin	Номер вывода из карты выводов GPIO
-----	------------------------------------

## Возвращаемые значения

GPIO_MODE_HI_Z	
GPIO_MODE_GPIO	
GPIO_MODE_AF	
GPIO_MODE_INVALID	

## 6.12.3.8 GPIO\_PinMode\_GPIO()

```
void GPIO_PinMode_GPIO (
    uint32_t pin,
    gpio_pin_direction_t direction)
```

Установка вывода в режим GPIO.

## Аргументы

pin	Номер вывода из карты выводов GPIO
direction	Направление вывода

## 6.12.3.9 GPIO\_PinMode\_HiZ()

```
void GPIO_PinMode_HiZ (  
    uint32_t pin)
```

Установка вывода в состояние Hi-Z.

## Аргументы

pin	Номер вывода из карты выводов GPIO
-----	------------------------------------

## 6.12.3.10 GPIO\_PinRead()

```
uint32_t GPIO_PinRead (  
    uint32_t pin)
```

Чтение логического уровня вывода GPIO.

## Аргументы

pin	Номер вывода из карты выводов GPIO
-----	------------------------------------

## Возвращаемые значения

0	- низкий логический уровень
1	- высокий логический уровень

## 6.12.3.11 GPIO\_PinSet\_MaxCurrent()

```
void GPIO_PinSet_MaxCurrent (  
    uint32_t pin,  
    gpio_pin_max_current_t current)
```

Настройка максимального выходного тока вывода GPIO.

## Аргументы

pin	Номер вывода из карты выводов GPIO
current	Номер максимального тока из списка доступных значений

## 6.12.3.12 GPIO\_PinSet\_PUPD()

```
void GPIO_PinSet_PUPD (  
    uint32_t pin,  
    gpio_pin_pupd_t pupd)
```

Настройка резистивной подтяжки вывода GPIO.

Аргументы

pin	Номер вывода из карты выводов GPIO
pull	Тип внешней подтяжки вывода

#### 6.12.3.13 GPIO\_PinSet\_Schmitt()

```
void GPIO_PinSet_Schmitt (  
    uint32_t pin,  
    uint32_t value)
```

Включение триггера Шмидта на входном выводе GPIO.

Аргументы

pin	Номер вывода из карты выводов GPIO
value	Значения: 0 - отключен, 1 - включен

#### 6.12.3.14 GPIO\_PinSet\_SpeedRaise()

```
void GPIO_PinSet_SpeedRaise (  
    uint32_t pin,  
    uint32_t value)
```

Настройка скорости нарастания сигнала на выводе GPIO.

Аргументы

pin	Номер вывода из карты выводов GPIO
value	Значения: 0 - fast, 1 - slow

#### 6.12.3.15 GPIO\_PinToggle()

```
void GPIO_PinToggle (  
    uint32_t pin)
```

Инвертирование логического уровня вывода GPIO.

Аргументы

pin	Номер вывода из карты выводов GPIO
-----	------------------------------------

#### 6.12.3.16 GPIO\_PinWrite()

```
void GPIO_PinWrite (  
    uint32_t pin,  
    uint32_t bit)
```

Установка логического уровня вывода GPIO.

## Аргументы

pin	Номер вывода из карты выводов GPIO
bit	Значения: 0 - низкий логический уровень, 1 - высокий логический уровень

## 6.12.3.17 GPIO\_PortIRQ\_Clear()

```
void GPIO_PortIRQ_Clear (
    GPIO_Type * port,
    uint32_t bit_mask)
```

Сброс прерываний на входных выводах порта GPIO.

## Аргументы

port	Базовый адрес порта GPIO
bit_mask	Битовая маска выбранных выводов

## 6.12.3.18 GPIO\_PortIRQ\_Disable()

```
void GPIO_PortIRQ_Disable (
    GPIO_Type * port,
    uint32_t bit_mask)
```

Отключение прерываний на входных выводах порта GPIO.

## Аргументы

port	Базовый адрес порта GPIO
bit_mask	Битовая маска выбранных выводов

## 6.12.3.19 GPIO\_PortIRQ\_Enable()

```
void GPIO_PortIRQ_Enable (
    GPIO_Type * port,
    uint32_t bit_mask,
    uint32_t int_type)
```

Включение прерываний на входных выводах порта GPIO.

## Аргументы

port	Базовый адрес порта GPIO
bit_mask	Битовая маска выбранных выводов
int_type	Тип прерывания из списка

## 6.12.3.20 GPIO\_PortIRQ\_GetStatus()

```
uint32_t GPIO_PortIRQ_GetStatus (
    GPIO_Type * port,
    uint32_t bit_mask)
```

Чтение статусов прерываний на входных выводах порта GPIO.

## Аргументы

port	Базовый адрес порта GPIO
bit_mask	Битовая маска выбранных выводов

## Возвращает

32-битная маска, каждый бит которой означает либо прерывание отсутствует (0), либо прерывание присутствует (1)

## 6.12.3.21 GPIO\_PortMode\_Function()

```
void GPIO_PortMode_Function (
    GPIO_Type * port,
    uint32_t bit_mask,
    gpio_pin_function_t func)
```

Установка выводов порта в режим альтернативной функции ЮСТР.

## Аргументы

port	Базовый адрес порта GPIO
bit_mask	Битовая маска выбранных выводов
func	Номер альтернативной функции вывода

## 6.12.3.22 GPIO\_PortMode\_GPIO()

```
void GPIO_PortMode_GPIO (
    GPIO_Type * port,
    uint32_t bit_mask,
    gpio_pin_direction_t direction)
```

Установка выводов порта в режим GPIO.

## Аргументы

port	Базовый адрес порта GPIO
bit_mask	Битовая маска выбранных выводов
direction	Направление вывода

## 6.12.3.23 GPIO\_PortMode\_HiZ()

```
void GPIO_PortMode_HiZ (
    GPIO_Type * port,
    uint32_t bit_mask)
```

Установка выводов порта в состояние Hi-Z.

## Аргументы

port	Базовый адрес порта GPIO
bit_mask	Битовая маска выбранных выводов

## 6.12.3.24 GPIO\_PortRead()

```
uint32_t GPIO_PortRead (
    GPIO_Type * port,
    uint32_t bit_mask)
```

Чтение логического уровня выводов порта GPIO.

## Аргументы

port	Базовый адрес порта GPIO
bit_mask	Битовая маска выбранных выводов

## Возвращает

32-битная маска, каждый бит которой означает либо низкий логический уровень вывода (0), либо высокий логический уровень (1)

## 6.12.3.25 GPIO\_PortSecureIRQ\_Clear()

```
void GPIO_PortSecureIRQ_Clear (
    GPIO_Type * port)
```

Сброс прерывания по нарушению безопасности порта GPIO.

## Аргументы

port	Базовый адрес порта GPIO
------	--------------------------

## 6.12.3.26 GPIO\_PortSecureIRQ\_GetInfo1()

```
uint32_t GPIO_PortSecureIRQ_GetInfo1 (
    GPIO_Type * port)
```

Чтение информации Info1 о прерывании по нарушению безопасности порта GPIO.

## Аргументы

port	Базовый адрес порта GPIO
------	--------------------------

## Возвращает

Данные информации размером 32 бита

## 6.12.3.27 GPIO\_PortSecureIRQ\_GetInfo2()

```
uint32_t GPIO_PortSecureIRQ_GetInfo2 (
    GPIO_Type * port)
```

Чтение информации Info2 о прерывании по нарушению безопасности порта GPIO.

Аргументы

port	Базовый адрес порта GPIO
------	--------------------------

Возвращает

Данные информации размером 32 бита

#### 6.12.3.28 GPIO\_PortSecureIRQ\_GetStatus()

```
uint32_t GPIO_PortSecureIRQ_GetStatus (
    GPIO_Type * port)
```

Чтение статуса прерывания по нарушению безопасности порта GPIO.

Аргументы

port	Базовый адрес порта GPIO
------	--------------------------

Возвращаемые значения

0	- прерывание отсутствует
1	- прерывание присутствует

#### 6.12.3.29 GPIO\_PortSecureIRQ\_SetMask()

```
void GPIO_PortSecureIRQ_SetMask (
    GPIO_Type * port,
    uint32_t mask)
```

Установка разрешений прерываний по нарушению безопасности порта GPIO.

Аргументы

port	Базовый адрес порта GPIO
mask	Маска разрешения прерываний (SEC_ACC_MASK и PORT_NONSEC_MASK)

#### 6.12.3.30 GPIO\_PortSet\_MaxCurrent()

```
void GPIO_PortSet_MaxCurrent (
    GPIO_Type * port,
    uint32_t bit_mask,
    gpio_pin_max_current_t current)
```

Настройка максимального выходного тока выводов порта GPIO.

## Аргументы

port	Базовый адрес порта GPIO
bit_mask	Битовая маска выбранных выводов
current	Номер максимального тока из списка доступных значений

## 6.12.3.31 GPIO\_PortSet\_NonsecureMask()

```
void GPIO_PortSet_NonsecureMask (
    GPIO_Type * port,
    uint32_t mask)
```

Установка разрешения для Non-secure обращений к порту GPIO.

## Аргументы

port	Базовый адрес порта GPIO
mask	Маска битов разрешения доступа (0 - secure, 1 - non-secure)

## 6.12.3.32 GPIO\_PortSet\_PUPD()

```
void GPIO_PortSet_PUPD (
    GPIO_Type * port,
    uint32_t bit_mask,
    gpio_pin_pupd_t pupd)
```

Настройка резистивной подтяжки выводов порта GPIO.

## Аргументы

port	Базовый адрес порта GPIO
bit_mask	Битовая маска выбранных выводов
pupd	Тип внешней подтяжки вывода

## 6.12.3.33 GPIO\_PortSet\_Schmitt()

```
void GPIO_PortSet_Schmitt (
    GPIO_Type * port,
    uint32_t bit_mask,
    uint32_t bit_value)
```

Включение триггера Шмидта на входных выводах порта GPIO.

## Аргументы

port	Базовый адрес порта GPIO
bit_mask	Битовая маска выбранных выводов
bit_value	Значения по маске битов: 0 - отключен, 1 - включен



## 6.12.3.34 GPIO\_PortSet\_SpeedRaise()

```
void GPIO_PortSet_SpeedRaise (
    GPIO_Type * port,
    uint32_t bit_mask,
    uint32_t bit_value)
```

Настройка скорости нарастания сигнала на выводах порта GPIO.

Аргументы

port	Базовый адрес порта GPIO
bit_mask	Битовая маска выбранных выводов
bit_value	Значения по маске битов: 0 - fast, 1 - slow

## 6.12.3.35 GPIO\_PortToggle()

```
void GPIO_PortToggle (
    GPIO_Type * port,
    uint32_t bit_mask)
```

Инвертирование логического уровня выводов порта GPIO.

Аргументы

port	Базовый адрес порта GPIO
bit_mask	Битовая маска выбранных выводов

## 6.12.3.36 GPIO\_PortWrite()

```
void GPIO_PortWrite (
    GPIO_Type * port,
    uint32_t bit_mask,
    uint32_t bit_value)
```

Установка логического уровня выводов порта GPIO.

Аргументы

port	Базовый адрес порта GPIO
bit_mask	Битовая маска выбранных выводов
bit_value	Значения по маске битов: 0 - низкий логический уровень, 1 - высокий логический уровень

## 6.13 hal\_gpio.h

См. документацию.

```

00001
00017 #ifndef HAL_GPIO_H
00018 #define HAL_GPIO_H
00019
00020 #ifdef __cplusplus
00021 extern "C" {
00022 #endif
00023
00024 #include "hal_common.h"
00025
00026 #ifndef GPIO_PORTA
00027 #define GPIO_PORTA          0U
00028 #endif
00029 #ifndef GPIO_PORTB
00030 #define GPIO_PORTB          1U
00031 #endif
00032 #ifndef GPIO_PORTC
00033 #define GPIO_PORTC          2U
00034 #endif
00035 #ifndef GPIO_PORTD
00036 #define GPIO_PORTD          3U
00037 #endif
00038
00039 #ifndef GPIO_PORTPIN
00040 #define GPIO_PORTPIN(port, pin) \
00041     (((port) & 0xF) << 4) | ((pin) & 0xF))
00042 #endif
00043 #ifndef GPIO_PORTPIN_GET_PIN_NUM
00044 #define GPIO_PORTPIN_GET_PIN_NUM(portpin) ((portpin) & 0xF)
00045 #endif
00046 #ifndef GPIO_PORTPIN_GET_MASK
00047 #define GPIO_PORTPIN_GET_MASK(portpin) \
00048     (1 << GPIO_PORTPIN_GET_PIN_NUM(portpin))
00049 #endif
00050 #ifndef GPIO_PORTPIN_GET_PORT_NUM
00051 #define GPIO_PORTPIN_GET_PORT_NUM(portpin) (((portpin) >> 4) & 0xF)
00052 #endif
00053
00062 typedef enum {
00063     GPIO_MODE_HI_Z    = 0x0,
00064     GPIO_MODE_GPIO    = 0x1,
00065     GPIO_MODE_AF      = 0x2,
00066     GPIO_MODE_INVALID = 0x3,
00067 } gpio_mode_t;
00074 typedef enum {
00075     GPIO_ALT_FUNC_TRACE_JTAG_FBIST = 0,
00076     GPIO_ALT_FUNC_PWM_VTU          = 1,
00077     GPIO_ALT_FUNC_I2C_I2S          = 2,
00078     GPIO_ALT_FUNC_SPI0_SPI1        = 3,
00079     GPIO_ALT_FUNC_UART             = 4,
00080     GPIO_ALT_FUNC_CAN_GNSS_USB     = 5,
00081     GPIO_ALT_FUNC_QSPI_SPI2        = 6,
00082     GPIO_ALT_FUNC_SDMMC_SMC        = 7,
00083 } gpio_pin_function_t;
00097 typedef enum {
00098     GPIO_DigitalInput = 0U,
00099     GPIO_DigitalOutput = 1U,
00100 } gpio_pin_direction_t;
00107 typedef enum {
00108     GPIO_PULL_NONE = 0,
00109     GPIO_PULL_DOWN = 1,
00110     GPIO_PULL_UP   = 3,
00111     GPIO_PULL_NONE_OD = 4,
00112     GPIO_PULL_DOWN_OD = 5,
00113     GPIO_PULL_UP_OD   = 7,
00114 } gpio_pin_pupd_t;
00121 typedef enum {
00122     GPIO_MAX_CURRENT_2mA = 0,
00123     GPIO_MAX_CURRENT_4mA = 1,
00124     GPIO_MAX_CURRENT_8mA = 2,
00125     GPIO_MAX_CURRENT_12mA = 3,
00126 } gpio_pin_max_current_t;
00133 typedef enum {
00134     GPIO_INT_LVL_LO = 0,
00135     GPIO_INT_LVL_HI = 1,
00136     GPIO_INT_EDGE_FALL = 2,
00137     GPIO_INT_EDGE_RISE = 3,
00138 } gpio_int_type_t;
00145 #define GPIO_SEC_INT_SEC_ACC_MASK 0x1
00146 #define GPIO_SEC_INT_PORT_NONSEC_MASK 0x2
00166 gpio_mode_t GPIO_PinMode_Get(uint32_t pin);
00167

```

```

00173 void GPIO_PinMode_HiZ(uint32_t pin);
00174
00181 void GPIO_PinMode_Function(uint32_t pin, gpio_pin_function_t func);
00182
00189 void GPIO_PinMode_GPIO(uint32_t pin, gpio_pin_direction_t direction);
00190
00197 void GPIO_PinSet_PUPD(uint32_t pin, gpio_pin_pupd_t pupd);
00198
00205 void GPIO_PinSet_MaxCurrent(uint32_t pin, gpio_pin_max_current_t current);
00206
00213 void GPIO_PinSet_Schmitt(uint32_t pin, uint32_t value);
00214
00221 void GPIO_PinSet_SpeedRaise(uint32_t pin, uint32_t value);
00222
00229 void GPIO_PinWrite(uint32_t pin, uint32_t bit);
00230
00236 void GPIO_PinToggle(uint32_t pin);
00237
00246 uint32_t GPIO_PinRead(uint32_t pin);
00247
00254 void GPIO_PinIRQ_Enable(uint32_t pin, gpio_int_type_t int_type);
00255
00261 void GPIO_PinIRQ_Disable(uint32_t pin);
00262
00271 uint32_t GPIO_PinIRQ_GetStatus(uint32_t pin);
00272
00278 void GPIO_PinIRQ_Clear(uint32_t pin);
00297 int32_t GPIO_GetInstance(GPIO_Type *port);
00298
00305 void GPIO_PortMode_HiZ(GPIO_Type *port, uint32_t bit_mask);
00306
00314 void GPIO_PortMode_Function(GPIO_Type *port, uint32_t bit_mask, gpio_pin_function_t func);
00315
00323 void GPIO_PortMode_GPIO(GPIO_Type *port, uint32_t bit_mask, gpio_pin_direction_t direction);
00324
00332 void GPIO_PortSet_PUPD(GPIO_Type *port, uint32_t bit_mask, gpio_pin_pupd_t pupd);
00333
00341 void GPIO_PortSet_MaxCurrent(GPIO_Type *port, uint32_t bit_mask, gpio_pin_max_current_t current);
00342
00350 void GPIO_PortSet_Schmitt(GPIO_Type *port, uint32_t bit_mask, uint32_t bit_value);
00351
00359 void GPIO_PortSet_SpeedRaise(GPIO_Type *port, uint32_t bit_mask, uint32_t bit_value);
00360
00368 void GPIO_PortWrite(GPIO_Type *port, uint32_t bit_mask, uint32_t bit_value);
00369
00376 void GPIO_PortToggle(GPIO_Type *port, uint32_t bit_mask);
00385 uint32_t GPIO_PortRead(GPIO_Type *port, uint32_t bit_mask);
00386
00394 void GPIO_PortIRQ_Enable(GPIO_Type *port, uint32_t bit_mask, uint32_t int_type);
00395
00402 void GPIO_PortIRQ_Disable(GPIO_Type *port, uint32_t bit_mask);
00403
00412 uint32_t GPIO_PortIRQ_GetStatus(GPIO_Type *port, uint32_t bit_mask);
00413
00420 void GPIO_PortIRQ_Clear(GPIO_Type *port, uint32_t bit_mask);
00421
00430 uint32_t GPIO_PortSecureIRQ_GetStatus(GPIO_Type *port);
00431
00437 void GPIO_PortSecureIRQ_Clear(GPIO_Type *port);
00438
00445 void GPIO_PortSecureIRQ_SetMask(GPIO_Type *port, uint32_t mask);
00446
00454 uint32_t GPIO_PortSecureIRQ_GetInfo1(GPIO_Type *port);
00455
00463 uint32_t GPIO_PortSecureIRQ_GetInfo2(GPIO_Type *port);
00464
00471 void GPIO_PortSet_NonsecureMask(GPIO_Type *port, uint32_t mask);
00474 #ifdef __cplusplus
00475 }
00476 #endif
00477
00478 #endif /* HAL_GPIO_H */
00479

```

## 6.14 Файл devices/eliot1/drivers/hal\_i2c.h

Интерфейс драйвера модуля I2C.

```
#include "hal_common.h"
```

## Структуры данных

- `struct i2c_master_config_t`  
Структура с настройками для инициализации Master-модуля I2C.
- `struct i2c_master_transfer_t`  
Структура дескриптора для неблокирующего обмена.
- `struct _i2c_master_handle`  
Дескриптор для работы по прерыванию.
- `struct i2c_slave_config_t`  
Структура с настройками для инициализации Slave-модуля I2C.
- `struct i2c_slave_transfer_t`  
I2C Slave структура обмена данными
- `struct _i2c_slave_handle`  
I2C Slave-дескриптор

## Макросы

- `#define I2C_CFG_MASK 0x1f`
- `#define HAL_I2C_DRIVER_VERSION (MAKE_VERSION(1, 0, 0))`  
Версия драйвера.
- `#define I2C_RETRY_TIMES 0U`  
Количество повторов при ожидании флага.
- `#define I2C_MASTER_TRANSMIT_IGNORE_LAST_NACK 0U`  
Игнорировать ли сигнал NACK последнего байта во время передачи Master.
- `#define I2C_STAT_MSTCODE_IDLE (0U)`
- `#define I2C_STAT_MSTCODE_RXREADY (1U)`
- `#define I2C_STAT_MSTCODE_TXREADY (2U)`
- `#define I2C_STAT_MSTCODE_NACKADR (3U)`
- `#define I2C_STAT_MSTCODE_NACKDAT (4U)`
- `#define I2C_STAT_SLVST_ADDR (0)`
- `#define I2C_STAT_SLVST_RX (1)`
- `#define I2C_STAT_SLVST_TX (2)`

## Определения типов

- `typedef struct _i2c_master_handle i2c_master_handle_t`  
Тип I2C Master дескриптор
- `typedef void(* i2c_master_transfer_callback_t) (I2C_Type *base, i2c_master_handle_t *handle, i2c_status_t completion_status, void *user_data)`  
Указатель на функцию обратного вызова при завершении Master-передачи.
- `typedef struct _i2c_slave_handle i2c_slave_handle_t`  
Тип I2C Slave дескриптор
- `typedef void(* i2c_slave_transfer_callback_t) (I2C_Type *base, volatile i2c_slave_transfer_t *transfer, void *user_data)`  
Slave тип функции обратного вызова.

## Перечисления

- enum `i2c_status_t`  
Коды возврата функций модуля I2C.
- enum `i2c_status_flags`  
Статусы модуля I2C (read-only регистр `IC_STATUS`).
- enum `i2c_abort_flags`  
Причины обрыва передачи (регистр `IC_TX_ABRT_SOURCE`).
- enum `i2c_interrupt`  
Флаги прерываний I2C.
- enum `i2c_direction_t`  
Направление передачи.
- enum `i2c_addr_size_t`  
Разрядность адреса.
- enum `i2c_master_transfer_states`  
Состояния для конечного автомата, используемого при обмене. FSM TODO.
- enum `i2c_speed_mode_t`  
Скоростной режим работы модуля в Master-режиме.
- enum `i2c_master_transfer_flags_t`  
Флаги вариантов передачи.
- enum `i2c_slave_event_transfer_t`  
События, вызывающие функцию обратного вызова для неблокирующих передач.
- enum `i2c_slave_fsm_t`  
I2C Slave состояния конечного автомата

## Функции

Инициализация и деинициализация.

- static bool `I2C_IsEnable` (`I2C_Type *base`)  
Возврат состояния модуля I2C.
- `i2c_status_t` `I2C_Enable` (`I2C_Type *base`, bool enable)  
Включение и отключение модуля I2C.
- uint32\_t `I2C_GetInstance` (`I2C_Type *base`)  
Возврат номера экземпляра по базовому адресу.
- `i2c_status_t` `I2C_Reset` (`I2C_Type *base`)  
Сброс модуля I2C.
- void `I2C_MasterGetDefaultConfig` (`i2c_master_config_t *master_config`)  
Получение конфигурации по умолчанию для Master-режима модуля I2C.
- `i2c_status_t` `I2C_MasterInit` (`I2C_Type *base`, const `i2c_master_config_t *master_config`, uint32\_t src\_clock\_hz)  
Инициализация Master-устройства I2C.
- `i2c_status_t` `I2C_MasterDeinit` (`I2C_Type *base`)  
Деинициализация Master-устройства I2C.

Прерывания.

- static void `I2C_EnableInterrupts` (`I2C_Type *base`, uint32\_t mask)  
Включение запросов на прерывание.
- static void `I2C_DisableInterrupts` (`I2C_Type *base`, uint32\_t mask)  
Отключение запросов на прерывание.
- static uint32\_t `I2C_GetEnabledInterrupts` (`I2C_Type *base`)  
Возврат набора текущих разрешенных запросов на прерывание.
- static void `I2C_ClearInterrupt` (`I2C_Type *base`, enum `i2c_interrupt` irq)

- Очищение флагов прерываний.
- `static void I2C_ClearAllInterrupts (I2C_Type *base)`  
Очищение всех доступных для очистки флагов прерываний.

#### Операции на шине

- `i2c_status_t I2C_MasterSetBaudRate (I2C_Type *base, uint32_t baudrate_bps, uint32_t src_clock_hz)`  
Установка частоты шины для Master-режима модуля I2C.
- `i2c_status_t I2C_MasterAddrSet (I2C_Type *base, uint32_t address, i2c_addr_size_t addr_size)`  
Установка адреса Slave-устройства на шине I2C, к которому будет обращение в цикле обмена.
- `static bool I2C_MasterGetBusActiveState (I2C_Type *base)`  
Возврат статуса активности шины.
- `i2c_status_t I2C_MasterWriteBlocking (I2C_Type *base, const void *tx_buff, size_t tx_size, uint32_t flags)`  
Выполнение блокирующей передачи по шине I2C.
- `i2c_status_t I2C_MasterReadBlocking (I2C_Type *base, void *rx_buff, size_t rx_size, uint32_t flags)`  
Выполнение блокирующего приема на шине I2C.
- `i2c_status_t I2C_MasterTransferBlocking (I2C_Type *base, i2c_master_transfer_t *xfer)`  
Выполнение обмена данными на шине I2C в режиме блокировки.

#### Неблокирующий обмен (по прерыванию).

- `void I2C_MasterTransferCreateHandle (I2C_Type *base, i2c_master_handle_t *handle, i2c_master_transfer_callback_t callback, void *user_data)`  
Создание нового дескриптора для неблокирующего Master-режима работы.
- `i2c_status_t I2C_MasterTransferNonBlocking (I2C_Type *base, i2c_master_handle_t *handle, i2c_master_transfer_t *xfer)`  
Выполнение неблокирующей транзакции на шине I2C.
- `i2c_status_t I2C_MasterTransferGetCount (I2C_Type *base, i2c_master_handle_t *handle, size_t *count)`  
Получение количества байтов, переданных неблокирующей транзакцией на данный момент.
- `i2c_status_t I2C_MasterTransferAbort (I2C_Type *base, i2c_master_handle_t *handle)`  
Досрочное завершение основной неблокирующей передачи I2C.

#### Обработчик запросов на прерывание для Master-режима.

- `void I2C_MasterTransferHandleIRQ (I2C_Type *base, i2c_master_handle_t *handle)`  
Обработчик запросов на прерывание для Master-режима.

#### Инициализация и деинициализация в Slave-режиме.

- `void I2C_SlaveGetDefaultConfig (i2c_slave_config_t *config)`  
Получение конфигурации по умолчанию для Slave-режима модуля I2C.
- `i2c_status_t I2C_SlaveInit (I2C_Type *base, const i2c_slave_config_t *config)`  
Инициализация Slave-модуля I2C.
- `i2c_status_t I2C_SlaveSetAddress (I2C_Type *base, uint16_t address, bool ack_gen_call)`  
Установка адреса Slave-устройства.
- `static i2c_status_t I2C_SlaveDeinit (I2C_Type *base)`  
Деинициализация периферийного Slave-устройства I2C.
- `static i2c_status_t I2C_SlaveEnable (I2C_Type *base, bool enable)`  
Включение/отключение модуля I2C.

#### Операции на шине в Slave-режиме.

- `i2c_status_t I2C_SlaveWriteBlocking` (`I2C_Type *base`, `const uint8_t *tx_buff`, `size_t tx_size`)  
Выполнение передачи из модуля на шину I2C в режиме блокировки.
- `i2c_status_t I2C_SlaveReadBlocking` (`I2C_Type *base`, `uint8_t *rx_buff`, `size_t rx_size`)  
Выполнение приема по шине I2C в модуль в режиме блокировки.

Неблокирующий обмен (по прерыванию) в Slave-режиме.

- `i2c_status_t I2C_SlaveTransferCreateHandle` (`I2C_Type *base`, `i2c_slave_handle_t *handle`, `i2c_slave_transfer_callback_t callback`, `void *user_data`)  
Создание нового дескриптора для I2C в Slave-режиме.
- `i2c_status_t I2C_SlaveTransferNonBlocking` (`I2C_Type *base`, `i2c_slave_handle_t *handle`, `uint32_t event_mask`)  
Инициализация приема для Slave-режима.
- `i2c_status_t I2C_SlaveSetSendBuffer` (`I2C_Type *base`, `volatile i2c_slave_transfer_t *transfer`, `const void *tx_data`, `size_t tx_size`, `uint32_t event_mask`)  
Запуск передачи данных из Slave в Master по прерыванию.
- `i2c_status_t I2C_SlaveSetReceiveBuffer` (`I2C_Type *base`, `volatile i2c_slave_transfer_t *transfer`, `void *rx_data`, `size_t rx_size`, `uint32_t event_mask`)  
Инициирование Slave-устройством приема запросов от Master-устройства.
- `i2c_status_t I2C_SlaveTransferAbort` (`I2C_Type *base`, `i2c_slave_handle_t *handle`)  
Завершение неблокирующей передачи Slave-устройства.
- `i2c_status_t I2C_SlaveTransferGetCount` (`I2C_Type *base`, `i2c_slave_handle_t *handle`, `size_t *count`)  
Получение количества оставшихся байтов во время неблокирующей передачи (через прерывание) в Slave-режиме.

Обработчик IRQ для Slave-режима.

- `void I2C_SlaveTransferHandleIRQ` (`I2C_Type *base`, `i2c_slave_handle_t *handle`)  
Процедура для обработки прерываний в Slave-режиме.

### 6.14.1 Подробное описание

Интерфейс драйвера модуля I2C.

### 6.14.2 Функции

#### 6.14.2.1 I2C\_SlaveDeinit()

```
static i2c_status_t I2C_SlaveDeinit (
    I2C_Type * base)  [inline], [static]
```

Деинициализация периферийного Slave-устройства I2C.

Эта функция отключает Slave-устройство I2C, а также выполняет программный сброс для восстановления модуля I2C до состояния сброса.

Аргументы

base	Базовый адрес модуля.
------	-----------------------

Возвращаемые значения

<a href="#">I2C_Status_Ok</a>	
<a href="#">I2C_Status_HwError</a>	

#### 6.14.2.2 I2C\_SlaveEnable()

```
static i2c\_status\_t I2C_SlaveEnable (
    I2C_Type * base,
    bool enable) [inline], [static]
```

Включение/отключение модуля I2C.

Заметки

Функция используется для Slave-режима.

Аргументы

base	Базовый адрес модуля.
enable	true - включить, false - выключить указанный модуль.

#### 6.14.2.3 I2C\_SlaveGetDefaultConfig()

```
void I2C_SlaveGetDefaultConfig (
    i2c\_slave\_config\_t * config)
```

Получение конфигурации по умолчанию для Slave-режима модуля I2C.

Эта функция обеспечивает следующую конфигурацию по умолчанию Slave-устройства I2C:

```
config->address      = I2C_SLAVE_ADDR_DEFAULT;
config->ack_gen_call = false;
config->i2c_addr_size = I2C\_Address7Bit;
config->enable_slave = true;
```

После вызова этой функции возможно изменить настройки, если требуется изменить конфигурацию, перед инициализацией с помощью [I2C\\_SlaveInit](#). Необходимо переопределить Slave-адрес желаемым адресом.

Аргументы

config	Пользовательская конфигурация.
--------	--------------------------------

#### 6.14.2.4 I2C\_SlaveInit()

```
i2c\_status\_t I2C_SlaveInit (
    I2C_Type * base,
    const i2c\_slave\_config\_t * config)
```

Инициализация Slave-модуля I2C.

Функция инициализирует ведомое периферийное устройство I2C, как описано пользователем в переданной конфигурации.

Заметки

Для получения значений по умолчанию для пользовательской конфигурации необходимо воспользоваться функцией [I2C\\_SlaveGetDefaultConfig](#), а затем переопределить их при необходимости.



## Аргументы

base	Базовый адрес модуля.
config	Пользовательская конфигурация.

## Возвращаемые значения

<a href="#">I2C_Status_Ok</a>	
<a href="#">I2C_Status_HwError</a>	

## 6.14.2.5 I2C\_SlaveReadBlocking()

```
i2c_status_t I2C_SlaveReadBlocking (
    I2C_Type * base,
    uint8_t * rx_buff,
    size_t rx_size)
```

Выполнение приема по шине I2C в модуль в режиме блокировки.

Функция выполняет фазу передачи адреса и фазу передачи данных.

## Аргументы

base	Базовый адрес модуля.
rx_buff	Буфер для хранения получаемых данных.
rx_size	Размер буфера получаемых данных в байтах.

## Возвращаемые значения

<a href="#">I2C_Status_Ok</a>	
<a href="#">I2C_Status_UnexpectedState</a>	Master пытается считать данные из Slave, в то время как Slave ожидает запись.

## 6.14.2.6 I2C\_SlaveSetAddress()

```
i2c_status_t I2C_SlaveSetAddress (
    I2C_Type * base,
    uint16_t address,
    bool ack_gen_call)
```

Установка адреса Slave-устройства.

Эта функция записывает новое значение в регистр адреса Slave-устройства.

## Аргументы

base	Базовый адрес модуля.
address	Адрес Slave-устройства.
ack_gen_call	Отвечать или нет General Call.

Возвращаемые значения

<a href="#">I2C_Status_Ok</a>	
<a href="#">I2C_Status_HwError</a>	

#### 6.14.2.7 I2C\_SlaveSetReceiveBuffer()

```
i2c_status_t I2C_SlaveSetReceiveBuffer (
    I2C_Type * base,
    volatile i2c_slave_transfer_t * transfer,
    void * rx_data,
    size_t rx_size,
    uint32_t event_mask)
```

Инициирование Slave-устройством приема запросов от Master-устройства.

Функция может быть вызвана в ответ на вызов функции обратного вызова по событию [I2C\\_SlaveEvent\\_Receive](#) для старта нового приема данных.

Набор событий, получаемых функцией обратного вызова, настраивается: для этого в параметр event\_mask передается комбинация из элементов [i2c\\_slave\\_event\\_transfer\\_t](#) для событий, которые требуется получать. События [I2C\\_SlaveEvent\\_Transmit](#) и [I2C\\_SlaveEvent\\_Receive](#) всегда включены и не требуют быть включенным в маску. Константа [I2C\\_SlaveEvents\\_All](#) включает все события. Константа [I2C\\_SlaveEvents\\_Ordinary](#) включает все события для работы в режиме без General Call.

Аргументы

base	Базовый адрес модуля.
transfer	Структура обмена данными.
rx_data	Буфер для хранения данных от Master.
rx_size	Размер буфера для хранения данных от Master в байтах.
event_mask	Битовая маска, определяющая события, приводящие к вызову функции обратного вызова.

Возвращаемые значения

<a href="#">I2C_Status_Ok</a>	
<a href="#">I2C_Status_Busy</a>	

#### 6.14.2.8 I2C\_SlaveSetSendBuffer()

```
i2c_status_t I2C_SlaveSetSendBuffer (
    I2C_Type * base,
    volatile i2c_slave_transfer_t * transfer,
    const void * tx_data,
    size_t tx_size,
    uint32_t event_mask)
```

Запуск передачи данных из Slave в Master по прерыванию.

Функция может быть вызвана в ответ на событие [I2C\\_SlaveEvent\\_Transmit](#), переданное в функцию обратного вызова, чтобы начать новую передачу из Slave в Master.

Набор событий, получаемых функцией обратного вызова, настраивается: для этого в параметр `event_mask` передается комбинация из элементов [i2c\\_slave\\_event\\_transfer\\_t](#) для событий, которые требуется получать. События [I2C\\_SlaveEvent\\_Transmit](#) и [I2C\\_SlaveEvent\\_Receive](#) всегда включены и не требуют быть включенным в маску. Константа [I2C\\_SlaveEvents\\_All](#) включает все события. Константа [I2C\\_SlaveEvents\\_Ordinary](#) включает все события для работы в режиме без General Call.

#### Аргументы

<code>base</code>	Базовый адрес модуля.
<code>transfer</code>	Структура обмена данными.
<code>tx_data</code>	Данные для отправки в Master.
<code>tx_size</code>	Количество данных для отправки в Master в байтах.
<code>event_mask</code>	Битовая маска, определяющая события, приводящие к вызову функции обратного вызова.

#### Возвращаемые значения

<a href="#">I2C_Status_Ok</a>	
<a href="#">I2C_Status_Busy</a>	Slave передача уже запущена для этого дескриптора.

#### 6.14.2.9 I2C\_SlaveTransferAbort()

```
i2c\_status\_t I2C_SlaveTransferAbort (
    I2C_Type * base,
    i2c\_slave\_handle\_t * handle)
```

Завершение неблокирующей передачи Slave-устройства.

#### Заметки

Функцию можно вызвать в любое время, чтобы остановить Slave-устройство для обработки событий шины.

#### Аргументы

<code>base</code>	Базовый адрес модуля.
<code>handle</code>	Дескриптор для Slave-режима.

#### Возвращаемые значения

<a href="#">I2C_Status_Ok</a>	
<a href="#">I2C_Status_Idle</a>	

## 6.14.2.10 I2C\_SlaveTransferCreateHandle()

```
i2c_status_t I2C_SlaveTransferCreateHandle (
    I2C_Type * base,
    i2c_slave_handle_t * handle,
    i2c_slave_transfer_callback_t callback,
    void * user_data)
```

Создание нового дескриптора для I2C в Slave-режиме.

## Заметки

Используется для работы в неблокирующих функциях (по прерыванию).

Создание дескриптора для использования в неблокирующих функциях. Однажды созданный дескриптор не требуется специально уничтожать. Если требуется завершить обмен данными используется функция [I2C\\_SlaveTransferAbort](#).

## Аргументы

base	Базовый адрес модуля.
handle	Дескриптор для Slave-режима.
callback	Функция обратного вызова.
user_data	Пользовательские данные для функции обратного вызова.

## Возвращаемые значения

<a href="#">I2C_Status_Ok</a>	
<a href="#">I2C_Status_UnexpectedState</a>	

## 6.14.2.11 I2C\_SlaveTransferGetCount()

```
i2c_status_t I2C_SlaveTransferGetCount (
    I2C_Type * base,
    i2c_slave_handle_t * handle,
    size_t * count)
```

Получение количества оставшихся байтов во время неблокирующей передачи (через прерывание) в Slave-режиме.

## Аргументы

base	Базовый адрес модуля.
handle	Дескриптор для Slave-режима.
count	Количество байтов, переданных на данный момент неблокирующей транзакцией.

## Возвращаемые значения

<a href="#">I2C_Status_InvalidParameter</a>	
<a href="#">I2C_Status_Ok</a>	

## 6.14.2.12 I2C\_SlaveTransferHandleIRQ()

```
void I2C_SlaveTransferHandleIRQ (
    I2C_Type * base,
    i2c_slave_handle_t * handle)
```

Процедура для обработки прерываний в Slave-режиме.

Заметки

Эту функцию не нужно вызывать самостоятельно.

Аргументы

base	Базовый адрес модуля.
handle	Дескриптор для Slave-режима.

## 6.14.2.13 I2C\_SlaveTransferNonBlocking()

```
i2c_status_t I2C_SlaveTransferNonBlocking (
    I2C_Type * base,
    i2c_slave_handle_t * handle,
    uint32_t event_mask)
```

Инициализация приема для Slave-режима.

Вызовите эту функцию после вызова [I2C\\_SlaveInit](#) и [I2C\\_SlaveTransferCreateHandle](#) для начала обмена, управляемого Master- устройством. Модуль в Slave-режиме отслеживает I2C шину и передает событие в функцию обратного вызова, которая была зарегистрирована при вызове [I2C\\_SlaveTransferCreateHandle](#). Функция обратного вызова вызывается из контекста прерывания.

Набор событий, получаемых функцией обратного вызова, настраивается: для этого в параметр event\_mask передается комбинация из элементов [i2c\\_slave\\_event\\_transfer\\_t](#) для событий, которые требуется получать. События [I2C\\_SlaveEvent\\_Transmit](#) и [I2C\\_SlaveEvent\\_Receive](#) всегда включены и не требуют быть включенным в маску. Константа [I2C\\_SlaveEvents\\_All](#) включает все события. Константа [I2C\\_SlaveEvents\\_Ordinary](#) включает все события для работы в режиме без General Call.

Аргументы

base	Базовый адрес модуля.
handle	Дескриптор для Slave-режима.
event_mask	Битовая маска, определяющая события, приводящие к вызову функции обратного вызова.

Возвращаемые значения

<a href="#">I2C_Status_Ok</a>	
<a href="#">I2C_Status_Busy</a>	На этом дескрипторе уже идет обмен данными.

#### 6.14.2.14 I2C\_SlaveWriteBlocking()

```
i2c_status_t I2C_SlaveWriteBlocking (  
    I2C_Type * base,  
    const uint8_t * tx_buff,  
    size_t tx_size)
```

Выполнение передачи из модуля на шину I2C в режиме блокировки.

Функция выполняет фазу передачи адреса и фазу передачи данных.

## Аргументы

base	Базовый адрес модуля.
tx_buff	Буфер для хранения передаваемых данных.
tx_size	Размер буфера передаваемых данных в байтах.

## Возвращаемые значения

I2C_Status_Ok	
I2C_Status_UnexpectedState	Master пытается считать данные из Slave, в то время как Slave ожидает запись.

## 6.15 hal\_i2c.h

См. документацию.

```

00001
00024 #ifndef HAL_I2C_H
00025 #define HAL_I2C_H
00026
00027 #include "hal_common.h"
00028
00029 #define I2C_CFG_MASK 0x1f
00034 #define HAL_I2C_DRIVER_VERSION (MAKE_VERSION(1, 0, 0))
00035
00039 #ifndef I2C_RETRY_TIMES
00040 #define I2C_RETRY_TIMES 0U
00041 #endif
00042
00046 #ifndef I2C_MASTER_TRANSMIT_IGNORE_LAST_NACK
00047 #define I2C_MASTER_TRANSMIT_IGNORE_LAST_NACK 0U
00048 #endif
00049
00050 /* TODO Master: Определения битов MSTCODE в регистре состояния I2C STAT */
00051 #define I2C_STAT_MSTCODE_IDLE (0U)
00052 #define I2C_STAT_MSTCODE_RXREADY (1U)
00053 #define I2C_STAT_MSTCODE_TXREADY (2U)
00054 #define I2C_STAT_MSTCODE_NACKADR (3U)
00055 #define I2C_STAT_MSTCODE_NACKDAT (4U)
00057 /* TODO Slave: Определения битов SLVSTATE в регистре состояния I2C STAT */
00058 #define I2C_STAT_SLVST_ADDR (0)
00059 #define I2C_STAT_SLVST_RX (1)
00060 #define I2C_STAT_SLVST_TX (2)
00065 typedef enum {
00066     I2C_Status_Ok           = 0U,
00067     I2C_Status_Busy        = 1U,
00068     I2C_Status_Idle        = 2U,
00069     I2C_Status_Nack        = 3U,
00070     I2C_Status_InvalidParameter = 4U,
00071     I2C_Status_BreakTransfer = 5U,
00072     I2C_Status_ArbitrationLost = 6U,
00073     I2C_Status_NoTransferInProgress = 7U,
00074     I2C_Status_DmaRequestFail = 8U,
00076     I2C_Status_Timeout      = 11U,
00077     I2C_Status_AddrNack     = 12U,
00078     I2C_Status_UserError    = 13U,
00079     I2C_Status_SetStartError = 14U,
00080     I2C_Status_HwError      = 15U,
00081     I2C_Status_HsCodeError  = 16U,
00083     I2C_Status_UnexpectedState = 100U,
00084     I2C_Status_UnexpectedState1 = 101U,
00085     I2C_Status_UnexpectedState2 = 102U,
00086     I2C_Status_UnexpectedState3 = 103U,
00087     I2C_Status_UnexpectedState4 = 104U,
00088     I2C_Status_UnexpectedState5 = 105U,
00089     I2C_Status_UnexpectedState6 = 106U,
00090     I2C_Status_UnexpectedState7 = 107U,
00091     I2C_Status_UnexpectedState8 = 108U,
00092     I2C_Status_UnexpectedState9 = 109U,
00094     I2C_Status_UnDef          = 255U,
00095 } i2c_status_t;
00096

```

```

00100 enum i2c_status_flags {
00101     I2C_Stat_Active = I2C_IC_STATUS_ACTIVITY_Msk,
00102     I2C_Stat_TxFifo_NotFull = I2C_IC_STATUS_TFNF_Msk,
00103     I2C_Stat_TxFifo_Empty = I2C_IC_STATUS_TFE_Msk,
00104     I2C_Stat_RxFifo_NotEmpty = I2C_IC_STATUS_RFNE_Msk,
00105     I2C_Stat_RxFifo_Full = I2C_IC_STATUS_RFF_Msk,
00106     I2C_Stat_Master_Active = I2C_IC_STATUS_MST_ACTIVITY_Msk,
00107     I2C_Stat_Slave_Active = I2C_IC_STATUS_SLV_ACTIVITY_Msk
00108 };
00109
00110 enum i2c_abort_flags {
00111     I2C_Abort_7B_Addr_Nack = I2C_IC_TX_ABRT_SOURCE_ABRT_7B_ADDR_NOACK_Msk,
00112     I2C_Abort_10B_Addr1_Nack = I2C_IC_TX_ABRT_SOURCE_ABRT_10ADDR1_NOACK_Msk,
00113     I2C_Abort_10B_Addr2_Nack = I2C_IC_TX_ABRT_SOURCE_ABRT_10ADDR2_NOACK_Msk,
00114     I2C_Abort_TxData_Nack = I2C_IC_TX_ABRT_SOURCE_ABRT_TXDATA_NOACK_Msk,
00115     I2C_Abort_GenCall_Nack = I2C_IC_TX_ABRT_SOURCE_ABRT_GCALL_NOACK_Msk,
00116     I2C_Abort_GenCall_Read = I2C_IC_TX_ABRT_SOURCE_ABRT_GCALL_READ_Msk,
00117     I2C_Abort_HsCode_Ack = I2C_IC_TX_ABRT_SOURCE_ABRT_HS_ACKDET_Msk,
00118     I2C_Abort_StartByte_Ack = I2C_IC_TX_ABRT_SOURCE_ABRT_SBYTE_ACKDET_Msk,
00119     I2C_Abort_HS_RStart_Dis = I2C_IC_TX_ABRT_SOURCE_ABRT_HS_NORSTRT_Msk,
00120     I2C_Abort_RStart_Dis = I2C_IC_TX_ABRT_SOURCE_ABRT_SBYTE_NORSTRT_Msk,
00121     I2C_Abort_Read_RStart_Dis = I2C_IC_TX_ABRT_SOURCE_ABRT_10B_RD_NORSTRT_Msk,
00122     I2C_Abort_Master_Dis = I2C_IC_TX_ABRT_SOURCE_ABRT_MASTER_DIS_Msk,
00123     I2C_Abort_Arbitr_Lost = I2C_IC_TX_ABRT_SOURCE_ARB_LOST_Msk,
00124     I2C_Abort_RxFifo_NotEmpty = I2C_IC_TX_ABRT_SOURCE_ABRT_SLVFLUSH_TXFIFO_Msk,
00125     I2C_Abort_SlaveArbitr_Lost = I2C_IC_TX_ABRT_SOURCE_ABRT_SLV_ARBLOST_Msk,
00126     I2C_Abort_SlaveDataCmd_Error = I2C_IC_TX_ABRT_SOURCE_ABRT_SLVRD_INTX_Msk,
00127     I2C_Abort_MasterDetect = I2C_IC_TX_ABRT_SOURCE_ABRT_USER_ABRT_Msk,
00128     I2C_Abort_Tx_FlushCnt = I2C_IC_TX_ABRT_SOURCE_TX_FLUSH_CNT_Msk
00129 };
00130
00131 enum i2c_interrupt {
00132     I2C_IRQ_RxUnder = I2C_IC_INTR_MASK_M_RX_UNDER_Msk,
00133     I2C_IRQ_RxOver = I2C_IC_INTR_MASK_M_RX_OVER_Msk,
00134     I2C_IRQ_RxFull = I2C_IC_INTR_MASK_M_RX_FULL_Msk,
00135     I2C_IRQ_TxOver = I2C_IC_INTR_MASK_M_TX_OVER_Msk,
00136     I2C_IRQ_TxEmpty = I2C_IC_INTR_MASK_M_TX_EMPTY_Msk,
00137     I2C_IRQ_RdReq = I2C_IC_INTR_MASK_M_RD_REQ_Msk,
00138     I2C_IRQ_TxAbrt = I2C_IC_INTR_MASK_M_TX_ABRT_Msk,
00139     I2C_IRQ_RxDone = I2C_IC_INTR_MASK_M_RX_DONE_Msk,
00140     I2C_IRQ_Activity = I2C_IC_INTR_MASK_M_ACTIVITY_Msk,
00141     I2C_IRQ_StopDet = I2C_IC_INTR_MASK_M_STOP_DET_Msk,
00142     I2C_IRQ_StartDet = I2C_IC_INTR_MASK_M_START_DET_Msk,
00143     I2C_IRQ_GenCall = I2C_IC_INTR_MASK_M_GEN_CALL_Msk,
00144     I2C_IRQ_Slave = I2C_IRQ_RxUnder | I2C_IRQ_RxOver | I2C_IRQ_RxFull |
00145         I2C_IRQ_TxOver | I2C_IRQ_RdReq |
00146         I2C_IRQ_TxAbrt | I2C_IRQ_RxDone |
00147         I2C_IRQ_StopDet | I2C_IRQ_StartDet | I2C_IRQ_GenCall,
00148     I2C_IRQ_SlaveTst = I2C_IRQ_RxFull /*TODO*/
00149     I2C_IRQ_RdReq |
00150     I2C_IRQ_TxAbrt | I2C_IRQ_RxDone |
00151     I2C_IRQ_StopDet | I2C_IRQ_StartDet | I2C_IRQ_GenCall,
00152     I2C_IRQ_All = I2C_IRQ_RxUnder | I2C_IRQ_RxOver | I2C_IRQ_RxFull |
00153         I2C_IRQ_TxOver | I2C_IRQ_TxEmpty | I2C_IRQ_RdReq |
00154         I2C_IRQ_TxAbrt | I2C_IRQ_RxDone | I2C_IRQ_Activity |
00155         I2C_IRQ_StopDet | I2C_IRQ_StartDet | I2C_IRQ_GenCall
00156 };
00157
00158 typedef enum {
00159     I2C_Write = 0U,
00160     I2C_Read = 1U
00161 } i2c_direction_t;
00162
00163 typedef enum {
00164     I2C_Address7Bit = 0U,
00165     I2C_Address10Bit = 1U
00166 } i2c_addr_size_t;
00167
00168 typedef enum {
00169     I2C_MTS_Idle = 0U,
00170     I2C_MTS_TransmitSubaddr = 1U,
00171     I2C_MTS_TransmitData = 2U,
00172     I2C_MTS_ReceiveDataBegin = 3U,
00173     I2C_MTS_ReceiveData = 4U,
00174     I2C_MTS_ReceiveLastData = 5U,
00175     I2C_MTS_Start = 6U,
00176     I2C_MTS_Stop = 7U,
00177     I2C_MTS_WaitForCompletion = 8U,
00178 } i2c_master_transfer_states;
00179
00180 typedef enum {
00181     I2C_UndefinedSpeedMode = 0U,
00182     I2C_StandardSpeedMode = 1U,

```



```

00239     I2C_FastSpeedMode    = 2U,
00240     I2C_HighSpeedMode    = 3U,
00241 } i2c_speed_mode_t;
00242
00251 typedef struct {
00252     bool    enable_master;
00253     uint32_t baudrate_bps;
00254     uint8_t  sda_setup;
00255     uint16_t sda_hold;
00256 } i2c_master_config_t;
00257
00261 typedef struct _i2c_master_handle i2c_master_handle_t;
00262
00275 typedef void (*i2c_master_transfer_callback_t)(I2C_Type *base,
00276     i2c_master_handle_t *handle, i2c_status_t completion_status,
00277     void *user_data);
00278
00284 typedef enum {
00285     I2C_TransferDataFlag    = 0x01U,
00286     I2C_TransferStartFlag   = 0x02U,
00287     I2C_TransferStopFlag    = 0x04U,
00288     I2C_TransferReStartFlag = 0x08U,
00289 } i2c_master_transfer_flags_t;
00290
00320 typedef struct {
00321     uint32_t    flags;
00322     uint16_t    slave_address;
00323     i2c_addr_size_t addr_size;
00324     i2c_direction_t direction;
00325     uint32_t    subaddress;
00326     size_t      subaddress_size;
00327     void        *data;
00328     size_t      data_size;
00329 } i2c_master_transfer_t;
00330
00334 struct i2c_master_handle {
00335     uint8_t    state;
00336     uint32_t    transfer_count;
00337     uint32_t    remaining_bytes;
00338     uint8_t     *buf;
00339     uint32_t    remaining_subaddr;
00340     uint8_t     subaddr_buf[4];
00341     bool        check_addr_nack;
00342     i2c_master_transfer_t transfer;
00343     i2c_master_transfer_callback_t completion_callback;
00344     void        *user_data;
00345 };
00346
00355 typedef struct {
00356     uint16_t    address;
00357     bool        ack_gen_call;
00358     i2c_addr_size_t i2c_addr_size;
00359     bool        enable_slave;
00360 } i2c_slave_config_t;
00361
00374 typedef enum {
00375     I2C_SlaveEvent_MatchAddrSlave = 0x01,
00376     I2C_SlaveEvent_MatchAddrGen   = 0x02,
00377     I2C_SlaveEvent_Transmit        = 0x08,
00378     I2C_SlaveEvent_Receive         = 0x10,
00379     I2C_SlaveEvent_Completion      = 0x20,
00381     I2C_SlaveEvents_Ordinary       = I2C_SlaveEvent_Transmit |
00382         I2C_SlaveEvent_Receive |
00383         I2C_SlaveEvent_Completion,
00384
00385     I2C_SlaveEvents_All            = I2C_SlaveEvent_MatchAddrSlave |
00386         I2C_SlaveEvent_MatchAddrGen |
00387         I2C_SlaveEvent_Transmit |
00388         I2C_SlaveEvent_Receive |
00389         I2C_SlaveEvent_Completion,
00390 } i2c_slave_event_transfer_t;
00391
00395 typedef struct _i2c_slave_handle i2c_slave_handle_t;
00396
00400 typedef struct {
00401     i2c_slave_handle_t *handle;
00402     i2c_slave_event_transfer_t event;
00403     uint32_t    event_mask;
00404     uint8_t     *rx_data;
00405     size_t      rx_size;
00406     const uint8_t *tx_data;
00407     size_t      tx_size;
00408     size_t      transferred_count;
00409     i2c_status_t completion_status;
00410 } i2c_slave_transfer_t;
00411
00424 typedef void (*i2c_slave_transfer_callback_t)(I2C_Type *base,

```

```

00425     volatile i2c_slave_transfer_t *transfer, void *user_data);
00426
00430 typedef enum {
00431     I2C_SlaveFsm_Idle    = 0U,
00432     I2C_SlaveFsm_Init    = 1U,
00433     I2C_SlaveFsm_Receive = 4U,
00434     I2C_SlaveFsm_Transmit = 6U,
00435     I2C_SlaveFsm_Stop    = 8U,
00436 } i2c_slave_fsm_t;
00437
00441 struct i2c_slave_handle {
00442     int32_t irq_num;
00443     volatile i2c_slave_transfer_t transfer;
00444     volatile i2c_slave_fsm_t slave_fsm;
00445     i2c_slave_transfer_callback_t callback;
00446     void *user_data;
00447 };
00448
00449 #if defined(__cplusplus)
00450 extern "C" {
00451 #endif
00452
00466 static inline bool I2C_IsEnable(I2C_Type *base)
00467 {
00468     return (bool) GET_VAL_MSK(base->IC_ENABLE_STATUS,
00469                               I2C_IC_ENABLE_STATUS_IC_EN_Msk, I2C_IC_ENABLE_STATUS_IC_EN_Pos);
00470 }
00471
00506 i2c_status_t I2C_Enable(I2C_Type *base, bool enable);
00507
00518 uint32_t I2C_GetInstance(I2C_Type *base);
00519
00530 i2c_status_t I2C_Reset(I2C_Type *base);
00531
00546 void I2C_MasterGetDefaultConfig(i2c_master_config_t *master_config);
00547
00566 static inline void I2C_EnableInterrupts(I2C_Type *base, uint32_t mask)
00567 {
00568     base->IC_INTR_MASK |= mask & I2C_IRQ_All;
00569 }
00570
00580 static inline void I2C_DisableInterrupts(I2C_Type *base, uint32_t mask)
00581 {
00582     base->IC_INTR_MASK &= ~(mask & I2C_IRQ_All);
00583 }
00584
00595 static inline uint32_t I2C_GetEnabledInterrupts(I2C_Type *base)
00596 {
00597     return base->IC_INTR_MASK & I2C_IRQ_All;
00598 }
00599
00609 static inline void I2C_ClearInterrupt(I2C_Type *base, enum i2c_interrupt irq)
00610 {
00611     volatile uint32_t temp = 0;
00612     UNUSED(temp);
00613
00614     switch (irq) {
00615         case I2C_IRQ_RxUnder: /* [0] Чтении из пустого RxFifo. */
00616             temp = base->IC_CLR_RX_UNDER; /* Сброс: чтение IC_CLR_RX_UNDER */
00617             break;
00618
00619         case I2C_IRQ_RxOver: /* [1] Переполнение RxFifo. */
00620             temp = base->IC_CLR_RX_OVER; /* Сброс: чтение IC_CLR_RX_OVER */
00621             break;
00622
00623         case I2C_IRQ_RxFull: /* [2] RxFifo заполнен до уровня IC_RX_TL. */
00624             assert(0);
00625             break;
00626
00627         case I2C_IRQ_TxOver: /* [3] Попытка записать в заполненный TxFifo */
00628             temp = base->IC_CLR_TX_OVER; /* Сброс: чтение IC_CLR_TX_OVER */
00629             break;
00630
00631         case I2C_IRQ_TxEmpty: /* [4] Опустошение TxFifo ниже уровня IC_TX_TL */
00632             assert(0);
00633             break;
00634
00635         case I2C_IRQ_RdReq: /* [5] Уст. при запросе данных удаленным Master. */
00636             temp = base->IC_CLR_RD_REQ; /* Сброс чтением: IC_CLR_RD_REQ.CLAR_RD_REQ */
00637             break;
00638
00639         case I2C_IRQ_TxABrt: /* [6] Передача прервана */
00640             temp = base->IC_CLR_TX_ABRT; /* Сброс чтением: IC_CLR_TX_ABRT.CLAR_TX_ABRT */
00641             break;
00642
00643         case I2C_IRQ_RxDone: /* [7] В режиме Slave-передатчика устанавливается в 1, если мастер не подтверждает
00644             передачу байта */

```

```

00644     temp = base->IC_CLR_RX_DONE; /* Сброс чтением: IC_CLR_RX_DONE.CLR_RX_DONE */
00645     break;
00646
00647     case I2C_IRQ_Activity: /* [8] Активность на шине I2C */
00648     temp = base->IC_CLR_ACTIVITY; /* Сброс чтением: 1) IC_CLR_INTR 2)
IC_CLR_ACTIVITY.CLR_ACTIVITY 3) Выключение I2C 4) Системный сброс */
00649     break;
00650
00651     case I2C_IRQ_StopDet: /* [9] Устанавливается в Slave- или Master-режиме, если на шине возникает состояние
STOP */
00652     temp = base->IC_CLR_STOP_DET; /* Сброс: чтение IC_CLR_STOP_DET */
00653     break;
00654
00655     case I2C_IRQ_StartDet: /* [10] На шине START или RESTART условия */
00656     temp = base->IC_CLR_START_DET; /* Сброс: чтение IC_CLR_START_DET */
00657     break;
00658
00659     case I2C_IRQ_GenCall: /* [11] Получен адрес General Call и отправлено подтверждение */
00660     temp = base->IC_CLR_GEN_CALL; /* Сброс: 1) чтением IC_CLR_GEN_CALL 2) выключение модуля */
00661     break;
00662
00663     default:
00664     assert(0);
00665     break;
00666 }
00667 }
00668 }
00669
00670 static inline void I2C_ClearAllInterrupts(I2C_Type *base)
00671 {
00672     volatile uint32_t temp;
00673     temp = base->IC_CLR_INTR;
00674
00675     UNUSED(temp);
00676 }
00677
00678 i2c_status_t I2C_MasterInit(I2C_Type *base,
00679     const i2c_master_config_t *master_config, uint32_t src_clock_hz);
00680
00681 i2c_status_t I2C_MasterDeinit(I2C_Type *base);
00682
00683 i2c_status_t I2C_MasterSetBaudRate(I2C_Type *base, uint32_t baudrate_bps,
00684     uint32_t src_clock_hz);
00685
00686 i2c_status_t I2C_MasterAddrSet(I2C_Type *base, uint32_t address,
00687     i2c_addr_size_t addr_size);
00688
00689 static inline bool I2C_MasterGetBusActiveState(I2C_Type *base)
00690 {
00691     /* Получить значение флага: Активность на шине */
00692     return (bool) GET_VAL_MSK(base->IC_STATUS, I2C_IC_STATUS_ACTIVITY_Msk,
00693         I2C_IC_STATUS_ACTIVITY_Pos);
00694 }
00695
00696 i2c_status_t I2C_MasterWriteBlocking(I2C_Type *base, const void *tx_buff,
00697     size_t tx_size, uint32_t flags);
00698
00699 i2c_status_t I2C_MasterReadBlocking(I2C_Type *base, void *rx_buff,
00700     size_t rx_size, uint32_t flags);
00701
00702 i2c_status_t I2C_MasterTransferBlocking(I2C_Type *base,
00703     i2c_master_transfer_t *xfer);
00704
00705 void I2C_MasterTransferCreateHandle(I2C_Type *base, i2c_master_handle_t *handle,
00706     i2c_master_transfer_callback_t callback, void *user_data);
00707
00708 i2c_status_t I2C_MasterTransferNonBlocking(I2C_Type *base,
00709     i2c_master_handle_t *handle, i2c_master_transfer_t *xfer);
00710
00711 i2c_status_t I2C_MasterTransferGetCount(I2C_Type *base,
00712     i2c_master_handle_t *handle, size_t *count);
00713
00714 i2c_status_t I2C_MasterTransferAbort(I2C_Type *base,
00715     i2c_master_handle_t *handle);
00716
00717 void I2C_MasterTransferHandleIRQ(I2C_Type *base, i2c_master_handle_t *handle);
00718
00719 void I2C_SlaveGetDefaultConfig(i2c_slave_config_t *config);
00720
00721 i2c_status_t I2C_SlaveInit(I2C_Type *base, const i2c_slave_config_t *config);
00722
00723 i2c_status_t I2C_SlaveSetAddress(I2C_Type *base, uint16_t address,
00724     bool ack_gen_call);
00725
00726 static inline i2c_status_t I2C_SlaveDeinit(I2C_Type *base)
00727 {
00728     return I2C_Enable(base, false);
00729 }

```

```

01097 }
01098
01107 static inline i2c_status_t I2C_SlaveEnable(I2C_Type *base, bool enable)
01108 {
01109     return I2C_Enable(base, enable);
01110 }
01111
01134 i2c_status_t I2C_SlaveWriteBlocking(I2C_Type *base, const uint8_t *tx_buff,
01135     size_t tx_size);
01136
01150 i2c_status_t I2C_SlaveReadBlocking(I2C_Type *base, uint8_t *rx_buff,
01151     size_t rx_size);
01152
01179 i2c_status_t I2C_SlaveTransferCreateHandle(I2C_Type *base,
01180     i2c_slave_handle_t *handle, i2c_slave_transfer_callback_t callback,
01181     void *user_data);
01182
01209 i2c_status_t I2C_SlaveTransferNonBlocking(I2C_Type *base,
01210     i2c_slave_handle_t *handle, uint32_t event_mask);
01211
01237 i2c_status_t I2C_SlaveSetSendBuffer(I2C_Type *base,
01238     volatile i2c_slave_transfer_t *transfer, const void *tx_data,
01239     size_t tx_size, uint32_t event_mask);
01240
01265 i2c_status_t I2C_SlaveSetReceiveBuffer(
01266     I2C_Type *base, volatile i2c_slave_transfer_t *transfer, void *rx_data,
01267     size_t rx_size, uint32_t event_mask);
01268
01281 i2c_status_t I2C_SlaveTransferAbort(I2C_Type *base, i2c_slave_handle_t *handle);
01282
01295 i2c_status_t I2C_SlaveTransferGetCount(I2C_Type *base,
01296     i2c_slave_handle_t *handle, size_t *count);
01297
01315 void I2C_SlaveTransferHandleIRQ(I2C_Type *base, i2c_slave_handle_t *handle);
01316
01325 #if defined(__cplusplus)
01326 }
01327 #endif
01328
01329 #endif /* HAL_I2C_H */
01330

```

## 6.16 Файл devices/eliot1/drivers/hal\_i2c\_dma.h

Дополнение драйвера I2C с пересылкой данных через DMA.

```

#include "hal_dma.h"
#include "hal_i2c.h"

```

Структуры данных

- struct [\\_i2c\\_master\\_dma\\_handle](#)  
Контекст данных прерывания I2C-DMA.

Макросы

- #define HAL\_SPI\_DMA\_DRIVER\_VERSION ([MAKE\\_VERSION](#)(0, 1, 0))  
Версия драйвера
- #define I2C\_MAX\_BUFFER\_SIZE (8U)  
Глубина буфера I2C контроллера

Определения типов

- typedef struct [\\_i2c\\_master\\_dma\\_handle](#) i2c\_master\_dma\_handle\_t  
Дескриптор I2C-DMA.
- typedef void(\* i2c\_master\_dma\_transfer\_callback\_t) (I2C\_Type \*base, [i2c\\_master\\_dma\\_handle\\_t](#) \*handle, void \*user\_data)  
Функция обратного вызова

## Функции

- void `I2C_MasterTransferCreateHandleDMA` (`I2C_Type *base`, `i2c_master_dma_handle_t *handle`, `i2c_master_dma_transfer_callback_t callback`, `void *user_data`, `dma_handle_t *tx_dma`, `dma_handle_t *rx_dma`)

Функция инициализации дескриптора I2C-DMA.

- `i2c_status_t I2C_MasterTransferDMA` (`I2C_Type *base`, `i2c_master_dma_handle_t *handle`, `i2c_master_transfer_t *xfer`)

Функция, запускающая I2C транзакцию. Данные в буфер/из буфера I2C передаются с помощью I2C.

- void `I2C_MasterTransferAbortDMA` (`I2C_Type *base`, `i2c_master_dma_handle_t *handle`)

Прекращение передачи I2C.

- static void `I2C_DMADescriptorInitTX` (`I2C_Type *base`, `dma_descriptor_t *desc`, `uint32_t count`, `uint32_t data_size`, `uint8_t src_incr`, `void *src_addr`)

Инициализация дескрипторов DMA для многоблочной передачи TX.

- static void `I2C_DMADescriptorInitRX` (`I2C_Type *base`, `dma_descriptor_t *desc`, `uint32_t count`, `uint32_t data_size`, `uint8_t dst_incr`, `void *dst_addr`)

Инициализация дескрипторов DMA для многоблочной передачи RX.

## 6.16.1 Подробное описание

Дополнение драйвера I2C с пересылкой данных через DMA.

## 6.17 hal\_i2c\_dma.h

См. документацию.

```

00001
00012 #ifndef HAL_I2C_DMA_H
00013 #define HAL_I2C_DMA_H
00014
00015 #include "hal_dma.h"
00016 #include "hal_i2c.h"
00017
00019 #define HAL_SPI_DMA_DRIVER_VERSION (MAKE_VERSION(0, 1, 0))
00021 #define I2C_MAX_BUFFER_SIZE (8U)
00022
00026 typedef struct i2c_master_dma_handle i2c_master_dma_handle_t;
00027
00029 typedef void (*i2c_master_dma_transfer_callback_t)
00030 (I2C_Type *base, i2c_master_dma_handle_t *handle, void *user_data);
00031
00035 struct i2c_master_dma_handle {
00036     i2c_master_transfer_states state;
00037     uint32_t remaining_bytes_DMA;
00038     uint8_t *buf;
00039     uint32_t *dummy_data;
00040     dma_handle_t *tx_dma;
00041     dma_handle_t *rx_dma;
00042     dma_descriptor_t *tx_desc;
00043     dma_descriptor_t *rx_desc;
00044     i2c_master_dma_transfer_callback_t completion_callback;
00045     void *user_data;
00046 };
00047
00048 #if defined(_cplusplus)
00049 extern "C" {
00050 #endif
00051
00062 void I2C_MasterTransferCreateHandleDMA(I2C_Type *base,
00063     i2c_master_dma_handle_t *handle,
00064     i2c_master_dma_transfer_callback_t callback,
00065     void *user_data,
00066     dma_handle_t *tx_dma, dma_handle_t *rx_dma);
00067
00083 i2c_status_t I2C_MasterTransferDMA(I2C_Type *base,
00084     i2c_master_dma_handle_t *handle, i2c_master_transfer_t *xfer);

```

```

00085
00092 void I2C_MasterTransferAbortDMA(I2C_Type *base,
00093     i2c_master_dma_handle_t *handle);
00094
00106 static inline void I2C_DMADescriptorInitTX(I2C_Type *base,
00107     dma_descriptor_t *desc, uint32_t count, uint32_t data_size,
00108     uint8_t src_incr, void *src_addr)
00109 {
00110     dma_multiblock_config_t config = {
00111         .count = count,
00112         .data_size = data_size,
00113         .transfer_type = DMA_MemoryToPeripheral_DMA,
00114         .scatter_en = false,
00115         .gather_en = false,
00116         .src_burst_size = DMA_BurstSize32,
00117         .dst_burst_size = DMA_BurstSize1,
00118         .src_incr = src_incr,
00119         .dst_incr = DMA_NoChange,
00120         .src_data_width = DMA_Transfer8BitWidth,
00121         .dst_data_width = DMA_Transfer8BitWidth,
00122         .src_addr = src_addr,
00123         .dst_addr = (void *) &base->IC_DATA_CMD,
00124         .int_en = true
00125     };
00126
00127     DMA_InitMultiblockDescriptor(desc, &config);
00128 }
00129
00141 static inline void I2C_DMADescriptorInitRX(I2C_Type *base,
00142     dma_descriptor_t *desc, uint32_t count, uint32_t data_size,
00143     uint8_t dst_incr, void *dst_addr)
00144 {
00145     dma_multiblock_config_t config = {
00146         .count = count,
00147         .data_size = data_size,
00148         .transfer_type = DMA_PeripheralToMemory_DMA,
00149         .scatter_en = false,
00150         .gather_en = false,
00151         .src_burst_size = DMA_BurstSize1,
00152         .dst_burst_size = DMA_BurstSize32,
00153         .src_incr = DMA_NoChange,
00154         .dst_incr = dst_incr,
00155         .src_data_width = DMA_Transfer8BitWidth,
00156         .dst_data_width = DMA_Transfer8BitWidth,
00157         .src_addr = (void *) &base->IC_DATA_CMD,
00158         .dst_addr = dst_addr,
00159         .int_en = true
00160     };
00161
00162     DMA_InitMultiblockDescriptor(desc, &config);
00163 }
00164
00165 #if defined(__cplusplus)
00166 }
00167 #endif
00168
00169 #endif /* HAL_I2C_DMA_H */
00170

```

## 6.18 Файл devices/eliot1/drivers/hal\_i2s.h

Интерфейс драйвера модуля I2S.

```
#include "hal_common.h"
```

Структуры данных

- struct [\\_i2s\\_config](#)  
Структура конфигурации контроллера I2S.
- struct [\\_i2s\\_transfer](#)  
Структура для передачи данных I2S в неблокирующем режиме (по прерыванию)
- struct [\\_i2s\\_handle](#)  
Структура обработчика драйвера I2S.

## Макросы

- `#define HAL_I2S_DRIVER_VERSION (MAKE_VERSION(0, 1, 0))`  
Версия драйвера I2S.

## Определения типов

- `typedef enum __i2s_status i2s_status_t`  
Коды возврата функций драйвера I2S.
- `typedef enum __i2s_flag i2s_flag_t`  
Флаги прерываний I2S.
- `typedef enum __i2s_sclk_per_sample i2s_sclk_per_sample_t`  
Перечисление возможных значений числа синхроимпульсов sclk на левый и правый поток
- `typedef enum __i2s_sclk_gating i2s_sclk_gating_t`  
Перечисление возможных значений обрезания числа синхроимпульсов sclk.
- `typedef enum __i2s_resolution i2s_resolution_t`  
Разрядность данных I2S.
- `typedef enum __i2s_interrupt_level i2s_interrupt_level_t`  
Уровни срабатывания прерывания по опустошению очереди I2S.
- `typedef struct __i2s_config i2s_config_t`  
Структура конфигурации контроллера I2S.
- `typedef struct __i2s_transfer i2s_transfer_t`  
Структура для передачи данных I2S в неблокирующем режиме (по прерыванию)
- `typedef struct __i2s_handle i2s_handle_t`  
Сокращённое название типа для структуры обработчика драйвера I2S.
- `typedef void(* i2s_transfer_callback_t) (I2S_Type *base, i2s_handle_t *handle, i2s_flag_t interrupt_flag, void *user_data)`  
Функция обратного вызова I2S.

## Перечисления

- `enum __i2s_status`  
Коды возврата функций драйвера I2S.
- `enum __i2s_flag`  
Флаги прерываний I2S.
- `enum __i2s_sclk_per_sample`  
Перечисление возможных значений числа синхроимпульсов sclk на левый и правый поток
- `enum __i2s_sclk_gating`  
Перечисление возможных значений обрезания числа синхроимпульсов sclk.
- `enum __i2s_resolution`  
Разрядность данных I2S.
- `enum __i2s_interrupt_level`  
Уровни срабатывания прерывания по опустошению очереди I2S.

## Функции

## Инициализация и деинициализация

- `i2s_status_t I2S_Init` (`I2S_Type *base`, `const i2s_config_t *config`, `uint32_t source_clock` ← `_hz`)  
Инициализация драйвера I2S.
- `void I2S_Deinit` (`I2S_Type *base`)  
Деинициализация драйвера I2S.
- `void I2S_GetDefaultConfig` (`i2s_config_t *config`)  
Получение параметров драйвера I2S по умолчанию

## Управление прерываниями

- `static void I2S_EnableInterrupt` (`I2S_Type *base`, `i2s_flag_t idx`)  
Разрешение прерывания
- `static void I2S_EnableInterruptMask` (`I2S_Type *base`, `uint32_t mask`)  
Разрешение прерываний по маске
- `static bool I2S_IsInterruptEnabled` (`I2S_Type *base`, `i2s_flag_t idx`)  
Запрос - разрешено ли прерывание I2S.
- `static uint32_t I2S_GetEnabledInterruptMask` (`I2S_Type *base`)  
Запрос маски разрешенных прерываний I2S.
- `static void I2S_DisableInterrupt` (`I2S_Type *base`, `i2s_flag_t idx`)  
Запрет прерывания
- `static void I2S_DisableInterruptMask` (`I2S_Type *base`, `uint32_t mask`)  
Запрет прерываний по маске
- `static bool I2S_GetInterruptStatus` (`I2S_Type *base`, `i2s_flag_t idx`)  
Получение состояния флага прерывания
- `static uint32_t I2S_GetInterruptStatusMask` (`I2S_Type *base`)  
Получение маски активных прерываний
- `static void I2S_ClearDataOverrunFlag` (`I2S_Type *base`)  
Сброс флага переполнения очереди выдачи

## Прямое управление выдачей

- `void I2S_WriteLeftFifo` (`I2S_Type *base`, `uint32_t sample`)  
Запись сэмпла в левый канал I2S.
- `void I2S_WriteRightFifo` (`I2S_Type *base`, `uint32_t sample`)  
Запись сэмпла в правый канал I2S.
- `static void I2S_AbortTx` (`I2S_Type *base`)  
Отмена выдачи по I2S.
- `static void I2S_EnableTx` (`I2S_Type *base`)  
Включение передатчика I2S.
- `static void I2S_DisableTx` (`I2S_Type *base`)  
Выключение передатчика I2S.

## Неблокирующая выдача по I2S

- `i2s_status_t I2S_TransferCreateHandle` (`I2S_Type *base`, `i2s_handle_t *handle`, `i2s_transfer_callback_t callback`, `void *user_data`)  
Инициализация обработчика событий I2S.
- `i2s_status_t I2S_TransferNonBlocking` (`I2S_Type *base`, `i2s_handle_t *handle`, `i2s_transfer_t *xfer`)  
Неблокирующая выдача через высокоприоритетный буфер
- `void I2S_TransferAbort` (`I2S_Type *base`, `i2s_handle_t *handle`)  
Отмена выдачи из высокоприоритетного буфера
- `void I2S_TransferHandleIRQ` (`I2S_Type *base`, `i2s_handle_t *handle`)  
Установка обработчика на прерывания от I2S, не связанных с приемом/выдачей



### 6.18.1 Подробное описание

Интерфейс драйвера модуля I2S.

## 6.19 hal\_i2s.h

[См. документацию.](#)

```

00001
00023 #ifndef HAL_I2S_H
00024 #define HAL_I2S_H
00025
00026 #ifdef __cplusplus
00027 extern "C" {
00028 #endif
00029
00030 #include "hal_common.h"
00031
00035 #define HAL_I2S_DRIVER_VERSION (MAKE_VERSION(0, 1, 0))
00036
00040 typedef enum i2s_status {
00041     I2S_Status_Ok = 0U,
00042     I2S_Status_Fail = 1U,
00043     I2S_Status_InvalidArgument = 2U,
00044     I2S_Status_TxBusy = 3U,
00045     I2S_Status_UnsupportedBitRate = 4U,
00046 } i2s_status_t;
00047
00051 typedef enum i2s_flag {
00052     I2S_FlagTxFifoEmpty = 4U,
00053     I2S_FlagTxFifoOverrun = 5U,
00054 } i2s_flag_t;
00055
00056 #if defined(__CC_ARM)
00057 #pragma anon_unions
00058 #endif
00059
00064 typedef enum i2s_sclk_per_sample {
00065     I2S_SclkCycles_16 = 0U,
00066     I2S_SclkCycles_24 = 1U,
00067     I2S_SclkCycles_32 = 2U,
00068 } i2s_sclk_per_sample_t;
00069
00077 typedef enum i2s_sclk_gating {
00078     I2S_NoSclkGating = 0U,
00079     I2S_SclkGatingCycles_12 = 1U,
00080     I2S_SclkGatingCycles_16 = 2U,
00081     I2S_SclkGatingCycles_20 = 3U,
00082     I2S_SclkGatingCycles_24 = 4U,
00083 } i2s_sclk_gating_t;
00084
00088 typedef enum i2s_resolution {
00089     I2S_ResolutionDefault = 0U,
00090     I2S_Resolution_12 = 1U,
00091     I2S_Resolution_16 = 2U,
00092     I2S_Resolution_20 = 3U,
00093     I2S_Resolution_24 = 4U,
00094     I2S_Resolution_32 = 5U,
00095 } i2s_resolution_t;
00096
00100 typedef enum i2s_interrupt_level {
00101     I2S_InterruptTriggerLevel_1 = 0U,
00102     I2S_InterruptTriggerLevel_2 = 1U,
00103     I2S_InterruptTriggerLevel_3 = 2U,
00104     I2S_InterruptTriggerLevel_4 = 3U,
00105     I2S_InterruptTriggerLevel_5 = 4U,
00106     I2S_InterruptTriggerLevel_6 = 5U,
00107     I2S_InterruptTriggerLevel_7 = 6U,
00108     I2S_InterruptTriggerLevel_8 = 7U,
00109     I2S_InterruptTriggerLevel_9 = 8U,
00110     I2S_InterruptTriggerLevel_10 = 9U,
00111     I2S_InterruptTriggerLevel_11 = 10U,
00112     I2S_InterruptTriggerLevel_12 = 11U,
00113     I2S_InterruptTriggerLevel_13 = 12U,
00114     I2S_InterruptTriggerLevel_14 = 13U,
00115     I2S_InterruptTriggerLevel_15 = 14U,
00116     I2S_InterruptTriggerLevel_16 = 15U,
00117 } i2s_interrupt_level_t;
00118
00122 typedef struct i2s_config {
00123     uint32_t sample_rate;

```

```

00124 i2s_sclk_per_sample_t sclk_per_sample;
00125 i2s_sclk_gating_t sclk_gating;
00126 i2s_resolution_t resolution;
00127 i2s_interrupt_level_t interrupt_level;
00128 } i2s_config_t;
00129
00134 typedef struct i2s_transfer {
00135     uint32_t *left_samples;
00136     uint32_t *right_samples;
00137     size_t nb_samples;
00138 } i2s_transfer_t;
00139
00143 typedef struct _i2s_handle i2s_handle_t;
00144
00153 typedef void (*i2s_transfer_callback_t)(I2S_Type *base, i2s_handle_t *handle,
00154     i2s_flag_t interrupt_flag, void *user_data);
00155
00159 struct i2s_handle {
00160     i2s_transfer_callback_t callback;
00161     void *user_data;
00163     uint32_t *left_samples;
00164     uint32_t *right_samples;
00165     size_t nb_samples;
00166 };
00167
00186 i2s_status_t I2S_Init(I2S_Type *base, const i2s_config_t *config,
00187     uint32_t source_clock_hz);
00188
00196 void I2S_Deinit(I2S_Type *base);
00197
00203 void I2S_GetDefaultConfig(i2s_config_t *config);
00204
00220 static inline void I2S_EnableInterrupt(I2S_Type *base, i2s_flag_t idx)
00221 {
00222     base->IMR0 &= ~(1 << idx);
00223 }
00224
00231 static inline void I2S_EnableInterruptMask(I2S_Type *base, uint32_t mask)
00232 {
00233     base->IMR0 &= ~mask;
00234 }
00235
00245 static inline bool I2S_IsInterruptEnabled(I2S_Type *base, i2s_flag_t idx)
00246 {
00247     return !(base->IMR0 & (1 << idx));
00248 }
00249
00250
00261 static inline uint32_t I2S_GetEnabledInterruptMask(I2S_Type *base)
00262 {
00263     return base->IMR0;
00264 }
00265
00272 static inline void I2S_DisableInterrupt(I2S_Type *base, i2s_flag_t idx)
00273 {
00274     base->IMR0 |= (1 << idx);
00275 }
00276
00283 static inline void I2S_DisableInterruptMask(I2S_Type *base, uint32_t mask)
00284 {
00285     base->IMR0 |= mask;
00286 }
00287
00297 static inline bool I2S_GetInterruptStatus(I2S_Type *base, i2s_flag_t idx)
00298 {
00299     return base->ISR0 & (1 << idx);
00300 }
00301
00309 static inline uint32_t I2S_GetInterruptStatusMask(I2S_Type *base)
00310 {
00311     return base->ISR0;
00312 }
00313
00319 static inline void I2S_ClearDataOverrunFlag(I2S_Type *base)
00320 {
00321     (void) base->TOR0;
00322 }
00323
00339 void I2S_WriteLeftFifo(I2S_Type *base, uint32_t sample);
00340
00347 void I2S_WriteRightFifo(I2S_Type *base, uint32_t sample);
00348
00354 static inline void I2S_AbortTx(I2S_Type *base)
00355 {
00356     base->TXFFR = (1 << I2S_TXFFR_TXFFR_Pos);
00357 }
00358

```

```

00364 static inline void I2S_EnableTx(I2S_Type *base)
00365 {
00366     SET_VAL_MSK(base->ITER, I2S_ITER_TXEN_Msk, I2S_ITER_TXEN_Pos, 1U);
00367     SET_VAL_MSK(base->CER, I2S_CER_CLKEN_Msk, I2S_CER_CLKEN_Pos, 1U);
00368 }
00369
00375 static inline void I2S_DisableTx(I2S_Type *base)
00376 {
00377     SET_VAL_MSK(base->CER, I2S_CER_CLKEN_Msk, I2S_CER_CLKEN_Pos, 0U);
00378     SET_VAL_MSK(base->ITER, I2S_ITER_TXEN_Msk, I2S_ITER_TXEN_Pos, 0U);
00379 }
00380
00401 i2s_status_t I2S_TransferCreateHandle(I2S_Type *base, i2s_handle_t *handle,
00402     i2s_transfer_callback_t callback, void *user_data);
00403
00419 i2s_status_t I2S_TransferNonBlocking(I2S_Type *base,
00420     i2s_handle_t *handle, i2s_transfer_t *xfer);
00421
00428 void I2S_TransferAbort(I2S_Type *base, i2s_handle_t *handle);
00429
00437 void I2S_TransferHandleIRQ(I2S_Type *base, i2s_handle_t *handle);
00438
00443 #ifdef __cplusplus
00444 }
00445 #endif
00446
00447 #endif /* HAL_I2S_H */
00448

```

## 6.20 Файл devices/eliot1/drivers/hal\_ioim.h

Интерфейс менеджера прерываний IO устройств

```
#include "hal_common.h"
```

Макросы

- `#define IOIM_NA_IRQ_NUM (-16)`

Перечисления

- `enum ioim_status_t`  
Возвращаемые статусы IOIM.

Функции

- `int32_t IOIM_GetIRQNumber (void *base)`  
Получение номера прерывания в системе
- `ioim_status_t IOIM_SetIRQHandler (void *base, void *handler, void *handle)`  
Установка обработчика прерывания для устройства IO.
- `ioim_status_t IOIM_ClearIRQHandler (void *base)`  
Сброс обработчика прерывания для устройства IO.
- `ioim_status_t IOIM_SetIRQHandler_DMA (void *base, uint32_t channel, void *handler, void *handle)`  
Установка обработчика прерывания для DMA.
- `ioim_status_t IOIM_ClearIRQHandler_DMA (void *base, uint32_t channel)`  
Сброс обработчика прерывания для DMA.

## 6.20.1 Подробное описание

Интерфейс менеджера прерываний IO устройств

## 6.21 hal\_ioim.h

[См. документацию.](#)

```

00001
00024 #ifndef HAL_IOIM_H
00025 #define HAL_IOIM_H
00026
00027 #if defined(__cplusplus)
00028 extern "C" {
00029 #endif /* __cplusplus */
00030
00031 #include "hal_common.h"
00032
00033 #define IOIM_NA_IRQ_NUM (-16)
00038 typedef enum {
00039     IOIM_Status_Ok = 0,
00040     IOIM_Status_UnknownBase = 1,
00041     IOIM_Status_NullHandler = 2,
00042 } ioim_status_t;
00043
00051 int32_t IOIM_GetIRQNumber(void *base);
00052
00068 ioim_status_t IOIM_SetIRQHandler(void *base, void *handler, void *handle);
00069
00081 ioim_status_t IOIM_ClearIRQHandler(void *base);
00082
00095 ioim_status_t IOIM_SetIRQHandler_DMA(void *base, uint32_t channel,
00096     void *handler, void *handle);
00097
00110 ioim_status_t IOIM_ClearIRQHandler_DMA(void *base, uint32_t channel);
00111
00112 #if defined(__cplusplus)
00113 }
00114 #endif /* __cplusplus */
00115
00116 #endif /* HAL_IOIM_H */
00117

```

## 6.22 hal\_jtm.h

```

00001
00022 #ifndef HAL_JTM_H
00023 #define HAL_JTM_H
00024
00025 #ifdef __cplusplus
00026 extern "C" {
00027 #endif
00028
00029 #include "hal_common.h"
00030
00034 #define HAL_JTM_DRIVER_VERSION (MAKE_VERSION(0, 1, 0))
00035
00039 typedef enum
00040 {
00041     JTM_Temperature = 0U,
00042     JTM_Vcasn = 4U,
00043     JTM_Vcore = 7U,
00044 } jtm_parameter_t;
00045
00049 typedef enum
00050 {
00051     JTM_Status_Ok = 0U,
00052     JTM_Status_Fail = 1U,
00053     JTM_Status_BadParameter = 2U,
00054     JTM_Status_Busy = 3U,
00055 } jtm_status_t;
00056
00060 typedef struct
00061 {
00062     int16_t tcal;
00063     int16_t wcal;
00064     int16_t wtcalf;

```

```

00065     int16_t wtconf;
00066 } jtm_config_t;
00067
00071 typedef struct _jtm_handle jtm_handle_t;
00072
00081 typedef void (*jtm_callback_t)(jtm_handle_t *handle, jtm_parameter_t parameter,
00082     int32_t value, void *user_data);
00083
00087 struct _jtm_handle {
00088     jtm_callback_t callback;
00089     jtm_parameter_t parameter;
00090     void *user_data;
00091 };
00092
00099 void JTM_Init(JTM_Type *base, jtm_config_t *config);
00100
00116 jtm_status_t JTM_GetParameterValue(JTM_Type *base, jtm_parameter_t parameter,
00117     int32_t *value);
00118
00130 jtm_status_t JTM_CreateHandle(JTM_Type *base, jtm_handle_t *handle,
00131     jtm_callback_t callback, void *user_data);
00132
00148 jtm_status_t JTM_GetParameterValueNonBlocking(JTM_Type *base,
00149     jtm_handle_t *handle, jtm_parameter_t parameter);
00150
00151 #ifdef __cplusplus
00152 }
00153 #endif
00154
00155 #endif /* HAL_JTM_H */
00156

```

## 6.23 Файл devices/eliot1/drivers/hal\_mhu.h

Интерфейс модуля программных прерываний MHU.

```
#include "hal_common.h"
```

### Функции

- static void [MHU\\_CPU0\\_SetInt](#) (MHU\_Type \*base, uint32\_t mask)  
Установка прерывания CPU0 по маске
- static void [MHU\\_CPU0\\_ClearInt](#) (MHU\_Type \*base, uint32\_t mask)  
Сброс прерывания CPU0 по маске
- static uint32\_t [MHU\\_CPU0\\_StatInt](#) (MHU\_Type \*base)  
Чтение статуса прерываний CPU0.
- static void [MHU\\_CPU1\\_SetInt](#) (MHU\_Type \*base, uint32\_t mask)  
Установка прерывания CPU1 по маске
- static void [MHU\\_CPU1\\_ClearInt](#) (MHU\_Type \*base, uint32\_t mask)  
Сброс прерывания CPU1 по маске
- static uint32\_t [MHU\\_CPU1\\_StatInt](#) (MHU\_Type \*base)  
Чтение статуса прерываний CPU1.

### 6.23.1 Подробное описание

Интерфейс модуля программных прерываний MHU.

## 6.24 hal\_mhu.h

См. документацию.

```

00001
00021 #ifndef HAL_MHU_H
00022 #define HAL_MHU_H
00023
00024 #ifdef __cplusplus
00025 extern "C" {
00026 #endif
00027
00028 #include "hal_common.h"
00029
00036 static inline void MHU_CPU0_SetInt(MHU_Type *base, uint32_t mask)
00037 {
00038     MHU_CPU0INTR_SET_CPU0INTR_SET_PTYPE_SET(base, mask);
00039 }
00040
00047 static inline void MHU_CPU0_ClearInt(MHU_Type *base, uint32_t mask)
00048 {
00049     MHU_CPU0INTR_CLR_CPU0INTR_CLR_PTYPE_SET(base, mask);
00050 }
00051
00059 static inline uint32_t MHU_CPU0_StatInt(MHU_Type *base)
00060 {
00061     return MHU_CPU0INTR_STAT_CPU0INTR_STAT_PTYPE_GET(base);
00062 }
00063
00070 static inline void MHU_CPU1_SetInt(MHU_Type *base, uint32_t mask)
00071 {
00072     MHU_CPU1INTR_SET_CPU1INTR_SET_PTYPE_SET(base, mask);
00073 }
00074
00081 static inline void MHU_CPU1_ClearInt(MHU_Type *base, uint32_t mask)
00082 {
00083     MHU_CPU1INTR_CLR_CPU1INTR_CLR_PTYPE_SET(base, mask);
00084 }
00085
00093 static inline uint32_t MHU_CPU1_StatInt(MHU_Type *base)
00094 {
00095     return MHU_CPU1INTR_STAT_CPU1INTR_STAT_PTYPE_GET(base);
00096 }
00097
00098 #ifdef __cplusplus
00099 }
00100 #endif
00101
00102 #endif /* HAL_MHU_H */
00103

```

## 6.25 Файл devices/eliot1/drivers/hal\_nor\_flash.h

Интерфейс драйвера флеш-памяти NOR.

```

#include "hal_common.h"
#include "hal_qspi.h"

```

Структуры данных

- struct [\\_nor\\_command\\_set](#)  
Основной набор команд для флеш-памяти NOR.
- struct [\\_nor\\_config](#)  
Структура первичной конфигурации флеш-памяти NOR.
- struct [\\_nor\\_handle](#)  
Контекст драйвера флеш-памяти NOR.

## Макросы

- `#define DUMMY_BYTE 0x00`

Поля внутреннего статусного регистра

- `#define FLASH_STAT_BUSY (1 << 0)`
- `#define FLASH_STAT_WEL (1 << 1)`

## Определения типов

- `typedef struct _nor_command_set nor_command_set_t`  
Основной набор команд для флеш-памяти NOR.
- `typedef struct _nor_config nor_config_t`  
Структура первичной конфигурации флеш-памяти NOR.
- `typedef struct _nor_handle nor_handle_t`  
Контекст драйвера флеш-памяти NOR.

## Перечисления

- `enum nor_status_t`  
Статусы драйвера флеш-памяти NOR.

## Функции

- `void QSPI_GetDefaultConfigXIP (qspi_xip_config_t *qspi_xip_config)`  
Получение конфигурации XIP QSPI по умолчанию
- `void QSPI_GetDefaultCommandSet (nor_command_set_t *command_set)`  
Получение стандартного набора команд SPI Flash.
- `nor_status_t QSPI_EnableXIP (nor_handle_t *handle)`  
Заполнение регистра XIPCFG контроллера QSPI.
- `void QSPI_DisableXIP (nor_handle_t *handle)`  
Выключение режима QSPI XIP.
- `nor_status_t QSPI_NorFlashInit (nor_config_t *config, nor_handle_t *handle)`  
Инициализация устройства флеш-памяти NOR.
- `nor_status_t NOR_FlashRead (nor_handle_t *handle, uint32_t address, uint8_t *buffer, uint32_t length)`  
Чтение данных с флеш-памяти NOR.
- `nor_status_t NOR_FlashPageProgram (nor_handle_t *handle, uint32_t address, uint8_t *buffer, uint32_t length)`  
Программирование страницы флеш-памяти NOR.
- `nor_status_t NOR_FlashEraseBlock (nor_handle_t *handle, uint32_t address, uint32_t size)`  
Очистка блока памяти
- `nor_status_t NOR_FlashEraseChip (nor_handle_t *handle)`  
Очистка чипа флеш-памяти NOR.
- `void NOR_FlashReadXIP (uint32_t address, uint8_t *buffer, uint32_t length)`  
Чтение в режиме XIP.
- `nor_status_t NOR_FlashProgramBlock (nor_handle_t *handle, uint32_t address, uint8_t *page_pointer, uint32_t length)`  
Заполнение блока флеш-памяти NOR.

### 6.25.1 Подробное описание

Интерфейс драйвера флеш-памяти NOR.

### 6.26 hal\_nor\_flash.h

[См. документацию.](#)

```

00001
00012 #ifndef HAL_NOR_FLASH_H
00013 #define HAL_NOR_FLASH_H
00014
00015 #ifdef __cplusplus
00016 extern "C" {
00017 #endif
00018
00019 #include "hal_common.h"
00020 #include "hal_qspi.h"
00021
00022 #if QSPI_USE_DMAC
00023 #include "hal_qspi_dma.h"
00024 #endif
00025
00026 #define DUMMY_BYTE 0x00
00033 #define FLASH_STAT_BUSY (1 < 0)
00034 #define FLASH_STAT_WEL (1 < 1)
00043 typedef struct _nor_command_set {
00044     uint8_t write_status_cmd;
00045     uint8_t page_write_memory_cmd;
00046     uint8_t read_memory_command;
00047     uint8_t write_disable_cmd;
00048     uint8_t read_status_cmd;
00049     uint8_t write_enable_cmd;
00050     uint8_t erase_sector_cmd;
00051     uint8_t erase_chip_cmd;
00052 } nor_command_set_t;
00053
00057 typedef enum {
00058     NOR_Status_Success = 0U,
00059     NOR_Status_Fail = 1U,
00060     NOR_Status_InvalidArgument = 2U,
00061     NOR_Status_Timeout = 3U,
00062 } nor_status_t;
00063
00067 typedef struct _nor_config {
00068     void *mem_control_config;
00069     void *quad_control_config;
00070     QSPI_Type *driver_base_addr;
00071 #if defined(QSPI_DRIVER_VERSION_1_6_0)
00072     QSPI_XIP_Type *driver_xip_base_addr;
00073 #endif
00074 } nor_config_t;
00075
00079 typedef struct _nor_handle {
00080     QSPI_Type *driver_base_addr;
00081 #if defined(QSPI_DRIVER_VERSION_1_6_0)
00082     QSPI_XIP_Type *driver_xip_base_addr;
00083 #endif
00084     uint32_t page_size_bytes;
00085     uint32_t sector_size_bytes;
00086     uint32_t memory_size_bytes;
00087     uint32_t max_sector_erase_time;
00088     uint32_t max_page_program_time;
00089     uint32_t max_chip_erase_time;
00090     void *device_specific;
00091     qspi_xip_config_t xip_config;
00092     qspi_config_t qspi_config;
00093 } nor_handle_t;
00094
00100 void QSPI_GetDefaultConfigXIP(qspi_xip_config_t *qspi_xip_config);
00101
00107 void QSPI_GetDefaultCommandSet(nor_command_set_t *command_set);
00108
00115 #if defined(QSPI_DRIVER_VERSION_1_6_0)
00116 nor_status_t QSPI_ConfigureXIP(QSPI_XIP_Type *base,
00117     qspi_xip_config_t *qspi_xip_config);
00118 #elif defined(QSPI_DRIVER_VERSION_2_0_0)
00119 nor_status_t QSPI_ConfigureXIP(QSPI_Type *base,
00120     qspi_xip_config_t *qspi_xip_config);
00121 #else
00122 #error "Please define QSPI controller version"

```



```

00123 #endif
00124
00130 nor_status_t QSPI_EnableXIP(nor_handle_t *handle);
00131
00137 void QSPI_DisableXIP(nor_handle_t *handle);
00138
00151 nor_status_t QSPI_NorFlashInit(nor_config_t *config, nor_handle_t *handle);
00152
00165 nor_status_t NOR_FlashRead(nor_handle_t *handle, uint32_t address,
00166     uint8_t *buffer, uint32_t length);
00167
00180 nor_status_t NOR_FlashPageProgram(nor_handle_t *handle, uint32_t address,
00181     uint8_t *buffer, uint32_t length);
00182
00195 nor_status_t NOR_FlashEraseBlock(nor_handle_t *handle, uint32_t address,
00196     uint32_t size);
00197
00207 nor_status_t NOR_FlashEraseChip(nor_handle_t *handle);
00208
00216 void NOR_FlashReadXIP(uint32_t address, uint8_t *buffer, uint32_t length);
00217
00230 nor_status_t NOR_FlashProgramBlock(nor_handle_t *handle,
00231     uint32_t address, uint8_t *page_pointer, uint32_t length);
00232
00233 #if QSPI_USE_DMACH
00234 void QSPI_SetDMAHandle(qspi_dma_handle_t *handle);
00235 #endif
00236
00237 #ifdef __cplusplus
00238 }
00239 #endif
00240
00241 #endif /* HAL_NOR_FLASH_H */
00242

```

## 6.27 Файл devices/eliot1/drivers/hal\_power.h

Интерфейс драйвера модуля POWER.

```
#include "hal_common.h"
```

Структуры данных

- struct [power\\_mode\\_config](#)  
Структура параметров режима питания
- struct [power\\_trim\\_config](#)  
Структура подстроечных параметров APC и DC-DC.
- struct [power\\_config](#)  
Структура конфигурации блока POWER.
- struct [power\\_state](#)  
Структура параметров состояния блока POWER.
- struct [power\\_handle](#)  
Структура обработчика драйвера I2S.

Определения типов

- typedef void(\* [power\\_callback\\_t](#)) (PWRCTR\_Type \*base, struct [power\\_handle](#) \*handle, uint8\_t interrupt\_mask, void \*user\_data)  
Функция обратного вызова для обработки прерывания POWER.

## Перечисления

- enum [power\\_status](#)  
Коды возврата функций драйвера POWER.
- enum [power\\_dcdc\\_vlevel](#)  
Перечисление выходных напряжений встроенного регулятора DC-DC.
- enum [power\\_dcdc\\_vlevel\\_value](#)  
Перечисление значений для уровней 0-2 выходных напряжений встроенного регулятора DC-DC.
- enum [power\\_dcdc\\_mode](#)  
Перечисление режимов работы встроенного регулятора DC-DC.
- enum [power\\_eco\\_mode](#)  
Перечисление режимов ECO DC-DC и APC.
- enum [power\\_dcdc\\_threshold](#)  
Перечисление пороговых напряжений DC-DC.
- enum [power\\_flash\\_mode](#)  
Перечисление пороговых напряжений DC-DC.
- enum [power\\_test\\_block](#)  
Перечисление блоков для тестирования
- enum [power\\_interrupt](#)  
Перечисление типов фронтов прерываний

## Функции

### Функции конфигурирования и чтения состояния

- void [POWER\\_GetCurrentConfig](#) (PWRCTR\_Type \*base, struct [power\\_config](#) \*config)  
Получение текущих значений параметров блока POWER.
- void [POWER\\_SetConfig](#) (PWRCTR\_Type \*base, struct [power\\_config](#) \*config)  
Установка параметров блока POWER.
- void [POWER\\_GetStatus](#) (PWRCTR\_Type \*base, struct [power\\_state](#) \*status)  
Установка параметров блока POWER.

### Функции управления прерываниями

- void [POWER\\_EnableInterrupt](#) (PWRCTR\_Type \*base, enum [power\\_interrupt](#) idx)  
Разрешение прерывания
- void [POWER\\_EnableInterruptMask](#) (PWRCTR\_Type \*base, uint8\_t mask)  
Разрешение прерываний по маске
- bool [POWER\\_IsInterruptEnabled](#) (PWRCTR\_Type \*base, enum [power\\_interrupt](#) idx)  
Запрос - разрешено ли прерывание
- uint8\_t [POWER\\_GetEnabledInterruptMask](#) (PWRCTR\_Type \*base)  
Запрос маски разрешенных прерываний
- void [POWER\\_DisableInterrupt](#) (PWRCTR\_Type \*base, enum [power\\_interrupt](#) idx)  
Запрет прерывания
- void [POWER\\_DisableInterruptMask](#) (PWRCTR\_Type \*base, uint8\_t mask)  
Запрет прерываний по маске
- bool [POWER\\_GetInterruptStatus](#) (PWRCTR\_Type \*base, enum [power\\_interrupt](#) idx)  
Получение состояния флага прерывания
- uint8\_t [POWER\\_GetInterruptStatusMask](#) (PWRCTR\_Type \*base)  
Получение маски активных прерываний
- void [POWER\\_ClearInterrupts](#) (PWRCTR\_Type \*base)  
Сброс признаков активных прерываний
- enum [power\\_status](#) [POWER\\_CreateHandle](#) (PWRCTR\_Type \*base, struct [power\\_handle](#) \*handle, [power\\_callback\\_t](#) callback, void \*user\_data)

Инициализация обработчика прерываний блока POWER.

Функции управления тестовыми режимами

- void `POWER_StartTestMode` (PWRCTR\_Type \*base, enum `power_test_block` test\_block)  
Запуск тестового режима
- void `POWER_StopTestMode` (PWRCTR\_Type \*base)  
Останов тестового режима

Функции переключения режимов

- void `POWER_DeepSleepThisCpu` ()  
Функция погружения ядра процессора в сон Погружается в сон то ядро, на котором выполнится данная функция.
- enum `power_status` `POWER_Standby` (PWRCTR\_Type \*base)  
Функция погружения процессора в сон
- enum `power_status` `POWER_Shutdown` (PWRCTR\_Type \*base)  
Функция погружения процессора в глубокий сон

### 6.27.1 Подробное описание

Интерфейс драйвера модуля POWER.

## 6.28 hal\_power.h

[См. документацию.](#)

```
00001
00020 #ifndef HAL_POWER_H
00021 #define HAL_POWER_H
00022
00023 #if defined(__cplusplus)
00024 extern "C" {
00025 #endif /* __cplusplus */
00026
00027 #include "hal_common.h"
00028
00032 enum power_status {
00033     POWER_Status_Ok = 0U,
00034     POWER_Status_Fail = 1U,
00035 };
00036
00040 enum power_dcdc_vlevel {
00041     POWER_DcdcVLevel0,
00042     POWER_DcdcVLevel1,
00043     POWER_DcdcVLevel2
00044 };
00045
00053 enum power_dcdc_vlevel_value {
00054     POWER_DcdcVLevel_0_85V,
00055     POWER_DcdcVLevel_0_86V,
00056     POWER_DcdcVLevel_0_87V,
00057     POWER_DcdcVLevel_0_88V,
00058     POWER_DcdcVLevel_0_89V,
00059     POWER_DcdcVLevel_0_90V,
00060     POWER_DcdcVLevel_0_91V,
00061     POWER_DcdcVLevel_0_92V,
00062     POWER_DcdcVLevel_0_93V,
00063     POWER_DcdcVLevel_0_94V,
00064     POWER_DcdcVLevel_0_95V,
00065     POWER_DcdcVLevel_0_96V,
00066     POWER_DcdcVLevel_0_97V,
00067     POWER_DcdcVLevel_0_98V,
00068     POWER_DcdcVLevel_0_99V,
00069     POWER_DcdcVLevel_1_00V,
00070     POWER_DcdcVLevel_1_01V,
00071     POWER_DcdcVLevel_1_02V,
00072     POWER_DcdcVLevel_1_03V,
```

```

00073 POWER_DcdcVLevel_1_04V,
00074 POWER_DcdcVLevel_1_05V,
00075 POWER_DcdcVLevel_1_06V,
00076 POWER_DcdcVLevel_1_07V,
00077 POWER_DcdcVLevel_1_08V,
00078 POWER_DcdcVLevel_1_09V,
00079 POWER_DcdcVLevel_1_10V,
00080 POWER_DcdcVLevel_1_11V,
00081 POWER_DcdcVLevel_1_12V,
00082 POWER_DcdcVLevel_1_13V,
00083 POWER_DcdcVLevel_1_14V,
00084 POWER_DcdcVLevel_1_15V,
00085 POWER_DcdcVLevel_1_16V
00086 };
00087
00091 enum power_dcdc_mode {
00092     POWER_DcdcModeAuto,
00093     POWER_DcdcModePwm,
00094     POWER_DcdcModePwmFccm,
00095     POWER_DcdcModePfm
00096 };
00097
00101 enum power_eco_mode {
00102     POWER_EcoOff,
00103     POWER_EcoDcdc,
00104     POWER_EcoReserved,
00105     POWER_EcoDcdcAndApc
00106 };
00107
00111 enum power_dcdc_threshold {
00112     POWER_DcdcThreshold_0_60V,
00113     POWER_DcdcThreshold_0_62V,
00114     POWER_DcdcThreshold_0_64V,
00115     POWER_DcdcThreshold_0_66V,
00116     POWER_DcdcThreshold_0_68V,
00117     POWER_DcdcThreshold_0_70V,
00118     POWER_DcdcThreshold_0_72V,
00119     POWER_DcdcThreshold_0_74V,
00120     POWER_DcdcThreshold_0_76V,
00121     POWER_DcdcThreshold_0_78V,
00122     POWER_DcdcThreshold_0_80V,
00123     POWER_DcdcThreshold_0_82V,
00124     POWER_DcdcThreshold_0_84V,
00125     POWER_DcdcThreshold_0_86V,
00126     POWER_DcdcThreshold_0_88V,
00127     POWER_DcdcThreshold_0_90V
00128 };
00129
00133 enum power_flash_mode {
00134     POWER_FlashModeNormal,
00135     POWER_FlashModeSleep,
00136     POWER_FlashModePowerDown
00137 };
00138
00142 enum power_test_block {
00143     POWER_TestBlockApc,
00144     POWER_TestBlockDcdc,
00145     POWER_TestBlockJtm,
00146     POWER_TestBlockRwc
00147 };
00148
00152 enum power_interrupt {
00153     POWER_VmonRising,
00154     POWER_VmonFalling,
00155 };
00156
00160 struct power_mode_config {
00161     enum power_dcdc_vlevel dcdc_level;
00162     enum power_dcdc_mode dcdc_mode;
00163     uint8_t dcdc_swdrv;
00164     bool dcdc_sink_enable;
00165     bool dcdc_ccm_enable;
00166     bool dcdc_low_consumption_enable;
00167     enum power_eco_mode eco_mode;
00168     bool apc_low_clk_enable;
00169     enum power_dcdc_threshold apc_eco_threshold;
00170     enum power_flash_mode flash_power_mode;
00171 };
00172
00176 struct power_trim_config {
00177     uint8_t apc_vref_it;
00178     uint8_t apc_vref_tt;
00179     uint8_t apc_vref_vt;
00180     bool apc_force_trim;
00181     uint8_t dcdc_imax;
00182     uint8_t dcdc_imin;
00183     uint8_t dcdc_trimlc;

```

```

00184 };
00185
00186
00190 struct power_config {
00191     struct power_mode_config    run_configuration;
00192     struct power_mode_config    standby_configuration;
00193     struct power_trim_config    trim_configuration;
00194     bool                        flash_low_voltage_read_enable;
00195     bool                        dcdc_enable;
00196     enum power_dcdc_vlevel_value vlevel0;
00197     enum power_dcdc_vlevel_value vlevel1;
00198     enum power_dcdc_vlevel_value vlevel2;
00199 };
00200
00204 struct power_state {
00205     bool                vdda_is_lower_threshold;
00206     bool                dcdc_is_ready;
00207     enum power_flash_mode flash_power_mode;
00208     bool                flash_low_voltage_read_enabled;
00209 };
00210
00211 struct power_handle;
00212
00221 typedef void (*power_callback_t)(PWRCTR_Type *base, struct power_handle *handle,
00222     uint8_t interrupt_mask, void *user_data);
00223
00227 struct power_handle {
00228     power_callback_t callback;
00229     void            *user_data;
00230 };
00231
00243 void POWER_GetCurrentConfig(PWRCTR_Type *base, struct power_config *config);
00244
00251 void POWER_SetConfig(PWRCTR_Type *base, struct power_config *config);
00252
00259 void POWER_GetStatus(PWRCTR_Type *base, struct power_state *status);
00260
00276 void POWER_EnableInterrupt(PWRCTR_Type *base, enum power_interrupt idx);
00277
00284 void POWER_EnableInterruptMask(PWRCTR_Type *base, uint8_t mask);
00285
00295 bool POWER_IsInterruptEnabled(PWRCTR_Type *base, enum power_interrupt idx);
00296
00308 uint8_t POWER_GetEnabledInterruptMask(PWRCTR_Type *base);
00309
00316 void POWER_DisableInterrupt(PWRCTR_Type *base, enum power_interrupt idx);
00317
00324 void POWER_DisableInterruptMask(PWRCTR_Type *base, uint8_t mask);
00325
00335 bool POWER_GetInterruptStatus(PWRCTR_Type *base, enum power_interrupt idx);
00336
00344 uint8_t POWER_GetInterruptStatusMask(PWRCTR_Type *base);
00345
00351 void POWER_ClearInterrupts(PWRCTR_Type *base);
00352
00364 enum power_status POWER_CreateHandle(PWRCTR_Type *base,
00365     struct power_handle *handle, power_callback_t callback, void *user_data);
00366
00382 void POWER_StartTestMode(PWRCTR_Type *base, enum power_test_block test_block);
00383
00389 void POWER_StopTestMode(PWRCTR_Type *base);
00390
00404 void POWER_DeepSleepThisCpu();
00405
00414 enum power_status POWER_Standby(PWRCTR_Type *base);
00415
00424 enum power_status POWER_Shutdown(PWRCTR_Type *base);
00425
00431 #if defined(__cplusplus)
00432 }
00433 #endif /* __cplusplus */
00434
00435 #endif /* HAL_POWER_H */
00436

```

## 6.29 Файл devices/eliot1/drivers/hal\_ppu.h

Интерфейс драйвера модуля PPU.

```
#include "hal_common.h"
```

## Структуры данных

- struct [ppu\\_config](#)  
Структура для инициализации

## Определения типов

- typedef void(\* [ReqOffForCPU](#)) (void)  
Тип функции запроса для выключения смежного ядра

## Перечисления

- enum [ppu\\_status](#)  
Статусы драйвера PPU.
- enum [ppu\\_domain\\_index](#)  
Индексы блоков PPU.
- enum [ppu\\_sense\\_index](#)  
Индексы бит блоков PPU, от которых зависят другие домены
- enum [ppu\\_power\\_mode](#)  
Состояние домена питания

## Регистры масок прерывания

- enum [ppu\\_event\\_name](#)  
Имена масок прерываний
- enum [ppu\\_add\\_event\\_name](#)  
Имена масок дополнительных прерываний

## Идентификационные регистры

- enum [ppu\\_opportunities\\_idr0](#)  
Идентификационный регистр PPU\_IDR0.
- enum [ppu\\_opportunities\\_idr1](#)  
Идентификационный регистр PPU\_IDR1.

## Функции

- void [PPU\\_StateOffRequestHandler](#) (void)  
Функция отключения питания текущего ядра

## Функции установки состояний

- enum [ppu\\_status](#) [PPU\\_SetState](#) (PPU\_Type \*base, enum [ppu\\_power\\_mode](#) mode)  
Функция запроса установки статического режима работы
- enum [ppu\\_status](#) [PPU\\_SetStateDynamic](#) (PPU\_Type \*base, enum [ppu\\_power\\_mode](#) mode)  
Функция запроса установки динамического режима работы
- enum [ppu\\_status](#) [PPU\\_SetPDCMPPUSense](#) (enum [ppu\\_domain\\_index](#) pd\_dst, enum [ppu\\_sense\\_index](#) pd\_src, uint32\_t sense)  
Функция установки зависимости доменов питания
- enum [ppu\\_status](#) [PPU\\_SetIRQStatus](#) (PPU\_Type \*base, enum [ppu\\_event\\_name](#) irq, enum [ppu\\_add\\_event\\_name](#) add\_irq)  
Функция установки состояний прерываний

- enum `ppu_status` `PPU_SetIRQMask` (`PPU_Type *base`, enum `ppu_event_name` `irq`, enum `ppu_add_event_name` `add_irq`)  
Функция установки масок прерываний
- enum `ppu_status` `PPU_ClrIRQStatus` (`PPU_Type *base`, enum `ppu_event_name` `irq`, enum `ppu_add_event_name` `add_irq`)  
Функция сброса состояний прерываний
- enum `ppu_status` `PPU_ClrIRQMask` (`PPU_Type *base`, enum `ppu_event_name` `irq`, enum `ppu_add_event_name` `add_irq`)  
Функция сброса масок прерываний

#### Функции получения состояния

- enum `ppu_power_mode` `PPU_GetPowerState` (`PPU_Type *base`)  
Функция получения состояния домена
- uint32\_t `PPU_GetPDxSenseFromPDy` (enum `ppu_domain_index` `pd_dst`, enum `ppu_sense_index` `pd_src`)  
Функция получения зависимости доменов питания
- enum `ppu_status` `PPU_GetLastAPIStatus` (void)  
Получение статуса выполнения функции, тип результата которой отличен от enum `ppu_status`.
- enum `ppu_status` `PPU_GetIRQStatus` (`PPU_Type *base`, enum `ppu_event_name` `*irq`, enum `ppu_add_event_name` `*add_irq`)  
Получение статуса прерываний
- enum `ppu_status` `PPU_GetIRQMask` (`PPU_Type *base`, enum `ppu_event_name` `*irq`, enum `ppu_add_event_name` `*add_irq`)  
Получение масок прерываний

#### Функции инициализации

- enum `ppu_status` `PPU_Init` (`PPU_Type *base`, struct `ppu_config` `*cfg`)  
Функция инициализации блока PPU.

### 6.29.1 Подробное описание

Интерфейс драйвера модуля PPU.

## 6.30 hal\_ppu.h

См. документацию.

```

00001
00020 #ifndef HAL_PPU_H
00021 #define HAL_PPU_H
00022
00023 #if defined(__cplusplus)
00024 extern "C" {
00025 #endif /* __cplusplus */
00026
00027 #include "hal_common.h"
00028
00032 enum ppu_status {
00033     PPU_Status_Ok = 0,
00034     PPU_Status_InvalidArgument = 1,
00035     PPU_Status_FeatureNotSupport = 2,
00036     PPU_Status_DriverError = 3,
00037     PPU_Status_ConfigError = 4,
00038 };
00039
00043 enum ppu_domain_index {
00044     PPU_DomainCPU0 = 0,
00045     PPU_DomainCPU1 = 1,
00046     PPU_DomainDEBUG = 2,
00047     PPU_DomainCRYPTO = 3,

```

```

00048 PPU_DomainGMS      = 4,
00049 PPU_DomainGNSS     = 5,
00050 PPU_DomainSRAM0     = 6,
00051 PPU_DomainSRAM1     = 7,
00052 PPU_DomainSRAM2     = 8,
00053 PPU_DomainSRAM3     = 9,
00054 PPU_DomainSYS       = 10,
00056 PPU_DomainMax      = 10,
00057 };
00058
00062 enum ppu_sense_index {
00063     PPU_SenseCPU0     = 1,
00064     PPU_SenseCPU1     = 2,
00065     PPU_SenseCRYPTO    = 12,
00066     PPU_SenseGMS      = 16,
00067     PPU_SenseGNSS     = 17,
00068     PPU_SenseSRAM0    = 3,
00069     PPU_SenseSRAM1    = 4,
00070     PPU_SenseSRAM2    = 5,
00071     PPU_SenseSRAM3    = 6,
00072     PPU_SenseSYS      = 0,
00073 };
00074
00080 enum ppu_power_mode {
00081     PPU_PowerModeOn    = 8,
00083     PPU_PowerModeOff   = 0,
00085     PPU_PowerModeMemRet = 2,
00086     /* Состояния не реализованы в первых ревизиях чипа */
00087     PPU_PowerModeWarmRst = 9,
00089     PPU_PowerModeDbgRecov = 10,
00097     PPU_PowerModeFuncRet = 7,
00099     PPU_PowerModeMemOff = 6,
00101     PPU_PowerModeFullRet = 5,
00102     PPU_PowerModeLogicRet = 4,
00104     PPU_PowerModeMemRetEmu = 3,
00108     PPU_PowerModeOffEmu = 1,
00113     PPU_PowerModeMax    = 10,
00114 };
00115
00124 enum ppu_event_name {
00125     PPU_Locked          = 0x00000020,
00126     PPU_EmuDeny         = 0x00000010,
00128     PPU_EmuAccept       = 0x00000008,
00130     PPU_StaDeny         = 0x00000004,
00132     PPU_StaAccept       = 0x00000002,
00134     PPU_StaPolicyTrn    = 0x00000001,
00137     PPU_EventNameAll = PPU_Locked |
00138                     PPU_EmuDeny |
00139                     PPU_EmuAccept |
00140                     PPU_StaDeny |
00141                     PPU_StaAccept |
00142                     PPU_StaPolicyTrn
00143                     ,
00144 };
00145
00149 enum ppu_add_event_name {
00150     PPU_StaPolicyOp      = 0x00000010,
00152     PPU_STA_PolicyPwr    = 0x00000008,
00154     PPU_DynDeny          = 0x00000004,
00156     PPU_DynAccept        = 0x00000002,
00158     PPU_UnsptPolicy      = 0x00000001,
00161     PPU_AddEventNameAll = PPU_StaPolicyOp |
00162                     PPU_STA_PolicyPwr |
00163                     PPU_DynDeny |
00164                     PPU_DynAccept |
00165                     PPU_UnsptPolicy
00166                     ,
00168 };
00169
00183 enum ppu_opportunities_idr0 {
00184     PPU_DYN_WRM_RST_SPT    = 1 « 29,
00185     PPU_DYN_ON_SPT         = 1 « 28,
00186     PPU_DYN_FUNC_RET_SPT   = 1 « 27,
00187     PPU_DYN_FULL_RET_SPT   = 1 « 26,
00188     PPU_DYN_MEM_OFF_SPT    = 1 « 25,
00189     PPU_DYN_LGC_RET_SPT    = 1 « 24,
00190     PPU_DYN_MEM_RET_EMU_SPT = 1 « 23,
00191     PPU_DYN_MEM_RET_SPT    = 1 « 22,
00192     PPU_DYN_OFF_EMU_SPT    = 1 « 21,
00193     PPU_DYN_OFF_SPT        = 1 « 20,
00194     PPU_STA_DBG_RECOV_SPT  = 1 « 18,
00195     PPU_STA_WRM_RST_SPT    = 1 « 17,
00196     PPU_STA_ON_SPT         = 1 « 16,
00197     PPU_STA_FUNC_RET_SPT   = 1 « 15,
00198     PPU_STA_FULL_RET_SPT   = 1 « 14,
00199     PPU_STA_MEM_OFF_SPT    = 1 « 13,
00200     PPU_STA_LGC_RET_SPT    = 1 « 12,

```



```

00201 PPU_STA_MEM_RET_EMU_SPT = 1 < 11,
00202 PPU_STA_MEM_RET_SPT = 1 < 10,
00203 PPU_STA_OFF_EMU_SPT = 1 < 9,
00204 PPU_STA_OFF_SPT = 1 < 8,
00205 };
00206
00210 enum ppu_opportunities_idr1 {
00211 PPU_OFF_MEM_RET_TRANS = 1 < 12,
00212 PPU_OP_ACTIVE = 1 < 10,
00214 PPU_STA_POLICY_OP_IRQ_SPT = 1 < 9,
00216 PPU_STA_POLICY_PWR_IRQ_SPT = 1 < 8,
00218 PPU_FUNC_RET_RAM_REG = 1 < 6,
00221 PPU_FULL_RET_RAM_REG = 1 < 5,
00223 PPU_MEM_RET_RAM_REG = 1 < 4,
00225 PPU_LOCK_SPT = 1 < 2,
00227 PPU_SW_DEV_DEL_SPT = 1 < 1,
00229 PPU_PWR_MODE_ENTRY_DEL_SPT = 1 < 0,
00231 };
00232
00245 typedef void (*ReqOffForCPU)(void);
00246
00250 struct ppu_config {
00251     ReqOffForCPU reqForCPU;
00252 };
00253
00272 enum ppu_status PPU_SetState(PPU_Type * base, enum ppu_power_mode mode);
00273
00289 enum ppu_status PPU_SetStateDynamic(PPU_Type * base, enum ppu_power_mode mode);
00290
00301 enum ppu_status PPU_SetPDCMPPUSense(enum ppu_domain_index pd_dst,
00302     enum ppu_sense_index pd_src, uint32_t sense);
00303
00314 enum ppu_status PPU_SetIRQStatus(PPU_Type * base, enum ppu_event_name irq,
00315     enum ppu_add_event_name add_irq);
00316
00327 enum ppu_status PPU_SetIRQMask(PPU_Type * base, enum ppu_event_name irq,
00328     enum ppu_add_event_name add_irq);
00329
00340 enum ppu_status PPU_ClrIRQStatus(PPU_Type * base, enum ppu_event_name irq,
00341     enum ppu_add_event_name add_irq);
00342
00353 enum ppu_status PPU_ClrIRQMask(PPU_Type * base, enum ppu_event_name irq,
00354     enum ppu_add_event_name add_irq);
00355
00373 enum ppu_power_mode PPU_GetPowerState(PPU_Type * base);
00374
00386 uint32_t PPU_GetPDxSenseFromPDy(enum ppu_domain_index pd_dst,
00387     enum ppu_sense_index pd_src);
00388
00395 enum ppu_status PPU_GetLastAPIStatus(void);
00396
00406 enum ppu_status PPU_GetIRQStatus(PPU_Type * base, enum ppu_event_name *irq,
00407     enum ppu_add_event_name *add_irq);
00408
00418 enum ppu_status PPU_GetIRQMask(PPU_Type * base, enum ppu_event_name *irq,
00419     enum ppu_add_event_name *add_irq);
00438 enum ppu_status PPU_Init(PPU_Type * base, struct ppu_config * cfg);
00439
00463 void PPU_StateOffRequestHandler(void);
00464
00465 #if defined(__cplusplus)
00466 }
00467 #endif /* __cplusplus */
00468
00469 #endif /* HAL_PPU_H */
00470

```

## 6.31 Файл devices/eliot1/drivers/hal\_pwm.h

Интерфейс драйвера модуля широтно-импульсного модулятора

```
#include "hal_common.h"
```

Структуры данных

- struct `pwm_channel_config`

Конфигурация канала широтно-импульсного модулятора

## Макросы

- `#define PWM_COUNT` (3)

## Перечисления

- enum `pwm_status`  
Статусы драйвера широтно-импульсного модулятора
- enum `pwm_prescaler_mode`  
Управление режимом работы предделителя канала
- enum `pwm_prescaler_cmd`  
Управление состоянием предделителя канала
- enum `pwm_run_command`  
Управление пуском/остановкой канала
- enum `pwm_prescaler_divmux`  
Управление мультиплексором делителя частоты (деление частоты после делителя)
- enum `pwm_prescaler_syncrst`  
Разрешения сброса предделителя при возникновении событий SYNCI или SWFSYNC.
- enum `pwm_dirsync`  
Направление счета после синхронизации
- enum `pwm_syncosel`  
Выбор источника выходного сигнала SYNCO.
- enum `pwm_loadprd`  
Управление моментом переписи данных из теневого регистра периода в активный
- enum `pwm_syncphsen`  
Сигнал разрешения загрузки счетчика из регистра фазы
- enum `pwm_cntmode`  
Режим работы счетчика CRTCNT.
- enum `pwm_scmpxmode`  
Режим работы регистра CMPx.
- enum `pwm_ldxmode`  
Выбор режима загрузки данных из теневого регистра в активный CMPx.
- enum `pwm_outx_cmd`  
Управление выходом OUTx.
- enum `pwm_ldcswrf`  
Механизм загрузки активного регистра из теневого регистра для регистра программного управления выходами
- enum `pwm_dz_signal`  
Источник сигнала для генерации запрещенной зоны
- enum `pwm_dz_outx_inv`  
Полярность OUTx после генерации запрещенной зоны
- enum `pwm_dz_mode`  
Выбор режима работы блока запрещенной зоны при формировании OUTx.
- enum `pwm_chopper_duty`  
Скважность дробящего сигнала
- enum `pwm_chopper_freq`  
Частота дробящего сигнала
- enum `pwm_chopper_first_width`  
Ширина первого импульса
- enum `pwm_chopper_work`  
Работа блока Chopper.

- enum [pwm\\_trip\\_unit\\_signal](#)  
Работа блока trip unit.
- enum [pwm\\_trip\\_unit\\_action](#)  
Реакции на событие блока trip unit.
- enum [pwm\\_int\\_en](#)  
Разрешение прерывания блока
- enum [pwm\\_eventprd](#)  
Выбор периода прерываний PWM\_INT.
- enum [pwm\\_int\\_source](#)  
Источник прерывания

## Функции

### Интерфейс драйвера

- enum [pwm\\_status PWM\\_GetChannelDefaultConfig](#) (struct [pwm\\_channel\\_config](#) \*cfg)  
Инициализация структуры "по умолчанию" для канала блока ШИМ
- enum [pwm\\_status PWM\\_InitChannel](#) (PWM\_Type \*base, struct [pwm\\_channel\\_config](#) cfg)  
Инициализация канала блока широтно-импульсного модулятора
- enum [pwm\\_status PWM\\_Deinit](#) (PWM\_Type \*base)  
Деинициализация блока широтно-импульсного модулятора
- enum [pwm\\_status PWM\\_Enable](#) (PWM\_Type \*base, uint32\_t channel, enum [pwm\\_run\\_command](#) cmd)  
Запуск/останов канала блока широтно-импульсного модулятора
- enum [pwm\\_status PWM\\_CmdForAllChannels](#) (PWM\_Type \*base, uint32\_t channel\_mask, enum [pwm\\_run\\_command](#) cmd0, enum [pwm\\_run\\_command](#) cmd1, enum [pwm\\_run\\_command](#) cmd2, enum [pwm\\_run\\_command](#) cmd3)  
Запуск/останов всех каналов блока широтно-импульсного модулятора
- enum [pwm\\_status PWM\\_ApplySoftOuts](#) (PWM\_Type \*base, uint32\_t channel, int8\_t mask↔\_outs)  
Программная не длительная установка значения выходов канала
- enum [pwm\\_status PWM\\_ApplyLongSoftOuts](#) (PWM\_Type \*base, uint32\_t channel, int8\_t mask\_outs, enum [pwm\\_outx\\_cmd](#) outa, enum [pwm\\_outx\\_cmd](#) outb)  
Программная длительная установка значения выходов канала
- uint32\_t [PWM\\_GetCntStat](#) (PWM\_Type \*base, uint32\_t channel)  
Определяет работает ли счетчик в канале
- enum [pwm\\_status PWM\\_SetPeriod](#) (PWM\_Type \*base, uint32\_t channel, uint32\_t period)  
Устанавливает значение периода

### 6.31.1 Подробное описание

Интерфейс драйвера модуля широтно-импульсного модулятора

## 6.32 hal\_pwm.h

[См. документацию.](#)

```
00001
00021 #ifndef HAL_PWM_H
00022 #define HAL_PWM_H
00023
00024 #ifdef __cplusplus
00025 extern "C" {
00026 #endif
00027
00028 #include "hal_common.h"
```

```

00029
00030 #define PWM_COUNT (3)
00035 enum pwm_status {
00036     PWM_Status_Ok          = 0,
00037     PWM_Status_InvalidArgument = 1,
00038     PWM_Status_BadConfigure  = 2,
00039 };
00040
00044 enum pwm_prescaler_mode {
00045     pwm_PrescModeTimerIsRun = 0,
00046     pwm_PrescModeAlways     = 1,
00048     pwm_PrescMax             = pwm_PrescModeAlways,
00049 };
00050
00054 enum pwm_prescaler_cmd {
00055     pwm_PrescCmdReset = 0,
00056     pwm_PrescCmdSave  = 1,
00058     pwm_PrescalerMax   = pwm_PrescCmdSave,
00059 };
00060
00064 enum pwm_run_command {
00065     pwm_RunCmdStop      = 0,
00066     pwm_RunCmdStopEvent = 1,
00070     pwm_RunCmdRun       = 2,
00072     pwm_RunCmdMax       = pwm_RunCmdRun,
00073 };
00074
00078 enum pwm_prescaler_divmux {
00079     pwm_PrescalerDivMux1  = 0,
00080     pwm_PrescalerDivMux2  = 1,
00081     pwm_PrescalerDivMux4  = 2,
00082     pwm_PrescalerDivMux8  = 3,
00083     pwm_PrescalerDivMux16 = 4,
00084     pwm_PrescalerDivMuxPWMClk = 5,
00086     pwm_PrescalerDivMuxMax = pwm_PrescalerDivMuxPWMClk,
00087 };
00088
00092 enum pwm_prescaler_syncrst {
00093     pwm_PrescalerSyncRstDis = 0,
00094     pwm_PrescalerSyncRstEn  = 1,
00096     pwm_PrescalerSyncRstEnMax = pwm_PrescalerSyncRstEn,
00097 };
00098
00103 enum pwm_dirsync {
00104     pwm_DirSyncDown = 0,
00105     pwm_DirSyncUp   = 1,
00107     pwm_DirSyncMax  = pwm_DirSyncUp,
00108 };
00109
00113 enum pwm_syncosel {
00114     pwm_SyncoSelSyncl = 0,
00115     pwm_SyncoSelCtrcntZero = 1,
00116     pwm_SyncoSelCtrcntEquCmpb = 2,
00117     pwm_SyncoSelOff     = 3,
00119     pwm_SyncoSelMax     = pwm_SyncoSelOff,
00120 };
00121
00125 enum pwm_loadprd {
00126     pwm_LoadPrdRegister = 0,
00130     pwm_LoadPrdDirect  = 1,
00134     pwm_LoadPrdMax     = pwm_LoadPrdDirect,
00135 };
00136
00140 enum pwm_syncphsen {
00141     pwm_SyncPhsEnDis = 0,
00142     pwm_SyncPhsEnEn  = 1,
00144     pwm_SyncPhsEnMax = pwm_SyncPhsEnEn,
00145 };
00146
00150 enum pwm_cntmode {
00151     pwm_CntModeUp      = 0,
00152     pwm_CntModeDown    = 1,
00153     pwm_CntModeUpDown = 2,
00154     pwm_CntModeOff     = 3,
00156     pwm_CntModeMax     = pwm_CntModeOff,
00157 };
00158
00162 enum pwm_scmpxmode {
00163     pwm_SCmpxModeReg    = 0,
00165     pwm_SCmpxModeDirect = 1,
00168     pwm_SCmpxModeMax    = pwm_SCmpxModeDirect,
00169 };
00170
00174 enum pwm_ldxmode {
00175     pwm_LdxModeCtrcntZero = 0,
00176     pwm_LdxModeCtrcntEquCtrprd = 1,
00177     pwm_LdxModeCtrcntZeroOrEquCtrprd = 2,

```

```

00178     pwm_LdxModeNoLoad          = 3,
00180     pwm_LdxModeMax             = pwm_LdxModeNoLoad,
00181 };
00182
00186 enum pwm_outx_cmd {
00187     pwm_OutxCmdNo      = 0,
00188     pwm_OutxCmdClear   = 1,
00189     pwm_OutxCmdSet      = 2,
00190     pwm_OutxCmdToggle  = 3,
00192     pwm_OutxCmdMax     = pwm_OutxCmdToggle,
00193 };
00194
00198 enum pwm_ldcswrf {
00199     pwm_LdcswrfCtrcntZero      = 0,
00200     pwm_LdcswrfCtrcntEquCtrprd = 1,
00201     pwm_LdcswrfCtrcntZeroEquCtrprd = 2,
00202     pwm_LdcswrfDirect          = 3,
00204     pwm_LdcswrfMax             = pwm_LdcswrfDirect,
00205 };
00206
00210 enum pwm_dz_signal {
00211     pwm_DzSignalOutA = 0,
00212     pwm_DzSignalOutB = 1,
00214     pwm_DzSignalMax  = pwm_DzSignalOutB,
00215 };
00216
00220 enum pwm_dz_outx_inv {
00221     pwm_DzSignalOutxInvOff = 0,
00222     pwm_DzSignalOutxInvOn  = 1,
00224     pwm_DzSignalOutxInvMax = pwm_DzSignalOutxInvOn,
00225 };
00226
00230 enum pwm_dz_mode {
00231     pwm_DzModeOff = 0,
00232     pwm_DzModeOn  = 1,
00234     pwm_DzModeMax = pwm_DzModeOn,
00235 };
00236
00240 enum pwm_chopper_duty {
00241     pwm_ChopperDuty_1_8 = 0,
00242     pwm_ChopperDuty_2_8 = 1,
00243     pwm_ChopperDuty_3_8 = 2,
00244     pwm_ChopperDuty_4_8 = 3,
00245     pwm_ChopperDuty_5_8 = 4,
00246     pwm_ChopperDuty_6_8 = 5,
00247     pwm_ChopperDuty_7_8 = 6,
00249     pwm_chopper_dutyMax = pwm_ChopperDuty_7_8,
00250 };
00251
00255 enum pwm_chopper_freq {
00256     pwm_ChopperFreqClk_8  = 0,
00257     pwm_ChopperFreqClk_16 = 1,
00258     pwm_ChopperFreqClk_24 = 2,
00259     pwm_ChopperFreqClk_32 = 3,
00260     pwm_ChopperFreqClk_40 = 4,
00261     pwm_ChopperFreqClk_48 = 5,
00262     pwm_ChopperFreqClk_56 = 6,
00263     pwm_ChopperFreqClk_64 = 7,
00265     pwm_ChopperFreqMax    = pwm_ChopperFreqClk_64,
00266 };
00267
00271 enum pwm_chopper_first_width {
00272     pwm_ChopperFirstWidth_0_8 = 0,
00273     pwm_ChopperFirstWidth_1_8 = 1,
00274     pwm_ChopperFirstWidth_2_8 = 2,
00275     pwm_ChopperFirstWidth_3_8 = 3,
00276     pwm_ChopperFirstWidth_4_8 = 4,
00277     pwm_ChopperFirstWidth_5_8 = 5,
00278     pwm_ChopperFirstWidth_6_8 = 6,
00279     pwm_ChopperFirstWidth_7_8 = 7,
00280     pwm_ChopperFirstWidth_8_8 = 8,
00281     pwm_ChopperFirstWidth_9_8 = 9,
00282     pwm_ChopperFirstWidth_10_8 = 10,
00283     pwm_ChopperFirstWidth_11_8 = 11,
00284     pwm_ChopperFirstWidth_12_8 = 12,
00285     pwm_ChopperFirstWidth_13_8 = 13,
00286     pwm_ChopperFirstWidth_14_8 = 14,
00287     pwm_ChopperFirstWidth_15_8 = 15,
00289     pwm_ChopperFirstWidthMax    = pwm_ChopperFirstWidth_15_8,
00290 };
00291
00295 enum pwm_chopper_work {
00296     pwm_ChopperWorkOff = 0,
00297     pwm_ChopperWorkOn  = 1,
00299     pwm_ChopperWorkMax = pwm_ChopperWorkOn,
00300 };
00301

```

```

00305 enum pwm_trip_unit_signal {
00306     pwm_TripUnitSignalNotUsed = 0,
00307     pwm_TripUnitSignalUsed    = 1,
00309     pwm_TripUnitMax           = pwm_TripUnitSignalUsed,
00310 };
00311
00315 enum pwm_trip_unit_action {
00316     pwm_TripUnitActionHigh = 0,
00317     pwm_TripUnitActionOne  = 1,
00318     pwm_TripUnitActionZero = 2,
00319     pwm_TripUnitActionNo   = 3,
00321     pwm_TripUnitActionMax  = pwm_TripUnitActionNo,
00322 };
00323
00327 enum pwm_int_en {
00328     pwm_IntEnNo = 0,
00329     pwm_IntEnYes = 1,
00331     pwm_IntEnMax = pwm_IntEnYes,
00332 };
00333
00337 enum pwm_eventprd {
00338     pwm_EventPrdNo    = 0,
00339     pwm_EventPrdOne   = 1,
00340     pwm_EventPrdTwo   = 2,
00341     pwm_EventPrdThree = 3,
00343     pwm_EventPrd      = pwm_EventPrdThree,
00344 };
00345
00349 enum pwm_int_source {
00350     pwm_IntSourceNo           = 0,
00351     pwm_IntSourceCtrcntEquZero = 2,
00352     pwm_IntSourceCtrcntEquCtrprd = 3,
00353     pwm_IntSourceCtrcntEquCmpAInc = 4,
00354     pwm_IntSourceCtrcntEquCmpADec = 5,
00355     pwm_IntSourceCtrcntEquCmpBInc = 6,
00356     pwm_IntSourceCtrcntEquCmpBDec = 7,
00358     pwm_IntSourceMax           = pwm_IntSourceCtrcntEquCmpBDec,
00359 };
00360
00364 struct pwm_channel_config {
00365     uint32_t channel;
00366     /* Группа 1 */
00367     /* Предделитель */
00368     enum pwm_prescaler_mode    prescaler_mode;
00369     enum pwm_prescaler_cmd     prescaler_cmd;
00370     enum pwm_prescaler_syncrst prescaler_syncrst;
00371     uint8_t                    prescaler;
00372     enum pwm_prescaler_divmux  prescaler_divmux;
00373     /* Основной счетчик */
00374     enum pwm_cntmode           cntmode;
00375     uint32_t                   counter;
00376     uint32_t                   period;
00377     enum pwm_loadprd           loadprd;
00378     uint32_t                   ctrphs;
00379     enum pwm_syncphsen         syncphsen;
00380     /* Блок сравнения */
00381     uint32_t                   cmpa;
00382     uint32_t                   cmpb;
00383     enum pwm_scmpxmode         scmpamode;
00384     enum pwm_scmpxmode         scmpbmode;
00385     enum pwm_ldxmode           ldamode;
00386     enum pwm_ldxmode           ldbmode;
00387     /* Группа 2 */
00388     /* Блок реакции на событие */
00389     enum pwm_outx_cmd          cnt_eq_prd_outa;
00390     enum pwm_outx_cmd          cnt_eq_prd_outb;
00391     enum pwm_outx_cmd          cnt_eq_cmpa_dec_outa;
00392     enum pwm_outx_cmd          cnt_eq_cmpa_inc_outa;
00393     enum pwm_outx_cmd          cnt_eq_cmpa_dec_outb;
00394     enum pwm_outx_cmd          cnt_eq_cmpa_inc_outb;
00395     enum pwm_outx_cmd          cnt_eq_cmpb_dec_outa;
00396     enum pwm_outx_cmd          cnt_eq_cmpb_inc_outa;
00397     enum pwm_outx_cmd          cnt_eq_cmpb_dec_outb;
00398     enum pwm_outx_cmd          cnt_eq_cmpb_inc_outb;
00399     enum pwm_outx_cmd          cnt_eq_zero_outa;
00400     enum pwm_outx_cmd          cnt_eq_zero_outb;
00401     enum pwm_outx_cmd          sw_forced_outa;
00402     enum pwm_outx_cmd          sw_forced_outb;
00403     enum pwm_outx_cmd          sw_forced_long_outa;
00404     enum pwm_outx_cmd          sw_forced_long_outb;
00405     enum pwm_ldcswrf           ldcswrf;
00406     /* Блок прерываний */
00407     enum pwm_int_en            pwm_int_enable;
00408     enum pwm_int_source        pwm_int_source;
00409     enum pwm_eventprd          eventprd;
00410     /* Группа 3 */
00411     /* Генератор запретной зоны */

```

```

00412     uint16_t          dz_rising_edge_delay_clk;
00413     uint16_t          dz_falling_edge_delay_clk;
00414     enum pwm_dz_signal dz_rising_edge_source;
00415     enum pwm_dz_signal dz_falling_edge_source;
00416     enum pwm_dz_outx_inv dz_rising_edge_outa_inv;
00417     enum pwm_dz_outx_inv dz_falling_edge_outb_inv;
00418     enum pwm_dz_mode     dz_outa_enable;
00419     enum pwm_dz_mode     dz_outb_enable;
00420     /* Блок дробления выходного сигнала */
00421     enum pwm_chopper_duty chopper_duty;
00422     enum pwm_chopper_freq chopper_freq;
00423     enum pwm_chopper_first_width chopper_first_width;
00424     enum pwm_chopper_work chopper_work;
00425     /* Блок реакции на внешнее воздействие */
00426     uint8_t          inputs_mask_one;
00427     uint8_t          inputs_mask_mult;
00428     enum pwm_trip_unit_action trip_unit_action_outa;
00429     enum pwm_trip_unit_action trip_unit_action_outb;
00430     enum pwm_int_en     pwmtu_int_one;
00431     enum pwm_int_en     pwmtu_int_mult;
00432     /* Запуск/останов канала */
00433     enum pwm_run_command cmd;
00434 };
00435
00436
00450 enum pwm_status PWM_GetChannelDefaultConfig(struct pwm_channel_config * cfg);
00451
00461 enum pwm_status PWM_InitChannel(PWM_Type *base,
00462     struct pwm_channel_config cfg);
00463
00472 enum pwm_status PWM_Deinit(PWM_Type *base);
00473
00484 enum pwm_status PWM_Enable(PWM_Type *base, uint32_t channel, enum pwm_run_command cmd);
00485
00499 enum pwm_status PWM_CmdForAllChannels(PWM_Type *base,
00500     uint32_t channel_mask,
00501     enum pwm_run_command cmd0,
00502     enum pwm_run_command cmd1,
00503     enum pwm_run_command cmd2,
00504     enum pwm_run_command cmd3);
00505
00521 enum pwm_status PWM_ApplySoftOuts(PWM_Type *base, uint32_t channel,
00522     int8_t mask_outs);
00523
00540 enum pwm_status PWM_ApplyLongSoftOuts(PWM_Type *base, uint32_t channel,
00541     int8_t mask_outs, enum pwm_outx_cmd outa, enum pwm_outx_cmd outb);
00542
00552 uint32_t PWM_GetCntStat(PWM_Type *base, uint32_t channel);
00553
00565 enum pwm_status PWM_SetPeriod(PWM_Type *base, uint32_t channel,
00566     uint32_t period);
00571 #ifdef __cplusplus
00572 }
00573 #endif
00574
00575 #endif /* HAL_PWM_H */
00576

```

## 6.33 hal\_pwm\_newgen.h

```

00001
00021 #ifndef HAL_PWM_H
00022 #define HAL_PWM_H
00023
00024 #ifdef __cplusplus
00025 extern "C" {
00026 #endif
00027
00028 #include "hal_common.h"
00029
00030 #define PWM_CHANNEL_COUNT 4
00031 #define PWM_UNITS { (PWM_Type *)0x40111000, (PWM_Type *)0x40111100, (PWM_Type *)0x40111200,
00032     (PWM_Type *)0x40111300 }
00033
00036 #define PWM_FREQ_HZ_TO_PERIOD_US(x) (1000000 / (x))
00040 #define PWM_PERIOD_MS(x) ((x) * 1000UL)
00044 #define PWM_PERIOD_S(x) ((x) * 1000000UL)
00045
00049 enum {
00050     PWM_OutA = 0x0UL,
00051     PWM_OutB = 0x1UL,
00052 };
00053
00057 enum pwm_eventprd {
00058     PWM_EventPrdNo = 0UL,

```

```

00059     PWM_EventPrdOne   = 1UL,
00060     PWM_EventPrdTwo    = 2UL,
00061     PWM_EventPrdThree  = 3UL,
00062 };
00063
00067 enum pwm_int_source {
00068     PWM_IntSourceCtrcntEquZero    = 2UL,
00069     PWM_IntSourceCtrcntEquCtrprd  = 3UL,
00070     PWM_IntSourceCtrcntEquCmpAInc = 4UL,
00071     PWM_IntSourceCtrcntEquCmpADec = 5UL,
00072     PWM_IntSourceCtrcntEquCmpBInc = 6UL,
00073     PWM_IntSourceCtrcntEquCmpBDec = 7UL,
00074 };
00075
00079 enum pwm_chopper_duty {
00080     PWM_ChopperDuty_1_8 = 0,
00081     PWM_ChopperDuty_2_8 = 1,
00082     PWM_ChopperDuty_3_8 = 2,
00083     PWM_ChopperDuty_4_8 = 3,
00084     PWM_ChopperDuty_5_8 = 4,
00085     PWM_ChopperDuty_6_8 = 5,
00086     PWM_ChopperDuty_7_8 = 6,
00087 };
00088
00092 enum pwm_chopper_freq {
00093     PWM_ChopperFreqClk_8  = 0,
00094     PWM_ChopperFreqClk_16 = 1,
00095     PWM_ChopperFreqClk_24 = 2,
00096     PWM_ChopperFreqClk_32 = 3,
00097     PWM_ChopperFreqClk_40 = 4,
00098     PWM_ChopperFreqClk_48 = 5,
00099     PWM_ChopperFreqClk_56 = 6,
00100     PWM_ChopperFreqClk_64 = 7,
00101 };
00102
00106 enum pwm_chopper_first_width {
00107     PWM_ChopperFirstWidth_0_8 = 0,
00108     PWM_ChopperFirstWidth_1_8 = 1,
00109     PWM_ChopperFirstWidth_2_8 = 2,
00110     PWM_ChopperFirstWidth_3_8 = 3,
00111     PWM_ChopperFirstWidth_4_8 = 4,
00112     PWM_ChopperFirstWidth_5_8 = 5,
00113     PWM_ChopperFirstWidth_6_8 = 6,
00114     PWM_ChopperFirstWidth_7_8 = 7,
00115     PWM_ChopperFirstWidth_8_8 = 8,
00116     PWM_ChopperFirstWidth_9_8 = 9,
00117     PWM_ChopperFirstWidth_10_8 = 10,
00118     PWM_ChopperFirstWidth_11_8 = 11,
00119     PWM_ChopperFirstWidth_12_8 = 12,
00120     PWM_ChopperFirstWidth_13_8 = 13,
00121     PWM_ChopperFirstWidth_14_8 = 14,
00122     PWM_ChopperFirstWidth_15_8 = 15,
00123 };
00124
00128 typedef struct _pwm_chopper {
00129     bool chopper_en;
00130     enum pwm_chopper_duty chopper_duty;
00131     enum pwm_chopper_freq chopper_freq;
00132     enum pwm_chopper_first_width chopper_first_width;
00133 } pwm_chopper_cfg_t;
00134
00138 enum pwm_dz_outx_inv {
00139     PWM_DzSignalOutxInvOff = 0,
00140     PWM_DzSignalOutxInvOn  = 1,
00141 };
00142
00146 enum pwm_dz_mode {
00147     PWM_DzModeOff = 0,
00148     PWM_DzModeOn  = 1,
00149 };
00150
00154 typedef struct _pwm_dz_cfg {
00155     bool dz_en;
00156     enum pwm_dz_outx_inv inv;
00157 } pwm_dz_cfg_t;
00158
00162 enum pwm_trip_unit_action {
00163     PWM_TripUnitActionHigh = 0,
00164     PWM_TripUnitActionOne  = 1,
00165     PWM_TripUnitActionZero = 2,
00166     PWM_TripUnitActionNo   = 3,
00167 };
00168
00169 typedef struct _pwm_trip_unit_cfg {
00170     bool trip_unit_en;
00171     enum pwm_trip_unit_action action;
00172 } pwm_trip_unit_cfg_t;

```



```

00173
00177 typedef struct _pwm_handle {
00178     uint32_t channel;
00179     uint32_t out;
00181     uint32_t ref_clk;
00183     float duty_cycle;
00184     uint32_t period_us;
00186     bool int_en;
00187     enum pwm_int_source int_source;
00188     enum pwm_eventprd int_freq;
00189     void (*pwm_callback)(struct _pwm_handle *hpwm);
00191     pwm_chopper_cfg_t chopper_cfg;
00192     pwm_dz_cfg_t deadzone_cfg;
00193     pwm_trip_unit_cfg_t trip_unit_cfg;
00194 } pwm_handle_t;
00195
00201 void PWM_Init(pwm_handle_t *hpwm);
00202
00209 void PWM_Enable(pwm_handle_t *hpwm, bool enable);
00210
00216 void PWM_IntCallback(pwm_handle_t *hpwm);
00217
00218 #ifdef __cplusplus
00219 }
00220 #endif
00221
00222 #endif /* HAL_PWM_H */
00223

```

## 6.34 Файл devices/eliot1/drivers/hal\_qspi.h

Интерфейс драйвера модуля QSPI.

```
#include "hal_common.h"
```

Структуры данных

- struct [\\_qspi\\_config](#)  
Количество бит во фрейме
- struct [\\_qspi\\_xip\\_config](#)  
Структура параметров конфигурации XIP контроллера QSPI.

Определения типов

- typedef enum [\\_qspi\\_qmode](#) qspi\_qmode\_t  
Режим работы контроллера QSPI.
- typedef struct [\\_qspi\\_config](#) qspi\_config\_t  
Количество бит во фрейме
- typedef struct [\\_qspi\\_xip\\_config](#) qspi\_xip\_config\_t  
Структура параметров конфигурации XIP контроллера QSPI.

Перечисления

- enum [\\_qspi\\_qmode](#)  
Режим работы контроллера QSPI.

## Функции

- `uint32_t QSPI_GetInstance` (`QSPI_Type *base`)  
Получение номера блока QSPI.
- `void QSPI_GetDefaultConfig` (`qspi_config_t *config`)  
Получение конфигурации QSPI по умолчанию
- `void QSPI_Init` (`QSPI_Type *base`, `const qspi_config_t *config`)  
Инициализация контроллера QSPI.
- `void QSPI_SetBitSize` (`QSPI_Type *base`, `qspi_bit_size_t bit_size`)  
Установка количества передаваемых бит
- `void QSPI_SetQMode` (`QSPI_Type *base`, `qspi_qmode_t spi_mode`)  
Установка режима SPI.
- `void QSPI_SetInhibitDin` (`QSPI_Type *base`, `bool inhibit_din`)  
Установка запрета записи в Tx FIFO.
- `void QSPI_SetInhibitDout` (`QSPI_Type *base`, `bool inhibit_dout`)  
Установка запрета чтения из Rx FIFO.
- `static void QSPI_Enable` (`QSPI_Type *base`)  
Включение контроллера QSPI.
- `static void QSPI_DeInit` (`QSPI_Type *base`)  
Деинициализация контроллера QSPI.
- `static void QSPI_EnableDMA` (`QSPI_Type *base`)  
Включение DMA.
- `static void QSPI_DisableDMA` (`QSPI_Type *base`)  
Выключение DMA.
- `static uint32_t QSPI_GetStatusFlag` (`QSPI_Type *base`)  
Получение значения статусного регистра
- `static void QSPI_SetSlaveSelect` (`QSPI_Type *base`, `uint32_t slave_select`)  
Переключение ведомого устройства
- `static void QSPI_EnableInterrupt` (`QSPI_Type *base`, `uint32_t mask`)  
Включение прерываний
- `static void QSPI_DisableInterrupt` (`QSPI_Type *base`, `uint32_t mask`)  
Отключение прерываний
- `static void QSPI_ClearInterrupt` (`QSPI_Type *base`, `uint32_t mask`)  
Сброс прерываний
- `static void QSPI_WriteData` (`QSPI_Type *base`, `uint32_t data`)  
Передача 32-битного слова в Tx FIFO.
- `static void QSPI_WriteDataByte` (`QSPI_Type *base`, `uint8_t data`)  
Передача байта данных в Tx FIFO.
- `static uint32_t QSPI_ReadData` (`QSPI_Type *base`)  
Чтение 32-битного слова из Rx FIFO.
- `static uint8_t QSPI_ReadDataByte` (`QSPI_Type *base`)  
Чтение байта данных из Rx FIFO.
- `static uint32_t QSPI_GetTXLVL` (`QSPI_Type *base`)  
Чтение уровня заполнения буфера передачи Tx FIFO.
- `static uint32_t QSPI_GetRXLVL` (`QSPI_Type *base`)  
Чтение уровня заполнения буфера приема Rx FIFO.

## 6.34.1 Подробное описание

Интерфейс драйвера модуля QSPI.

## 6.35 hal\_qspi.h

[См. документацию.](#)

```

00001
00021 #ifndef HAL_QSPI_H
00022 #define HAL_QSPI_H
00023
00024 #ifdef __cplusplus
00025 extern "C" {
00026 #endif
00027
00028 #include "hal_common.h"
00029
00033 typedef enum __qspi_qmode {
00034     QSPI_NormalSPI = 0x0,
00035     QSPI_DualSPI   = 0x2,
00036     QSPI_QuadSPI   = 0x3
00037 } qspi_qmode_t;
00038
00042 #if defined(QSPI_DRIVER_VERSION_1_6_0)
00043 typedef enum __qspi_bit_size_t {
00044     QSPI_FRAME_BITS_4  = 0x03,
00045     QSPI_FRAME_BITS_5  = 0x04,
00046     QSPI_FRAME_BITS_6  = 0x05,
00047     QSPI_FRAME_BITS_7  = 0x06,
00048     QSPI_FRAME_BITS_8  = 0x07,
00049     QSPI_FRAME_BITS_9  = 0x08,
00050     QSPI_FRAME_BITS_10 = 0x09,
00051     QSPI_FRAME_BITS_11 = 0x0A,
00052     QSPI_FRAME_BITS_12 = 0x0B,
00053     QSPI_FRAME_BITS_13 = 0x0C,
00054     QSPI_FRAME_BITS_14 = 0x0D,
00055     QSPI_FRAME_BITS_15 = 0x0E,
00056     QSPI_FRAME_BITS_16 = 0x0F,
00057     QSPI_FRAME_BITS_17 = 0x10,
00058     QSPI_FRAME_BITS_18 = 0x11,
00059     QSPI_FRAME_BITS_19 = 0x12,
00060     QSPI_FRAME_BITS_20 = 0x13,
00061     QSPI_FRAME_BITS_21 = 0x14,
00062     QSPI_FRAME_BITS_22 = 0x15,
00063     QSPI_FRAME_BITS_23 = 0x16,
00064     QSPI_FRAME_BITS_24 = 0x17,
00065     QSPI_FRAME_BITS_25 = 0x18,
00066     QSPI_FRAME_BITS_26 = 0x19,
00067     QSPI_FRAME_BITS_27 = 0x1A,
00068     QSPI_FRAME_BITS_28 = 0x1B,
00069     QSPI_FRAME_BITS_29 = 0x1C,
00070     QSPI_FRAME_BITS_30 = 0x1D,
00071     QSPI_FRAME_BITS_31 = 0x1E,
00072     QSPI_FRAME_BITS_32 = 0x1F
00073 } qspi_bit_size_t;
00074 #elif defined(QSPI_DRIVER_VERSION_2_0_0)
00075 typedef enum __qspi_bit_size_t {
00076     QSPI_FRAME_BITS_4  = 0x00,
00077     QSPI_FRAME_BITS_8  = 0x01,
00078     QSPI_FRAME_BITS_12 = 0x02,
00079     QSPI_FRAME_BITS_16 = 0x03,
00080     QSPI_FRAME_BITS_20 = 0x04,
00081     QSPI_FRAME_BITS_24 = 0x05,
00082     QSPI_FRAME_BITS_28 = 0x06,
00083     QSPI_FRAME_BITS_32 = 0x07
00084 } qspi_bit_size_t;
00085 #else
00086 #error "Please define QSPI controller version"
00087 #endif
00088
00089 #if defined(QSPI_DRIVER_VERSION_1_6_0)
00093 typedef enum __qspi_format_spi_t {
00094     QSPI_FORMAT_MOTOROLA = 0x00,
00095     QSPI_FORMAT_TI       = 0x02,
00096     QSPI_FORMAT_MICROWIRE = 0x03
00097 } qspi_format_spi_t;
00098 #endif
00099
00103 typedef struct __qspi_config {
00104     uint32_t delay_en;
00105     uint32_t cpol;
00106     uint32_t cpha;
00107     uint32_t msb;
00108     uint32_t cont_trans_en;
00109     uint16_t cont_transfer_ext;
00110     qspi_qmode_t spi_mode;
00111     uint32_t slave_select;
00112     uint32_t slave_pol;
00113     qspi_bit_size_t bit_size;

```

```

00114     uint32_t mode;
00115     uint32_t dma_en;
00116     uint32_t inhibit_din;
00117     uint32_t inhibit_dout;
00118     #if defined(QSPI_DRIVER_VERSION_1_6_0)
00119     uint32_t xfer_format;
00120     #endif
00121 } qspi_config_t;
00122
00123 typedef struct _qspi_xip_config {
00124     uint32_t cmd;
00125     uint32_t hpen;
00126     uint32_t cpha;
00127     uint32_t cpol;
00128     uint32_t addr4;
00129     uint32_t le32;
00130     uint32_t hp_mode;
00131     uint32_t dummy_cycles;
00132     uint32_t hp_end_dummy;
00133 } qspi_xip_config_t;
00134
00135 uint32_t QSPI_GetInstance(QSPI_Type *base);
00136
00137 void QSPI_GetDefaultConfig(qspi_config_t *config);
00138
00139 void QSPI_Init(QSPI_Type *base, const qspi_config_t *config);
00140
00141 void QSPI_SetBitSize(QSPI_Type *base, qspi_bit_size_t bit_size);
00142
00143 #if defined(QSPI_DRIVER_VERSION_1_6_0)
00144 void QSPI_SetFormatSPI(QSPI_Type *base, qspi_format_spi_t xfer_format);
00145 #endif
00146
00147 void QSPI_SetQMode(QSPI_Type *base, qspi_qmode_t spi_mode);
00148
00149 void QSPI_SetInhibitDin(QSPI_Type *base, bool inhibit_din);
00150
00151 void QSPI_SetInhibitDout(QSPI_Type *base, bool inhibit_dout);
00152
00153 static inline void QSPI_Enable(QSPI_Type *base)
00154 {
00155     #if defined(QSPI_DRIVER_VERSION_1_6_0)
00156     base->ENABLE = QSPI_ENABLE_ENABLEREQ_Msk;
00157     while (base->ENABLE != QSPI_ENABLE_ENABLEREQ_Msk)
00158         ;
00159     #elif defined(QSPI_DRIVER_VERSION_2_0_0)
00160     base->ENABLE = QSPI_ENABLE_ENABLE_Msk;
00161     while (base->ENABLE != QSPI_ENABLE_ENABLE_Msk)
00162         ;
00163     #else
00164     #error "Please define QSPI controller version"
00165     #endif
00166 }
00167
00168 static inline void QSPI_DeInit(QSPI_Type *base)
00169 {
00170     base->ENABLE = 0x0;
00171     while (base->ENABLE != 0)
00172         ;
00173 }
00174
00175 static inline void QSPI_EnableDMA(QSPI_Type *base)
00176 {
00177     base->CTRL |= QSPI_CTRL_DMA_Msk;
00178 }
00179
00180 static inline void QSPI_DisableDMA(QSPI_Type *base)
00181 {
00182     base->CTRL &= ~QSPI_CTRL_DMA_Msk;
00183 }
00184
00185 static inline uint32_t QSPI_GetStatusFlag(QSPI_Type *base)
00186 {
00187     return base->STAT;
00188 }
00189
00190 static inline void QSPI_SetSlaveSelect(QSPI_Type *base, uint32_t slave_select)
00191 {
00192     base->SS = slave_select;
00193 }
00194
00195 static inline void QSPI_EnableInterrupt(QSPI_Type *base, uint32_t mask)
00196 {
00197     base->INTR_EN |= mask;

```

```

00289 }
00290
00297 static inline void QSPI_DisableInterrupt(QSPI_Type *base, uint32_t mask)
00298 {
00299     base->INTR_EN &= ~mask;
00300 }
00301
00308 static inline void QSPI_ClearInterrupt(QSPI_Type *base, uint32_t mask)
00309 {
00310     base->INTR_CLR = mask;
00311 }
00312
00319 static inline void QSPI_WriteData(QSPI_Type *base, uint32_t data)
00320 {
00321     base->TX_DATA = data;
00322 }
00323
00330 static inline void QSPI_WriteDataByte(QSPI_Type *base, uint8_t data)
00331 {
00332     *((volatile uint8_t *)&(base->TX_DATA)) = data;
00333 }
00334
00341 static inline uint32_t QSPI_ReadData(QSPI_Type *base)
00342 {
00343     return base->RX_DATA;
00344 }
00345
00352 static inline uint8_t QSPI_ReadDataByte(QSPI_Type *base)
00353 {
00354     return *((volatile uint8_t *)&(base->RX_DATA));
00355 }
00356
00363 static inline uint32_t QSPI_GetTXLVL(QSPI_Type *base)
00364 {
00365     return base->TX_FIFO_LVL;
00366 }
00367
00374 static inline uint32_t QSPI_GetRXLVL(QSPI_Type *base)
00375 {
00376     return base->RX_FIFO_LVL;
00377 }
00378
00379 #ifdef __cplusplus
00380 }
00381 #endif
00382
00383 #endif /* HAL_QSPI_H */
00384

```

## 6.36 Файл devices/eliot1/drivers/hal\_qspi\_dma.h

Дополнение драйвера QSPI для обмена данными с помощью DMA.

```

#include "hal_qspi.h"
#include "hal_dma.h"

```

Структуры данных

- struct `qspi_dma_handle_t`  
Дескриптор QSPI-DMA передачи

Перечисления

- enum `qspi_dma_status_t`  
Статусы выполнения функций

## Функции

- void `QSPI_TransferCreateHandleDMA` (`QSPI_Type` \*base, `qspi_dma_handle_t` \*handle, `dma_handle_t` \*tx\_handle, `dma_handle_t` \*rx\_handle)  
Инициализация контекста передачи QSPI-DMA.
- `qspi_dma_status_t` `QSPI_WriteDataDMA` (`qspi_dma_handle_t` \*handle, void \*addr, uint8\_t incr, uint8\_t transfer\_width, uint32\_t size)  
Запись данных в QSPI TX.
- `qspi_dma_status_t` `QSPI_ReadDataDMA` (`qspi_dma_handle_t` \*handle, void \*addr, uint8\_t incr, uint8\_t transfer\_width, uint32\_t size)  
Чтение данных из QSPI RX.
- uint32\_t `QSPI_GetReadDMADescriptorsCount` (uint32\_t size\_in\_bytes)  
Расчет количества дескрипторов многоблочной передачи, нужных для считывания данных из NOR Flash памяти
- void `QSPI_DMAREadDescriptorInitRX` (`QSPI_Type` \*base, `dma_descriptor_t` \*desc, uint32\_t data\_size, void \*dst\_addr, uint8\_t dst\_addr\_incr)  
Инициализация группы дескрипторов DMA (многоблочная передача) для приема данных в RX буфер.
- static uint32\_t `QSPI_GetDummyDMADescriptorsCount` (uint32\_t size\_in\_bytes)  
Расчет количества дескрипторов многоблочной передачи, нужных для считывания данных из NOR Flash памяти при посылке фиктивных данных
- static void `QSPI_DMADescriptorInitTX` (`QSPI_Type` \*base, `dma_descriptor_t` \*desc, uint32\_t count, uint32\_t data\_size, uint32\_t data\_width, void \*src\_addr, uint8\_t src\_addr\_incr)  
Инициализация группы дескрипторов DMA (многоблочная передача) для отправки данных в TX буфер.
- static void `QSPI_DMADescriptorInitRX` (`QSPI_Type` \*base, `dma_descriptor_t` \*desc, uint32\_t count, uint32\_t data\_size, uint32\_t data\_width, void \*dst\_addr)  
Инициализация группы дескрипторов DMA (многоблочная передача) для приема данных в RX буфер.

## 6.36.1 Подробное описание

Дополнение драйвера QSPI для обмена данными с помощью DMA.

## 6.37 hal\_qspi\_dma.h

См. документацию.

```

00001
00012 #ifndef HAL_QSPI_DMA_H
00013 #define HAL_QSPI_DMA_H
00014
00015 #include "hal_qspi.h"
00016 #include "hal_dma.h"
00017
00018 #ifdef __cplusplus
00019 extern "C" {
00020 #endif
00021
00022 typedef struct {
00023     QSPI_Type *base;
00024     dma_handle_t *tx_handle;
00025     dma_handle_t *rx_handle;
00026     void *dummy_data;
00027     dma_descriptor_t *tx_desc;
00028     dma_descriptor_t *rx_desc;
00029 } qspi_dma_handle_t;
00030
00031 typedef enum {
00032     QSPI_DMA_Status_Success = 0U,
00033     QSPI_DMA_Status_Fail = 1U,

```

```

00040     QSPI_DMA_Status_InvalidArgument = 2U,
00041 } qspi_dma_status_t;
00042
00051 void QSPI_TransferCreateHandleDMA(QSPI_Type *base, qspi_dma_handle_t *handle,
00052     dma_handle_t *tx_handle, dma_handle_t *rx_handle);
00053
00067 qspi_dma_status_t QSPI_WriteDataDMA(qspi_dma_handle_t *handle, void *addr,
00068     uint8_t incr, uint8_t transfer_width, uint32_t size);
00069
00083 qspi_dma_status_t QSPI_ReadDataDMA(qspi_dma_handle_t *handle, void *addr,
00084     uint8_t incr, uint8_t transfer_width, uint32_t size);
00085
00094 uint32_t QSPI_GetReadDMADescriptorsCount(uint32_t size_in_bytes);
00095
00105 void QSPI_DMAReadDescriptorInitRX(QSPI_Type *base, dma_descriptor_t *desc,
00106     uint32_t data_size, void *dst_addr, uint8_t dst_addr_incr);
00107
00117 static inline uint32_t QSPI_GetDummyDMADescriptorsCount(uint32_t size_in_bytes)
00118 {
00119     return DMA_GetDescriptorCount(size_in_bytes, DMA_Transfer8BitWidth);
00120 }
00121
00132 static inline void QSPI_DMADescriptorInitTX(QSPI_Type *base, dma_descriptor_t *desc,
00133     uint32_t count, uint32_t data_size, uint32_t data_width, void *src_addr, uint8_t src_addr_incr)
00134 {
00135     dma_multiblock_config_t config = {
00136         .count = count,
00137         .data_size = data_size,
00138         .transfer_type = DMA_MemoryToPeripheral_DMA,
00139         .scatter_en = false,
00140         .gather_en = false,
00141         .src_burst_size = DMA_BurstSize32,
00142         .dst_burst_size = DMA_BurstSize1,
00143         .src_incr = src_addr_incr,
00144         .dst_incr = DMA_NoChange,
00145         .src_data_width = data_width,
00146         .dst_data_width = data_width,
00147         .src_addr = src_addr,
00148         .dst_addr = (void *) &base->TX_DATA,
00149         .int_en = true
00150     };
00151
00152     DMA_InitMultiblockDescriptor(desc, &config);
00153 }
00154
00164 static inline void QSPI_DMADescriptorInitRX(QSPI_Type *base, dma_descriptor_t *desc,
00165     uint32_t count, uint32_t data_size, uint32_t data_width, void *dst_addr)
00166 {
00167     dma_multiblock_config_t config = {
00168         .count = count,
00169         .data_size = data_size,
00170         .transfer_type = DMA_PeripheralToMemory_DMA,
00171         .scatter_en = false,
00172         .gather_en = false,
00173         .src_burst_size = DMA_BurstSize1,
00174         .dst_burst_size = DMA_BurstSize32,
00175         .src_incr = DMA_NoChange,
00176         .dst_incr = DMA_Incr,
00177         .src_data_width = data_width,
00178         .dst_data_width = data_width,
00179         .src_addr = (void *) &base->RX_DATA,
00180         .dst_addr = dst_addr,
00181         .int_en = true
00182     };
00183
00184     DMA_InitMultiblockDescriptor(desc, &config);
00185 }
00186
00187 #ifdef __cplusplus
00188 }
00189 #endif
00190
00191 #endif /* HAL_QSPI_DMA_H */
00192

```

## 6.38 Файл devices/eliot1/drivers/hal\_qspi\_nor\_flash.h

Интерфейс драйвера модуля QSPI-NOR-FLASH.

```

#include "hal_common.h"
#include "hal_nor_flash.h"

```

```
#include "hal_qspi.h"
```

### Структуры данных

- `struct _qspi_nor_config`  
Конфигурационный блок для режима Quad.
- `struct _qspi_nor_init_config`  
Первоначальная конфигурация QSPI.
- `struct _qspi_nor_handle`  
Контекст драйвера NOR Flash.

### Определения типов

- `typedef struct _qspi_nor_config qspi_nor_config_t`  
Конфигурационный блок для режима Quad.
- `typedef enum _qspi_command_format qspi_command_format_t`  
Формат команды QSPI.
- `typedef struct _qspi_nor_init_config qspi_nor_init_config_t`  
Первоначальная конфигурация QSPI.
- `typedef enum _qspi_command_type qspi_command_type_t`  
Тип команды QSPI.
- `typedef struct _qspi_nor_handle qspi_nor_handle_t`  
Контекст драйвера NOR Flash.

### Перечисления

- `enum _serial_nor_command`  
Коды операций микросхемы флеш-памяти
- `enum`  
Требования для включение режима Quad.
- `enum _qspi_command_format`  
Формат команды QSPI.
- `enum _qspi_command_type`  
Тип команды QSPI.

#### 6.38.1 Подробное описание

Интерфейс драйвера модуля QSPI-NOR-FLASH.



## 6.39 hal\_qspi\_nor\_flash.h

[См. документацию.](#)

```

00001
00020 #ifndef HAL_QSPI_NOR_FLASH_H
00021 #define HAL_QSPI_NOR_FLASH_H
00022
00023 #ifdef __cplusplus
00024 extern "C" {
00025 #endif
00026
00027 #include "hal_common.h"
00028 #include "hal_nor_flash.h"
00029 #include "hal_qspi.h"
00030
00034 enum serial_nor_command {
00035     NOR_CmdInvalid = 0x00U,
00036     NOR_CmdWriteStatus = 0x01U,
00037     NOR_CmdWriteSecStatus_31 = 0x31U,
00038     NOR_CmdWriteSecStatus_3E = 0x3EU,
00039     NOR_CmdWriteMemory = 0x02U,
00040     NOR_CmdWriteMemoryA32 = 0x12U,
00041     NOR_CmdWriteEnable = 0x06U,
00042     NOR_CmdWriteDisable = 0x04U,
00043     NOR_CmdReadStatus = 0x05U,
00044     NOR_CmdReadSecStatus_35 = 0x35U,
00045     NOR_CmdReadSecStatus_3F = 0x3FU,
00046     NOR_CmdReadMemory = 0x03U,
00047     NOR_CmdReadMemoryA32 = 0x13U,
00048     NOR_CmdReadMemorySDR_1_1_1 = 0x0BU,
00049     NOR_CmdReadMemorySDR_1_1_2 = 0x3BU,
00050     NOR_CmdReadMemorySDR_1_2_2 = 0xBBU,
00051     NOR_CmdReadMemorySDR_1_1_4 = 0x6BU,
00052     NOR_CmdReadMemorySDR_1_4_4 = 0xEBU,
00053     NOR_CmdReadMemorySDR_1_4_4_A32 = 0xECU,
00054     NOR_CmdReadMemorySDR_1_1_4_A32 = 0x6CU,
00055     NOR_CmdEraseChipNor = 0x60U,
00056     NOR_CmdEraseChip = 0xC7U,
00057     NOR_CmdErasePage = 0x42U,
00058     NOR_CmdEraseSector4KB = 0x20U,
00059     NOR_CmdEraseSector32KB = 0x52U,
00060     NOR_CmdEraseSector = 0xD8U,
00061     NOR_CmdEraseSector4KBA32 = 0x21U,
00062     NOR_CmdEraseSectorA32 = 0xDCU,
00063 };
00064
00070 enum {
00071     NOR_QuadModeNotConfig = 0,
00072     NOR_QuadModeStatusReg1_Bit6 = 1,
00073     NOR_QuadModeStatusReg2_Bit1 = 2,
00074     NOR_QuadModeStatusReg2_Bit7 = 3,
00075     NOR_QuadModeStatusReg2_Bit1_0x31 = 4,
00076 };
00077
00081 typedef struct _qspi_nor_config {
00082     bool is_quad_need_enable;
00083     bool write_two_status_bytes;
00084     uint8_t quad_enable_command;
00085     uint8_t quad_enable_bit_shift;
00086     uint8_t quad_read_command;
00087 } qspi_nor_config_t;
00088
00092 typedef enum _qspi_command_format {
00093     QSPI_CommandAllSerial = 0x0,
00094     QSPI_CommandDataQuad = 0x1U,
00095     QSPI_CommandOpcodeSerial = 0x2U,
00096 } qspi_command_format_t;
00097
00101 typedef struct _qspi_nor_init_config {
00102     qspi_command_format_t cmd_format;
00103     uint32_t quad_mode_setting;
00104 } qspi_nor_init_config_t;
00105
00109 typedef enum _qspi_command_type {
00110     QSPI_CommandOpcodeOnly = 0x1U,
00111     QSPI_CommandOpcodeAddrOneByte = 0x2U,
00112     QSPI_CommandOpcodeAddrTwoBytes = 0x3U,
00113     QSPI_CommandOpcodeAddrThreeBytes = 0x4U,
00114     QSPI_CommandOpcodeAddrFourBytes = 0x5U,
00115     QSPI_CommandNoOpcodeAddrThreeBytes = 0x6U,
00116     QSPI_CommandNoOpcodeAddrFourBytes = 0x7U,
00117 } qspi_command_type_t;
00118
00122 typedef struct _qspi_nor_handle {
00123     qspi_command_type_t command_type;

```

```

00124     qspi_command_format_t read_cmd_format;
00125     uint8_t intermediate_len;
00126     nor_command_set_t command_set;
00127 } qspi_nor_handle_t;
00128
00129 #ifdef __cplusplus
00130 }
00131 #endif
00132
00133 #endif /* HAL_QSPI_NOR_FLASH_H */
00134

```

## 6.40 Файл devices/eliot1/drivers/hal\_rwc.h

Интерфейс драйвера модуля RWC.

```
#include "hal_common.h"
```

Структуры данных

- struct [rwc\\_trimload\\_reg](#)  
Регистр записи значения подстройки из регистра TRIM.
- struct [rwc\\_time\\_reg](#)  
Регистр текущего значения счетчика времени
- struct [rwc\\_alarm\\_reg](#)  
Регистр времени пробуждения
- struct [rwc\\_trim\\_reg](#)  
Регистр подстройки осцилляторов
- struct [rwc\\_config\\_reg](#)  
Конфигурационный регистр
- struct [rwc\\_general\\_reg](#)  
Регистр общего назначения
- struct [rwc\\_wake\\_config\\_reg](#)  
Регистр настройки контроллера пробуждения
- union [rwc\\_union\\_reg](#)  
Объединение для доступа к регистрам
- struct [\\_rtc\\_datetime](#)  
Структура используемая для хранения даты и времени
- struct [rwc\\_config](#)  
Структура используемая для конфигурирования RWC.

Макросы

- #define [RWC\\_SYNC\\_RETRY\\_TIMES](#) 0U  
Количество циклов ожидания
- #define [RWC\\_SET\\_RETRY\\_TIMES](#) 10U  
Количество циклов установки времени

Макросы для операций перевода времени

- #define [SECONDS\\_IN\\_A\\_DAY](#) (86400U)
- #define [SECONDS\\_IN\\_A\\_HOUR](#) (3600U)
- #define [SECONDS\\_IN\\_A\\_MINUTE](#) (60U)
- #define [DAYS\\_IN\\_A\\_YEAR](#) (365U)
- #define [YEAR\\_RANGE\\_START](#) (1970U)
- #define [YEAR\\_RANGE\\_END](#) (2099U)

## Определения типов

- typedef struct [\\_rtc\\_datetime](#) rtc\_datetime\_t  
Структура используемая для хранения даты и времени

## Перечисления

- enum [rwc\\_time\\_clk\\_sel](#)  
Выбор сигнала для тактирования счетчика времени
- enum [rwc\\_wake\\_up\\_polarity](#)  
Уровень активного сигнала WKUP для генерирования прерывания
- enum [rwc\\_wake\\_up\\_irq\\_enable](#)  
Разрешение прерывания RWC\_WKUP.
- enum [rwc\\_lfe\\_bypass](#)  
Режимы работы осциллятора LFE.
- enum [rwc\\_internal\\_register](#)  
Внутренние регистры с батарейным питанием
- enum [rwc\\_rtclk\\_type](#)  
Источник частоты, подаваемой на RTCCLK.
- enum [rwc\\_reset\\_type](#)  
Виды сбросов внутренних регистров при сигнале SRSTh.
- enum [rwc\\_rtclk\\_divisor](#)  
Делители частоты RTCCLK.
- enum [rwc\\_cmd](#)  
Команды доступа к внутренним регистрам
- enum [rwc\\_status](#)  
Статусы драйвера CLKCTR.
- enum [rwc\\_timer\\_status](#)  
Статусы таймера
- enum [rwc\\_alarm\\_enable](#)  
Разрешение прерывания ALARM.
- enum [rwc\\_wkup\\_enable](#)  
Разрешение работы входа WKUP.
- enum [rwc\\_shutdown\\_force](#)  
Принудительный переход в SHUTDOWN.
- enum [rwc\\_freq\\_serial](#)  
Значения делителей частоты внутреннего интерфейса

## Функции

- enum [rwc\\_status](#) [RWC\\_SetSecondsTimerMatch](#) (RWC\_Type \*base, uint32\_t match\_value)  
Устанавливает время будильника в секундах
- uint32\_t [RWC\\_GetSecondsTimerMatch](#) (RWC\_Type \*base)  
Получает актуальное время срабатывания будильника в секундах
- enum [rwc\\_status](#) [RWC\\_SetSecondsTimerCount](#) (RWC\_Type \*base, uint32\_t count\_value)  
Устанавливает текущее время в секундах
- enum [rwc\\_status](#) [RWC\\_GetSecondsTimerCount](#) (RWC\_Type \*base, uint32\_t \*sec)  
Получение текущее время в секундах
- enum [rwc\\_status](#) [RWC\\_GetDefaultConfig](#) (struct [rwc\\_config](#) \*config)  
Получение конфигурации таймера по умолчанию

- enum [rwc\\_status RWC\\_SetLFEBypass](#) (RWC\_Type \*base, enum [rwc\\_lfe\\_bypass](#) value)  
Выбор режима работы осциллятора LFE.
- enum [rwc\\_status RWC\\_SetLFX](#) (RWC\_Type \*base, enum [rwc\\_rtcclk\\_type](#) value)  
Выбор генератора LFE или LFI.
- enum [rwc\\_status RWC\\_GetLastAPIStatus](#) ()  
Получение статуса выполнения функции, тип результата которой отличен от enum [rwc\\_status](#).
- enum [rwc\\_status RWC\\_SetResetCtrl](#) (RWC\_Type \*base, enum [rwc\\_reset\\_type](#) value)  
Управление сбросом регистров при сигнале на входе SRSTn.

#### Функции чтения/записи внутренних регистров

- enum [rwc\\_status RWC\\_GetInternalRegister](#) (RWC\_Type \*base, enum [rwc\\_internal\\_register](#) reg, union [rwc\\_union\\_reg](#) \*value)  
Чтение значения из внутреннего регистра RWC.
- enum [rwc\\_status RWC\\_SetInternalRegister](#) (RWC\_Type \*base, enum [rwc\\_internal\\_register](#) reg, union [rwc\\_union\\_reg](#) value)  
Запись значения во внутренний регистр RWC.

#### Функции для работы с частотой тактирования

- enum [rwc\\_status RWC\\_GetRTCClkParam](#) (uint32\_t \*div, enum [rwc\\_rtcclk\\_type](#) \*src)  
Чтение делителя и источника частоты RTCCLK.
- enum [rwc\\_status RWC\\_SetRTCClkParam](#) (uint32\_t div, enum [rwc\\_rtcclk\\_type](#) src)  
Запись делителя и источника частоты RTCCLK.
- uint32\_t [RWC\\_GetTime](#) (RWC\_Type \*base)  
Чтение значения счетчика реального времени

#### Инициализация и деинициализация

- enum [rwc\\_status RWC\\_Init](#) (RWC\_Type \*base, struct [rwc\\_config](#) cfg)  
Инициализирует таймер реального времени
- enum [rwc\\_status RWC\\_Deinit](#) (RWC\_Type \*base)  
Деинициализирует таймер реального времени

#### Текущее время и предупреждение

- enum [rwc\\_status RWC\\_SetDatetime](#) (RWC\_Type \*base, const [rtc\\_datetime\\_t](#) \*datetime)  
Устанавливает текущее время и дату согласно заданной структуре
- enum [rwc\\_status RWC\\_GetDatetime](#) (RWC\_Type \*base, [rtc\\_datetime\\_t](#) \*datetime)  
Получает текущее дату/время и сохраняет в указанную структуру
- enum [rwc\\_status RWC\\_SetAlarm](#) (RWC\_Type \*base, const [rtc\\_datetime\\_t](#) \*alarmTime)  
Устанавливает время будильника
- enum [rwc\\_status RWC\\_GetAlarm](#) (RWC\_Type \*base, [rtc\\_datetime\\_t](#) \*datetime)  
Возвращает время будильника

#### Interrupt Interface

- enum [rwc\\_status RWC\\_EnableWakeUpTimerInterruptFromDPD](#) (RWC\_Type \*base, bool enable)  
Разрешение прерывания по сигналу wake-up из режима глубокого отключения питания
- enum [rwc\\_status RWC\\_EnableAlarmTimerInterruptFromDPD](#) (RWC\_Type \*base, bool enable)  
Разрешение прерывания по сигналу будильника из режима глубокого отключения питания
- enum [rwc\\_status RWC\\_InterruptClear](#) (RWC\_Type \*base)

Сброс прерывания RWC\_ALARM.

#### Status Interface

- `uint32_t RWC_GetStatusFlags (RWC_Type *base)`  
Получение статусов таймера реального времени

Функции работы с внешним сигналом пробуждения (wake-up)

- `enum rwc_status RWC_SetWakeUpEnable (RWC_Type *base, bool enable)`  
Разрешение работы входа wake\_up.
- `enum rwc_status RWC_SetWakeUpActiveLevel (RWC_Type *base, enum rwc_wake_up_polarity value)`  
Установка полярности сигнала wake\_up.

Функции для удобного взаимодействия с типом данных datetime

- `uint32_t RWC_ConvertDatetimeToSeconds (const rtc_datetime_t *datetime)`  
Преобразование переменной типа данных datetime в секунды

### 6.40.1 Подробное описание

Интерфейс драйвера модуля RWC.

### 6.40.2 Функции

#### 6.40.2.1 RWC\_ConvertDatetimeToSeconds()

```
uint32_t RWC_ConvertDatetimeToSeconds (
    const rtc_datetime_t * datetime)
```

Преобразование переменной типа данных datetime в секунды

Аргументы

datetime	Преобразуемая дата
----------	--------------------

Возвращает

Количество секунд, прошедших с 1 января 1970 года по указанную дату

## 6.41 hal\_rwc.h

[См. документацию.](#)

```

00001
00021 #ifndef HAL_RWC_H
00022 #define HAL_RWC_H
00023
00024 #include "hal_common.h"
00025
00026 #if defined(__cplusplus)
00027 extern "C" {
00028 #endif
00029
00036 #ifndef RWC_SYNC_RETRY_TIMES
00037 #define RWC_SYNC_RETRY_TIMES 0U
00038 #endif /* RWC_SYNC_RETRY_TIMES */
00039
00043 #define RWC_SET_RETRY_TIMES 10U
00044
00049 #define SECONDS_IN_A_DAY (86400U)
00050 #define SECONDS_IN_A_HOUR (3600U)
00051 #define SECONDS_IN_A_MINUTE (60U)
00052 #define DAYS_IN_A_YEAR (365U)
00053 #define YEAR_RANGE_START (1970U)
00054 #define YEAR_RANGE_END (2099U)
00070 struct rwc_trimload_reg {
00071     uint32_t trimload :
00072         FIELD_BIT(0, 0);
00073     uint32_t :
00074         FIELD_BIT(31, 1);
00075 };
00076
00083 struct rwc_time_reg {
00084     uint32_t time :
00085         FIELD_BIT(31, 0);
00086 };
00087
00093 struct rwc_alarm_reg {
00094     uint32_t alarm :
00095         FIELD_BIT(31, 0);
00096 };
00097
00107 struct rwc_trim_reg {
00108     uint32_t trim_lfe :
00109         FIELD_BIT(10, 0);
00110     uint32_t trim_lfi :
00111         FIELD_BIT(21, 11);
00112     uint32_t lfe_bypass :
00113         FIELD_BIT(22, 22);
00114     uint32_t :
00115         FIELD_BIT(24, 23);
00122     uint32_t wake_stat2 :
00123         FIELD_BIT(25, 25);
00124
00129     uint32_t wake_stat3 :
00130         FIELD_BIT(26, 26);
00131     uint32_t :
00132         FIELD_BIT(31, 27);
00133 };
00134
00144 struct rwc_config_reg {
00149     uint32_t time_clk_sel :
00150         FIELD_BIT(0, 0);
00151
00152     uint32_t :
00153         FIELD_BIT(3, 1);
00154     uint32_t osc_sel :
00155         FIELD_BIT(4, 4);
00156     uint32_t :
00157         FIELD_BIT(5, 5);
00158     uint32_t clk_div :
00159         FIELD_BIT(10, 6);
00165     uint32_t reset_en :
00166         FIELD_BIT(11, 11);
00167
00172     uint32_t alarm_en :
00173         FIELD_BIT(12, 12);
00174
00179     uint32_t pz :
00180         FIELD_BIT(13, 13);
00181
00186     uint32_t pl :
00187         FIELD_BIT(14, 14);
00188     uint32_t wake_in_en :
00189         FIELD_BIT(15, 15);

```

```

00190 uint32_t      :
00191     FIELD_BIT(29, 16);
00198 uint32_t shutdown_force      :
00199     FIELD_BIT(30, 30);
00200
00205 uint32_t wake_stat1      :
00206     FIELD_BIT(31, 31);
00207 };
00208
00216 struct rwc_general_reg {
00217     uint32_t time :
00218         FIELD_BIT(31, 0);
00219 };
00220
00227 struct rwc_wake_config_reg {
00228     uint32_t wake_en :
00229         FIELD_BIT(0, 0);
00230     uint32_t      :
00231         FIELD_BIT(15, 1);
00232     uint32_t wake_pol :
00233         FIELD_BIT(16, 16);
00234     uint32_t      :
00235         FIELD_BIT(31, 17);
00236 };
00237
00244 union rwc_union_reg {
00245     uint32_t      reg_value;
00246     struct rwc_trimload_reg trimload;
00247     struct rwc_time_reg time;
00248     struct rwc_alarm_reg alarm;
00249     struct rwc_trim_reg trim;
00250     struct rwc_config_reg config;
00251     struct rwc_general_reg general;
00252     struct rwc_wake_config_reg wake_config;
00253 };
00254
00263 enum rwc_time_clk_sel {
00264     RWC_TimeClock32kHz = 0,
00265     RWC_TimeClock1Hz  = 1,
00266 };
00267
00272 enum rwc_wake_up_polarity {
00273     RWC_WkUpPolarityHigh = 0,
00274     RWC_WkUpPolarityLow  = 1,
00275 };
00276
00281 enum rwc_wake_up_irq_enable {
00282     RWC_IRQWkUpDisable = 0,
00283     RWC_IRQWkUpEnable  = 1,
00284 };
00285
00290 enum rwc_lfe_bypass {
00291     RWC_QuartzResonator = 0,
00292     RWC_CMOSSignal      = 1,
00293 };
00294
00299 enum rwc_internal_register {
00300     RWC_TrimLoad    = 2,
00301     RWC_Time        = 3,
00302     RWC_Alarm        = 4,
00303     RWC_Trim         = 5,
00304     RWC_Config       = 6,
00305     RWC_GeneralReg   = 7,
00306     RWC_WakeConfig   = 8,
00307 };
00308
00313 enum rwc_rtclk_type {
00314     RWC_RTCLKTypeLFI = 0,
00315     RWC_RTCLKTypeLFE = 1,
00316 };
00317
00318
00323 enum rwc_reset_type {
00324     RWC_ResetOnlyWakeConfigAndConfig = 0,
00325     RWC_ResetAllExpectedGeneralReg   = 1,
00326 };
00327
00332 enum rwc_rtclk_divisor {
00333     RWC_Div1    = 0,
00334     RWC_Div2    = 1,
00335     RWC_Div4    = 2,
00336     RWC_Div8    = 3,
00337     RWC_Div16   = 4,
00338     RWC_Div32   = 5,
00339     RWC_Div64   = 6,
00340     RWC_Div128  = 7,
00341     RWC_Div256  = 8,

```

```

00342 RWC_Div512 = 9,
00343 RWC_Div1024 = 10,
00344 RWC_Div2048 = 11,
00345 RWC_Div4096 = 12,
00346 RWC_Div8192 = 13,
00347 RWC_Div16384 = 14,
00348 RWC_Div32768 = 15,
00350 RWC_DivMax = RWC_Div32768,
00351 };
00352
00357 enum rwc_cmd {
00358     RWC_CmdWait = 0,
00359     RWC_CmdWrite = 1,
00360     RWC_CmdRead = 2,
00361 };
00362
00367 enum rwc_status {
00368     RWC_Status_Ok = 0,
00369     RWC_Status_InvalidArgument = 1,
00370     RWC_Status_CheckError = 2,
00371     RWC_Status_VerifyError = 3,
00372     RWC_Status_ConfigureError = 4,
00373     RWC_Status_HardwareBusy = 5,
00374     RWC_Status_Timeout = 6,
00375 };
00376
00381 enum rwc_timer_status {
00386     RWC_WakeStat3 = 1,
00387
00393     RWC_WakeStat2 = 2,
00394
00399     RWC_WakeStat1 = 4,
00400 };
00401
00406 enum rwc_alarm_enable {
00407     RWC_AlarmDisable = 0,
00408     RWC_AlarmEnable = 1,
00409 };
00410
00415 enum rwc_wkup_enable {
00416     RWC_WkUpDisable = 0,
00417     RWC_WkUpEnable = 1,
00418 };
00419
00424 enum rwc_shutdown_force {
00425     RWC_ShutdownNoSet = 0,
00426     RWC_ShutdownSet = 1,
00427 };
00428
00433 enum rwc_freq_serial {
00434     RWC_FS8MHz = 0,
00435     RWC_FS4MHz = 1,
00436     RWC_FS2MHz = 2,
00437     RWC_FS1MHz = 3,
00438     RWC_FS500kHz = 4,
00439     RWC_FS250kHz = 5,
00440     RWC_FS125kHz = 6,
00441     RWC_FS625hHz = 7,
00442 };
00443
00448 typedef struct _rtc_datetime {
00449     uint16_t year;
00450     uint8_t month;
00451     uint8_t day;
00452     uint8_t hour;
00453     uint8_t minute;
00454     uint8_t second;
00455 } rtc_datetime_t;
00456
00461 struct rwc_config {
00462     bool trimload;
00463     uint32_t time;
00464     uint32_t alarm_time;
00465     enum rwc_lfe_bypass lfe_bypass;
00466     uint32_t trim_lfi;
00467     uint32_t trim_lfe;
00473     uint32_t wake_stat1;
00474
00479     enum rwc_shutdown_force shutdown_force;
00480     enum rwc_wkup_enable wake_in_en;
00481     uint32_t pl;
00482     uint32_t pz;
00488     enum rwc_alarm_enable alarm_en;
00489     enum rwc_reset_type reset_en;
00490     enum rwc_rtclk_divisor clkdiv;
00491     enum rwc_rtclk_type osc_sel;
00492     enum rwc_time_clk_sel time_clk_sel;

```



```

00493     uint32_t          general;
00494     enum rwc_wake_up_polarity wake_pol;
00495     enum rwc_wake_up_irq_enable wake_en;
00496     enum rwc_freq_serial   presc;
00497 };
00498 };
00499
00500 enum rwc_status RWC_GetInternalRegister(RWC_Type *base,
00501     enum rwc_internal_register reg, union rwc_union_reg *value);
00502
00503 enum rwc_status RWC_SetInternalRegister(RWC_Type *base,
00504     enum rwc_internal_register reg, union rwc_union_reg value);
00505
00506 enum rwc_status RWC_GetRTCClkParam(uint32_t *div, enum rwc_rtclk_type *src);
00507
00508 enum rwc_status RWC_SetRTCClkParam(uint32_t div, enum rwc_rtclk_type src);
00509
00510 uint32_t RWC_GetTime(RWC_Type *base);
00511 enum rwc_status RWC_Init(RWC_Type *base, struct rwc_config cfg);
00512
00513 enum rwc_status RWC_Deinit(RWC_Type *base);
00514 enum rwc_status RWC_SetDatetime(RWC_Type *base, const rtc_datetime_t *datetime);
00515
00516 enum rwc_status RWC_GetDatetime(RWC_Type *base, rtc_datetime_t *datetime);
00517
00518 enum rwc_status RWC_SetAlarm(RWC_Type *base, const rtc_datetime_t *alarmTime);
00519
00520 enum rwc_status RWC_GetAlarm(RWC_Type *base, rtc_datetime_t *datetime);
00521
00522 enum rwc_status RWC_SetSecondsTimerMatch(RWC_Type *base, uint32_t match_value);
00523
00524 uint32_t RWC_GetSecondsTimerMatch(RWC_Type *base);
00525
00526 enum rwc_status RWC_SetSecondsTimerCount(RWC_Type *base, uint32_t count_value);
00527
00528 enum rwc_status RWC_GetSecondsTimerCount(RWC_Type *base, uint32_t *sec);
00529
00530 enum rwc_status RWC_EnableWakeUpTimerInterruptFromDPD(RWC_Type *base,
00531     bool enable);
00532
00533 enum rwc_status RWC_EnableAlarmTimerInterruptFromDPD(RWC_Type *base,
00534     bool enable);
00535
00536 enum rwc_status RWC_InterruptClear(RWC_Type *base);
00537
00538 uint32_t RWC_GetStatusFlags(RWC_Type *base);
00539
00540 enum rwc_status RWC_SetWakeUpEnable(RWC_Type *base, bool enable);
00541
00542 enum rwc_status RWC_SetWakeUpActiveLevel(RWC_Type *base,
00543     enum rwc_wake_up_polarity value);
00544
00545 enum rwc_status RWC_GetDefaultConfig(struct rwc_config *config);
00546
00547 enum rwc_status RWC_SetLFEbypass(RWC_Type *base,
00548     enum rwc_lfe_bypass value);
00549
00550 enum rwc_status RWC_SetLFX(RWC_Type *base,
00551     enum rwc_rtclk_type value);
00552
00553 enum rwc_status RWC_GetLastAPIStatus();
00554
00555 enum rwc_status RWC_SetResetCtrl(RWC_Type *base,
00556     enum rwc_reset_type value);
00557
00558 #if defined(__cplusplus)
00559 }
00560 #endif
00561
00562 uint32_t RWC_ConvertDatetimeToSeconds(const rtc_datetime_t *datetime);
00563
00564 #endif /* HAL_RWC_H */
00565

```

## 6.42 Файл devices/eliot1/drivers/hal\_sdmmc.h

Интерфейс драйвера модуля SDMMC.

```
#include "hal_common.h"
```

## Структуры данных

- struct `sdmmc_card_t`  
Контекст драйвера контроллера SDMMC.

## Макросы

- #define `SDMMC_CALC_DIVIDER`(input\_freq\_hz, required\_freq\_hz) (((input\_freq\_hz) / (required\_freq\_hz)) / 2)  
Формула подсчета делителя выходной частоты контроллера SDMMC.

## Константы контроллера SDMMC

- #define `SDMMC_SDHC_SECTOR_SIZE` 512
- #define `SDMMC_SD_SEND_IF_COND_PATTERN` 0x1AA
- #define `SDMMC_SD_OCR_INIT_VALUE` 0xFF80
- #define `SDMMC_MMC_RCA_ADDR` 0x00010000
- #define `SDMMC_TIMEOUTCONTROL_MAX_VALUE` 0xE

## Типы слота карты контроллера SDMMC

- #define `SDMMC_CORECFG0_SLOTTYPE_REMOVABLE` 0
- #define `SDMMC_CORECFG0_SLOTTYPE_EMBEDDED` 1
- #define `SDMMC_CORECFG0_SLOTTYPE_SHARED_BUS` 2

## Режимы UHS-I карты SD

- #define `SDMMC_SD_UHS_MODE_DEFAULT_SDR12` 0
- #define `SDMMC_SD_UHS_MODE_HIGHSEED_SDR25` 1
- #define `SDMMC_SD_UHS_MODE_SDR50` 2
- #define `SDMMC_SD_UHS_MODE_SDR104` 3
- #define `SDMMC_SD_UHS_MODE_DDR50` 4

## Выравнивание адреса буфера данных SDMA

Возможные варианты значений:

- 0 - 4KB (Detects A11 Carry out)
- 1 - 8KB (Detects A12 Carry out)
- 2 - 16KB (Detects A13 Carry out)
- 3 - 32KB (Detects A14 Carry out)
- 4 - 64KB (Detects A15 Carry out)
- 5 - 128KB (Detects A16 Carry out)
- 6 - 256KB (Detects A17 Carry out)
- 7 - 512KB (Detects A18 Carry out)
- #define `SDMMC_SDMA_ALIGN` 0
- #define `SDMMC_SDMA_BLOCK_SIZE` (4096 << `SDMMC_SDMA_ALIGN`)
- #define `SDMMC_SDMA_BLOCK_ALIGN` \_\_attribute\_\_((aligned(`SDMMC_SDMA_BLOCK_SIZE`))))
- #define `SDMMC_SDMA_IS_BLOCK_ALIGN_ADDR`(x) (((uint32\_t)(x) & (`SDMMC_SDMA_BLOCK_SIZE` - 1)) == 0)

## Перечисления

- enum [sdmmc\\_status\\_t](#)  
Статусы драйвера SDMMC.
- enum [sdmmc\\_voltage\\_t](#)  
Рабочие напряжения питания карты

## Рабочие напряжения контроллера SDMMC

- enum

## Направления передачи SDMA канала контроллера SDMMC

- enum

## Типы и размеры ответов карты

- enum

## Ширина шины данных карты в битах

- enum

## Функции

- [sdmmc\\_status\\_t](#) [SDMMC\\_InitCard](#) ([sdmmc\\_card\\_t](#) \*sd, [uint32\\_t](#) num, [sdmmc\\_voltage\\_t](#) vol, void \*init\_buf)  
Инициализация SDMMC контроллера и вставленной карты SD или MMC.
- void [SDMMC\\_DisableCard](#) ([sdmmc\\_card\\_t](#) \*sd)  
Остановка SDMMC контроллера и выключение питания вставленной карты
- [sdmmc\\_status\\_t](#) [SDMMC\\_CalcMemorySpace](#) ([sdmmc\\_card\\_t](#) \*sd, void \*sector\_buf, bool unsafe)  
Подсчет размера пространства памяти карты
- [sdmmc\\_status\\_t](#) [SDMMC\\_Read](#) ([sdmmc\\_card\\_t](#) \*sd, [uint32\\_t](#) start\_block, void \*data, [uint32\\_t](#) nblocks)  
Чтение карты блоками размером 512 байт
- [sdmmc\\_status\\_t](#) [SDMMC\\_ReadAsync](#) ([sdmmc\\_card\\_t](#) \*sd, [uint32\\_t](#) start\_block, void \*data, [uint32\\_t](#) nblocks)  
Асинхронное чтение карты блоками размером 512 байт
- [sdmmc\\_status\\_t](#) [SDMMC\\_ReadWait](#) ([sdmmc\\_card\\_t](#) \*sd)  
Ожидание завершения операции асинхронного чтения памяти карты
- [sdmmc\\_status\\_t](#) [SDMMC\\_Write](#) ([sdmmc\\_card\\_t](#) \*sd, [uint32\\_t](#) start\_block, const void \*data, [uint32\\_t](#) nblocks)  
Запись карты блоками размером 512 байт
- [sdmmc\\_status\\_t](#) [SDMMC\\_WriteAsync](#) ([sdmmc\\_card\\_t](#) \*sd, [uint32\\_t](#) start\_block, const void \*data, [uint32\\_t](#) nblocks)  
Асинхронная запись карты блоками размером 512 байт
- [sdmmc\\_status\\_t](#) [SDMMC\\_WriteWait](#) ([sdmmc\\_card\\_t](#) \*sd)  
Ожидание завершения операции асинхронной записи памяти карты

Количество слотов под карты и их типы

- `#define SDMMC_IS_MMC(x) ((x)->type == SDMMC_TypeMMC)`
- `#define SDMMC_IS_SD(x) ((x)->type == SDMMC_TypeSD)`
- `enum`

### 6.42.1 Подробное описание

Интерфейс драйвера модуля SDMMC.

## 6.43 `hal_sdmmc.h`

[См. документацию.](#)

```
00001
00021 #ifndef HAL_SDMMC_H
00022 #define HAL_SDMMC_H
00023
00024 #if defined(_cplusplus)
00025 extern "C" {
00026 #endif /* _cplusplus */
00027
00028 #include "hal_common.h"
00029
00034 enum {
00035     SDMMC_TypeMMC = 0,
00036     SDMMC_TypeSD = 1
00037 };
00038
00039 #define SDMMC_IS_MMC(x) ((x)->type == SDMMC_TypeMMC)
00040 #define SDMMC_IS_SD(x) ((x)->type == SDMMC_TypeSD)
00053 #define SDMMC_CALC_DIVIDER(input_freq_hz, required_freq_hz) (((input_freq_hz) / (required_freq_hz)) / 2)
00054
00055
00060 #define SDMMC_SDHC_SECTOR_SIZE 512
00061 #define SDMMC_SD_SEND_IF_COND_PATTERN 0x1AA
00062 #define SDMMC_SD_OCR_INIT_VALUE 0xFF80
00063 #define SDMMC_MMC_RCA_ADDR 0x00010000
00064 #define SDMMC_TIMEOUTCONTROL_MAX_VALUE 0xE
00073 enum {
00074     SDMMC_HostPWR_3V3 = 0x7,
00075     SDMMC_HostPWR_3V0 = 0x6,
00076     SDMMC_HostPWR_1V8 = 0x5
00077 };
00086 #define SDMMC_CORECFG0_SLOTTYPE_REMOVABLE 0
00087 #define SDMMC_CORECFG0_SLOTTYPE_EMBEDDED 1
00088 #define SDMMC_CORECFG0_SLOTTYPE_SHARED_BUS 2
00097 enum {
00098     SDMMC_SDMA_TransferWrite = 0,
00099     SDMMC_SDMA_TransferRead = 1
00100 };
00109 enum {
00110     SDMMC_NoResponse = 0,
00111     SDMMC_ResponseLength136 = 1,
00112     SDMMC_ResponseLength48 = 2,
00113     SDMMC_ResponseLength48_Check = 3
00114 };
00123 enum {
00124     SDMMC_DataBusTransferWidth_1Bit = 0,
00125     SDMMC_DataBusTransferWidth_4bit = 1,
00126     SDMMC_ExtDataBusTransferWidth_8bit = 1
00127 };
00136 #define SDMMC_SD_UHS_MODE_DEFAULT_SDR12 0
00137 #define SDMMC_SD_UHS_MODE_HIGHSPEED_SDR25 1
00138 #define SDMMC_SD_UHS_MODE_SDR50 2
00139 #define SDMMC_SD_UHS_MODE_SDR104 3
00140 #define SDMMC_SD_UHS_MODE_DDR50 4
00160 #define SDMMC_SDMA_ALIGN 0
00161 #define SDMMC_SDMA_BLOCK_SIZE (4096 « SDMMC_SDMA_ALIGN)
00162 #define SDMMC_SDMA_BLOCK_ALIGN __attribute__((aligned(SDMMC_SDMA_BLOCK_SIZE)))
00163 #define SDMMC_SDMA_IS_BLOCK_ALIGN_ADDR(x) (((uint32_t) (x) & (SDMMC_SDMA_BLOCK_SIZE - 1)) == 0)
00171 typedef enum {
00172     SDMMC_Status_Ok = 0,
00173     SDMMC_Status_Err = 1
00174 } sdmmc_status_t;
```

```

00175
00179 typedef enum {
00180     SDMMC_3v3      = 0,
00181     SDMMC_1v8      = 1,
00182     SDMMC_3v3To1v8 = 2
00183 } sdmmc_voltage_t;
00184
00188 typedef struct {
00189     int32_t      dev_num;
00190     const sdmmc_port_cfg_t *cfg;
00191     SDMMC_Type    *regs;
00192     uint32_t      rca;
00193     int32_t      type;
00194     int32_t      sdhc_mode;
00195     int32_t      hs_mode;
00196     int32_t      ddr_mode;
00197     int32_t      version;
00198     sdmmc_voltage_t gpio_vol;
00199     int32_t      need_1V8en;
00200     uint32_t      freq_input;
00201     int32_t      freq_divider;
00202     int32_t      lock;
00203     uint64_t      total_size;
00204     uint32_t      cid[4];
00205     uint32_t      csd[4];
00206 } sdmmc_card_t;
00207
00222 sdmmc_status_t SDMMC_InitCard(sdmmc_card_t *sd, uint32_t num,
00223     sdmmc_voltage_t vol, void *init_buf);
00224
00230 void SDMMC_DisableCard(sdmmc_card_t *sd);
00231
00251 sdmmc_status_t SDMMC_CalcMemorySpace(sdmmc_card_t *sd, void *sector_buf,
00252     bool unsafe);
00253
00267 sdmmc_status_t SDMMC_Read(sdmmc_card_t *sd, uint32_t start_block, void *data,
00268     uint32_t nblocks);
00269
00283 sdmmc_status_t SDMMC_ReadAsync(sdmmc_card_t *sd, uint32_t start_block,
00284     void *data, uint32_t nblocks);
00285
00294 sdmmc_status_t SDMMC_ReadWait(sdmmc_card_t *sd);
00295
00309 sdmmc_status_t SDMMC_Write(sdmmc_card_t *sd, uint32_t start_block,
00310     const void *data, uint32_t nblocks);
00311
00325 sdmmc_status_t SDMMC_WriteAsync(sdmmc_card_t *sd, uint32_t start_block,
00326     const void *data, uint32_t nblocks);
00327
00336 sdmmc_status_t SDMMC_WriteWait(sdmmc_card_t *sd);
00337
00338 #if defined(__cplusplus)
00339 }
00340 #endif /* __cplusplus */
00341
00342 #endif /* HAL_SDMMC_H */
00343

```

## 6.44 Файл devices/eliot1/drivers/hal\_sdmmc\_cfg.h

Интерфейс конфигурации модуля SDMMC.

```
#include <stdint.h>
```

Структуры данных

- struct `sdmmc_cfg_pin_map`  
Конфигурация выводов контроллера SDMMC.
- struct `sdmmc_port_cfg_t`  
Конфигурация контроллера SDMMC.

### 6.44.1 Подробное описание

Интерфейс конфигурации модуля SDMMC.

## 6.45 hal\_sdmmc\_cfg.h

[См. документацию.](#)

```

00001
00011 #ifndef HAL_SDMMC_CFG_H
00012 #define HAL_SDMMC_CFG_H
00013
00014 #include <stdint.h>
00015
00019 typedef struct {
00020     uint8_t pull_en;
00021     uint8_t max_current;
00022     uint8_t CK;
00023     uint8_t CMD;
00024     uint8_t D0;
00025     uint8_t D1;
00026     uint8_t D2;
00027     uint8_t D3;
00028     uint8_t D4;
00029     uint8_t D5;
00030     uint8_t D6;
00031     uint8_t D7;
00032     uint8_t CD;
00033 } sdmmc_cfg_pin_map;
00034
00038 typedef struct {
00039     uint32_t reg_base[2];
00040     uint32_t slot_count;
00041     uint8_t slot_type;
00042     uint8_t emmc_8bit_en;
00043     uint8_t hs_en;
00044     uint8_t sd_uhs_mode;
00045     uint32_t freq_out_init;
00046     uint32_t freq_out;
00047     uint32_t timeout_cd;
00048     uint32_t timeout_init;
00049     void (*delay_us)(uint32_t);
00050     sdmmc_cfg_pin_map pin_map;
00051 } sdmmc_port_cfg_t;
00052
00053 #endif /* HAL_SDMMC_CFG_H */
00054

```

## 6.46 Файл devices/eliot1/drivers/hal\_smc.h

Интерфейс драйвера внешней статической памяти

```
#include "hal_common.h"
```

Макросы

- #define SMC\_NO\_DEACTIVATION 0  
Деактивация не выполняется

## Перечисления

- enum [smc\\_status](#)  
Статусы драйвера SMC.
- enum [smc\\_cmd\\_type](#)  
Тип конфигурационной команды
- enum [smc\\_cre](#)  
Значение выхода CRE при выполнении команды ModeReg.
- enum [smc\\_burst\\_align](#)  
Граница пакета памяти
- enum [smc\\_bls](#)  
Поведение выводов SMC\_NBLS.
- enum [smc\\_adv](#)  
Использование сигнала NADV.
- enum [smc\\_packet\\_lenght](#)  
Длина пакета данных в 16-битных словах при записи или чтении
- enum [smc\\_rd\\_wr\\_type](#)  
Тип интерфейса при записи или чтении
- enum [smc\\_bit\\_depth](#)  
Разрядность интерфейса памяти
- enum [smc\\_incr\\_to\\_incr4](#)  
Управление преобразованием АНВ-пакетов типа INCR в пакеты типа INCR4.

## Функции

### Интерфейс драйвера

- enum [smc\\_status SMC\\_PowerSaveOn](#) (SMC\_Type \*base)  
Включение энергосберегающего режима
- enum [smc\\_status SMC\\_PowerSaveOff](#) (SMC\_Type \*base)  
Выключение энергосберегающего режима
- enum [smc\\_status SMC\\_DirectCmd](#) (SMC\_Type \*base, uint32\_t chip\_select, enum [smc\\_cmd\\_type](#) cmd\_type, enum [smc\\_cre](#) set\_cre, uint32\_t addr)  
Отправка конфигурационных команд
- enum [smc\\_status SMC\\_SetCycles](#) (SMC\_Type \*base, uint32\_t ttr, uint32\_t tpc, uint32\_t twp, uint32\_t tceoe, uint32\_t twc, uint32\_t trc)  
Хранение новой конфигурации временных параметров интерфейса.
- enum [smc\\_status SMC\\_SetOpmode](#) (SMC\_Type \*base, enum [smc\\_burst\\_align](#) align, enum [smc\\_bls](#) bls, enum [smc\\_adv](#) adv, enum [smc\\_packet\\_lenght](#) wr\_lenght, enum [smc\\_rd\\_wr\\_type](#) wr\_sync, enum [smc\\_packet\\_lenght](#) rd\_lenght, enum [smc\\_rd\\_wr\\_type](#) rd\_sync, enum [smc\\_bit\\_depth](#) depth)  
Установка регистра SET\_OPMODE.
- enum [smc\\_status SMC\\_RefreshPeriod](#) (SMC\_Type \*base, uint32\_t period)  
Управление деактивацией микросхемы
- enum [smc\\_status SMC\\_UserConfig](#) (SMC\_Type \*base, enum [smc\\_incr\\_to\\_incr4](#) bank0, enum [smc\\_incr\\_to\\_incr4](#) bank1, uint32\_t smcclkdiv)  
Запись в регистр USER\_CONFIG.
- bool [SMC\\_CheckConfigure](#) (SMC\_Type \*base, uint32\_t chip\_select, uint32\_t cycles, uint32\_t mode)  
Проверка завершения установки конфигурации

### 6.46.1 Подробное описание

#### Интерфейс драйвера внешней статической памяти

## 6.47 hal\_smc.h

См. документацию.

```

00001
00020 #ifndef HAL_SMC_H
00021 #define HAL_SMC_H
00022
00023 #ifdef __cplusplus
00024 extern "C" {
00025 #endif
00026
00027 #include "hal_common.h"
00028
00032 enum smc_status {
00033     SMC_Status_Ok = 0,
00034     SMC_Status_InvalidArgument = 1,
00035 };
00036
00040 enum smc_cmd_type {
00041     SMC_CmdTypeUpdateRegsAndAHBCommand = 0,
00042     SMC_CmdTypeModeReg = 1,
00043     SMC_CmdTypeUpdateRegs = 2,
00044     SMC_CmdTypeModeRegAndUpdateRegs = 3,
00045 };
00046
00050 enum smc_cre {
00051     SMC_CRE0 = 0,
00052     SMC_CRE1 = 1,
00053 };
00054
00058 enum smc_burst_align {
00059     SMC_BurstAlignNo = 0,
00060     SMC_BurstAlign32 = 0,
00061     SMC_BurstAlign64 = 0,
00062     SMC_BurstAlign128 = 0,
00063     SMC_BurstAlign256 = 0,
00064 };
00065
00069 enum smc_bls {
00070     SMC_BlsAsNcs = 0,
00071     SMC_BlsAsNwe = 1,
00072 };
00073
00077 enum smc_adv {
00078     SMC_AdvNotUsed = 0,
00079     SMC_AdvUsed = 1,
00080     SMC_AdvLcd = SMC_AdvNotUsed,
00081     SMC_AdvMemory = SMC_AdvUsed,
00082 };
00083
00087 enum smc_packet_lenght {
00088     SMC_PacketLenght1 = 0,
00089     SMC_PacketLenght4 = 1,
00090     SMC_PacketLenght8 = 2,
00091     SMC_PacketLenghtEndless = 5,
00092 };
00093
00097 enum smc_rd_wr_type {
00098     SMC_RdWrAsync = 0,
00099     SMC_RdWrSync = 1,
00100 };
00101
00105 enum smc_bit_depth {
00106     SMC_BitDepth8 = 0,
00107     SMC_BitDepth16 = 1,
00108     SMC_BitDepth24 = 2,
00109     SMC_BitDepth32 = 3,
00110 };
00111
00115 enum smc_incr_to_incr4 {
00116     SMC_IncrToIncr4Enable = 0,
00117     SMC_IncrToIncr4Disable = 1,
00118 };
00119
00123 #define SMC_NO_DEACTIVATION 0
00124
00141 enum smc_status SMC_PowerSaveOn(SMC_Type *base);
00142
00154 enum smc_status SMC_PowerSaveOff(SMC_Type *base);
00155
00178 enum smc_status SMC_DirectCmd(SMC_Type *base, uint32_t chip_select,
00179     enum smc_cmd_type cmd_type, enum smc_cre set_cre, uint32_t addr);
00180
00201 enum smc_status SMC_SetCycles(SMC_Type *base, uint32_t ttr, uint32_t tpc,
00202     uint32_t twp, uint32_t tceoe, uint32_t twc, uint32_t trc);

```



```

00203
00225 enum smc_status SMC_SetOpmode(SMC_Type *base, enum smc_burst_align align,
00226     enum smc_bls bls, enum smc_adv adv, enum smc_packet_length wr_length,
00227     enum smc_rd_wr_type wr_sync, enum smc_packet_length rd_length,
00228     enum smc_rd_wr_type rd_sync, enum smc_bit_depth depth);
00229
00246 enum smc_status SMC_RefreshPeriod(SMC_Type *base, uint32_t period);
00247
00266 enum smc_status SMC_UserConfig(SMC_Type *base, enum smc_incr_to_incr4 bank0,
00267     enum smc_incr_to_incr4 bank1, uint32_t smcclkdiv);
00268
00282 bool SMC_CheckConfigure(SMC_Type *base, uint32_t chip_select,
00283     uint32_t cycles, uint32_t mode);
00284
00289 #ifdef __cplusplus
00290 }
00291 #endif
00292
00293 #endif /* HAL_SMC_H */
00294

```

## 6.48 Файл devices/eliot1/drivers/hal\_spi.h

Интерфейс драйвера модуля SPI.

```
#include "hal_common.h"
```

Структуры данных

- struct [spi\\_microwire\\_cfg\\_t](#)  
Конфигурация для протокола Microwire National Semiconductor.
- struct [spi\\_motorola\\_cfg\\_t](#)  
Конфигурация для протокола Motorola SPI.
- struct [spi\\_config\\_t](#)  
Структура конфигурации для Master SPI.
- struct [spi\\_transfer\\_t](#)  
Структура SPI для приемо-передачи
- struct [spi\\_half\\_duplex\\_transfer\\_t](#)  
Структура SPI для полудуплексной приемо-передачи в режиме Master.
- struct [spi\\_config\\_internal\\_t](#)  
Внутренняя структура конфигурации модуля SPI.
- struct [spi\\_handle](#)  
SPI структура дескриптора для работы по прерыванию

Макросы

- #define HAL\_SPI\_DRIVER\_VERSION ([MAKE\\_VERSION](#)(1, 1, 0))  
Версия драйвера SPI.
- #define [SPI\\_DUMMYDATA](#) (0xA5U)  
SPI фиктивные данные для передачи по умолчанию.
- #define [SPI\\_RETRY\\_TIMES](#) 0U

## Определения типов

- typedef struct [spi\\_handle](#) spi\_master\_handle\_t  
Дескриптор Master SPI для работы по прерыванию
- typedef struct [spi\\_handle](#) spi\_slave\_handle\_t  
Дескриптор Slave SPI для работы по прерыванию
- typedef void(\* spi\_master\_callback\_t) (SPI\_Type \*base, [spi\\_master\\_handle\\_t](#) \*handle, uint32\_t status, void \*user\_data)  
Прототип пользовательской функции обратного вызова Master SPI для вызова по окончании обмена
- typedef void(\* spi\_slave\_callback\_t) (SPI\_Type \*base, [spi\\_slave\\_handle\\_t](#) \*handle, uint32\_t status, void \*user\_data)  
Прототип пользовательской функции обратного вызова Slave SPI для вызова по окончании обмена

## Перечисления

- enum [spi\\_status](#)  
Статусы возврата из функций для драйвера SPI.
- enum [spi\\_shift\\_direction\\_t](#)  
Формат передачи данных (MSB или LSB)
- enum [spi\\_txfifo\\_watermark\\_t](#)  
Триггер уровня заполнения TxFIFO.
- enum [spi\\_rxfifo\\_watermark\\_t](#)  
Триггер уровня заполнения RxFIFO.
- enum [spi\\_frame\\_width\\_t](#)  
Размер кадра данных в 32-х битном режиме передачи данных (CTRLR0.DFS\_32)
- enum [spi\\_frame\\_format\\_t](#)  
Формат кадра передачи данных
- enum [spi\\_status\\_flags](#)  
SPI флаги статусов
- enum [spi\\_mode\\_t](#)  
Режим передачи (CTRLR0.TMOD)
- enum [spi\\_trans\\_status](#)  
Состояния приемо-передачи SPI.
- enum [spi\\_interrupt\\_enable](#)  
Источники прерываний SPI.

## National Semiconductor Microwire

- enum [microwire\\_ctrlword\\_len\\_t](#)  
Выбор длины управляющего слова для формата кадра передачи данных National Semiconductor Microwire.
- enum [microwire\\_busy\\_ready\\_check\\_t](#)  
Включить/отключить проверку busy/ready флага (регистр SR) для формата кадра передачи данных National Semiconductor Microwire.
- enum [microwire\\_tx\\_rx\\_t](#)  
Направление передачи слова данных для формата кадра передачи данных National Semiconductor Microwire.
- enum [microwire\\_single\\_serial\\_t](#)  
Выбор типа передачи (одиночная или последовательная) для формата кадра передачи данных National Semiconductor Microwire.

## Motorola SPI

- enum [spi\\_motorola\\_clk\\_pol\\_t](#)  
Выбор полярности тактового сигнала при отсутствии передаваемых данных в режиме Master для формата кадра передачи данных Motorola SPI.
- enum [spi\\_motorola\\_cap\\_data\\_t](#)  
Выбор фронта для захвата данных для формата кадра передачи данных Motorola SPI.

## Функции

- uint8\_t [SPI\\_GetInstance](#) (SPI\_Type \*base)  
Получение индекса модуля SPI.

## Инициализация и деинициализация

- void [SPI\\_MasterGetDefaultConfig](#) (spi\_config\_t \*config)  
Инициализация конфигурации SPI значениями по умолчанию
- enum spi\_status [SPI\\_MasterInit](#) (SPI\_Type \*base, const spi\_config\_t \*config, uint32\_t src←\_clock\_hz)  
Инициализация SPI модуля как Master с заданной конфигурацией
- void [SPI\\_SlaveGetDefaultConfig](#) (spi\_config\_t \*config)  
Заполнение структуры конфигурации SPI Slave-устройства значениями по умолчанию
- enum spi\_status [SPI\\_SlaveInit](#) (SPI\_Type \*base, const spi\_config\_t \*config)  
Инициализация SPI заданной конфигурацией
- void [SPI\\_Deinit](#) (SPI\_Type \*base)  
Деинициализация SPI.
- static void [SPI\\_Enable](#) (SPI\_Type \*base, bool enable)  
Включение или отключение модуль SPI Master или Slave.
- static bool [SPI\\_IsEnable](#) (SPI\_Type \*base)  
Получение статуса модуля SPI включено/выключено
- static void [SPI\\_Reset](#) (SPI\_Type \*base)  
Останов всех операций SPI.

## Статусы

- static uint32\_t [SPI\\_GetStatusFlags](#) (SPI\_Type \*base)  
Получение флага состояния SPI.

## Прерывания

- static void [SPI\\_EnableInterrupts](#) (SPI\_Type \*base, uint32\_t irq)
  - Включение прерываний SPI.
- static void [SPI\\_DisableInterrupts](#) (SPI\_Type \*base, uint32\_t irq)
  - Отключение прерываний SPI.
- static uint32\_t [SPI\\_CurrentStatusInterrupts](#) (SPI\_Type \*base)
  - Получение статуса прерываний
- static void [SPI\\_TakeDownInterrupts](#) (SPI\_Type \*base, uint32\_t irq)
  - Сброс флагов прерывания

## DMA управление

- void [SPI\\_EnableTxDMA](#) (SPI\_Type \*base, bool enable)  
Включение или отключение запроса DMA от SPI TxFIFO.
- void [SPI\\_EnableRxDMA](#) (SPI\_Type \*base, bool enable)  
Включение или отключение запроса DMA от SPI RxFIFO.

## Операции на шине SPI

- spi\_config\_internal\_t \* [SPI\\_GetConfig](#) (SPI\_Type \*base)  
Получение внутренней конфигурации
- enum spi\_status [SPI\\_MasterSetBaud](#) (SPI\_Type \*base, uint32\_t baudrate\_bps, uint32\_t←src\_clock\_hz)  
Установка скорости передачи для SPI Master.
- void [SPI\\_WriteData](#) (SPI\_Type \*base, uint32\_t data)

- Запись данных в регистр данных SPI.
- `uint32_t SPI_ReadData (SPI_Type *base)`  
Получение данных из регистра данных SPI.
- `void SPI_SetDummyData (SPI_Type *base, uint8_t dummy_data)`  
Запись фиктивных данных для передачи

#### Приемо-передача

- `enum spi_status SPI_MasterTransferCreateHandle (SPI_Type *base, spi_master_handle_t *handle, spi_master_callback_t callback, void *user_data)`  
Инициализация дескриптора SPI Master.
- `enum spi_status SPI_MasterTransferBlocking (SPI_Type *base, spi_transfer_t *xfer)`  
Блокирующая дуплексная передача данных (с ожиданием завершения операции)
- `enum spi_status SPI_MasterTransferNonBlocking (SPI_Type *base, spi_master_handle_t *handle, spi_transfer_t *xfer)`  
Неблокирующий SPI обмен по прерыванию
- `enum spi_status SPI_MasterHalfDuplexTransferBlocking (SPI_Type *base, spi_half_duplex_transfer_t *xfer)`  
Блокирующая полудуплексная передача данных (с ожиданием завершения операции)
- `status_t SPI_MasterHalfDuplexTransferNonBlocking (SPI_Type *base, spi_master_handle_t *handle, spi_half_duplex_transfer_t *xfer)`  
Выполнение неблокирующего SPI обмена по прерыванию
- `status_t SPI_MasterTransferGetByte (spi_master_handle_t *handle, size_t *count)`  
Получение количества переданных и принятых байт
- `status_t SPI_MasterTransferGetRemainingByte (spi_master_handle_t *handle, size_t *count)`  
Получение количества оставшихся байт для передачи и приема
- `status_t SPI_MasterTransferGetTotalByte (spi_master_handle_t *handle, size_t *count)`  
Получение общего количества байт на передачу и прием
- `void SPI_MasterTransferAbort (SPI_Type *base, spi_master_handle_t *handle)`  
Останов передачи для мастер режима
- `void SPI_MasterTransferHandleIRQ (SPI_Type *base, spi_master_handle_t *handle)`
- `static status_t SPI_SlaveTransferCreateHandle (SPI_Type *base, spi_slave_handle_t *handle, spi_slave_callback_t callback, void *userData)`  
Инициализация slave SPI дескриптор
- `static status_t SPI_SlaveTransferNonBlocking (SPI_Type *base, spi_slave_handle_t *handle, spi_transfer_t *xfer)`  
Неблокирующая дуплексная передача данных (без ожидания завершения операции)
- `static status_t SPI_SlaveHalfDuplexTransferNonBlocking (SPI_Type *base, spi_slave_handle_t *handle, spi_half_duplex_transfer_t *xfer)`
- `static status_t SPI_SlaveTransferGetByte (spi_slave_handle_t *handle, size_t *count)`  
Получение количества байт для обмена
- `static void SPI_SlaveTransferAbort (SPI_Type *base, spi_slave_handle_t *handle)`  
Останов передачи для режима Slave по прерыванию.
- `static void SPI_SlaveTransferHandleIRQ (SPI_Type *base, spi_slave_handle_t *handle)`  
Обработчик прерываний для SPI.

#### 6.48.1 Подробное описание

Интерфейс драйвера модуля SPI.

## 6.49 hal\_spi.h

[См. документацию.](#)

```

00001
00023 #ifndef HAL_SPI_H
00024 #define HAL_SPI_H
00025
00026 #include "hal_common.h"
00027
00031 #define HAL_SPI_DRIVER_VERSION (MAKE_VERSION(1, 1, 0))
00032
00038 #ifndef SPI_DUMMYDATA
00039 #define SPI_DUMMYDATA (0xA5U)
00040 #endif
00041
00042 #ifndef SPI_RETRY_TIMES
00043 #define SPI_RETRY_TIMES 0U
00044 #endif
00045
00049 enum spi_status {
00050     SPI_Status_Ok                = 0,
00051     SPI_Status_Fail              = 1,
00052     SPI_Status_ReadOnly          = 2,
00053     SPI_Status_InvalidArgument  = 3,
00054     SPI_Status_Timeout           = 4,
00055     SPI_Status_BaudrateNotSupport = 5,
00056     SPI_Status_Busy              = 6,
00057     SPI_Status_Idle              = 9,
00058     SPI_Status_TxError           = 10,
00059     SPI_Status_RxError           = 11,
00060     SPI_Status_RxRingBufferOverrun = 12,
00061     SPI_Status_RxFifoBufferOverrun = 13,
00062     SPI_Status_NoTransferInProgress = 14,
00063     SPI_Status_IncorrectCall      = 15,
00064     SPI_Status_UnexpectedState    = 128U,
00065 };
00066
00070 typedef enum {
00071     SPI_ShiftDirMsbFirst = 0,
00072     SPI_ShiftDirLsbFirst = 1,
00073 } spi_shift_direction_t;
00074
00078 typedef enum {
00079     SPI_TxFifoWatermark0 = 0,
00080     SPI_TxFifoWatermark1 = 1,
00081     SPI_TxFifoWatermark2 = 2,
00082     SPI_TxFifoWatermark3 = 3,
00083     SPI_TxFifoWatermark4 = 4,
00084     SPI_TxFifoWatermark5 = 5,
00085     SPI_TxFifoWatermark6 = 6,
00086     SPI_TxFifoWatermark7 = 7,
00087 } spi_txfifo_watermark_t;
00088
00092 typedef enum {
00093     SPI_RxFifoWatermark1 = 0,
00094     SPI_RxFifoWatermark2 = 1,
00095     SPI_RxFifoWatermark3 = 2,
00096     SPI_RxFifoWatermark4 = 3,
00097     SPI_RxFifoWatermark5 = 4,
00098     SPI_RxFifoWatermark6 = 5,
00099     SPI_RxFifoWatermark7 = 6,
00100     SPI_RxFifoWatermark8 = 7,
00101 } spi_rxfifo_watermark_t;
00102
00107 typedef enum {
00108     SPI_Data4Bits = 3,
00109     SPI_Data5Bits = 4,
00110     SPI_Data6Bits = 5,
00111     SPI_Data7Bits = 6,
00112     SPI_Data8Bits = 7,
00113     SPI_Data9Bits = 8,
00114     SPI_Data10Bits = 9,
00115     SPI_Data11Bits = 10,
00116     SPI_Data12Bits = 11,
00117     SPI_Data13Bits = 12,
00118     SPI_Data14Bits = 13,
00119     SPI_Data15Bits = 14,
00120     SPI_Data16Bits = 15,
00121     SPI_Data17Bits = 16,
00122     SPI_Data18Bits = 17,
00123     SPI_Data19Bits = 18,
00124     SPI_Data20Bits = 19,
00125     SPI_Data21Bits = 20,
00126     SPI_Data22Bits = 21,
00127     SPI_Data23Bits = 22,

```

```

00128     SPI_Data24Bits = 23,
00129     SPI_Data25Bits = 24,
00130     SPI_Data26Bits = 25,
00131     SPI_Data27Bits = 26,
00132     SPI_Data28Bits = 27,
00133     SPI_Data29Bits = 28,
00134     SPI_Data30Bits = 29,
00135     SPI_Data31Bits = 30,
00136     SPI_Data32Bits = 31,
00137 } spi_frame_width_t;
00138
00159 typedef enum {
00160     SPI_FfMotorola = 0,
00161     SPI_FfTexas = 1,
00162     SPI_FfMicrowire = 2,
00163 } spi_frame_format_t;
00164
00174 typedef enum {
00175     SPI_MicrowireCtrlWordLen1Bit = 0,
00176     SPI_MicrowireCtrlWordLen2Bit = 1,
00177     SPI_MicrowireCtrlWordLen3Bit = 2,
00178     SPI_MicrowireCtrlWordLen4Bit = 3,
00179     SPI_MicrowireCtrlWordLen5Bit = 4,
00180     SPI_MicrowireCtrlWordLen6Bit = 5,
00181     SPI_MicrowireCtrlWordLen7Bit = 6,
00182     SPI_MicrowireCtrlWordLen8Bit = 7,
00183     SPI_MicrowireCtrlWordLen9Bit = 8,
00184     SPI_MicrowireCtrlWordLen10Bit = 9,
00185     SPI_MicrowireCtrlWordLen11Bit = 10,
00186     SPI_MicrowireCtrlWordLen12Bit = 11,
00187     SPI_MicrowireCtrlWordLen13Bit = 12,
00188     SPI_MicrowireCtrlWordLen14Bit = 13,
00189     SPI_MicrowireCtrlWordLen15Bit = 14,
00190     SPI_MicrowireCtrlWordLen16Bit = 15,
00191 } microwire_ctrlword_len_t;
00192
00200 typedef enum {
00201     SPI_MicrowireBusyReadyCheckDisable = 0,
00202     SPI_MicrowireBusyReadyCheckEnable = 1,
00203 } microwire_busy_ready_check_t;
00204
00209 typedef enum {
00210     SPI_MicrowireTx = 0,
00211     SPI_MicrowireRx = 1,
00212 } microwire_tx_rx_t;
00213
00218 typedef enum {
00219     SPI_MicrowireSingle = 0,
00220     SPI_MicrowireSerial = 1,
00221 } microwire_single_serial_t;
00222
00226 typedef struct {
00227     microwire_ctrlword_len_t ctrl_word_len;
00228     microwire_busy_ready_check_t busy_ready_check;
00229     microwire_tx_rx_t tx_rx;
00230     microwire_single_serial_t single_serial;
00231 } spi_microwire_cfg_t;
00232
00246 typedef enum {
00247     SPI_MotorolaClkPolLow = 0,
00248     SPI_MotorolaClkPolHi = 1,
00249 } spi_motorola_clk_pol_t;
00250
00255 typedef enum {
00256     SPI_MotorolaCapDataRising = 0,
00257     SPI_MotorolaCapDataFalling = 1,
00258 } spi_motorola_cap_data_t;
00259
00263 typedef struct {
00264     spi_motorola_clk_pol_t clk_pol;
00265     spi_motorola_cap_data_t cap_data;
00266 } spi_motorola_cfg_t;
00267
00275 enum spi_status_flags {
00276     SPI_TxNotFullFlag = SPI_SR_TFNF_Msk,
00277     SPI_TxEmptyFlag = SPI_SR_TFE_Msk,
00278     SPI_RxNotEmptyFlag = SPI_SR_RFNE_Msk,
00279     SPI_RxFullFlag = SPI_SR_RFF_Msk,
00280 };
00281
00299 typedef enum {
00300     SPI_ModeDuplex = 0,
00301     SPI_ModeSimplexTx = 1,
00302     SPI_ModeSimplexRx = 2,
00303     SPI_ModeHalfDuplex = 3,
00304 } spi_mode_t;
00305

```

```

00309 typedef struct {
00310     struct {
00311         bool                loopback_enable;
00312         uint32_t            baud_rate_bps;
00313     } master;
00314     spi_frame_width_t       data_width_bits;
00315     spi_shift_direction_t   direction;
00316     spi_frame_format_t      frame_format;
00317     spi_microwire_cfg_t     microwire_cfg;
00318     spi_motorola_cfg_t      motorola_cfg;
00319 } spi_config_t;
00320
00324 enum spi_trans_status {
00325     SPI_TransStatus_Busy    = 0,
00326     SPI_TransStatus_Idle    = 1,
00327     SPI_TransStatus_Error   = 128,
00328     SPI_TransStatus_Error_1 = 129,
00329     SPI_TransStatus_Error_2 = 130,
00330     SPI_TransStatus_Error_3 = 132,
00331     SPI_TransStatus_Error_4 = 144,
00332 };
00333
00342 enum spi_interrupt_enable {
00343     SPI_IRQ_MultiMaster    = SPI_IMR_MSTIM_Msk,
00344     SPI_IRQ_RxFifoTrigger  = SPI_IMR_RXFIM_Msk,
00345     SPI_IRQ_RxFifoOverflow = SPI_IMR_RXOIM_Msk,
00346     SPI_IRQ_RxFifoUnderflow = SPI_IMR_RXUIM_Msk,
00347     SPI_IRQ_TxFifoOverflow = SPI_IMR_TXOIM_Msk,
00348     SPI_IRQ_TxFifoTrigger  = SPI_IMR_TXEIM_Msk,
00349     SPI_IRQ_TxOnly
00350     = SPI_IRQ_MultiMaster
00351     | SPI_IRQ_TxFifoOverflow
00352     | SPI_IRQ_TxFifoTrigger,
00353     /* Мультимастер */
00354     /* Переполнение TxFIFO */
00355     /* Прерывание по триггеру уровня TxFIFO */
00356     SPI_IRQ_RxOnly
00357     = SPI_IRQ_MultiMaster
00358     | SPI_IRQ_RxFifoTrigger
00359     | SPI_IRQ_RxFifoOverflow
00360     | SPI_IRQ_RxFifoUnderflow,
00361     /* Мультимастер */
00362     /* Прерывание по триггеру уровня Rx FIFO */
00363     /* Переполнение Rx FIFO */
00364     /* Чтение из пустого Rx FIFO */
00365     SPI_IRQ_All
00366     = SPI_IRQ_MultiMaster
00367     | SPI_IRQ_RxFifoTrigger
00368     | SPI_IRQ_RxFifoOverflow
00369     | SPI_IRQ_RxFifoUnderflow
00370     | SPI_IRQ_TxFifoOverflow
00371     | SPI_IRQ_TxFifoTrigger,
00372 };
00373
00377 typedef struct {
00378     uint8_t *tx_data;
00379     uint8_t *rx_data;
00380     size_t data_size;
00381 } spi_transfer_t;
00382
00386 typedef struct {
00387     uint8_t *tx_data;
00388     uint8_t *rx_data;
00389     size_t tx_data_size;
00390     size_t rx_data_size;
00391 } spi_half_duplex_transfer_t;
00392
00396 typedef struct {
00397     spi_shift_direction_t shift_dir;
00398     uint8_t                frame_width_bits;
00399     uint8_t                frame_width_bytes;
00400 } spi_config_internal_t;
00401
00405 struct spi_handle;
00406
00410 typedef struct spi_handle spi_master_handle_t;
00411
00415 typedef struct spi_handle spi_slave_handle_t;
00416
00420 typedef void (*spi_master_callback_t)(SPI_Type *base,
00421     spi_master_handle_t *handle, uint32_t status, void *user_data);
00422
00426 typedef void (*spi_slave_callback_t)(SPI_Type *base, spi_slave_handle_t *handle,
00427     uint32_t status, void *user_data);
00428
00437 struct spi_handle {
00438     volatile uint8_t *tx_data;
00439     volatile uint8_t *rx_data;
00440     volatile size_t tx_remaining_bytes;
00441     volatile size_t rx_remaining_bytes;
00442     int8_t instance;
00443     size_t total_byte_to_transfer;

```

```

00444     int32_t          state;
00445     spi_master_callback_t callback;
00446     void             *user_data;
00447     uint8_t          frame_width_bits;
00448     uint8_t          frame_width_bytes;
00449     spi_mode_t        mode;
00450 };
00451
00452 #if defined(__cplusplus)
00453 extern "C" {
00454 #endif
00455
00463 uint8_t SPI_GetInstance(SPI_Type *base);
00464
00485 void SPI_MasterGetDefaultConfig(spi_config_t *config);
00486
00513 enum spi_status SPI_MasterInit(SPI_Type *base, const spi_config_t *config,
00514     uint32_t src_clock_hz);
00515
00532 void SPI_SlaveGetDefaultConfig(spi_config_t *config);
00533
00556 enum spi_status SPI_SlaveInit(SPI_Type *base, const spi_config_t *config);
00557
00566 void SPI_Deinit(SPI_Type *base);
00567
00574 static inline void SPI_Enable(SPI_Type *base, bool enable)
00575 {
00576     /* Включить/выключить SPI (SSIENR.SSI_EN). */
00577     SET_VAL_MSK(base->SSIENR, SPI_SSIENR_SSI_EN_Msk, SPI_SSIENR_SSI_EN_Pos,
00578         enable);
00579 }
00580
00589 static inline bool SPI_IsEnable(SPI_Type *base)
00590 {
00591     /* SPI включено/выключено (SSIENR.SSI_EN). */
00592     return (bool)GET_VAL_MSK(base->SSIENR, SPI_SSIENR_SSI_EN_Msk,
00593         SPI_SSIENR_SSI_EN_Pos);
00594 }
00595
00604 static inline void SPI_Reset(SPI_Type *base)
00605 {
00606     /* Выключить SPI. */
00607     SET_VAL_MSK(base->SSIENR, SPI_SSIENR_SSI_EN_Msk, SPI_SSIENR_SSI_EN_Pos, 0U);
00608
00609     /* Включить SPI. */
00610     SET_VAL_MSK(base->SSIENR, SPI_SSIENR_SSI_EN_Msk, SPI_SSIENR_SSI_EN_Pos, 1U);
00611 }
00612
00632 static inline uint32_t SPI_GetStatusFlags(SPI_Type *base)
00633 {
00634     assert(NULL != base);
00635     return base->SR;
00636 }
00637
00663 static inline void SPI_EnableInterrupts(SPI_Type *base, uint32_t irqs)
00664 {
00665     assert(NULL != base);
00666
00667     /*
00668      * IMR - регистр маскирования прерываний:
00669      * 0 - прерывание замаскировано, 1 - активно.
00670      */
00671     base->IMR |= irqs;
00672 }
00673
00690 static inline void SPI_DisableInterrupts(SPI_Type *base, uint32_t irqs)
00691 {
00692     assert(NULL != base);
00693
00694     /*
00695      * IMR - регистр маскирования прерываний:
00696      * 0 - прерывание замаскировано, 1 - активно.
00697      */
00698     base->IMR &= ~irqs;
00699 }
00700
00708 static inline uint32_t SPI_CurrentStatusInterrupts(SPI_Type *base)
00709 {
00710
00711     assert(NULL != base);
00712
00713     /* ISR - регистр статуса прерываний после маскирования. */
00714     return (base->ISR & SPI_IRQ_All);
00715 }
00716
00730 static inline void SPI_TakeDownInterrupts(SPI_Type *base, uint32_t irqs)
00731 {

```



```

00732     assert(NULL != base);
00733     assert(0 == (irqs & SPI_IRQ_RxFifoTrigger));
00734     assert(0 == (irqs & SPI_IRQ_TxFifoTrigger));
00735
00736     volatile uint32_t temp = 0;
00737     UNUSED(temp);
00738
00739     /* xxxICR - Регистры снятия прерываний. */
00740
00741     /* Требуется сброс всех прерываний (допустимых для сброса таким образом). */
00742     if ((SPI_IRQ_MultiMaster
00743          | SPI_IRQ_RxFifoOverflow
00744          | SPI_IRQ_RxFifoUnderflow
00745          | SPI_IRQ_TxFifoOverflow) == irq) {
00746         temp = GET_VAL_MSK(base->ICR, SPI_ICR_ICR_Msk, SPI_ICR_ICR_Pos);
00747     }
00748     /* Требуется сброс некоторых прерываний. */
00749     else {
00750         if (0 != (irqs & SPI_IRQ_MultiMaster)) {
00751             /* MSTICR - Регистр сброса прерывания: "Возможный конфликт master устройств". */
00752             temp = GET_VAL_MSK(base->MSTICR, SPI_MSTICR_MSTICR_Msk, SPI_MSTICR_MSTICR_Pos);
00753         }
00754
00755         if (0 != (irqs & SPI_IRQ_RxFifoOverflow)) {
00756             /* RXOICR - Регистр сброса прерывания: "Переполнен FIFO приемника". */
00757             temp = GET_VAL_MSK(base->RXOICR, SPI_RXOICR_RXOICR_Msk, SPI_RXOICR_RXOICR_Pos);
00758         }
00759
00760         if (0 != (irqs & SPI_IRQ_RxFifoUnderflow)) {
00761             /* RXUICR - Регистр сброса прерывания: "FIFO приемника пуст". */
00762             temp = GET_VAL_MSK(base->RXUICR, SPI_RXUICR_RXUICR_Msk, SPI_RXUICR_RXUICR_Pos);
00763         }
00764
00765         if (0 != (irqs & SPI_IRQ_TxFifoOverflow)) {
00766             /* TXOICR - Регистр сброса прерывания: "Переполнен FIFO передатчика". */
00767             temp = GET_VAL_MSK(base->TXOICR, SPI_TXOICR_TXOICR_Msk, SPI_TXOICR_TXOICR_Pos);
00768         }
00769     }
00770
00771     return;
00772 }
00773
00789 void SPI_EnableTxDMA(SPI_Type *base, bool enable);
00790
00797 void SPI_EnableRxDMA(SPI_Type *base, bool enable);
00798
00815 spi_config_internal_t *SPI_GetConfig(SPI_Type *base);
00816
00828 enum spi_status SPI_MasterSetBaud(SPI_Type *base, uint32_t baudrate_bps,
00829     uint32_t src_clock_hz);
00830
00837 void SPI_WriteData(SPI_Type *base, uint32_t data);
00838
00846 uint32_t SPI_ReadData(SPI_Type *base);
00847
00857 void SPI_SetDummyData(SPI_Type *base, uint8_t dummy_data);
00858
00879 enum spi_status SPI_MasterTransferCreateHandle(SPI_Type *base,
00880     spi_master_handle_t *handle, spi_master_callback_t callback,
00881     void *user_data);
00882
00900 enum spi_status SPI_MasterTransferBlocking(SPI_Type *base,
00901     spi_transfer_t *xfer);
00902
00926 enum spi_status SPI_MasterTransferNonBlocking(SPI_Type *base,
00927     spi_master_handle_t *handle, spi_transfer_t *xfer);
00928
00944 enum spi_status SPI_MasterHalfDuplexTransferBlocking(SPI_Type *base,
00945     spi_half_duplex_transfer_t *xfer);
00946
00962 status_t SPI_MasterHalfDuplexTransferNonBlocking(SPI_Type *base,
00963     spi_master_handle_t *handle,
00964     spi_half_duplex_transfer_t *xfer);
00965
00975 status_t SPI_MasterTransferGetByte(spi_master_handle_t *handle, size_t *count);
00976
00986 status_t SPI_MasterTransferGetRemainingByte(spi_master_handle_t *handle,
00987     size_t *count);
00988
00998 status_t SPI_MasterTransferGetTotalByte(spi_master_handle_t *handle,
00999     size_t *count);
01000
01009 void SPI_MasterTransferAbort(SPI_Type *base, spi_master_handle_t *handle);
01010
01017 void SPI_MasterTransferHandleIRQ(SPI_Type *base, spi_master_handle_t *handle);
01018
01033 static inline status_t SPI_SlaveTransferCreateHandle(SPI_Type *base,

```

```

01034     spi_slave_handle_t *handle,
01035     spi_slave_callback_t callback,
01036     void *userData)
01037 {
01038     return SPI_MasterTransferCreateHandle(base, handle, callback, userData);
01039 }
01040
01053 static inline status_t SPI_SlaveTransferNonBlocking(SPI_Type *base,
01054     spi_slave_handle_t *handle, spi_transfer_t *xfer)
01055 {
01056     return SPI_MasterTransferNonBlocking(base, handle, xfer);
01057 }
01058
01059 static inline status_t SPI_SlaveHalfDuplexTransferNonBlocking(SPI_Type *base,
01060     spi_slave_handle_t *handle, spi_half_duplex_transfer_t *xfer)
01061 {
01062     return SPI_MasterHalfDuplexTransferNonBlocking(base, handle, xfer);
01063 }
01073 static inline status_t SPI_SlaveTransferGetByte(spi_slave_handle_t *handle,
01074     size_t *count)
01075 {
01076     return SPI_MasterTransferGetByte((spi_master_handle_t *) handle, count);
01077 }
01078
01085 static inline void SPI_SlaveTransferAbort(SPI_Type *base, spi_slave_handle_t *handle)
01086 {
01087     SPI_MasterTransferAbort(base, (spi_master_handle_t *) handle);
01088 }
01089
01096 static inline void SPI_SlaveTransferHandleIRQ(SPI_Type *base,
01097     spi_slave_handle_t *handle)
01098 {
01099     SPI_MasterTransferHandleIRQ(base, handle);
01100 }
01101
01106 #if defined(__cplusplus)
01107 }
01108 #endif
01109
01110 #endif /* HAL_SPI_H */
01111

```

## 6.50 Файл devices/eliot1/drivers/hal\_spi\_dma.h

Дополнение драйвера SPI с пересылкой данных через DMA.

```

#include "hal_dma.h"
#include "hal_spi.h"

```

Структуры данных

- struct `_spi_dma_handle`  
Дескриптор SPI-DMA.

Макросы

- #define HAL\_SPI\_DMA\_DRIVER\_VERSION (`MAKE_VERSION`(1, 0, 0))  
Версия драйвера

Определения типов

- typedef struct `_spi_dma_handle` spi\_dma\_handle\_t  
Дескриптор SPI-DMA.
- typedef void(\* spi\_dma\_callback\_t) (SPI\_Type \*base, `spi_dma_handle_t` \*handle, void \*user←  
\_data, `dma_irq_t` inttype)  
Функция обратного вызова

## Функции

- enum `spi_status SPI_MasterTransferCreateHandleDMA` (`SPI_Type *base`, `spi_dma_callback_t` `callback`, `void *user_data`, `spi_dma_handle_t *handle`, `dma_handle_t *tx_handle`, `dma_handle_t *rx_handle`)  
Функция инициализации дескриптора SPI-DMA.
- enum `spi_status SPI_MasterTransferDMA` (`SPI_Type *base`, `spi_dma_handle_t *handle`, `spi_transfer_t *xfer`)  
Функция для SPI master приема/передачи данных в полнодуплексном режиме через DMA каналы в порт SPI.
- enum `spi_status SPI_MasterHalfDuplexTransferDMA` (`SPI_Type *base`, `spi_dma_handle_t *handle`, `spi_half_duplex_transfer_t *xfer`)  
Функция приема/передачи данных в полудуплексном режиме через DMA каналы в порт SPI.
- static enum `spi_status SPI_SlaveTransferCreateHandleDMA` (`SPI_Type *base`, `spi_dma_callback_t` `callback`, `void *user_data`, `spi_dma_handle_t *handle`, `dma_handle_t *tx_handle`, `dma_handle_t *rx_handle`)  
Функция инициализации дескриптора SPI-DMA.
- static enum `spi_status SPI_SlaveTransferDMA` (`SPI_Type *base`, `spi_dma_handle_t *handle`, `spi_transfer_t *xfer`)  
Функция для SPI slave приема/передачи данных в полнодуплексном режиме через DMA каналы в порт SPI.
- void `SPI_MasterTransferAbortDMA` (`spi_dma_handle_t *handle`)  
Прекращение передачи SPI.
- static void `SPI_SlaveTransferAbortDMA` (`spi_dma_handle_t *handle`)  
Прекращение передачи SPI.
- static void `SPI_DMADescriptorInitTX` (`SPI_Type *base`, `dma_descriptor_t *desc`, `uint32_t count`, `uint32_t data_size`, `uint8_t src_incr`, `uint32_t data_width`, `void *src_addr`)  
Инициализация дескрипторов DMA для многоблочной передачи TX.
- static void `SPI_DMADescriptorInitRX` (`SPI_Type *base`, `dma_descriptor_t *desc`, `uint32_t count`, `uint32_t data_size`, `uint8_t dst_incr`, `uint32_t data_width`, `void *dst_addr`)  
Инициализация дескрипторов DMA для многоблочной передачи RX.

## 6.50.1 Подробное описание

Дополнение драйвера SPI с пересылкой данных через DMA.

## 6.51 hal\_spi\_dma.h

См. документацию.

```

00001
00012 #ifndef HAL_SPI_DMA_H
00013 #define HAL_SPI_DMA_H
00014
00015 #include "hal_dma.h"
00016 #include "hal_spi.h"
00017
00019 #define HAL_SPI_DMA_DRIVER_VERSION (MAKE_VERSION(1, 0, 0))
00020
00024 typedef struct _spi_dma_handle spi_dma_handle_t;
00025
00027 typedef void (*spi_dma_callback_t)(SPI_Type *base, spi_dma_handle_t *handle,
00028     void *user_data, dma_irq_t inttype);
00029
00033 struct spi_dma_handle {
00034     volatile bool tx_in_progress;
00035     volatile bool rx_in_progress;
00036     dma_handle_t *tx_handle;
00037     dma_descriptor_t *tx_desc;

```

```

00038 dma_handle_t *rx_handle;
00039 dma_descriptor_t *rx_desc;
00040 uint8_t bytes_per_frame;
00041 spi_dma_callback_t callback;
00042 void *user_data;
00043 enum spi_trans_status state;
00044 size_t transfer_size;
00045 void *dummy_data;
00046 };
00047
00048 #if defined(__cplusplus)
00049 extern "C" {
00050 #endif
00051
00052 enum spi_status SPI_MasterTransferCreateHandleDMA(SPI_Type *base,
00053 spi_dma_callback_t callback, void *user_data, spi_dma_handle_t *handle,
00054 dma_handle_t *tx_handle, dma_handle_t *rx_handle);
00055
00056 enum spi_status SPI_MasterTransferDMA(SPI_Type *base,
00057 spi_dma_handle_t *handle, spi_transfer_t *xfer);
00058
00059 enum spi_status SPI_MasterHalfDuplexTransferDMA(SPI_Type *base,
00060 spi_dma_handle_t *handle, spi_half_duplex_transfer_t *xfer);
00061
00062 static inline enum spi_status SPI_SlaveTransferCreateHandleDMA(
00063 SPI_Type *base, spi_dma_callback_t callback, void *user_data,
00064 spi_dma_handle_t *handle, dma_handle_t *tx_handle, dma_handle_t *rx_handle)
00065 {
00066 return SPI_MasterTransferCreateHandleDMA(base, callback, user_data,
00067 handle, tx_handle, rx_handle);
00068 }
00069
00070 static inline enum spi_status SPI_SlaveTransferDMA(SPI_Type *base,
00071 spi_dma_handle_t *handle, spi_transfer_t *xfer)
00072 {
00073 return SPI_MasterTransferDMA(base, handle, xfer);
00074 }
00075
00076 void SPI_MasterTransferAbortDMA(spi_dma_handle_t *handle);
00077
00078 static inline void SPI_SlaveTransferAbortDMA(spi_dma_handle_t *handle)
00079 {
00080 SPI_MasterTransferAbortDMA(handle);
00081 }
00082
00083 static inline void SPI_DMADescriptorInitTX(SPI_Type *base,
00084 dma_descriptor_t *desc, uint32_t count, uint32_t data_size,
00085 uint8_t src_incr, uint32_t data_width, void *src_addr)
00086 {
00087 dma_multiblock_config_t config = {
00088 .count = count,
00089 .data_size = data_size,
00090 .transfer_type = DMA_MemoryToPeripheral_DMA,
00091 .scatter_en = false,
00092 .gather_en = false,
00093 .src_burst_size = DMA_BurstSize32,
00094 .dst_burst_size = DMA_BurstSize1,
00095 .src_incr = src_incr,
00096 .dst_incr = DMA_NoChange,
00097 .src_data_width = data_width,
00098 .dst_data_width = data_width,
00099 .src_addr = src_addr,
00100 .dst_addr = (void *) &base->DR0,
00101 .int_en = true
00102 };
00103
00104 DMA_InitMultiblockDescriptor(desc, &config);
00105 }
00106
00107 static inline void SPI_DMADescriptorInitRX(SPI_Type *base,
00108 dma_descriptor_t *desc, uint32_t count, uint32_t data_size,
00109 uint8_t dst_incr, uint32_t data_width, void *dst_addr)
00110 {
00111 dma_multiblock_config_t config = {
00112 .count = count,
00113 .data_size = data_size,
00114 .transfer_type = DMA_PeripheralToMemory_DMA,
00115 .scatter_en = false,
00116 .gather_en = false,
00117 .src_burst_size = DMA_BurstSize1,
00118 .dst_burst_size = DMA_BurstSize32,
00119 .src_incr = DMA_NoChange,
00120 .dst_incr = dst_incr,
00121 .src_data_width = data_width,
00122 .dst_data_width = data_width,
00123 .src_addr = (void *) &base->DR0,
00124 .dst_addr = dst_addr,

```

```

00221     .int_en      = true
00222 };
00223
00224 DMA_InitMultiblockDescriptor(desc, &config);
00225 }
00226
00227 #if defined(__cplusplus)
00228 }
00229 #endif
00230
00231 #endif /* HAL_SPI_DMA_H */
00232

```

## 6.52 Файл devices/eliot1/drivers/hal\_timer.h

Интерфейс драйвера модуля таймеров общего назначения

```
#include "hal_common.h"
```

Структуры данных

- struct [timer\\_hardware\\_config](#)  
Конфигурация аппаратной части таймера общего назначения

Макросы

- #define [TIMER\\_COUNT](#) 3
- #define [TIMER\\_HARDWARE\\_FIELD\\_MAX](#) (0xFFFFFFFFFUL)
- #define [TIMER\\_SOFTWARE\\_FIELD\\_MAX](#) (0xFFFFFFFFFFFFFFFFFULL)
- #define [TIMER\\_SOFTWARE\\_FIELD\\_HIGH\\_OFFSET](#) (32)

Определения типов

- typedef void(\* callback\_t) (TIM\_Type \*base)  
Функция обратного вызова

Перечисления

- enum [timer\\_status](#)  
Статусы драйвера таймеров общего назначения
- enum [timer\\_type\\_of\\_counting](#)  
Режимы счета импульсов таймером
- enum [timer\\_work\\_mode](#)  
Режим работы таймера общего назначения

## Функции

## Интерфейс драйвера

- enum `timer_status` `TIMER_Init` (`TIM_Type *base`, struct `timer_hardware_config` `config`, enum `timer_work_mode` `mode`, `callback_t` `callback`, `uint32_t` `ticks_h`)  
Инициализация таймера общего назначения
- enum `timer_status` `TIMER_Deinit` (`TIM_Type *base`)  
Деинициализация таймера общего назначения
- enum `timer_status` `TIMER_Run` (`TIM_Type *base`)  
Запуск таймера общего назначения
- enum `timer_status` `TIMER_Stop` (`TIM_Type *base`)  
Остановка таймера общего назначения
- enum `timer_status` `TIMER_Reset` (`TIM_Type *base`)  
Сброс таймера общего назначения
- `uint64_t` `TIMER_GetTicks` (`TIM_Type *base`)  
Получение количества тиков
- enum `timer_status` `TIMER_SetTick` (`TIM_Type *base`, `uint64_t` `ticks`)  
Установка количества тиков
- enum `timer_status` `TIMER_GetAPIStatus` ()  
Получение результата выполнения последней функции
- static `uint32_t` `TIMER_GetTimerHardwareValue` (`TIM_Type *base`)  
Получение значения регистра счетчика таймера
- enum `timer_status` `TIMER_SetConfig` (`TIM_Type *base`, struct `timer_hardware_config` `config`, enum `timer_work_mode` `mode`, `callback_t` `callback`, `uint32_t` `ticks_h`)  
Инициализация структуры таймера общего назначения
- enum `timer_status` `TIMER_IRQEnable` (`TIM_Type *base`)  
Включение прерывания
- enum `timer_status` `TIMER_IRQDisable` (`TIM_Type *base`)  
Отключение прерывания
- `uint32_t` `TIMER_IRQGetStatus` (`TIM_Type *base`)  
Чтение статуса прерывания
- enum `timer_status` `TIMER_IRQClear` (`TIM_Type *base`)  
Сброс прерывания

## 6.52.1 Подробное описание

Интерфейс драйвера модуля таймеров общего назначения

6.53 `hal_timer.h`

[См. документацию.](#)

```

00001
00021 #ifndef HAL_TIMER_H
00022 #define HAL_TIMER_H
00023
00024 #if defined(__cplusplus)
00025 extern "C" {
00026 #endif /* __cplusplus */
00027
00028 #include "hal_common.h"
00029
00030 #define TIMER_COUNT 3
00031 #define TIMER_HARDWARE_FIELD_MAX (0xFFFFFFFFFUL)
00032 #define TIMER_SOFTWARE_FIELD_MAX (0xFFFFFFFFFFFFFFFFULL)
00033 #define TIMER_SOFTWARE_FIELD_HIGH_OFFSET (32)
00038 enum timer_status {
00039     TIMER_Status_Ok = 0,
00040     TIMER_Status_InvalidArgument = 1,
00041     TIMER_Status_TimerBusy = 2,
00042     TIMER_Status_BadConfigure = 3,

```

```

00043     TIMER_Status_NotIni           = 4,
00044     TIMER_Status_NotSupport       = 5,
00045 };
00046
00050 enum timer_type_of_counting {
00051     TIMER_Work = 0,
00052     TIMER_Debug = 1,
00053 };
00054
00058 enum timer_work_mode {
00059     TIMER_Hardware = 0,
00060     TIMER_Software = 1,
00061 };
00062
00066 struct timer_hardware_config {
00067     uint32_t start_value;
00068     uint32_t reload_value;
00069     uint32_t interrupt_enable;
00070     enum timer_type_of_counting work_type;
00071     uint32_t start_enable;
00072 };
00073
00077 typedef void (*callback_t)(TIM_Type *base);
00078
00102 enum timer_status TIMER_Init(TIM_Type *base,
00103     struct timer_hardware_config config, enum timer_work_mode mode,
00104     callback_t callback, uint32_t ticks_h);
00105
00114 enum timer_status TIMER_Deinit(TIM_Type *base);
00115
00124 enum timer_status TIMER_Run(TIM_Type *base);
00125
00134 enum timer_status TIMER_Stop(TIM_Type *base);
00135
00144 enum timer_status TIMER_Reset(TIM_Type *base);
00145
00153 uint64_t TIMER_GetTicks(TIM_Type *base);
00154
00163 enum timer_status TIMER_SetTick(TIM_Type *base, uint64_t ticks);
00164
00178 enum timer_status TIMER_GetAPIStatus();
00179
00187 static inline uint32_t TIMER_GetTimerHardwareValue(TIM_Type *base)
00188 {
00189     return base->VALUE;
00190 }
00191
00210 enum timer_status TIMER_SetConfig(TIM_Type *base,
00211     struct timer_hardware_config config, enum timer_work_mode mode,
00212     callback_t callback, uint32_t ticks_h);
00213
00224 enum timer_status TIMER_IRQEnable(TIM_Type *base);
00225
00236 enum timer_status TIMER_IRQDisable(TIM_Type *base);
00237
00246 uint32_t TIMER_IRQGetStatus(TIM_Type *base);
00247
00256 enum timer_status TIMER_IRQClear(TIM_Type *base);
00257
00262 #if defined(__cplusplus)
00263 }
00264 #endif /* __cplusplus */
00265
00266 #endif /* HAL_TIMER_H */
00267

```

## 6.54 Файл devices/eliot1/drivers/hal\_uart.h

Интерфейс драйвера UART.

```
#include "hal_common.h"
```

Структуры данных

- struct `uart_config`

- Конфигурация UART.
- struct [uart\\_transfer](#)  
Указатель на буфер приема или передачи
- struct [uart\\_handle](#)  
Дескриптор состояния приема/передачи для неблокирующих функций обмена

## Макросы

- #define HAL\_UART\_DRIVER\_VERSION ([MAKE\\_VERSION](#)(0, 1, 0))  
Версия драйвера UART.
- #define UART\_RETRY\_TIMES 0U /\* 0 - ожидание до получения значения \*/  
Количество циклов ожидания

## Определения типов

- typedef void(\* [uart\\_transfer\\_callback\\_t](#)) (UART\_Type \*base, struct [uart\\_handle](#) \*handle, enum [uart\\_status](#) status, void \*user\_data)  
Callback-функция

## Перечисления

- enum [uart\\_status](#)  
Статусы драйвера UART.
- enum [uart\\_interrupt\\_enable](#)  
Конфигурация прерываний для UART.
- enum [uart\\_lsr\\_flags](#)  
Флаги состояния UART LSR.
- enum [uart\\_parity\\_mode](#)  
Режимы четности UART.
- enum [uart\\_stop\\_bit\\_count](#)  
Количество стоп-битов для UART.
- enum [uart\\_data\\_len](#)  
Количество бит данных в передаваемом символе
- enum [uart\\_txfifo\\_watermark](#)  
Триггер уровня заполненности TxFIFO.
- enum [uart\\_rxfifo\\_watermark](#)  
Триггер уровня заполненности RxFIFO.
- enum [uart\\_rs485\\_mode](#)  
Режим работы RS485.
- enum [uart\\_rs485\\_active\\_state](#)  
Активное состояние линии для RS485.



## Функции

## Инициализация и деинициализация

- enum [uart\\_status](#) [UART\\_Init](#) (UART\_Type \*base, const struct [uart\\_config](#) \*config, uint32\_t src\_clock\_hz)  
Инициализирует модуль UART структурой конфигурации пользователя и частотой периферии
- enum [uart\\_status](#) [UART\\_Deinit](#) (UART\_Type \*base)  
Деинициализирует модуль UART.
- enum [uart\\_status](#) [UART\\_GetDefaultConfig](#) (struct [uart\\_config](#) \*config)  
Получает структуру конфигурации по умолчанию
- enum [uart\\_status](#) [UART\\_SetBaudRate](#) (UART\_Type \*base, uint32\_t baudrate\_bps, uint32\_t src\_clock\_hz)  
Устанавливает скорость модуля UART.

## Состояние

- static uint32\_t [UART\\_GetStatusFlags](#) (UART\_Type \*base)  
Извлекает флаги состояния UART.

## Включение/выключение и настройка прерываний

- static void [UART\\_EnableInterrupts](#) (UART\_Type \*base, uint32\_t mask)  
Разрешает прерывания UART в соответствии с предоставленной маской
- static void [UART\\_DisableInterrupts](#) (UART\_Type \*base, uint32\_t mask)  
Отключает прерывания UART в соответствии с предоставленной маской
- static uint32\_t [UART\\_GetEnabledInterrupts](#) (UART\_Type \*base)  
Запрос маски включенных прерываний в UART.
- static void [UART\\_SetRxFifoWatermark](#) (UART\_Type \*base, enum [uart\\_rxfifo\\_watermark](#) water)  
Устанавливает триггер уровня заполненности RxFIFO.
- static void [UART\\_SetTxFifoWatermark](#) (UART\_Type \*base, enum [uart\\_txfifo\\_watermark](#) water)  
Устанавливает триггер уровня заполненности TxFIFO.

## Включение/выключение и настройка расширенных режимов работы UART

- static void [UART\\_SetLoopback](#) (UART\_Type \*base, bool enable)  
Включение/выключение режима петли
- static void [UART\\_SetIr](#) (UART\_Type \*base, bool enable)  
Включение/выключение инфракрасного режима работы
- static void [UART\\_SetRs485](#) (UART\_Type \*base, bool enable)  
Включение/выключение RS485 режима работы
- static void [UART\\_Rs485Mode](#) (UART\_Type \*base, enum [uart\\_rs485\\_mode](#) mode, enum [uart\\_rs485\\_active\\_state](#) de, enum [uart\\_rs485\\_active\\_state](#) re)  
Установка режима работы RS485.

## Прием и передача без использования прерываний

- static void [UART\\_WriteByte](#) (UART\_Type \*base, uint8\_t data)  
Записывает данные на передачу в регистр передачи
- static void [UART\\_WriteByteWait](#) (UART\_Type \*base, uint8\_t data)  
Записывает данные на передачу в регистр передачи с ожиданием освобождения места
- static uint8\_t [UART\\_ReadByte](#) (UART\_Type \*base)  
Вычитывает байт из регистра приема
- static uint8\_t [UART\\_ReadByteWait](#) (UART\_Type \*base)

- Вычитывает байт из регистра приема с ожиданием получения
- `static uint8_t UART_GetRxFifoCount (UART_Type *base)`  
Получить количество байтов в RxFIFO.
- `static uint8_t UART_GetTxFifoCount (UART_Type *base)`  
Получить количество байтов в TxFIFO.
- `enum uart_status UART_WriteBlocking (UART_Type *base, const uint8_t *data, size_t length)`  
Записывает в регистр TX с использованием метода блокировки
- `enum uart_status UART_ReadBlocking (UART_Type *base, uint8_t *data, size_t length)`  
Чтение регистра данных RX с использованием метода блокировки

Прием данных через прерывания с использованием буферов

- `enum uart_status UART_TransferStartRingBuffer (UART_Type *base, struct uart_handle *handle, uint8_t *ring_buffer, size_t ring_buffer_size)`  
Инициализация кольцевого буфера на прием
- `enum uart_status UART_TransferStopRingBuffer (UART_Type *base, struct uart_handle *handle)`  
Прерывает фоновую передачу и удаляет кольцевой буфер
- `size_t UART_TransferGetRxRingBufferLength (struct uart_handle *handle)`  
Получить длину данных, принятых в кольцевом RX буфере
- `enum uart_status UART_TransferReceiveNonBlocking (UART_Type *base, struct uart_handle *handle, struct uart_transfer *xfer, size_t *received_bytes)`  
Прием данных в асинхронном режиме (без ожидания) по прерыванию
- `enum uart_status UART_TransferAbortReceive (UART_Type *base, struct uart_handle *handle)`  
Отмена приема данных по прерыванию через линейный буфер в дескрипторе
- `enum uart_status UART_TransferGetReceiveCount (UART_Type *base, struct uart_handle *handle, uint32_t *count)`  
Возвращает количество принятых байтов

Отправка данных через прерывания с использованием буферов

- `enum uart_status UART_TransferSendNonBlocking (UART_Type *base, struct uart_handle *handle, struct uart_transfer *xfer)`  
Передает буфер данных по прерыванию
- `enum uart_status UART_TransferAbortSend (UART_Type *base, struct uart_handle *handle)`  
Останавливает передачу данных, управляемую прерыванием
- `enum uart_status UART_TransferGetSendCount (UART_Type *base, struct uart_handle *handle, uint32_t *count)`  
Возвращает количество байтов, отправленных в шину
- `enum uart_status UART_TransferStartTxRingBuffer (UART_Type *base, struct uart_handle *handle, uint8_t *tx_ring_buffer, size_t ring_buffer_size)`  
Инициализация кольцевого буфера на передачу
- `size_t UART_TransferGetTxRingBufferLength (struct uart_handle *handle)`  
Получить количество байт данных для отправки в кольцевом Tx буфере
- `enum uart_status UART_WriteTxRing (UART_Type *base, struct uart_handle *handle, const uint8_t *data, size_t length)`  
Передать линейный буфер на передачу через кольцевой буфер
- `enum uart_status UART_TransferStopTxRingBuffer (UART_Type *base, struct uart_handle *handle)`  
Отключение кольцевого буфера передатчика

Прием и передача через прерывания с использованием буферов

- `enum uart_status UART_TransferCreateHandle (UART_Type *base, struct uart_handle *handle, uart_transfer_callback_t callback, void *user_data)`  
Инициализирует дескриптор UART.
- `void UART_TransferHandleIRQ (UART_Type *base, struct uart_handle *handle)`  
Функция-обработчик UART IRQ.

## 6.54.1 Подробное описание

Интерфейс драйвера UART.

## 6.55 hal\_uart.h

[См. документацию.](#)

```

00001
00025 #ifndef HAL_UART_H
00026 #define HAL_UART_H
00027
00028 #include "hal_common.h"
00029
00033 #define HAL_UART_DRIVER_VERSION (MAKE_VERSION(0, 1, 0))
00034
00038 #ifndef UART_RETRY_TIMES
00039 #define UART_RETRY_TIMES 0U /* 0 - ожидание до получения значения */
00040 #endif /* UART_RETRY_TIMES */
00041
00045 enum uart_status {
00046     UART_Status_Ok           = 0U,
00047     UART_Status_Fail         = 1U,
00048     UART_Status_ReadOnly     = 2U,
00049     UART_Status_InvalidArgument = 3U,
00050     UART_Status_Timeout      = 4U,
00051     UART_Status_NoTransferInProgress = 5U,
00052     UART_Status_TxBusy       = 6U,
00053     UART_Status_RxBusy       = 7U,
00054     UART_Status_TxIdle       = 8U,
00055     UART_Status_RxIdle       = 9U,
00056     UART_Status_TxError      = 10U,
00057     UART_Status_RxError      = 11U,
00058     UART_Status_RxRingBufferOverrun = 12U,
00059     UART_Status_RxFifoBufferOverrun = 13U,
00060     UART_Status_BreakLineError = 14U,
00061     UART_Status_FramingError  = 15U,
00062     UART_Status_ParityError   = 16U,
00063     UART_Status_BaudrateNotSupport = 17U,
00064     UART_Status_TxRingBufferNull = 18U,
00065 };
00066
00070 enum uart_interrupt_enable {
00071     UART_ThresholdInterruptEnable = (UART_IER_PTIME_Msk),
00072     UART_ModemInterruptEnable    = (UART_IER_EDSSI_Msk),
00073     UART_RxLineInterruptEnable   = (UART_IER_ELSI_Msk),
00074     UART_TxInterruptEnable       = (UART_IER_ETBEI_Msk),
00075     UART_RxInterruptEnable       = (UART_IER_ERBFI_Msk),
00076     UART_AllInterruptsEnable     = (UART_ThresholdInterruptEnable
00077     | UART_ModemInterruptEnable
00078     | UART_RxLineInterruptEnable
00079     | UART_TxInterruptEnable
00080     | UART_RxInterruptEnable),
00081 };
00082
00086 enum uart_lsr_flags {
00087     UART_LSR_FlagRxError      = (UART_LSR_RFE_Msk),
00088     UART_LSR_FlagTxHwEmpty    = (UART_LSR_TEMT_Msk),
00089     UART_LSR_FlagTxSwEmpty    = (UART_LSR_THRE_Msk),
00090     UART_LSR_FlagRxLinebreakError = (UART_LSR_BI_Msk),
00091     UART_LSR_FlagRxFrameError  = (UART_LSR_FE_Msk),
00092     UART_LSR_FlagRxParityError = (UART_LSR_PE_Msk),
00093     UART_LSR_FlagRxOverflowError = (UART_LSR_OE_Msk),
00094     UART_LSR_FlagRxReady      = (UART_LSR_DR_Msk),
00095 };
00096
00100 enum uart_parity_mode {
00101     UART_ParityOdd = 0U,
00102     UART_ParityEven = 1U,
00103 };
00104
00108 enum uart_stop_bit_count {
00109     UART_OneStopBit = 0U,
00110     UART_TwoOrOneAndHalfStopBit = 1U,
00111 };
00112
00116 enum uart_data_len {
00117     UART_5BitsPerChar = 0U,
00118     UART_6BitsPerChar = 1U,
00119     UART_7BitsPerChar = 2U,
00120     UART_8BitsPerChar = 3U,

```

```

00121 };
00122
00126 enum uart_txfifo_watermark {
00127     UART_TxFifoEmpty      = 0U,
00128     UART_TxFifoTwoChars   = 1U,
00129     UART_TxFifoQuarterFull = 2U,
00130     UART_TxFifoHalfFull   = 3U,
00131 };
00132
00136 enum uart_rxfifo_watermark {
00137     UART_RxFifoOneChar     = 0U,
00138     UART_RxFifoQuarterFull = 1U,
00139     UART_RxFifoHalfFull    = 2U,
00140     UART_RxFifoTwoToFull   = 3U,
00141 };
00142
00146 enum uart_rs485_mode {
00147     UART_RS485_ModeFullDuplex      = 0U,
00148     UART_RS485_ModeHalfDuplexManual = 1U,
00149     UART_RS485_ModeHalfDuplexAuto  = 2U,
00150 };
00151
00155 enum uart_rs485_active_state {
00156     UART_RS485_ActiveStateHigh = 0U,
00157     UART_RS485_ActiveStateLow  = 1U,
00158 };
00159
00163 struct uart_config {
00164     uint32_t      baudrate_bps;
00165     bool          enable_parity;
00166     enum uart_parity_mode parity_mode;
00167     bool          parity_manual;
00168     enum uart_stop_bit_count stop_bit_count;
00169     enum uart_data_len      bit_count_per_char;
00170     bool                  enable_rxfifo;
00171     bool                  enable_txfifo;
00172     bool                  enable_loopback;
00173     bool                  enable_infrared;
00174     bool                  enable_hardware_flow_control;
00175     bool                  break_line;
00176     /* enum uart_txfifo_watermark tx_watermark; */
00177     /* enum uart_rxfifo_watermark rx_watermark; */
00178     /* bool          rs485_enable; */
00179     /* enum uart_rs485_mode      rs485_mode; */
00180     /* bool          rs485_de_active_state; */
00181     /* bool          rs485_re_active_state; */
00186 };
00187
00193 struct uart_transfer {
00194     union {
00195         uint8_t *rx_data;
00196         uint8_t const *tx_data;
00197     };
00198     size_t data_size;
00199 };
00200
00201 /* Объявление, описание ниже. */
00202 struct uart_handle;
00203
00207 typedef void (*uart_transfer_callback_t)(UART_Type *base,
00208     struct uart_handle *handle, enum uart_status status, void *user_data);
00209
00214 struct uart_handle {
00215     volatile const uint8_t *tx_data;
00216     volatile size_t tx_data_size;
00217     size_t tx_data_size_all;
00219     uint8_t *tx_ring_buffer;
00220     size_t tx_ring_buffer_size;
00221     volatile uint16_t tx_ring_buffer_head;
00222     volatile uint16_t tx_ring_buffer_tail;
00224     volatile uint8_t *rx_data;
00225     volatile size_t rx_data_size;
00226     size_t rx_data_size_all;
00228     uint8_t *rx_ring_buffer;
00229     size_t rx_ring_buffer_size;
00230     volatile uint16_t rx_ring_buffer_head;
00231     volatile uint16_t rx_ring_buffer_tail;
00233     uart_transfer_callback_t callback;
00234     void *user_data;
00236     volatile uint8_t tx_state;
00237     volatile uint8_t rx_state;
00238 };
00239
00240 #if defined(__cplusplus)
00241 extern "C" {
00242 #endif /* __cplusplus */
00243

```

```

00280 enum uart_status UART_Init(UART_Type *base, const struct uart_config *config,
00281     uint32_t src_clock_hz);
00282
00292 enum uart_status UART_Deinit(UART_Type *base);
00293
00318 enum uart_status UART_GetDefaultConfig(struct uart_config *config);
00319
00336 enum uart_status UART_SetBaudRate(UART_Type *base, uint32_t baudrate_bps,
00337     uint32_t src_clock_hz);
00338
00366 static inline uint32_t UART_GetStatusFlags(UART_Type *base)
00367 {
00368     return (base->LSR & 0xFFUL);
00369 }
00370
00394 static inline void UART_EnableInterrupts(UART_Type *base, uint32_t mask)
00395 {
00396     /*
00397      * Работаем только с прерываниями, зарегистрированными в
00398      * @ref uart_interrupt_enable.
00399      */
00400     base->IER |= mask & UART_AllInterruptsEnable;
00401 }
00402
00418 static inline void UART_DisableInterrupts(UART_Type *base, uint32_t mask)
00419 {
00420     mask &= UART_AllInterruptsEnable;
00421     base->IER &= ~mask;
00422 }
00423
00434 static inline uint32_t UART_GetEnabledInterrupts(UART_Type *base)
00435 {
00436     return base->IER & UART_AllInterruptsEnable;
00437 }
00438
00446 static inline void UART_SetRxFifoWatermark(UART_Type *base,
00447     enum uart_rxfifo_watermark water)
00448 {
00449     SET_VAL_MSK(base->FCR, UART_FCR_RT_Msk, UART_FCR_RT_Pos, water);
00450 }
00451
00459 static inline void UART_SetTxFifoWatermark(UART_Type *base,
00460     enum uart_txfifo_watermark water)
00461 {
00462     SET_VAL_MSK(base->FCR, UART_FCR_TET_Msk, UART_FCR_TET_Pos, water);
00463 }
00464
00481 static inline void UART_SetLoopback(UART_Type *base, bool enable)
00482 {
00483     SET_VAL_MSK(base->MCR, UART_MCR_LOOPBACK_Msk, UART_MCR_LOOPBACK_Pos,
00484         enable);
00485 }
00486
00494 static inline void UART_SetIr(UART_Type *base, bool enable)
00495 {
00496     SET_VAL_MSK(base->MCR, UART_MCR_SIRE_Msk, UART_MCR_SIRE_Pos, enable);
00497 }
00498
00506 static inline void UART_SetRs485(UART_Type *base, bool enable)
00507 {
00508     SET_VAL_MSK(base->TCR, UART_TCR_RS485_EN_Msk, UART_TCR_RS485_EN_Pos,
00509         enable);
00510 }
00511
00521 static inline void UART_Rs485Mode(UART_Type *base, enum uart_rs485_mode mode,
00522     enum uart_rs485_active_state de, enum uart_rs485_active_state re)
00523 {
00524     SET_VAL_MSK(base->TCR, UART_TCR_XFER_MODE_Msk, UART_TCR_XFER_MODE_Pos,
00525         mode);
00526     SET_VAL_MSK(base->TCR, UART_TCR_DE_POL_Msk, UART_TCR_DE_POL_Pos, de);
00527     SET_VAL_MSK(base->TCR, UART_TCR_RE_POL_Msk, UART_TCR_RE_POL_Pos, re);
00528 }
00529
00549 static inline void UART_WriteByte(UART_Type *base, uint8_t data)
00550 {
00551     /* Доступен, если LCR[7](DLAB) == 0x0. */
00552     base->THR = data;
00553 }
00554
00566 static inline void UART_WriteByteWait(UART_Type *base, uint8_t data)
00567 {
00568     /* Пока есть данные в буфере передатчика. */
00569     while (GET_VAL_MSK(base->LSR, UART_LSR_THRE_Msk, UART_LSR_THRE_Pos) == 0U)
00570         ;
00571
00572     /* Доступен, если LCR[7](DLAB) == 0x0. */
00573     base->THR = data;

```

```

00574 }
00575
00587 static inline uint8_t UART_ReadByte(UART_Type *base)
00588 {
00589     /* Доступен, если LCR[7](DLAB) == 0x0. */
00590     return (uint8_t) base->RBR;
00591 }
00592
00604 static inline uint8_t UART_ReadByteWait(UART_Type *base)
00605 {
00606     /* Пока нет данных в приемнике. */
00607     while (GET_VAL_MSK(base->LSR, UART_LSR_DR_Msk, UART_LSR_DR_Pos) == 0U);
00608
00609     /* Доступен, если LCR[7](DLAB) == 0x0. */
00610     return (uint8_t) base->RBR;
00611 }
00612
00621 static inline uint8_t UART_GetRxFifoCount(UART_Type *base)
00622 {
00623     return (uint8_t) GET_VAL_MSK(base->TFL, UART_RFL_RFL_Msk, UART_RFL_RFL_Pos);
00624 }
00625
00634 static inline uint8_t UART_GetTxFifoCount(UART_Type *base)
00635 {
00636     return (uint8_t) GET_VAL_MSK(base->TFL, UART_TFL_TFL_Msk, UART_TFL_TFL_Pos);
00637 }
00638
00653 enum uart_status UART_WriteBlocking(UART_Type *base, const uint8_t *data,
00654     size_t length);
00655
00674 enum uart_status UART_ReadBlocking(UART_Type *base, uint8_t *data,
00675     size_t length);
00676
00711 enum uart_status UART_TransferStartRingBuffer(UART_Type *base,
00712     struct uart_handle *handle, uint8_t *ring_buffer, size_t ring_buffer_size);
00713
00725 enum uart_status UART_TransferStopRingBuffer(UART_Type *base,
00726     struct uart_handle *handle);
00727
00735 size_t UART_TransferGetRxRingBufferLength(struct uart_handle *handle);
00736
00771 enum uart_status UART_TransferReceiveNonBlocking(UART_Type *base,
00772     struct uart_handle *handle, struct uart_transfer *xfer,
00773     size_t *received_bytes);
00774
00790 enum uart_status UART_TransferAbortReceive(UART_Type *base,
00791     struct uart_handle *handle);
00792
00804 enum uart_status UART_TransferGetReceiveCount(UART_Type *base,
00805     struct uart_handle *handle, uint32_t *count);
00806
00842 enum uart_status UART_TransferSendNonBlocking(UART_Type *base,
00843     struct uart_handle *handle, struct uart_transfer *xfer);
00844
00858 enum uart_status UART_TransferAbortSend(UART_Type *base,
00859     struct uart_handle *handle);
00860
00874 enum uart_status UART_TransferGetSendCount(UART_Type *base,
00875     struct uart_handle *handle, uint32_t *count);
00876
00891 enum uart_status UART_TransferStartTxRingBuffer(UART_Type *base,
00892     struct uart_handle *handle,
00893     uint8_t *tx_ring_buffer, size_t ring_buffer_size);
00894
00902 size_t UART_TransferGetTxRingBufferLength(struct uart_handle *handle);
00903
00918 enum uart_status UART_WriteTxRing(UART_Type *base, struct uart_handle *handle,
00919     const uint8_t *data, size_t *length);
00920
00930 enum uart_status UART_TransferStopTxRingBuffer(UART_Type *base,
00931     struct uart_handle *handle);
00932
00956 enum uart_status UART_TransferCreateHandle(UART_Type *base,
00957     struct uart_handle *handle, uart_transfer_callback_t callback,
00958     void *user_data);
00959
00968 void UART_TransferHandleIRQ(UART_Type *base, struct uart_handle *handle);
00969
00974 #if defined(__cplusplus)
00975 }
00976 #endif /* __cplusplus */
00977
00982 #endif /* HAL_UART_H */

```

## 6.56 Файл devices/eliot1/drivers/hal\_uart\_dma.h

Дополнение драйвера UART с пересылкой данных через DMA.

```
#include "hal_dma.h"
#include "hal_uart.h"
```

Структуры данных

- struct `_uart_dma_handle`  
Дескриптор UART-DMA передачи

Макросы

- #define HAL\_UART\_DMA\_DRIVER\_VERSION ([MAKE\\_VERSION](#)(1, 0, 0))  
Версия драйвера

Определения типов

- typedef struct `_uart_dma_handle` `uart_dma_handle_t`  
Дескриптор UART-DMA передачи
- typedef void(\* `uart_dma_transfer_callback_t`) (UART\_Type \*base, `uart_dma_handle_t` \*handle, enum `uart_status` status, void \*user\_data)  
Функция обратного вызова

Функции

- void [UART\\_TransferCreateHandleDMA](#) (UART\_Type \*base, `uart_dma_handle_t` \*handle, `uart_dma_transfer_callback_t` callback, void \*user\_data, `dma_handle_t` \*tx\_dma\_handle, `dma_handle_t` \*rx\_dma\_handle)  
Функция для инициализации полей дескриптора UART-DMA.
- enum `uart_status` [UART\\_TransferSendDMA](#) (UART\_Type \*base, `uart_dma_handle_t` \*handle, struct `uart_transfer` \*xfer)  
Функция осуществляющая передачу данных по UART, данные в буфер попадают через DMA канал
- enum `uart_status` [UART\\_TransferReceiveDMA](#) (UART\_Type \*base, `uart_dma_handle_t` \*handle, struct `uart_transfer` \*xfer)  
Функция осуществляющая передачу данных по UART, данные в буфер попадают через DMA канал
- void [UART\\_TransferAbortSendDMA](#) (UART\_Type \*base, `uart_dma_handle_t` \*handle)  
Функция прерывающая передачу данных между UART(TX) и DMA.
- void [UART\\_TransferAbortReceiveDMA](#) (UART\_Type \*base, `uart_dma_handle_t` \*handle)  
Функция прерывающая передачу данных между UART(RX) и DMA.
- static void [UART\\_WaitWhileActive](#) (UART\_Type \*base)  
Ожидание завершения передачи. Выход из функции будет осуществлен по окончании UART транзакций
- static void [UART\\_DMADescriptorInitTX](#) (UART\_Type \*base, `dma_descriptor_t` \*desc, uint32\_t count, uint32\_t data\_size, uint32\_t data\_width, void \*src\_addr)  
Инициализация дескрипторов DMA для многоблочной передачи TX.
- static void [UART\\_DMADescriptorInitRX](#) (UART\_Type \*base, `dma_descriptor_t` \*desc, uint32\_t count, uint32\_t data\_size, uint32\_t data\_width, void \*dst\_addr)  
Инициализация дескрипторов DMA для многоблочной передачи TX.

### 6.56.1 Подробное описание

Дополнение драйвера UART с пересылкой данных через DMA.

### 6.57 hal\_uart\_dma.h

[См. документацию.](#)

```

00001
00012 #ifndef HAL_SPI_DMA_H
00013 #define HAL_SPI_DMA_H
00014
00015 #include "hal_dma.h"
00016 #include "hal_uart.h"
00017
00019 #define HAL_UART_DMA_DRIVER_VERSION (MAKE_VERSION(1, 0, 0))
00020
00024 typedef struct _uart_dma_handle uart_dma_handle_t;
00025
00027 typedef void (*uart_dma_transfer_callback_t)(UART_Type *base,
00028     uart_dma_handle_t *handle,
00029     enum uart_status status,
00030     void *user_data);
00031
00035 struct uart_dma_handle {
00036     UART_Type *base;
00038     uart_dma_transfer_callback_t callback;
00039     void *user_data;
00040     uint32_t rx_data_size_all;
00041     uint32_t tx_data_size_all;
00043     dma_handle_t *tx_dma_handle;
00044     dma_handle_t *rx_dma_handle;
00046     dma_descriptor_t *tx_desc;
00047     dma_descriptor_t *rx_desc;
00048
00049     volatile uint8_t tx_state;
00050     volatile uint8_t rx_state;
00051 };
00052
00053 #if defined(__cplusplus)
00054 extern "C" {
00055 #endif
00056
00067 void UART_TransferCreateHandleDMA(UART_Type *base,
00068     uart_dma_handle_t *handle,
00069     uart_dma_transfer_callback_t callback,
00070     void *user_data, dma_handle_t *tx_dma_handle,
00071     dma_handle_t *rx_dma_handle);
00072
00088 enum uart_status UART_TransferSendDMA(UART_Type *base,
00089     uart_dma_handle_t *handle, struct uart_transfer *xfer);
00090
00106 enum uart_status UART_TransferReceiveDMA(UART_Type *base,
00107     uart_dma_handle_t *handle, struct uart_transfer *xfer);
00108
00115 void UART_TransferAbortSendDMA(UART_Type *base, uart_dma_handle_t *handle);
00116
00123 void UART_TransferAbortReceiveDMA(UART_Type *base, uart_dma_handle_t *handle);
00124
00131 static inline void UART_WaitWhileActive(UART_Type *base)
00132 {
00133     /* Ожидание завершения передачи */
00134     while (base->USR & UART_USR_BUSY_Msk);
00135 }
00136
00147 static inline void UART_DMADescriptorInitTX(UART_Type *base,
00148     dma_descriptor_t *desc, uint32_t count, uint32_t data_size,
00149     uint32_t data_width, void *src_addr)
00150 {
00151     dma_multiblock_config_t config = {
00152         .count = count,
00153         .data_size = data_size,
00154         .transfer_type = DMA_MemoryToPeripheral_DMA,
00155         .scatter_en = false,
00156         .gather_en = false,
00157         .src_burst_size = DMA_BurstSize32,
00158         .dst_burst_size = DMA_BurstSize1,
00159         .src_incr = DMA_Incr,
00160         .dst_incr = DMA_NoChange,
00161         .src_data_width = data_width,
00162         .dst_data_width = data_width,

```



```

00163     .src_addr      = src_addr,
00164     .dst_addr      = (void *) &base->THR,
00165     .int_en        = true
00166 };
00167
00168 DMA_InitMultiblockDescriptor(desc, &config);
00169 }
00170
00181 static inline void UART_DMADescriptorInitRX(UART_Type *base,
00182 dma_descriptor_t *desc, uint32_t count, uint32_t data_size,
00183 uint32_t data_width, void *dst_addr)
00184 {
00185     dma_multiblock_config_t config = {
00186         .count      = count,
00187         .data_size   = data_size,
00188         .transfer_type = DMA_PeripheralToMemory_DMA,
00189         .scatter_en   = false,
00190         .gather_en    = false,
00191         .src_burst_size = DMA_BurstSize1,
00192         .dst_burst_size = DMA_BurstSize32,
00193         .src_incr      = DMA_NoChange,
00194         .dst_incr      = DMA_Incr,
00195         .src_data_width = data_width,
00196         .dst_data_width = data_width,
00197         .src_addr      = (void *) &base->RBR,
00198         .dst_addr      = dst_addr,
00199         .int_en        = true
00200     };
00201
00202 DMA_InitMultiblockDescriptor(desc, &config);
00203 }
00204
00205 #if defined(__cplusplus)
00206 }
00207 #endif
00208
00209 #endif /* HAL_UART_DMA_H */
00210

```

## 6.58 Файл devices/eliot1/drivers/hal\_vtu.h

Интерфейс драйвера универсального блока таймеров

```
#include "hal_common.h"
```

Структуры данных

- struct [vtu\\_config](#)  
Структура для конфигурации VTU.

Определения типов

- typedef void(\* vtu\_callback) (VTU\_Type \*base, uint32\_t timer, enum [vtu\\_interrupt\\_control](#) value)  
Функция обратного вызова

Перечисления

- enum [vtu\\_status](#)  
Статусы драйвера универсального блока таймеров
- enum [vtu\\_mode](#)  
Режимы работы тамеров универсального блока таймеров
- enum [vtu\\_capture\\_edge\\_control](#)

- Управление фронтами при режиме захвата
- enum [vtu\\_pwm\\_polarity](#)
  - Управление полярностью ШИМ
- enum [vtu\\_interrupt\\_control](#)
  - Управление прерываниями
- enum [timer\\_num\\_mode](#)
  - Номер таймера и его режим работы для прерываний

## Функции

- enum [vtu\\_status](#) [VTU\\_GetLastAPIStatus](#) (void)
  - Получение статуса выполнения функции, тип результата которой отличен от enum [vtu\\_status](#).

## Инициализация и деинициализации таймера

- enum [vtu\\_status](#) [VTU\\_GetDefaultConfig](#) (struct [vtu\\_config](#) \*config)
  - Создание конфигурации по умолчанию
- enum [vtu\\_status](#) [VTU\\_Init](#) (VTU\_Type \*base, uint32\_t timer, struct [vtu\\_config](#) \*config)
  - Инициализация таймера
- enum [vtu\\_status](#) [VTU\\_Deinit](#) (VTU\_Type \*base, uint32\_t timer)
  - Деинициализация таймера

## Функции управления VTU

- enum [vtu\\_status](#) [VTU\\_EnableTimer](#) (VTU\_Type \*base, uint32\_t timer, bool enable)
  - Разрешение работы таймера
- enum [vtu\\_status](#) [VTU\\_SetCounter](#) (VTU\_Type \*base, uint32\_t timer, uint16\_t value, bool extended)
  - Установка значения счетчика
- uint16\_t [VTU\\_GetCounter](#) (VTU\_Type \*base, uint32\_t timer, bool extended)
  - Получение значения счетчика
- enum [vtu\\_status](#) [VTU\\_SetPrescaler](#) (VTU\_Type \*base, uint32\_t timer, uint8\_t value)
  - Установка значения предделителя
- uint8\_t [VTU\\_GetPrescaler](#) (VTU\_Type \*base, uint32\_t timer)
  - Получение значения предделителя
- enum [vtu\\_status](#) [VTU\\_SetPeriodCapture](#) (VTU\_Type \*base, uint32\_t timer, uint16\_t value, bool extended)
  - Установка значения периода генерации шим без учета предделителя
- uint16\_t [VTU\\_GetPeriodCapture](#) (VTU\_Type \*base, uint32\_t timer, bool extended)
  - Получение значения периода генерации шим без учета предделителя
- enum [vtu\\_status](#) [VTU\\_SetDutyCycleCapture](#) (VTU\_Type \*base, uint32\_t timer, uint16\_t value, bool extended)
  - Установка периода импульса шим без учета предделителя
- uint16\_t [VTU\\_GetDutyCycleCapture](#) (VTU\_Type \*base, uint32\_t timer, bool extended)
  - Получение периода импульса шим без учета предделителя
- void [VTU\\_SetCallback](#) (uint32\_t timer, [vtu\\_callback](#) callback)
  - Установка функции обратного вызова
- enum [vtu\\_status](#) [VTU\\_EnableTimerIRQ](#) (VTU\_Type \*base, uint32\_t timer, enum [vtu\\_interrupt\\_control](#) value, bool enable, enum [vtu\\_mode](#) mode)
  - Разрешение работы прерывания
- enum [vtu\\_interrupt\\_control](#) [VTU\\_GetTimerIRQ](#) (VTU\_Type \*base, uint32\_t timer, enum [vtu\\_mode](#) mode)
  - Получение прерываний
- enum [vtu\\_status](#) [VTU\\_ClearTimerIRQ](#) (VTU\_Type \*base, uint32\_t timer, enum [vtu\\_interrupt\\_control](#) values, enum [vtu\\_mode](#) mode)

- Очистка прерываний
- enum vtu\_status VTU\_SetPWMPolarity (VTU\_Type \*base, uint32\_t timer, enum vtu\_pwm\_polarity value1, enum vtu\_pwm\_polarity value2, bool use\_value2)
- Установка полярности ШИМ
- enum vtu\_status VTU\_GetPWMPolarity (VTU\_Type \*base, uint32\_t timer, enum vtu\_pwm\_polarity \*value1, enum vtu\_pwm\_polarity \*value2, bool use\_value2)
- Получение полярности ШИМ
- enum vtu\_status VTU\_SetCaptureEdgeCtrl (VTU\_Type \*base, uint32\_t timer, enum vtu\_capture\_edge\_control value1, enum vtu\_capture\_edge\_control value2, bool use\_value2)
- Установка типа захвата
- enum vtu\_status VTU\_GetCaptureEdgeCtrl (VTU\_Type \*base, uint32\_t timer, enum vtu\_capture\_edge\_control \*value1, enum vtu\_capture\_edge\_control \*value2, bool use\_value2)
- Получение типа захвата

### 6.58.1 Подробное описание

Интерфейс драйвера универсального блока таймеров

## 6.59 hal\_vtu.h

[См. документацию.](#)

```

00001
00020 #ifndef HAL_VTU_H
00021 #define HAL_VTU_H
00022
00023 #include "hal_common.h"
00024
00028 enum vtu_status {
00029     VTU_Status_Ok = 0,
00030     VTU_Status_InvalidArgument = 1,
00031     VTU_Status_TimerBusy = 2,
00032     VTU_Status_BadConfigure = 3,
00033     VTU_Status_DriverError = 4,
00034     VTU_Status_DualTimerNotCanRun = 5,
00036     VTU_Status_TimerNotInit = 6,
00037 };
00038
00042 enum vtu_mode {
00043     VTU_LowPower = 0,
00044     VTU_PWMDual8Bit = 1,
00045     VTU_PWM16Bit = 2,
00046     VTU_Capture = 3,
00047 };
00048
00052 enum vtu_capture_edge_control {
00053     VTU_CaptureRisingEdgeResetNo = 0,
00054     VTU_CaptureFallingEdgeResetNo = 1,
00055     VTU_CaptureRisingEdgeResetYes = 2,
00056     VTU_CaptureFallingEdgeResetYes = 3,
00057     VTU_CaptureBothEdgeResetNo = 4,
00058     VTU_CaptureBothEdgeResetRisingEdge = 5,
00059     VTU_CaptureBothEdgeResetFallingEdge = 6,
00060     VTU_CaptureBothEdgeResetBothEdge = 7,
00061 };
00062
00066 enum vtu_pwm_polarity {
00067     VTU_One = 0,
00068     VTU_Zero = 1,
00069 };
00070
00074 enum vtu_interrupt_control {
00075     VTU_NoInterrupt = 0,
00076     VTU_LowByteDutyCycleMatch = 1,
00077     VTU_LowBytePeriodMatch = 2,
00078     VTU_HighByteDutyCycleMatch = 4,
00079     VTU_HighBytePeriodMatch = 8,
00080     VTU_DutyCycleMatch = 1,
00081     VTU_PeriodMatch = 2,
00082     VTU_CaptureToPERCAPx = 1,

```

```

00083     VTU_CaptureToDTYCAPx      = 2,
00084     VTU_CounterOverflow        = 4,
00085 };
00086
00087 typedef void (*vtu_callback)(VTU_Type *base, uint32_t timer,
00088     enum vtu_interrupt_control value);
00089
00090 struct vtu_config {
00091     enum vtu_mode                mode;
00092     enum vtu_capture_edge_control capture_edge_control1;
00093     enum vtu_capture_edge_control capture_edge_control2;
00094     enum vtu_pwm_polarity        pwm_polarity;
00095     enum vtu_pwm_polarity        pwm_polarity2;
00096     enum vtu_interrupt_control   interrupt_control;
00097     uint8_t                      prescaler;
00098     uint16_t                     counter;
00099     uint16_t                     period;
00100     uint16_t                     duty_cycle_capture;
00101 };
00102
00103 enum timer_num_mode {
00104     VTU_Timer0Mode8bit = 5,
00105     VTU_Timer0Mode16bit = 9,
00106     VTU_Timer1Mode8bit = 96,
00107     VTU_Timer2Mode8bit = 1280,
00108     VTU_Timer2Mode16bit = 2304,
00109     VTU_Timer3Mode8bit = 24576,
00110 };
00111
00112 enum vtu_status VTU_GetDefaultConfig(struct vtu_config *config);
00113
00114 enum vtu_status VTU_Init(VTU_Type *base, uint32_t timer, struct vtu_config *config);
00115
00116 enum vtu_status VTU_Deinit(VTU_Type *base, uint32_t timer);
00117
00118 enum vtu_status VTU_EnableTimer(VTU_Type *base, uint32_t timer, bool enable);
00119
00120 enum vtu_status VTU_SetCounter(VTU_Type *base, uint32_t timer,
00121     uint16_t value, bool extended);
00122
00123 uint16_t VTU_GetCounter(VTU_Type *base, uint32_t timer, bool extended);
00124
00125 enum vtu_status VTU_SetPrescaler(VTU_Type *base, uint32_t timer,
00126     uint8_t value);
00127
00128 uint8_t VTU_GetPrescaler(VTU_Type *base, uint32_t timer);
00129
00130 enum vtu_status VTU_SetPeriodCapture(VTU_Type *base, uint32_t timer,
00131     uint16_t value, bool extended);
00132
00133 uint16_t VTU_GetPeriodCapture(VTU_Type *base, uint32_t timer,
00134     bool extended);
00135
00136 enum vtu_status VTU_SetDutyCycleCapture(VTU_Type *base, uint32_t timer,
00137     uint16_t value, bool extended);
00138
00139 uint16_t VTU_GetDutyCycleCapture(VTU_Type *base, uint32_t timer,
00140     bool extended);
00141
00142 void VTU_SetCallback(uint32_t timer, vtu_callback callback);
00143
00144 enum vtu_status VTU_EnableTimerIRQ(VTU_Type *base, uint32_t timer,
00145     enum vtu_interrupt_control value, bool enable, enum vtu_mode mode);
00146
00147 enum vtu_interrupt_control VTU_GetTimerIRQ(VTU_Type *base, uint32_t timer,
00148     enum vtu_mode mode);
00149
00150 enum vtu_status VTU_ClearTimerIRQ(VTU_Type *base, uint32_t timer,
00151     enum vtu_interrupt_control values, enum vtu_mode mode);
00152
00153 enum vtu_status VTU_SetPWMPolarity(VTU_Type *base, uint32_t timer,
00154     enum vtu_pwm_polarity value1, enum vtu_pwm_polarity value2,
00155     bool use_value2);
00156
00157 enum vtu_status VTU_GetPWMPolarity(VTU_Type *base, uint32_t timer,
00158     enum vtu_pwm_polarity *value1, enum vtu_pwm_polarity *value2,
00159     bool use_value2);
00160
00161 enum vtu_status VTU_SetCaptureEdgeCtrl(VTU_Type *base, uint32_t timer,
00162     enum vtu_capture_edge_control value1,
00163     enum vtu_capture_edge_control value2,
00164     bool use_value2);
00165
00166 enum vtu_status VTU_GetCaptureEdgeCtrl(VTU_Type *base, uint32_t timer,
00167     enum vtu_capture_edge_control *value1,
00168     enum vtu_capture_edge_control *value2,

```

```

00393     bool use_value2);
00394
00405 enum vtu_status VTU_GetLastAPIStatus(void);
00406
00411 #endif /* HAL_VTU_H */
00412

```

## 6.60 Файл devices/eliot1/drivers/hal\_wdt.h

Интерфейс драйвера сторожевого таймера

```
#include "hal_common.h"
```

Структуры данных

- struct `wdt_config`  
Структура инициализации сторожевого таймера

Макросы

- #define `WDT_NUMBER_OF_TIMERS` 3

Перечисления

- enum `wdt_status`  
Статусы драйвера сторожевого таймера
- enum `wdt_resen_type`  
Управление сбросом при таймауте сторожевого таймера
- enum `wdt_inten_type`  
Управление прерыванием предупреждения от сторожевого таймера\ и разрешением работы таймера

Функции

Инициализация и деинициализации таймера

- enum `wdt_status` `WDT_GetDefaultConfig` (struct `wdt_config` \*config)  
Создание конфигурации по умолчанию
- enum `wdt_status` `WDT_Init` (WDT\_Type \*base, const struct `wdt_config` \*config)  
Инициализация таймера
- enum `wdt_status` `WDT_Deinit` (WDT\_Type \*base)  
Деинициализация таймера

Функции управления WDT

- enum `wdt_status` `WDT_Enable` (WDT\_Type \*base)  
Разрешение работы таймера
- enum `wdt_status` `WDT_Disable` (WDT\_Type \*base)  
Запрещение работы таймера
- uint32\_t `WDT_GetStatusFlagsRaw` (WDT\_Type \*base)  
Получение немаскированных статусов таймера

- `uint32_t WDT_GetStatusFlagsMsk (WDT_Type *base)`  
Получение маскированных статусов таймера
- `enum wdt_status WDT_ClearStatusFlags (WDT_Type *base, uint32_t mask)`  
Очищение статусов таймера
- `enum wdt_status WDT_SetWarningValue (WDT_Type *base, uint32_t warning_value)`  
Установка времени срабатывания предупреждения
- `enum wdt_status WDT_SetTimeoutValue (WDT_Type *base, uint32_t timeout_count)`  
Установка времени таймаута таймера
- `uint32_t WDT_GetWarningValue (WDT_Type *base)`  
Получение значения счетчика
- `enum wdt_status WDT_Refresh (WDT_Type *base)`  
Обновление времени сторожевого таймера
- `enum wdt_status WDT_GetLastAPIStatus ()`  
Получение статуса выполнения функции, тип результата которой отличен от `enum wdt_status`.

### 6.60.1 Подробное описание

Интерфейс драйвера сторожевого таймера

## 6.61 hal\_wdt.h

[См. документацию.](#)

```

00001
00020 #ifndef HAL_WDT_H
00021 #define HAL_WDT_H
00022
00023 #include "hal_common.h"
00024
00025 #if defined(__cplusplus)
00026 extern "C" {
00027 #endif /* __cplusplus */
00028
00029 #define WDT_NUMBER_OF_TIMERS 3
00034 enum wdt_status {
00035     WDT_Status_Ok = 0,
00036     WDT_Status_InvalidArgument = 1,
00037     WDT_Status_TimerBusy = 2,
00038     WDT_Status_BadConfigure = 3,
00039 };
00040
00044 enum wdt_resen_type {
00045     WDT_ResenDisable = 0,
00046     WDT_ResenEnable = 1,
00047 };
00048
00053 enum wdt_inten_type {
00054     WDT_IntenDisable = 0,
00055     WDT_IntenEnable = 1,
00056 };
00057
00061 struct wdt_config {
00062     uint32_t load;
00063     enum wdt_resen_type resen;
00064     enum wdt_inten_type inten;
00065 };
00066
00090 enum wdt_status WDT_GetDefaultConfig(struct wdt_config *config);
00091
00103 enum wdt_status WDT_Init(WDT_Type *base,
00104     const struct wdt_config *config);
00105
00116 enum wdt_status WDT_Deinit(WDT_Type *base);
00117
00135 enum wdt_status WDT_Enable(WDT_Type *base);
00136
00145 enum wdt_status WDT_Disable(WDT_Type *base);
00146
00152 uint32_t WDT_GetStatusFlagsRaw(WDT_Type *base);
00153
00159 uint32_t WDT_GetStatusFlagsMsk(WDT_Type *base);

```

```
00160
00170 enum wdt_status WDT_ClearStatusFlags(WDT_Type *base, uint32_t mask);
00171
00187 enum wdt_status WDT_SetWarningValue(WDT_Type *base, uint32_t warning_value);
00188
00204 enum wdt_status WDT_SetTimeoutValue(WDT_Type *base, uint32_t timeout_count);
00205
00213 uint32_t WDT_GetWarningValue(WDT_Type *base);
00214
00223 enum wdt_status WDT_Refresh(WDT_Type *base);
00224
00234 enum wdt_status WDT_GetLastAPIStatus();
00235
00244 #if defined(__cplusplus)
00245 }
00246 #endif /* __cplusplus */
00247
00248 #endif /* HAL_WDT_H */
00249
```





# Предметный указатель

- \_\_pad0\_\_
  - \_can\_frame\_filter, [359](#)
  - \_can\_rx\_buffer\_frame, [364](#)
  - \_can\_tx\_buffer\_frame, [371](#)
  - rcw\_config\_reg, [448](#)
  - rcw\_trim\_reg, [451](#)
  - rcw\_trimload\_reg, [453](#)
  - rcw\_wake\_config\_reg, [455](#)
- \_\_pad1\_\_
  - \_can\_frame\_filter, [359](#)
  - rcw\_config\_reg, [448](#)
  - rcw\_trim\_reg, [451](#)
  - rcw\_wake\_config\_reg, [455](#)
- \_\_pad2\_\_
  - rcw\_config\_reg, [448](#)
- \_can\_bytes\_in\_datafield
  - Драйвер модуля CAN, [14](#)
- \_can\_config, [357](#)
  - can\_fd\_mode, [357](#)
  - enable\_listen\_only, [357](#)
  - enable\_loopback\_ext, [358](#)
  - enable\_loopback\_int, [358](#)
  - filter\_config, [358](#)
  - ptb\_config, [358](#)
  - rxb\_config, [358](#)
  - stb\_config, [358](#)
  - timing\_config, [358](#)
- \_can\_flag
  - Драйвер модуля CAN, [14](#)
- \_can\_frame\_filter, [359](#)
  - \_\_pad0\_\_, [359](#)
  - \_\_pad1\_\_, [359](#)
  - accepted\_id, [359](#)
  - enable\_id\_check, [359](#)
  - id, [360](#)
  - mask, [360](#)
- \_can\_frame\_filter\_config, [360](#)
  - filter, [360](#)
  - nb\_filters\_used, [360](#)
- \_can\_handle, [361](#)
  - callback, [361](#)
  - rx\_frames, [361](#)
  - rx\_nb\_frames\_all, [361](#)
  - rx\_nb\_frames\_rest, [362](#)
  - tx\_frames\_prim, [362](#)
  - tx\_frames\_sec, [362](#)
  - tx\_nb\_frames\_all\_prim, [362](#)
  - tx\_nb\_frames\_all\_sec, [362](#)
  - tx\_nb\_frames\_rest\_prim, [362](#)
  - tx\_nb\_frames\_rest\_sec, [362](#)
  - user\_data, [362](#)
- \_can\_kind\_of\_error
  - Драйвер модуля CAN, [15](#)
- \_can\_ptb\_config, [363](#)
  - tx\_single\_shot, [363](#)
- \_can\_rx\_buffer\_frame, [363](#)
  - \_\_pad0\_\_, [364](#)
  - brs, [364](#)
  - cycle\_time, [364](#)
  - data, [364](#)
  - dlc, [365](#)
  - esi, [365](#)
  - fdf, [365](#)
  - id, [365](#)
  - ide, [365](#)
  - koer, [365](#)
  - rtr, [365](#)
  - rts, [365](#)
  - tx, [366](#)
- \_can\_rx\_transfer, [366](#)
  - frames, [366](#)
  - nb\_frames, [366](#)
- \_can\_rxb\_config, [367](#)
  - almost\_full\_level, [367](#)
  - prohibit\_overflow, [367](#)
  - self\_acknowledge, [367](#)
- \_can\_status
  - Драйвер модуля CAN, [16](#)
- \_can\_stb\_config, [367](#)
  - tx\_discipline, [368](#)
  - tx\_single\_shot, [368](#)
- \_can\_stb\_discipline
  - Драйвер модуля CAN, [16](#)
- \_can\_timing\_config, [368](#)
  - data\_prescaler, [369](#)
  - data\_seg1, [369](#)
  - data\_seg2, [369](#)
  - data\_sjw, [369](#)
  - delay\_compensation\_enable, [369](#)
  - prescaler, [369](#)
  - secondary\_sample\_point\_offset, [369](#)
  - seg1, [369](#)
  - seg2, [370](#)
  - sjw, [370](#)
- \_can\_tx\_buffer\_frame, [370](#)
  - \_\_pad0\_\_, [371](#)
  - brs, [371](#)
  - data, [371](#)

- dlc, 371
- fdf, 371
- id, 371
- ide, 371
- rtr, 371
- ttsen, 372
- \_can\_tx\_transfer, 372
  - frames, 372
  - nb\_frames, 372
- \_canfd\_mode
  - Драйвер модуля CAN, 16
- \_dma\_channel\_config, 373
  - ctlx\_cfg, 373
  - dst\_addr, 373
  - is\_dst\_periph, 373
  - is\_src\_periph, 373
  - next\_desc, 373
  - src\_addr, 374
- \_dma\_channel\_reg, 374
  - CFG\_HI, 375
  - CFG\_LO, 375
  - CTL\_HI, 375
  - CTL\_LO, 375
  - DAR\_HI, 375
  - DAR\_LO, 375
  - DSR\_HI, 375
  - DSR\_LO, 375
  - DSTAT\_HI, 376
  - DSTAT\_LO, 376
  - DSTATAR\_HI, 376
  - DSTATAR\_LO, 376
  - LLP\_HI, 376
  - LLP\_LO, 376
  - SAR\_HI, 376
  - SAR\_LO, 376
  - SGR\_HI, 377
  - SGR\_LO, 377
  - SSTAT\_HI, 377
  - SSTAT\_LO, 377
  - SSTATAR\_HI, 377
  - SSTATAR\_LO, 377
- \_dma\_descriptor, 378
  - CTL\_HI, 378
  - CTL\_LO, 378
  - DAR, 378
  - DSTAT, 378
  - LLP, 378
  - SAR, 379
  - SSTAT, 379
- \_dma\_handle, 379
  - base, 379
  - callback, 379
  - channel, 380
  - user\_data, 380
- \_dma\_int
  - Драйвер модуля DMA, 89
- \_dma\_multiblock\_config, 380
  - count, 381
  - data\_size, 381
  - dst\_addr, 381
  - dst\_burst\_size, 381
  - dst\_data\_width, 381
  - dst\_incr, 381
  - gather\_en, 381
  - int\_en, 381
  - scatter\_en, 382
  - src\_addr, 382
  - src\_burst\_size, 382
  - src\_data\_width, 382
  - src\_incr, 382
  - transfer\_type, 382
- \_dma\_priority
  - Драйвер модуля DMA, 89
- \_dma\_status
  - Драйвер модуля DMA, 90
- \_dma\_transfer\_type
  - Драйвер модуля DMA, 90
- \_i2c\_master\_dma\_handle, 383
  - buf, 383
  - completion\_callback, 383
  - dummy\_data, 383
  - remaining\_bytes\_DMA, 383
  - rx\_desc, 383
  - rx\_dma, 384
  - state, 384
  - tx\_desc, 384
  - tx\_dma, 384
  - user\_data, 384
- \_i2c\_master\_handle, 384
  - buf, 385
  - check\_addr\_nack, 385
  - completion\_callback, 385
  - remaining\_bytes, 385
  - remaining\_subaddr, 385
  - state, 385
  - subaddr\_buf, 386
  - transfer, 386
  - transfer\_count, 386
  - user\_data, 386
- \_i2c\_slave\_handle, 386
  - callback, 387
  - irq\_num, 387
  - slave\_fsm, 387
  - transfer, 387
  - user\_data, 387
- \_i2s\_config, 387
  - interrupt\_level, 388
  - resolution, 388
  - sample\_rate, 388
  - sclk\_gating, 388
  - sclk\_per\_sample, 388
- \_i2s\_flag
  - Драйвер модуля I2S, 148
- \_i2s\_handle, 389
  - callback, 389
  - left\_samples, 389

- nb\_samples, 389
- right\_samples, 389
- user\_data, 389
- \_i2s\_interrupt\_level
  - Драйвер модуля I2S, 148
- \_i2s\_resolution
  - Драйвер модуля I2S, 148
- \_i2s\_sclk\_gating
  - Драйвер модуля I2S, 149
- \_i2s\_sclk\_per\_sample
  - Драйвер модуля I2S, 149
- \_i2s\_status
  - Драйвер модуля I2S, 149
- \_i2s\_transfer, 390
  - left\_samples, 390
  - nb\_samples, 390
  - right\_samples, 390
- \_jtm\_handle, 391
  - callback, 391
  - parameter, 391
  - user\_data, 391
- \_nor\_command\_set, 391
  - erase\_chip\_cmd, 392
  - erase\_sector\_cmd, 392
  - page\_write\_memory\_cmd, 392
  - read\_memory\_command, 392
  - read\_status\_cmd, 392
  - write\_disable\_cmd, 392
  - write\_enable\_cmd, 393
  - write\_status\_cmd, 393
- \_nor\_config, 393
  - driver\_base\_addr, 393
  - mem\_control\_config, 393
  - quad\_control\_config, 394
- \_nor\_handle, 394
  - device\_specific, 394
  - driver\_base\_addr, 394
  - max\_chip\_erase\_time, 395
  - max\_page\_program\_time, 395
  - max\_sector\_erase\_time, 395
  - memory\_size\_bytes, 395
  - page\_size\_bytes, 395
  - qspi\_config, 395
  - sector\_size\_bytes, 395
  - xip\_config, 395
- \_pwm\_chopper, 396
- \_pwm\_dz\_cfg, 396
- \_pwm\_handle, 397
  - channel, 397
  - duty\_cycle, 397
  - int\_en, 397
  - int\_freq, 398
  - int\_source, 398
  - out, 398
  - period\_us, 398
  - pwm\_callback, 398
  - ref\_clk, 398
- \_pwm\_trip\_unit\_cfg, 399
- \_qspi\_command\_format
  - Драйвер модуля QSPI, 212
- \_qspi\_command\_type
  - Драйвер модуля QSPI, 212
- \_qspi\_config, 399
  - bit\_size, 399
  - cont\_trans\_en, 399
  - cont\_transfer\_ext, 400
  - cpha, 400
  - cpol, 400
  - delay\_en, 400
  - dma\_en, 400
  - inhibit\_din, 400
  - inhibit\_dout, 400
  - mode, 400
  - msb, 401
  - slave\_pol, 401
  - slave\_select, 401
  - spi\_mode, 401
- \_qspi\_nor\_config, 401
  - is\_quad\_need\_enable, 402
  - quad\_enable\_bit\_shift, 402
  - quad\_enable\_command, 402
  - quad\_read\_command, 402
  - write\_two\_status\_bytes, 402
- \_qspi\_nor\_handle, 402
  - command\_set, 403
  - command\_type, 403
  - intermediate\_len, 403
  - read\_cmd\_format, 403
- \_qspi\_nor\_init\_config, 403
  - cmd\_format, 404
  - quad\_mode\_setting, 404
- \_qspi\_qmode
  - Драйвер модуля QSPI, 213
- \_qspi\_xip\_config, 404
  - addr4, 405
  - cmd, 405
  - cpha, 405
  - cpol, 405
  - dummy\_cycles, 405
  - hp\_end\_dummy, 405
  - hp\_mode, 405
  - hpen, 405
  - le32, 406
- \_rtc\_datetime, 406
  - day, 406
  - hour, 406
  - minute, 406
  - month, 407
  - second, 407
  - year, 407
- \_serial\_nor\_command
  - Драйвер модуля QSPI, 213
- \_spi\_dma\_handle, 407
  - bytes\_per\_frame, 408
  - callback, 408
  - dummy\_data, 408

- rx\_desc, 408
- rx\_handle, 408
- rx\_in\_progress, 408
- state, 408
- transfer\_size, 408
- tx\_desc, 409
- tx\_handle, 409
- tx\_in\_progress, 409
- user\_data, 409
- \_ttcan\_config, 409
  - prescaler, 410
  - reference\_id, 410
  - reference\_ide, 410
  - transmit\_enable\_window, 410
  - transmit\_trigger\_pointer, 410
  - trigger\_time0, 410
  - trigger\_time1, 410
  - trigger\_type, 410
  - watch\_trigger\_time0, 411
  - watch\_trigger\_time1, 411
- \_ttcan\_timer\_prescaler
  - Драйвер модуля CAN, 17
- \_ttcan\_trigger\_type
  - Драйвер модуля CAN, 17
- \_uart\_dma\_handle, 411
  - base, 412
  - callback, 412
  - rx\_data\_size\_all, 412
  - rx\_dma\_handle, 412
  - rx\_state, 412
  - tx\_data\_size\_all, 412
  - tx\_dma\_handle, 412
  - tx\_state, 412
  - user\_data, 413
- accepted\_ide
  - \_can\_frame\_filter, 359
- ack\_gen\_call
  - i2c\_slave\_config\_t, 423
- addr4
  - \_qspi\_xip\_config, 405
- addr\_size
  - i2c\_master\_transfer\_t, 421
- address
  - i2c\_slave\_config\_t, 423
- alarm
  - rtc\_alarm\_reg, 443
  - rtc\_union\_reg, 454
- alarm\_en
  - rtc\_config, 444
  - rtc\_config\_reg, 448
- alarm\_time
  - rtc\_config, 444
- almost\_full\_level
  - \_can\_rxb\_config, 367
- apc\_eco\_threshold
  - power\_mode\_config, 429
- apc\_force\_trim
  - power\_trim\_config, 432
- apc\_low\_clk\_enable
  - power\_mode\_config, 429
- apc\_vref\_it
  - power\_trim\_config, 432
- apc\_vref\_tt
  - power\_trim\_config, 432
- apc\_vref\_vt
  - power\_trim\_config, 432
- base
  - \_dma\_handle, 379
  - \_uart\_dma\_handle, 412
- baud\_rate\_bps
  - spi\_config\_t, 465
- baudrate\_bps
  - i2c\_master\_config\_t, 420
  - uart\_config, 473
- BE\_TO\_LE16
  - Общие определения для библиотеки HAL, 80
- BE\_TO\_LE24
  - Общие определения для библиотеки HAL, 80
- BE\_TO\_LE32
  - Общие определения для библиотеки HAL, 80
- bg\_load
  - dualtimer\_hardware\_config, 418
- bit\_count\_per\_char
  - uart\_config, 473
- bit\_size
  - \_qspi\_config, 399
- block\_size
  - dma\_channel\_ctl\_cfg, 416
- break\_line
  - uart\_config, 474
- brs
  - \_can\_rx\_buffer\_frame, 364
  - \_can\_tx\_buffer\_frame, 371
- buf
  - i2c\_master\_dma\_handle, 383
  - i2c\_master\_handle, 385
- busy\_ready\_check
  - spi\_microwire\_cfg\_t, 469
- bytes\_per\_frame
  - \_spi\_dma\_handle, 408
- callback
  - \_can\_handle, 361
  - \_dma\_handle, 379
  - i2c\_slave\_handle, 387
  - i2s\_handle, 389
  - jtm\_handle, 391
  - spi\_dma\_handle, 408
  - \_uart\_dma\_handle, 412
  - power\_handle, 428
  - spi\_handle, 467
  - uart\_handle, 476
- CAN\_0ByteDatafield
  - Драйвер модуля CAN, 14
- CAN\_12ByteDatafield
  - Драйвер модуля CAN, 14

- CAN\_16ByteDatafield
  - Драйвер модуля CAN, [14](#)
- CAN\_1ByteDatafield
  - Драйвер модуля CAN, [14](#)
- CAN\_20ByteDatafield
  - Драйвер модуля CAN, [14](#)
- CAN\_24ByteDatafield
  - Драйвер модуля CAN, [14](#)
- CAN\_2ByteDatafield
  - Драйвер модуля CAN, [14](#)
- CAN\_32ByteDatafield
  - Драйвер модуля CAN, [14](#)
- CAN\_3ByteDatafield
  - Драйвер модуля CAN, [14](#)
- CAN\_48ByteDatafield
  - Драйвер модуля CAN, [14](#)
- CAN\_4ByteDatafield
  - Драйвер модуля CAN, [14](#)
- CAN\_5ByteDatafield
  - Драйвер модуля CAN, [14](#)
- CAN\_64ByteDatafield
  - Драйвер модуля CAN, [14](#)
- CAN\_6ByteDatafield
  - Драйвер модуля CAN, [14](#)
- CAN\_7ByteDatafield
  - Драйвер модуля CAN, [14](#)
- CAN\_8ByteDatafield
  - Драйвер модуля CAN, [14](#)
- CAN\_AbortPrimaryTxBuffer
  - Драйвер модуля CAN, [17](#)
- CAN\_AbortSecondaryTxBuffer
  - Драйвер модуля CAN, [18](#)
- CAN\_BoschFd
  - Драйвер модуля CAN, [17](#)
- CAN\_CalculateImprovedTimingValues
  - Драйвер модуля CAN, [18](#)
- CAN\_ClearStatusFlag
  - Драйвер модуля CAN, [18](#)
- CAN\_ClearStatusFlagMask
  - Драйвер модуля CAN, [19](#)
- CAN\_Deinit
  - Драйвер модуля CAN, [19](#)
- CAN\_DisableInterrupt
  - Драйвер модуля CAN, [19](#)
- CAN\_DisableInterruptMask
  - Драйвер модуля CAN, [20](#)
- CAN\_EnableInterrupt
  - Драйвер модуля CAN, [20](#)
- CAN\_EnableInterruptMask
  - Драйвер модуля CAN, [20](#)
- CAN\_EnterNormalMode
  - Драйвер модуля CAN, [21](#)
- CAN\_EnterStandbyMode
  - Драйвер модуля CAN, [21](#)
- CAN\_ErrorAckError
  - Драйвер модуля CAN, [15](#)
- CAN\_ErrorBitError
  - Драйвер модуля CAN, [15](#)
- CAN\_ErrorCrcError
  - Драйвер модуля CAN, [16](#)
- CAN\_ErrorFormError
  - Драйвер модуля CAN, [15](#)
- CAN\_ErrorNone
  - Драйвер модуля CAN, [15](#)
- CAN\_ErrorOtherError
  - Драйвер модуля CAN, [16](#)
- CAN\_ErrorReserved
  - Драйвер модуля CAN, [16](#)
- CAN\_ErrorStuffError
  - Драйвер модуля CAN, [15](#)
- can\_fd\_mode
  - \_can\_config, [357](#)
- CAN\_Fifo
  - Драйвер модуля CAN, [16](#)
- can\_flag\_t
  - Драйвер модуля CAN, [14](#)
- CAN\_FlagAbortState
  - Драйвер модуля CAN, [15](#)
- CAN\_FlagArbitrationLost
  - Драйвер модуля CAN, [15](#)
- CAN\_FlagBusError
  - Драйвер модуля CAN, [15](#)
- CAN\_FlagError
  - Драйвер модуля CAN, [15](#)
- CAN\_FlagErrorPassiveInterrupt
  - Драйвер модуля CAN, [15](#)
- CAN\_FlagErrorPassiveState
  - Драйвер модуля CAN, [15](#)
- CAN\_FlagErrorWarningState
  - Драйвер модуля CAN, [15](#)
- CAN\_FlagRBAAlmostFull
  - Драйвер модуля CAN, [15](#)
- CAN\_FlagRBFull
  - Драйвер модуля CAN, [15](#)
- CAN\_FlagRBOverrun
  - Драйвер модуля CAN, [15](#)
- CAN\_FlagReceive
  - Драйвер модуля CAN, [15](#)
- CAN\_FlagsNumber
  - Драйвер модуля CAN, [15](#)
- CAN\_FlagTimeTriggered
  - Драйвер модуля CAN, [15](#)
- CAN\_FlagTransmissionPrimary
  - Драйвер модуля CAN, [15](#)
- CAN\_FlagTransmissionSecondary
  - Драйвер модуля CAN, [15](#)
- CAN\_FlagTransmitBufferFull
  - Драйвер модуля CAN, [15](#)
- CAN\_FlagTriggerError
  - Драйвер модуля CAN, [15](#)
- CAN\_FlagWatchTriggerError
  - Драйвер модуля CAN, [15](#)
- CAN\_GetDefaultConfig
  - Драйвер модуля CAN, [21](#)
- CAN\_GetEnabledInterruptMask
  - Драйвер модуля CAN, [21](#)

- CAN\_GetStatusFlag  
Драйвер модуля CAN, 22
- CAN\_GetStatusFlagMask  
Драйвер модуля CAN, 22
- CAN\_Init  
Драйвер модуля CAN, 22
- CAN\_IsInterruptEnabled  
Драйвер модуля CAN, 23
- CAN\_IsoFd  
Драйвер модуля CAN, 17
- CAN\_IsPrimaryTransmitRequestPending  
Драйвер модуля CAN, 23
- CAN\_IsRxBufferAlmostFull  
Драйвер модуля CAN, 23
- CAN\_IsRxBufferEmpty  
Драйвер модуля CAN, 24
- CAN\_IsRxBufferFull  
Драйвер модуля CAN, 24
- CAN\_IsSecondaryTransmitRequestPending  
Драйвер модуля CAN, 24
- CAN\_IsSecondaryTxBufferEmpty  
Драйвер модуля CAN, 25
- CAN\_IsSecondaryTxBufferFull  
Драйвер модуля CAN, 25
- CAN\_IsSecondaryTxBufferMoreThanHalfFull  
Драйвер модуля CAN, 25
- can\_kind\_of\_error\_t  
Драйвер модуля CAN, 14
- CAN\_Priority  
Драйвер модуля CAN, 16
- CAN\_ReadRxBuffer  
Драйвер модуля CAN, 26
- CAN\_SetArbitrationTimingConfig  
Драйвер модуля CAN, 26
- CAN\_SetFilterConfig  
Драйвер модуля CAN, 26
- CAN\_SetPrimaryTxBufferConfig  
Драйвер модуля CAN, 27
- CAN\_SetRxBufferConfig  
Драйвер модуля CAN, 27
- CAN\_SetSecondaryTxBufferConfig  
Драйвер модуля CAN, 27
- CAN\_SetTTCANConfig  
Драйвер модуля CAN, 27
- CAN\_Status\_Fail  
Драйвер модуля CAN, 16
- CAN\_Status\_InvalidArgument  
Драйвер модуля CAN, 16
- CAN\_Status\_Ok  
Драйвер модуля CAN, 16
- CAN\_Status\_RxBusy  
Драйвер модуля CAN, 16
- CAN\_Status\_RxEmpty  
Драйвер модуля CAN, 16
- CAN\_Status\_RxIdle  
Драйвер модуля CAN, 16
- CAN\_Status\_TxBusy  
Драйвер модуля CAN, 16
- CAN\_Status\_TxIdle  
Драйвер модуля CAN, 16
- CAN\_TransferAbortReceive  
Драйвер модуля CAN, 28
- CAN\_TransferAbortSendPrimary  
Драйвер модуля CAN, 28
- CAN\_TransferAbortSendSecondary  
Драйвер модуля CAN, 28
- CAN\_TransferCreateHandle  
Драйвер модуля CAN, 28
- CAN\_TransferGetReceivedCount  
Драйвер модуля CAN, 29
- CAN\_TransferGetSentPrimaryCount  
Драйвер модуля CAN, 29
- CAN\_TransferGetSentSecondaryCount  
Драйвер модуля CAN, 30
- CAN\_TransferHandleIRQ  
Драйвер модуля CAN, 30
- CAN\_TransferReceiveFifoBlocking  
Драйвер модуля CAN, 30
- CAN\_TransferReceiveFifoNonBlocking  
Драйвер модуля CAN, 31
- CAN\_TransferSendPrimaryBlocking  
Драйвер модуля CAN, 31
- CAN\_TransferSendPrimaryNonBlocking  
Драйвер модуля CAN, 32
- CAN\_TransferSendSecondaryBlocking  
Драйвер модуля CAN, 32
- CAN\_TransferSendSecondaryNonBlocking  
Драйвер модуля CAN, 33
- CAN\_TriggerImmediate  
Драйвер модуля CAN, 17
- CAN\_TriggerSingleShotTransmit  
Драйвер модуля CAN, 17
- CAN\_TriggerTime  
Драйвер модуля CAN, 17
- CAN\_TriggerTransmitStart  
Драйвер модуля CAN, 17
- CAN\_TriggerTransmitStop  
Драйвер модуля CAN, 17
- CAN\_TTCANDiv1  
Драйвер модуля CAN, 17
- CAN\_TTCANDiv2  
Драйвер модуля CAN, 17
- CAN\_TTCANDiv4  
Драйвер модуля CAN, 17
- CAN\_TTCANDiv8  
Драйвер модуля CAN, 17
- CAN\_WritePrimaryTxBuffer  
Драйвер модуля CAN, 33
- CAN\_WriteSecondaryTxBuffer  
Драйвер модуля CAN, 34
- cap\_data  
spi\_motorola\_cfg\_t, 470
- capture\_edge\_control1  
vtu\_config, 480
- capture\_edge\_control2  
vtu\_config, 480



- CD
  - sdmmc\_cfg\_pin\_map, 459
- cfg
  - sdmmc\_card\_t, 457
- CFG\_HI
  - \_dma\_channel\_reg, 375
- CFG\_LO
  - \_dma\_channel\_reg, 375
- channel
  - \_dma\_handle, 380
  - \_pwm\_handle, 397
  - pwm\_channel\_config, 435
- check\_addr\_nack
  - \_i2c\_master\_handle, 385
- chopper\_duty
  - pwm\_channel\_config, 435
- chopper\_first\_width
  - pwm\_channel\_config, 435
- chopper\_freq
  - pwm\_channel\_config, 435
- chopper\_work
  - pwm\_channel\_config, 435
- cid
  - sdmmc\_card\_t, 457
- CK
  - sdmmc\_cfg\_pin\_map, 459
- clk\_div
  - rcw\_config\_reg, 448
- clk\_pol
  - spi\_motorola\_cfg\_t, 470
- CLKCTR\_BasePikClkForce
  - Драйвер модуля CLKCTR, 47
- clkctr\_clk\_force\_type
  - Драйвер модуля CLKCTR, 46
- CLKCTR\_ClkForceTypeDynamic
  - Драйвер модуля CLKCTR, 47
- CLKCTR\_ClkForceTypeForce
  - Драйвер модуля CLKCTR, 47
- CLKCTR\_CPUDBGPikClkForce
  - Драйвер модуля CLKCTR, 47
- CLKCTR\_CPUFClkForce
  - Драйвер модуля CLKCTR, 47
- CLKCTR\_CPUSysClkForce
  - Драйвер модуля CLKCTR, 47
- CLKCTR\_CryptoSysClkForce
  - Драйвер модуля CLKCTR, 47
- clkctr\_device\_clk\_force
  - Драйвер модуля CLKCTR, 47
- clkctr\_div, 413
  - clkctr\_fclk\_div, 413
  - clkctr\_gnssclk\_div, 413
  - clkctr\_i2selk\_div, 413
  - clkctr\_mco\_div, 414
  - clkctr\_qspiclk\_div, 414
  - clkctr\_sysclk\_div, 414
- clkctr\_extern\_freq
  - Драйвер модуля CLKCTR, 47
- CLKCTR\_ExternFreqHFI
  - Драйвер модуля CLKCTR, 47
- CLKCTR\_ExternFreqI2SExtClk
  - Драйвер модуля CLKCTR, 47
- CLKCTR\_ExternFreqLFI
  - Драйвер модуля CLKCTR, 47
- CLKCTR\_ExternFreqXTI
  - Драйвер модуля CLKCTR, 47
- CLKCTR\_ExternFreqXTI32
  - Драйвер модуля CLKCTR, 47
- clkctr\_fclk\_div
  - clkctr\_div, 413
- CLKCTR\_FCLK\_MAX
  - Драйвер модуля CLKCTR, 40
- CLKCTR\_FCLK\_MIN
  - Драйвер модуля CLKCTR, 40
- CLKCTR\_FREQ\_NOT\_SET
  - Драйвер модуля CLKCTR, 40
- CLKCTR\_GetAllDiv
  - Драйвер модуля CLKCTR, 50
- CLKCTR\_GetClk
  - Драйвер модуля CLKCTR, 51
- CLKCTR\_GetClkForce
  - Драйвер модуля CLKCTR, 51
- CLKCTR\_GetDivClk
  - Драйвер модуля CLKCTR, 51
- CLKCTR\_GetFClk
  - Драйвер модуля CLKCTR, 52
- CLKCTR\_GetFClkDiv
  - Драйвер модуля CLKCTR, 52
- CLKCTR\_GetFClkInt
  - Драйвер модуля CLKCTR, 52
- CLKCTR\_GetFrequency
  - Драйвер модуля CLKCTR, 53
- CLKCTR\_GetGNSSClk
  - Драйвер модуля CLKCTR, 53
- CLKCTR\_GetGNSSClkDiv
  - Драйвер модуля CLKCTR, 53
- CLKCTR\_GetHFI
  - Драйвер модуля CLKCTR, 54
- CLKCTR\_GetHFIClk
  - Драйвер модуля CLKCTR, 54
- CLKCTR\_GetHFIClkForMainClk
  - Драйвер модуля CLKCTR, 54
- CLKCTR\_GetHFITrim
  - Драйвер модуля CLKCTR, 55
- CLKCTR\_GetI2SClk
  - Драйвер модуля CLKCTR, 55
- CLKCTR\_GetI2SClkDiv
  - Драйвер модуля CLKCTR, 55
- CLKCTR\_GetI2SExtClk
  - Драйвер модуля CLKCTR, 56
- CLKCTR\_GetLFI
  - Драйвер модуля CLKCTR, 56
- CLKCTR\_GetLPClk
  - Драйвер модуля CLKCTR, 56
- CLKCTR\_GetMainClk
  - Драйвер модуля CLKCTR, 57
- CLKCTR\_GetMCOClk

Драйвер модуля CLKCTR, 57  
 CLKCTR\_GetMCOdiv  
 Драйвер модуля CLKCTR, 57  
 CLKCTR\_GetMCOEn  
 Драйвер модуля CLKCTR, 57  
 CLKCTR\_GetPLLClk  
 Драйвер модуля CLKCTR, 58  
 CLKCTR\_GetPLLConfig  
 Драйвер модуля CLKCTR, 58  
 CLKCTR\_GetPLLRef  
 Драйвер модуля CLKCTR, 58  
 CLKCTR\_GetQSPIClk  
 Драйвер модуля CLKCTR, 59  
 CLKCTR\_GetQSPIClkDiv  
 Драйвер модуля CLKCTR, 59  
 CLKCTR\_GetRTCClk  
 Драйвер модуля CLKCTR, 59  
 CLKCTR\_GetSwitchClk  
 Драйвер модуля CLKCTR, 60  
 CLKCTR\_GetSwitchI2SClk  
 Драйвер модуля CLKCTR, 60  
 CLKCTR\_GetSwitchLPClk  
 Драйвер модуля CLKCTR, 60  
 CLKCTR\_GetSwitchMainClk  
 Драйвер модуля CLKCTR, 61  
 CLKCTR\_GetSwitchMCOClk  
 Драйвер модуля CLKCTR, 61  
 CLKCTR\_GetSwitchPLLRef  
 Драйвер модуля CLKCTR, 61  
 CLKCTR\_GetSwitchPMUDIS  
 Драйвер модуля CLKCTR, 62  
 CLKCTR\_GetSwitchRTCClk  
 Драйвер модуля CLKCTR, 62  
 CLKCTR\_GetSwitchUSBClk  
 Драйвер модуля CLKCTR, 62  
 CLKCTR\_GetSysClk  
 Драйвер модуля CLKCTR, 63  
 CLKCTR\_GetSysClkDiv  
 Драйвер модуля CLKCTR, 63  
 CLKCTR\_GetUSBClk  
 Драйвер модуля CLKCTR, 63  
 CLKCTR\_GetXTI  
 Драйвер модуля CLKCTR, 64  
 CLKCTR\_GetXTI32  
 Драйвер модуля CLKCTR, 64  
 CLKCTR\_GetXTIClk  
 Драйвер модуля CLKCTR, 64  
 CLKCTR\_GMSSysClkForce  
 Драйвер модуля CLKCTR, 47  
 clkctr\_gnssclk\_div  
 clkctr\_div, 413  
 CLKCTR\_GNSSCLK\_MAX  
 Драйвер модуля CLKCTR, 40  
 CLKCTR\_GNSSCLK\_MIN  
 Драйвер модуля CLKCTR, 41  
 clkctr\_hfi\_for\_main\_type  
 Драйвер модуля CLKCTR, 47  
 CLKCTR\_HFI\_MAX

Драйвер модуля CLKCTR, 41  
 CLKCTR\_HFI\_MIN  
 Драйвер модуля CLKCTR, 41  
 CLKCTR\_HFIForMainTypeHFI  
 Драйвер модуля CLKCTR, 48  
 CLKCTR\_HFIForMainTypeXTI  
 Драйвер модуля CLKCTR, 48  
 CLKCTR\_I2S\_EXTCLK\_MAX  
 Драйвер модуля CLKCTR, 41  
 CLKCTR\_I2S\_EXTCLK\_MIN  
 Драйвер модуля CLKCTR, 41  
 clkctr\_i2sclk  
 Драйвер модуля CLKCTR, 48  
 clkctr\_i2sclk\_div  
 clkctr\_div, 413  
 CLKCTR\_I2SCLK\_MAX  
 Драйвер модуля CLKCTR, 41  
 CLKCTR\_I2SCLK\_MIN  
 Драйвер модуля CLKCTR, 41  
 CLKCTR\_I2SClkTypeI2SClk  
 Драйвер модуля CLKCTR, 48  
 CLKCTR\_I2SClkTypeSysClk  
 Драйвер модуля CLKCTR, 48  
 clkctr\_int\_freq  
 Драйвер модуля CLKCTR, 48  
 CLKCTR\_IntFreqFClk  
 Драйвер модуля CLKCTR, 48  
 CLKCTR\_IntFreqFClkInt  
 Драйвер модуля CLKCTR, 48  
 CLKCTR\_IntFreqGNSSClk  
 Драйвер модуля CLKCTR, 48  
 CLKCTR\_IntFreqHFIClk  
 Драйвер модуля CLKCTR, 48  
 CLKCTR\_IntFreqHFIForMain  
 Драйвер модуля CLKCTR, 48  
 CLKCTR\_IntFreqI2SClk  
 Драйвер модуля CLKCTR, 48  
 CLKCTR\_IntFreqLPClk  
 Драйвер модуля CLKCTR, 48  
 CLKCTR\_IntFreqMainClk  
 Драйвер модуля CLKCTR, 48  
 CLKCTR\_IntFreqMCOClk  
 Драйвер модуля CLKCTR, 48  
 CLKCTR\_IntFreqPLLClk  
 Драйвер модуля CLKCTR, 48  
 CLKCTR\_IntFreqPLLRefClk  
 Драйвер модуля CLKCTR, 48  
 CLKCTR\_IntFreqQSPIClk  
 Драйвер модуля CLKCTR, 48  
 CLKCTR\_IntFreqRTCClk  
 Драйвер модуля CLKCTR, 48  
 CLKCTR\_IntFreqSysClk  
 Драйвер модуля CLKCTR, 48  
 CLKCTR\_IntFreqUSBClk  
 Драйвер модуля CLKCTR, 48  
 CLKCTR\_IntFreqXTIClk  
 Драйвер модуля CLKCTR, 48  
 CLKCTR\_LFI\_MAX



- Драйвер модуля CLKCTR, 41
- CLKCTR\_LFI\_MIN
- Драйвер модуля CLKCTR, 42
- clkctr\_lpcclk
- Драйвер модуля CLKCTR, 48
- CLKCTR\_LPClkType500
- Драйвер модуля CLKCTR, 49
- CLKCTR\_LPClkTypeRTCClk
- Драйвер модуля CLKCTR, 49
- clkctr\_mainclk
- Драйвер модуля CLKCTR, 49
- CLKCTR\_MainClkTypeHFIClk
- Драйвер модуля CLKCTR, 49
- CLKCTR\_MainClkTypeMax
- Драйвер модуля CLKCTR, 49
- CLKCTR\_MainClkTypePLLClk
- Драйвер модуля CLKCTR, 49
- CLKCTR\_MainClkTypeXTIClk
- Драйвер модуля CLKCTR, 49
- CLKCTR\_MAN\_MAX
- Драйвер модуля CLKCTR, 42
- CLKCTR\_MAX\_FCLK\_DIV
- Драйвер модуля CLKCTR, 42
- CLKCTR\_MAX\_GNSSCLK\_DIV
- Драйвер модуля CLKCTR, 42
- CLKCTR\_MAX\_I2SCLK\_DIV
- Драйвер модуля CLKCTR, 42
- CLKCTR\_MAX\_MCOCLK\_DIV
- Драйвер модуля CLKCTR, 42
- CLKCTR\_MAX\_QSPICLK\_DIV
- Драйвер модуля CLKCTR, 42
- CLKCTR\_MAX\_SYSCLK\_DIV
- Драйвер модуля CLKCTR, 43
- clkctr\_mco\_div
- clkctr\_div, 414
- clkctr\_mcoclk
- Драйвер модуля CLKCTR, 49
- CLKCTR\_MCOClkTypeFClk
- Драйвер модуля CLKCTR, 49
- CLKCTR\_MCOClkTypeFClkInt
- Драйвер модуля CLKCTR, 49
- CLKCTR\_MCOClkTypeHFIClk
- Драйвер модуля CLKCTR, 49
- CLKCTR\_MCOClkTypeLPClk
- Драйвер модуля CLKCTR, 49
- CLKCTR\_MCOClkTypeMainClk
- Драйвер модуля CLKCTR, 49
- CLKCTR\_MCOClkTypePLLClk
- Драйвер модуля CLKCTR, 49
- CLKCTR\_MCOClkTypeRTCClk
- Драйвер модуля CLKCTR, 49
- CLKCTR\_MCOClkTypeSysClk
- Драйвер модуля CLKCTR, 49
- CLKCTR\_MIN\_FCLK\_DIV
- Драйвер модуля CLKCTR, 43
- CLKCTR\_MIN\_GNSSCLK\_DIV
- Драйвер модуля CLKCTR, 43
- CLKCTR\_MIN\_I2SCLK\_DIV
- Драйвер модуля CLKCTR, 43
- CLKCTR\_MIN\_MCOCLK\_DIV
- Драйвер модуля CLKCTR, 43
- CLKCTR\_MIN\_QSPICLK\_DIV
- Драйвер модуля CLKCTR, 43
- CLKCTR\_MIN\_SYSCLK\_DIV
- Драйвер модуля CLKCTR, 43
- CLKCTR\_NF\_MAN\_MAX
- Драйвер модуля CLKCTR, 43
- CLKCTR\_NR\_MAN\_MAX
- Драйвер модуля CLKCTR, 44
- CLKCTR\_OD\_MAN\_MAX
- Драйвер модуля CLKCTR, 44
- clkctr\_pll\_cfg, 414
- lock, 415
- man, 415
- nf\_man, 415
- nr\_man, 415
- od\_man, 415
- sel, 415
- CLKCTR\_PLLCLK\_MAX
- Драйвер модуля CLKCTR, 44
- CLKCTR\_PLLCLK\_MIN
- Драйвер модуля CLKCTR, 44
- CLKCTR\_PLLCLK\_OD\_MAX
- Драйвер модуля CLKCTR, 44
- CLKCTR\_PLLCLK\_OD\_MIN
- Драйвер модуля CLKCTR, 44
- clkctr\_pllref
- Драйвер модуля CLKCTR, 49
- CLKCTR\_PLLREF\_MAX
- Драйвер модуля CLKCTR, 45
- CLKCTR\_PLLREF\_MIN
- Драйвер модуля CLKCTR, 45
- CLKCTR\_PLLRefTypeHFIClk
- Драйвер модуля CLKCTR, 49
- CLKCTR\_PLLRefTypeXTIClk
- Драйвер модуля CLKCTR, 49
- clkctr\_qspiclk\_div
- clkctr\_div, 414
- CLKCTR\_QSPICLK\_MAX
- Драйвер модуля CLKCTR, 45
- CLKCTR\_QSPICLK\_MIN
- Драйвер модуля CLKCTR, 45
- clkctr\_rtccclk
- Драйвер модуля CLKCTR, 49
- CLKCTR\_RTCClkTypeLFE
- Драйвер модуля CLKCTR, 50
- CLKCTR\_RTCClkTypeLFI
- Драйвер модуля CLKCTR, 50
- CLKCTR\_SEL\_MAX
- Драйвер модуля CLKCTR, 45
- CLKCTR\_SetClkForce
- Драйвер модуля CLKCTR, 65
- CLKCTR\_SetDivClk
- Драйвер модуля CLKCTR, 65
- CLKCTR\_SetFClkDiv
- Драйвер модуля CLKCTR, 65

- CLKCTR\_SetFrequency
  - Драйвер модуля CLKCTR, 66
- CLKCTR\_SetGNSSClkDiv
  - Драйвер модуля CLKCTR, 66
- CLKCTR\_SetHFI
  - Драйвер модуля CLKCTR, 67
- CLKCTR\_SetHFITrim
  - Драйвер модуля CLKCTR, 67
- CLKCTR\_SetI2SClkDiv
  - Драйвер модуля CLKCTR, 68
- CLKCTR\_SetI2SExtClk
  - Драйвер модуля CLKCTR, 68
- CLKCTR\_SetLFI
  - Драйвер модуля CLKCTR, 69
- CLKCTR\_SetMCOdiv
  - Драйвер модуля CLKCTR, 69
- CLKCTR\_SetMCOEn
  - Драйвер модуля CLKCTR, 70
- CLKCTR\_SetPll
  - Драйвер модуля CLKCTR, 70
- CLKCTR\_SetPLLConfig
  - Драйвер модуля CLKCTR, 70
- CLKCTR\_SetPllMan
  - Драйвер модуля CLKCTR, 71
- CLKCTR\_SetQSPIClkDiv
  - Драйвер модуля CLKCTR, 72
- CLKCTR\_SetSwitchClk
  - Драйвер модуля CLKCTR, 72
- CLKCTR\_SetSwitchI2SClk
  - Драйвер модуля CLKCTR, 72
- CLKCTR\_SetSwitchLPClk
  - Драйвер модуля CLKCTR, 73
- CLKCTR\_SetSwitchMainClk
  - Драйвер модуля CLKCTR, 73
- CLKCTR\_SetSwitchMCOClk
  - Драйвер модуля CLKCTR, 74
- CLKCTR\_SetSwitchPLLRef
  - Драйвер модуля CLKCTR, 74
- CLKCTR\_SetSwitchPMUDIS
  - Драйвер модуля CLKCTR, 75
- CLKCTR\_SetSwitchRTCClk
  - Драйвер модуля CLKCTR, 76
- CLKCTR\_SetSwitchUSBClk
  - Драйвер модуля CLKCTR, 76
- CLKCTR\_SetSysClkDiv
  - Драйвер модуля CLKCTR, 77
- CLKCTR\_SetSysDiv
  - Драйвер модуля CLKCTR, 77
- CLKCTR\_SetXTI
  - Драйвер модуля CLKCTR, 77
- CLKCTR\_SetXTI32
  - Драйвер модуля CLKCTR, 78
- CLKCTR\_SMCClkForce
  - Драйвер модуля CLKCTR, 47
- CLKCTR\_SRAMFClkForce
  - Драйвер модуля CLKCTR, 47
- CLKCTR\_SRAMSysClkForce
  - Драйвер модуля CLKCTR, 47
- clkctr\_status
  - Драйвер модуля CLKCTR, 50
- CLKCTR\_Status\_CheckError
  - Драйвер модуля CLKCTR, 50
- CLKCTR\_Status\_ConfigureError
  - Драйвер модуля CLKCTR, 50
- CLKCTR\_Status\_InvalidArgument
  - Драйвер модуля CLKCTR, 50
- CLKCTR\_Status\_Ok
  - Драйвер модуля CLKCTR, 50
- CLKCTR\_Status\_VerifyError
  - Драйвер модуля CLKCTR, 50
- clkctr\_sysclk\_div
  - clkctr\_div, 414
- CLKCTR\_SYSCLK\_MAX
  - Драйвер модуля CLKCTR, 45
- CLKCTR\_SYSCLK\_MIN
  - Драйвер модуля CLKCTR, 45
- CLKCTR\_SysFClkForce
  - Драйвер модуля CLKCTR, 47
- CLKCTR\_SysSysClkForce
  - Драйвер модуля CLKCTR, 47
- clkctr\_usbcclk
  - Драйвер модуля CLKCTR, 50
- CLKCTR\_USBClkTypeHFIClk
  - Драйвер модуля CLKCTR, 50
- CLKCTR\_USBClkTypeXTIClk
  - Драйвер модуля CLKCTR, 50
- CLKCTR\_XTI32\_MAX
  - Драйвер модуля CLKCTR, 46
- CLKCTR\_XTI32\_MIN
  - Драйвер модуля CLKCTR, 46
- CLKCTR\_XTI\_MAX
  - Драйвер модуля CLKCTR, 46
- CLKCTR\_XTI\_MIN
  - Драйвер модуля CLKCTR, 46
- clkdiv
  - rw\_c\_config, 444
- CMD
  - sdmmc\_cfg\_pin\_map, 460
- cmd
  - \_qspi\_xip\_config, 405
  - pwm\_channel\_config, 435
- cmd\_format
  - \_qspi\_nor\_init\_config, 404
- cmpa
  - pwm\_channel\_config, 436
- cmpb
  - pwm\_channel\_config, 436
- cnt\_eq\_cmpa\_dec\_outa
  - pwm\_channel\_config, 436
- cnt\_eq\_cmpa\_dec\_outb
  - pwm\_channel\_config, 436
- cnt\_eq\_cmpa\_inc\_outa
  - pwm\_channel\_config, 436
- cnt\_eq\_cmpa\_inc\_outb
  - pwm\_channel\_config, 436
- cnt\_eq\_cmpb\_dec\_outa

- pwm\_channel\_config, 436
- cnt\_eq\_cmpb\_dec\_outb
  - pwm\_channel\_config, 436
- cnt\_eq\_cmpb\_inc\_outa
  - pwm\_channel\_config, 437
- cnt\_eq\_cmpb\_inc\_outb
  - pwm\_channel\_config, 437
- cnt\_eq\_prd\_outa
  - pwm\_channel\_config, 437
- cnt\_eq\_prd\_outb
  - pwm\_channel\_config, 437
- cnt\_eq\_zero\_outa
  - pwm\_channel\_config, 437
- cnt\_eq\_zero\_outb
  - pwm\_channel\_config, 437
- cntmode
  - pwm\_channel\_config, 437
- command\_set
  - \_qspi\_nor\_handle, 403
- command\_type
  - \_qspi\_nor\_handle, 403
- completion\_callback
  - \_i2c\_master\_dma\_handle, 383
  - \_i2c\_master\_handle, 385
- completion\_status
  - i2c\_slave\_transfer\_t, 424
- config
  - rwu\_union\_reg, 454
- cont\_trans\_en
  - \_qspi\_config, 399
- cont\_transfer\_ext
  - \_qspi\_config, 400
- count
  - \_dma\_multiblock\_config, 381
- counter
  - pwm\_channel\_config, 437
  - vtu\_config, 480
- cpha
  - \_qspi\_config, 400
  - \_qspi\_xip\_config, 405
- cpol
  - \_qspi\_config, 400
  - \_qspi\_xip\_config, 405
- csd
  - sdmmc\_card\_t, 457
- CTL\_HI
  - \_dma\_channel\_reg, 375
  - \_dma\_descriptor, 378
- CTL\_LO
  - \_dma\_channel\_reg, 375
  - \_dma\_descriptor, 378
- ctlx\_cfg
  - \_dma\_channel\_config, 373
- ctrl\_word\_len
  - spi\_microwire\_cfg\_t, 469
- ctrphs
  - pwm\_channel\_config, 438
- cycle\_time
  - \_can\_rx\_buffer\_frame, 364
- cyclicity
  - dualtimer\_hardware\_config, 418
- D0
  - sdmmc\_cfg\_pin\_map, 460
- D1
  - sdmmc\_cfg\_pin\_map, 460
- D2
  - sdmmc\_cfg\_pin\_map, 460
- D3
  - sdmmc\_cfg\_pin\_map, 460
- D4
  - sdmmc\_cfg\_pin\_map, 460
- D5
  - sdmmc\_cfg\_pin\_map, 460
- D6
  - sdmmc\_cfg\_pin\_map, 460
- D7
  - sdmmc\_cfg\_pin\_map, 461
- DAR
  - \_dma\_descriptor, 378
- DAR\_HI
  - \_dma\_channel\_reg, 375
- DAR\_LO
  - \_dma\_channel\_reg, 375
- data
  - \_can\_rx\_buffer\_frame, 364
  - \_can\_tx\_buffer\_frame, 371
  - i2c\_master\_transfer\_t, 421
- data\_prescaler
  - \_can\_timing\_config, 369
- data\_seg1
  - \_can\_timing\_config, 369
- data\_seg2
  - \_can\_timing\_config, 369
- data\_size
  - \_dma\_multiblock\_config, 381
  - i2c\_master\_transfer\_t, 422
  - spi\_transfer\_t, 471
  - uart\_transfer, 479
- data\_sjw
  - \_can\_timing\_config, 369
- data\_width\_bits
  - spi\_config\_t, 465
- day
  - \_rtc\_datetime, 406
- DAYS\_IN\_A\_YEAR
  - Драйвер модуля RWC, 232
- dcdc\_ccm\_enable
  - power\_mode\_config, 429
- dcdc\_enable
  - power\_config, 427
- dcdc\_imax
  - power\_trim\_config, 432
- dcdc\_imin
  - power\_trim\_config, 432
- dcdc\_is\_ready
  - power\_state, 431

- dcdc\_level
  - power\_mode\_config, 429
- dcdc\_low\_consumption\_enable
  - power\_mode\_config, 429
- dcdc\_mode
  - power\_mode\_config, 430
- dcdc\_sink\_enable
  - power\_mode\_config, 430
- dcdc\_swdrv
  - power\_mode\_config, 430
- dcdc\_trimlc
  - power\_trim\_config, 433
- ddr\_mode
  - sdmmc\_card\_t, 457
- delay\_compensation\_enable
  - \_can\_timing\_config, 369
- delay\_en
  - \_qspi\_config, 400
- delay\_us
  - sdmmc\_port\_cfg\_t, 462
- dev\_num
  - sdmmc\_card\_t, 457
- device\_specific
  - \_nor\_handle, 394
- devices/eliot1/drivers/hal\_can.h, 483, 487
- devices/eliot1/drivers/hal\_clkctr.h, 492, 497
- devices/eliot1/drivers/hal\_common.h, 501
- devices/eliot1/drivers/hal\_dma.h, 502, 506
- devices/eliot1/drivers/hal\_dualtimer.h, 510, 511
- devices/eliot1/drivers/hal\_flash.h, 512, 513
- devices/eliot1/drivers/hal\_gpio.h, 514, 530
- devices/eliot1/drivers/hal\_i2c.h, 531, 543
- devices/eliot1/drivers/hal\_i2c\_dma.h, 548, 549
- devices/eliot1/drivers/hal\_i2s.h, 550, 553
- devices/eliot1/drivers/hal\_ioim.h, 555, 556
- devices/eliot1/drivers/hal\_jtm.h, 556
- devices/eliot1/drivers/hal\_mhu.h, 557, 558
- devices/eliot1/drivers/hal\_nor\_flash.h, 558, 560
- devices/eliot1/drivers/hal\_power.h, 561, 563
- devices/eliot1/drivers/hal\_ppu.h, 565, 567
- devices/eliot1/drivers/hal\_pwm.h, 569, 571
- devices/eliot1/drivers/hal\_pwm\_newgen.h, 575
- devices/eliot1/drivers/hal\_qspi.h, 577, 579
- devices/eliot1/drivers/hal\_qspi\_dma.h, 581, 582
- devices/eliot1/drivers/hal\_qspi\_nor\_flash.h, 583, 585
- devices/eliot1/drivers/hal\_rwc.h, 586, 590
- devices/eliot1/drivers/hal\_sdmmc.h, 593, 596
- devices/eliot1/drivers/hal\_sdmmc\_cfg.h, 597, 598
- devices/eliot1/drivers/hal\_smc.h, 598, 600
- devices/eliot1/drivers/hal\_spi.h, 601, 605
- devices/eliot1/drivers/hal\_spi\_dma.h, 610, 611
- devices/eliot1/drivers/hal\_timer.h, 613, 614
- devices/eliot1/drivers/hal\_uart.h, 615, 619
- devices/eliot1/drivers/hal\_uart\_dma.h, 623, 624
- devices/eliot1/drivers/hal\_vtu.h, 625, 627
- devices/eliot1/drivers/hal\_wdt.h, 629, 630
- DIM
  - Общие определения для библиотеки HAL, 81
- direction
  - i2c\_master\_transfer\_t, 422
  - spi\_config\_t, 465
- dlc
  - \_can\_rx\_buffer\_frame, 365
  - \_can\_tx\_buffer\_frame, 371
- DMA\_AbortTransfer
  - Драйвер модуля DMA, 91
- DMA\_AllIRQ
  - Драйвер модуля DMA, 89
- DMA\_BurstSize1
  - Драйвер модуля DMA, 89
- DMA\_BurstSize128
  - Драйвер модуля DMA, 89
- DMA\_BurstSize16
  - Драйвер модуля DMA, 89
- DMA\_BurstSize256
  - Драйвер модуля DMA, 89
- DMA\_BurstSize32
  - Драйвер модуля DMA, 89
- DMA\_BurstSize4
  - Драйвер модуля DMA, 89
- DMA\_BurstSize64
  - Драйвер модуля DMA, 89
- DMA\_BurstSize8
  - Драйвер модуля DMA, 89
- DMA\_CHANNEL\_CTL
  - Драйвер модуля DMA, 87
- DMA\_CHANNEL\_CTL\_BLOCKSIZE\_MASK
  - Драйвер модуля DMA, 88
- DMA\_CHANNEL\_CTL\_BLOCKSIZE\_SHIFT
  - Драйвер модуля DMA, 88
- dma\_channel\_ctl\_cfg, 415
  - block\_size, 416
  - dst\_burst\_size, 416
  - dst\_incr, 416
  - dst\_tr\_width, 416
  - gather\_en, 416
  - int\_en, 417
  - llp\_dst\_en, 417
  - llp\_src\_en, 417
  - scatter\_en, 417
  - src\_burst\_size, 417
  - src\_incr, 417
  - src\_tr\_width, 417
  - transfer\_type, 417
- DMA\_ChannelIRQHandle
  - Драйвер модуля DMA, 91
- DMA\_ChannelsActive
  - Драйвер модуля DMA, 91
- DMA\_ChannelPriority0
  - Драйвер модуля DMA, 90
- DMA\_ChannelPriority1
  - Драйвер модуля DMA, 90
- DMA\_ChannelPriority2
  - Драйвер модуля DMA, 90
- DMA\_ChannelPriority3
  - Драйвер модуля DMA, 90

- Драйвер модуля DMA, 90
- DMA\_ChannelPriority4
  - Драйвер модуля DMA, 90
- DMA\_ChannelPriority5
  - Драйвер модуля DMA, 90
- DMA\_ChannelPriority6
  - Драйвер модуля DMA, 90
- DMA\_ChannelPriority7
  - Драйвер модуля DMA, 90
- DMA\_CreateHandle
  - Драйвер модуля DMA, 91
- DMA\_Descr
  - Драйвер модуля DMA, 88
- DMA\_Deinit
  - Драйвер модуля DMA, 92
- DMA\_DisableChannel
  - Драйвер модуля DMA, 92
- DMA\_DisableChannelInterrupt
  - Драйвер модуля DMA, 92
- dma\_en
  - \_qspi\_config, 400
- DMA\_EnableChannel
  - Драйвер модуля DMA, 92
- DMA\_EnableChannelInterrupt
  - Драйвер модуля DMA, 93
- DMA\_EnableChannelPeriphRq
  - Драйвер модуля DMA, 93
- DMA\_GetChannelPriority
  - Драйвер модуля DMA, 93
- DMA\_GetCTLCfgMask
  - Драйвер модуля DMA, 94
- DMA\_GetDescriptorCount
  - Драйвер модуля DMA, 94
- DMA\_HardwareHandshakeEnable
  - Драйвер модуля DMA, 94
- DMA\_Incr
  - Драйвер модуля DMA, 88
- DMA\_Init
  - Драйвер модуля DMA, 95
- DMA\_InitMultiblockDescriptor
  - Драйвер модуля DMA, 95
- DMA\_IntBlock
  - Драйвер модуля DMA, 89
- DMA\_IntDstTran
  - Драйвер модуля DMA, 89
- DMA\_IntError
  - Драйвер модуля DMA, 89
- DMA\_IntSrcTran
  - Драйвер модуля DMA, 89
- DMA\_IntTfr
  - Драйвер модуля DMA, 89
- DMA\_MemoryToMemory\_DMA
  - Драйвер модуля DMA, 90
- DMA\_MemoryToPeripheral\_DMA
  - Драйвер модуля DMA, 90
- DMA\_MemoryToPeripheral\_Peripheral
  - Драйвер модуля DMA, 90
- DMA\_NoChange
  - Драйвер модуля DMA, 88
- DMA\_PeripheralToMemory\_DMA
  - Драйвер модуля DMA, 90
- DMA\_PeripheralToMemory\_Peripheral
  - Драйвер модуля DMA, 90
- DMA\_PeripheralToPeripheral\_DMA
  - Драйвер модуля DMA, 90
- DMA\_PeripheralToPeripheral\_DST
  - Драйвер модуля DMA, 90
- DMA\_PeripheralToPeripheral\_SRC
  - Драйвер модуля DMA, 90
- DMA\_PrepareChannelTransfer
  - Драйвер модуля DMA, 95
- DMA\_ScatterGatherEnable
  - Драйвер модуля DMA, 96
- DMA\_SetCallback
  - Драйвер модуля DMA, 96
- DMA\_SetChannelPriority
  - Драйвер модуля DMA, 96
- DMA\_SetupDescriptor
  - Драйвер модуля DMA, 97
- DMA\_SetupDstScatter
  - Драйвер модуля DMA, 97
- DMA\_SetupSrcGather
  - Драйвер модуля DMA, 97
- DMA\_StartTransfer
  - Драйвер модуля DMA, 98
- DMA\_Status\_Busy
  - Драйвер модуля DMA, 90
- DMA\_Status\_Fail
  - Драйвер модуля DMA, 90
- DMA\_Status\_NoBase
  - Драйвер модуля DMA, 90
- DMA\_Status\_Success
  - Драйвер модуля DMA, 90
- DMA\_SubmitChannelDescriptor
  - Драйвер модуля DMA, 98
- DMA\_SubmitChannelTransfer
  - Драйвер модуля DMA, 98
- DMA\_SubmitChannelTransferParameter
  - Драйвер модуля DMA, 99
- DMA\_Transfer128BitWidth
  - Драйвер модуля DMA, 89
- DMA\_Transfer16BitWidth
  - Драйвер модуля DMA, 89
- DMA\_Transfer256BitWidth
  - Драйвер модуля DMA, 89
- DMA\_Transfer32BitWidth
  - Драйвер модуля DMA, 89
- DMA\_Transfer64BitWidth
  - Драйвер модуля DMA, 89
- DMA\_Transfer8BitWidth
  - Драйвер модуля DMA, 89
- driver\_base\_addr
  - \_nor\_config, 393
  - \_nor\_handle, 394
- DSR\_HI
  - \_dma\_channel\_reg, 375

- DSR\_LO
  - \_dma\_channel\_reg, 375
- dst\_addr
  - \_dma\_channel\_config, 373
  - \_dma\_multiblock\_config, 381
- dst\_burst\_size
  - \_dma\_multiblock\_config, 381
  - dma\_channel\_ctl\_cfg, 416
- dst\_data\_width
  - \_dma\_multiblock\_config, 381
- dst\_incr
  - \_dma\_multiblock\_config, 381
  - dma\_channel\_ctl\_cfg, 416
- dst\_tr\_width
  - dma\_channel\_ctl\_cfg, 416
- DSTAT
  - \_dma\_descriptor, 378
- DSTAT\_HI
  - \_dma\_channel\_reg, 376
- DSTAT\_LO
  - \_dma\_channel\_reg, 376
- DSTATAR\_HI
  - \_dma\_channel\_reg, 376
- DSTATAR\_LO
  - \_dma\_channel\_reg, 376
- DUALTIMER\_Deinit
  - Драйвер модуля DTIM, 103
- DUALTIMER\_Disable
  - Драйвер модуля DTIM, 103
- DUALTIMER\_Enable
  - Драйвер модуля DTIM, 103
- DUALTIMER\_FreeRunning
  - Драйвер модуля DTIM, 102
- DUALTIMER\_GetAPIStatus
  - Драйвер модуля DTIM, 104
- DUALTIMER\_GetDefaultConfig
  - Драйвер модуля DTIM, 104
- DUALTIMER\_GetRawStatus
  - Драйвер модуля DTIM, 104
- DUALTIMER\_GetStatus
  - Драйвер модуля DTIM, 105
- DUALTIMER\_GetTick
  - Драйвер модуля DTIM, 105
- dualtimer\_hardware\_config, 418
  - bg\_load, 418
  - cyclicity, 418
  - enable, 418
  - int\_ctrl, 419
  - load, 419
  - mode, 419
  - prescale, 419
  - size, 419
- DUALTIMER\_Init
  - Драйвер модуля DTIM, 105
- dualtimer\_interrupt\_control
  - Драйвер модуля DTIM, 101
- DUALTIMER\_InterruptDisable
  - Драйвер модуля DTIM, 102
- DUALTIMER\_InterruptEnable
  - Драйвер модуля DTIM, 102
- DUALTIMER\_IrqClr
  - Драйвер модуля DTIM, 106
- DUALTIMER\_MAX\_INDEX
  - Драйвер модуля DTIM, 101
- dualtimer\_mode
  - Драйвер модуля DTIM, 102
- DUALTIMER\_NUMBER\_OF\_DUALTIMERS
  - Драйвер модуля DTIM, 101
- dualtimer\_number\_of\_repetitions
  - Драйвер модуля DTIM, 102
- DUALTIMER\_OneShot
  - Драйвер модуля DTIM, 102
- DUALTIMER\_Periodic
  - Драйвер модуля DTIM, 102
- dualtimer\_prescale
  - Драйвер модуля DTIM, 102
- DUALTIMER\_Prescale1
  - Драйвер модуля DTIM, 102
- DUALTIMER\_Prescale16
  - Драйвер модуля DTIM, 102
- DUALTIMER\_Prescale256
  - Драйвер модуля DTIM, 102
- DUALTIMER\_Reload
  - Драйвер модуля DTIM, 106
- DUALTIMER\_Run
  - Драйвер модуля DTIM, 107
- dualtimer\_status
  - Драйвер модуля DTIM, 102
- DUALTIMER\_Status\_BadConfigure
  - Драйвер модуля DTIM, 103
- DUALTIMER\_Status\_InvalidArgument
  - Драйвер модуля DTIM, 103
- DUALTIMER\_Status\_Ok
  - Драйвер модуля DTIM, 103
- DUALTIMER\_Status\_TimerBusy
  - Драйвер модуля DTIM, 103
- DUALTIMER\_Stop
  - Драйвер модуля DTIM, 107
- dualtimer\_timer\_size
  - Драйвер модуля DTIM, 103
- DUALTIMER\_TimerSize16
  - Драйвер модуля DTIM, 103
- DUALTIMER\_TimerSize32
  - Драйвер модуля DTIM, 103
- dualtimer\_work\_enable
  - Драйвер модуля DTIM, 103
- DUALTIMER\_WrappingMode
  - Драйвер модуля DTIM, 102
- DUMMY\_BYTE
  - Драйвер модуля QSPI, 211
- dummy\_cycles
  - \_qspi\_xip\_config, 405
- dummy\_data
  - \_i2c\_master\_dma\_handle, 383
  - \_spi\_dma\_handle, 408
- duty\_cycle



- `_pwm_handle`, 397
- `duty_cycle_capture`
  - `vtu_config`, 480
- `dz_falling_edge_delay_clk`
  - `pwm_channel_config`, 438
- `dz_falling_edge_outb_inv`
  - `pwm_channel_config`, 438
- `dz_falling_edge_source`
  - `pwm_channel_config`, 438
- `dz_outa_enable`
  - `pwm_channel_config`, 438
- `dz_outb_enable`
  - `pwm_channel_config`, 438
- `dz_rising_edge_delay_clk`
  - `pwm_channel_config`, 438
- `dz_rising_edge_outa_inv`
  - `pwm_channel_config`, 438
- `dz_rising_edge_source`
  - `pwm_channel_config`, 439
- `eco_mode`
  - `power_mode_config`, 430
- `emmc_8bit_en`
  - `sdmmc_port_cfg_t`, 462
- `enable`
  - `dualtimer_hardware_config`, 418
- `enable_hardware_flow_control`
  - `uart_config`, 474
- `enable_ide_check`
  - `_can_frame_filter`, 359
- `enable_infrared`
  - `uart_config`, 474
- `enable_listen_only`
  - `_can_config`, 357
- `enable_loopback`
  - `uart_config`, 474
- `enable_loopback_ext`
  - `_can_config`, 358
- `enable_loopback_int`
  - `_can_config`, 358
- `enable_master`
  - `i2c_master_config_t`, 420
- `enable_parity`
  - `uart_config`, 474
- `enable_rxfifo`
  - `uart_config`, 474
- `enable_slave`
  - `i2c_slave_config_t`, 423
- `enable_txfifo`
  - `uart_config`, 474
- `erase_chip_cmd`
  - `_nor_command_set`, 392
- `erase_sector_cmd`
  - `_nor_command_set`, 392
- `esi`
  - `_can_rx_buffer_frame`, 365
- `event`
  - `i2c_slave_transfer_t`, 424
- `event_mask`
  - `i2c_slave_transfer_t`, 424
- `eventprd`
  - `pwm_channel_config`, 439
- `FCTR_CMD_ERASE`
  - Драйвер модуля FLASH., 109
- `FCTR_CMD_MASS_ERASE`
  - Драйвер модуля FLASH., 109
- `FCTR_CMD_READ`
  - Драйвер модуля FLASH., 109
- `FCTR_CMD_ROW_WRITE`
  - Драйвер модуля FLASH., 109
- `FCTR_CMD_WRITE`
  - Драйвер модуля FLASH., 109
- `FCTR_IRQ_STS_CLR_SUCCESS_FLAGS`
  - Драйвер модуля FLASH., 109
- `FCTR_IRQ_STS_SET_RESULT_FLAGS`
  - Драйвер модуля FLASH., 109
- `fdf`
  - `_can_rx_buffer_frame`, 365
  - `_can_tx_buffer_frame`, 371
- `FIELD_BIT`
  - Общие определения для библиотеки HAL, 81
- `filter`
  - `_can_frame_filter_config`, 360
- `filter_config`
  - `_can_config`, 358
- `flags`
  - `i2c_master_transfer_t`, 422
- `FLASH_Erase`
  - Драйвер модуля FLASH., 111
- `FLASH_Init`
  - Драйвер модуля FLASH., 111
- `flash_low_voltage_read_enable`
  - `power_config`, 427
- `flash_low_voltage_read_enabled`
  - `power_state`, 431
- `FLASH_MainRegion`
  - Драйвер модуля FLASH., 110
- `FLASH_MassErase`
  - Драйвер модуля FLASH., 112
- `flash_power_mode`
  - `power_mode_config`, 430
  - `power_state`, 431
- `FLASH_Program`
  - Драйвер модуля FLASH., 112
- `FLASH_Read`
  - Драйвер модуля FLASH., 113
- `flash_region`
  - Драйвер модуля FLASH., 110
- `FLASH_STAT_BUSY`
  - Драйвер модуля QSPI, 211
- `FLASH_STAT_WEL`
  - Драйвер модуля QSPI, 211
- `flash_status`
  - Драйвер модуля FLASH., 110
- `FLASH_Status_AddressAlignmentError`
  - Драйвер модуля FLASH., 111
- `FLASH_Status_AddressOutOfRange`

- Драйвер модуля FLASH., 111
- FLASH\_Status\_CheckError
  - Драйвер модуля FLASH., 111
- FLASH\_Status\_ConfigureError
  - Драйвер модуля FLASH., 111
- FLASH\_Status\_InvalidArgument
  - Драйвер модуля FLASH., 111
- FLASH\_Status\_Ok
  - Драйвер модуля FLASH., 111
- FLASH\_Status\_VerifyError
  - Драйвер модуля FLASH., 111
- FLASH\_SystemRegion
  - Драйвер модуля FLASH., 110
- FLASH\_TEST\_ADDRESSES
  - Драйвер модуля FLASH., 110
- FLASH\_VerifyErase
  - Драйвер модуля FLASH., 113
- FLASH\_VerifyProgram
  - Драйвер модуля FLASH., 114
- FLASH\_WriteWord
  - Драйвер модуля FLASH., 115
- frame\_format
  - spi\_config\_t, 465
- frame\_width\_bits
  - spi\_config\_internal\_t, 464
  - spi\_handle, 467
- frame\_width\_bytes
  - spi\_config\_internal\_t, 464
  - spi\_handle, 468
- frames
  - \_can\_rx\_transfer, 366
  - \_can\_tx\_transfer, 372
- freq\_divider
  - sdmmc\_card\_t, 457
- freq\_input
  - sdmmc\_card\_t, 457
- freq\_out
  - sdmmc\_port\_cfg\_t, 462
- freq\_out\_init
  - sdmmc\_port\_cfg\_t, 462
- gather\_en
  - \_dma\_multiblock\_config, 381
  - dma\_channel\_ctl\_cfg, 416
- general
  - rcw\_config, 445
  - rcw\_union\_reg, 454
- GPIO\_ALT\_FUNC\_CAN\_GNSS\_USB
  - Драйвер модуля GPIO, 118
- GPIO\_ALT\_FUNC\_I2C\_I2S
  - Драйвер модуля GPIO, 118
- GPIO\_ALT\_FUNC\_PWM\_VTU
  - Драйвер модуля GPIO, 118
- GPIO\_ALT\_FUNC\_QSPI\_SPI2
  - Драйвер модуля GPIO, 118
- GPIO\_ALT\_FUNC\_SDMMC\_SMC
  - Драйвер модуля GPIO, 118
- GPIO\_ALT\_FUNC\_SPI0\_SPI1
  - Драйвер модуля GPIO, 118
- GPIO\_ALT\_FUNC\_TRACE\_JTAG\_FBIST
  - Драйвер модуля GPIO, 118
- GPIO\_ALT\_FUNC\_UART
  - Драйвер модуля GPIO, 118
- GPIO\_DigitalInput
  - hal\_gpio.h, 519
- GPIO\_DigitalOutput
  - hal\_gpio.h, 519
- GPIO\_GetInstance
  - hal\_gpio.h, 519
- GPIO\_INT\_EDGE\_FALL
  - hal\_gpio.h, 517
- GPIO\_INT\_EDGE\_RISE
  - hal\_gpio.h, 517
- GPIO\_INT\_LVL\_HI
  - hal\_gpio.h, 517
- GPIO\_INT\_LVL\_LO
  - hal\_gpio.h, 517
- gpio\_int\_type\_t
  - hal\_gpio.h, 517
- GPIO\_MAX\_CURRENT\_12mA
  - hal\_gpio.h, 519
- GPIO\_MAX\_CURRENT\_2mA
  - hal\_gpio.h, 519
- GPIO\_MAX\_CURRENT\_4mA
  - hal\_gpio.h, 519
- GPIO\_MAX\_CURRENT\_8mA
  - hal\_gpio.h, 519
- GPIO\_MODE\_AF
  - Драйвер модуля GPIO, 118
- GPIO\_MODE\_GPIO
  - Драйвер модуля GPIO, 118
- GPIO\_MODE\_HI\_Z
  - Драйвер модуля GPIO, 118
- GPIO\_MODE\_INVALID
  - Драйвер модуля GPIO, 118
- gpio\_mode\_t
  - Драйвер модуля GPIO, 117
- gpio\_pin\_direction\_t
  - hal\_gpio.h, 517
- gpio\_pin\_function\_t
  - Драйвер модуля GPIO, 118
- gpio\_pin\_max\_current\_t
  - hal\_gpio.h, 519
- gpio\_pin\_pup\_d\_t
  - hal\_gpio.h, 519
- GPIO\_PinIRQ\_Clear
  - hal\_gpio.h, 520
- GPIO\_PinIRQ\_Disable
  - hal\_gpio.h, 520
- GPIO\_PinIRQ\_Enable
  - hal\_gpio.h, 520
- GPIO\_PinIRQ\_GetStatus
  - hal\_gpio.h, 520
- GPIO\_PinMode\_Function
  - hal\_gpio.h, 521
- GPIO\_PinMode\_Get
  - hal\_gpio.h, 521



- GPIO\_PinMode\_GPIO
  - hal\_gpio.h, [521](#)
- GPIO\_PinMode\_HiZ
  - hal\_gpio.h, [522](#)
- GPIO\_PinRead
  - hal\_gpio.h, [522](#)
- GPIO\_PinSet\_MaxCurrent
  - hal\_gpio.h, [522](#)
- GPIO\_PinSet\_PUPD
  - hal\_gpio.h, [522](#)
- GPIO\_PinSet\_Schmitt
  - hal\_gpio.h, [523](#)
- GPIO\_PinSet\_SpeedRaise
  - hal\_gpio.h, [523](#)
- GPIO\_PinToggle
  - hal\_gpio.h, [523](#)
- GPIO\_PinWrite
  - hal\_gpio.h, [523](#)
- GPIO\_PORTA
  - Драйвер модуля GPIO, [116](#)
- GPIO\_PORTB
  - Драйвер модуля GPIO, [116](#)
- GPIO\_PORTC
  - Драйвер модуля GPIO, [116](#)
- GPIO\_PORTD
  - Драйвер модуля GPIO, [117](#)
- GPIO\_PortIRQ\_Clear
  - hal\_gpio.h, [524](#)
- GPIO\_PortIRQ\_Disable
  - hal\_gpio.h, [524](#)
- GPIO\_PortIRQ\_Enable
  - hal\_gpio.h, [524](#)
- GPIO\_PortIRQ\_GetStatus
  - hal\_gpio.h, [524](#)
- GPIO\_PortMode\_Function
  - hal\_gpio.h, [525](#)
- GPIO\_PortMode\_GPIO
  - hal\_gpio.h, [525](#)
- GPIO\_PortMode\_HiZ
  - hal\_gpio.h, [525](#)
- GPIO\_PORTPIN
  - Драйвер модуля GPIO, [117](#)
- GPIO\_PORTPIN\_GET\_MASK
  - Драйвер модуля GPIO, [117](#)
- GPIO\_PORTPIN\_GET\_PIN\_NUM
  - Драйвер модуля GPIO, [117](#)
- GPIO\_PORTPIN\_GET\_PORT\_NUM
  - Драйвер модуля GPIO, [117](#)
- GPIO\_PortRead
  - hal\_gpio.h, [526](#)
- GPIO\_PortSecureIRQ\_Clear
  - hal\_gpio.h, [526](#)
- GPIO\_PortSecureIRQ\_GetInfo1
  - hal\_gpio.h, [526](#)
- GPIO\_PortSecureIRQ\_GetInfo2
  - hal\_gpio.h, [526](#)
- GPIO\_PortSecureIRQ\_GetStatus
  - hal\_gpio.h, [527](#)
- GPIO\_PortSecureIRQ\_SetMask
  - hal\_gpio.h, [527](#)
- GPIO\_PortSet\_MaxCurrent
  - hal\_gpio.h, [527](#)
- GPIO\_PortSet\_NonsecureMask
  - hal\_gpio.h, [528](#)
- GPIO\_PortSet\_PUPD
  - hal\_gpio.h, [528](#)
- GPIO\_PortSet\_Schmitt
  - hal\_gpio.h, [528](#)
- GPIO\_PortSet\_SpeedRaise
  - hal\_gpio.h, [528](#)
- GPIO\_PortToggle
  - hal\_gpio.h, [529](#)
- GPIO\_PortWrite
  - hal\_gpio.h, [529](#)
- GPIO\_PULL\_DOWN
  - hal\_gpio.h, [519](#)
- GPIO\_PULL\_DOWN\_OD
  - hal\_gpio.h, [519](#)
- GPIO\_PULL\_NONE
  - hal\_gpio.h, [519](#)
- GPIO\_PULL\_NONE\_OD
  - hal\_gpio.h, [519](#)
- GPIO\_PULL\_UP
  - hal\_gpio.h, [519](#)
- GPIO\_PULL\_UP\_OD
  - hal\_gpio.h, [519](#)
- GPIO\_SEC\_INT\_PORT\_NONSEC\_MASK
  - hal\_gpio.h, [517](#)
- GPIO\_SEC\_INT\_SEC\_ACC\_MASK
  - hal\_gpio.h, [517](#)
- gpio\_vol
  - sdmhc\_card\_t, [457](#)
- hal\_gpio.h
  - GPIO\_DigitalInput, [519](#)
  - GPIO\_DigitalOutput, [519](#)
  - GPIO\_GetInstance, [519](#)
  - GPIO\_INT\_EDGE\_FALL, [517](#)
  - GPIO\_INT\_EDGE\_RISE, [517](#)
  - GPIO\_INT\_LVL\_HI, [517](#)
  - GPIO\_INT\_LVL\_LO, [517](#)
  - gpio\_int\_type\_t, [517](#)
  - GPIO\_MAX\_CURRENT\_12mA, [519](#)
  - GPIO\_MAX\_CURRENT\_2mA, [519](#)
  - GPIO\_MAX\_CURRENT\_4mA, [519](#)
  - GPIO\_MAX\_CURRENT\_8mA, [519](#)
  - gpio\_pin\_direction\_t, [517](#)
  - gpio\_pin\_max\_current\_t, [519](#)
  - gpio\_pin\_pupdt, [519](#)
  - GPIO\_PinIRQ\_Clear, [520](#)
  - GPIO\_PinIRQ\_Disable, [520](#)
  - GPIO\_PinIRQ\_Enable, [520](#)
  - GPIO\_PinIRQ\_GetStatus, [520](#)
  - GPIO\_PinMode\_Function, [521](#)
  - GPIO\_PinMode\_Get, [521](#)
  - GPIO\_PinMode\_GPIO, [521](#)
  - GPIO\_PinMode\_HiZ, [522](#)

- GPIO\_PinRead, [522](#)
- GPIO\_PinSet\_MaxCurrent, [522](#)
- GPIO\_PinSet\_PUPD, [522](#)
- GPIO\_PinSet\_Schmitt, [523](#)
- GPIO\_PinSet\_SpeedRaise, [523](#)
- GPIO\_PinToggle, [523](#)
- GPIO\_PinWrite, [523](#)
- GPIO\_PortIRQ\_Clear, [524](#)
- GPIO\_PortIRQ\_Disable, [524](#)
- GPIO\_PortIRQ\_Enable, [524](#)
- GPIO\_PortIRQ\_GetStatus, [524](#)
- GPIO\_PortMode\_Function, [525](#)
- GPIO\_PortMode\_GPIO, [525](#)
- GPIO\_PortMode\_HiZ, [525](#)
- GPIO\_PortRead, [526](#)
- GPIO\_PortSecureIRQ\_Clear, [526](#)
- GPIO\_PortSecureIRQ\_GetInfo1, [526](#)
- GPIO\_PortSecureIRQ\_GetInfo2, [526](#)
- GPIO\_PortSecureIRQ\_GetStatus, [527](#)
- GPIO\_PortSecureIRQ\_SetMask, [527](#)
- GPIO\_PortSet\_MaxCurrent, [527](#)
- GPIO\_PortSet\_NonsecureMask, [528](#)
- GPIO\_PortSet\_PUPD, [528](#)
- GPIO\_PortSet\_Schmitt, [528](#)
- GPIO\_PortSet\_SpeedRaise, [528](#)
- GPIO\_PortToggle, [529](#)
- GPIO\_PortWrite, [529](#)
- GPIO\_PULL\_DOWN, [519](#)
- GPIO\_PULL\_DOWN\_OD, [519](#)
- GPIO\_PULL\_NONE, [519](#)
- GPIO\_PULL\_NONE\_OD, [519](#)
- GPIO\_PULL\_UP, [519](#)
- GPIO\_PULL\_UP\_OD, [519](#)
- GPIO\_SEC\_INT\_PORT\_NONSEC\_MASK, [517](#)
- GPIO\_SEC\_INT\_SEC\_ACC\_MASK, [517](#)
- hal\_i2c.h
  - I2C\_SlaveDeinit, [535](#)
  - I2C\_SlaveEnable, [536](#)
  - I2C\_SlaveGetDefaultConfig, [536](#)
  - I2C\_SlaveInit, [536](#)
  - I2C\_SlaveReadBlocking, [537](#)
  - I2C\_SlaveSetAddress, [537](#)
  - I2C\_SlaveSetReceiveBuffer, [538](#)
  - I2C\_SlaveSetSendBuffer, [538](#)
  - I2C\_SlaveTransferAbort, [539](#)
  - I2C\_SlaveTransferCreateHandle, [539](#)
  - I2C\_SlaveTransferGetCount, [540](#)
  - I2C\_SlaveTransferHandleIRQ, [540](#)
  - I2C\_SlaveTransferNonBlocking, [541](#)
  - I2C\_SlaveWriteBlocking, [541](#)
- hal\_rwc.h
  - RWC\_ConvertDatetimeToSeconds, [589](#)
- handle
  - i2c\_slave\_transfer\_t, [424](#)
- HFI\_FREQUENCY
  - Драйвер модуля CLKCTR, [46](#)
- hour
  - \_rtc\_datetime, [406](#)
- hp\_end\_dummy
  - \_qspi\_xip\_config, [405](#)
- hp\_mode
  - \_qspi\_xip\_config, [405](#)
- hpen
  - \_qspi\_xip\_config, [405](#)
- hs\_en
  - sdmmc\_port\_cfg\_t, [462](#)
- hs\_mode
  - sdmmc\_card\_t, [458](#)
- I2C\_Abort\_10B\_Addr1\_Nack
  - Драйвер модуля I2C, [124](#)
- I2C\_Abort\_10B\_Addr2\_Nack
  - Драйвер модуля I2C, [124](#)
- I2C\_Abort\_7B\_Addr\_Nack
  - Драйвер модуля I2C, [124](#)
- I2C\_Abort\_Arbitr\_Lost
  - Драйвер модуля I2C, [125](#)
- i2c\_abort\_flags
  - Драйвер модуля I2C, [124](#)
- I2C\_Abort\_GenCall\_Nack
  - Драйвер модуля I2C, [124](#)
- I2C\_Abort\_GenCall\_Read
  - Драйвер модуля I2C, [124](#)
- I2C\_Abort\_HS\_RStart\_Dis
  - Драйвер модуля I2C, [125](#)
- I2C\_Abort\_HsCode\_Ack
  - Драйвер модуля I2C, [124](#)
- I2C\_Abort\_Master\_Dis
  - Драйвер модуля I2C, [125](#)
- I2C\_Abort\_MasterDetect
  - Драйвер модуля I2C, [125](#)
- I2C\_Abort\_Read\_RStart\_Dis
  - Драйвер модуля I2C, [125](#)
- I2C\_Abort\_RStart\_Dis
  - Драйвер модуля I2C, [125](#)
- I2C\_Abort\_RxFifo\_NotEmpty
  - Драйвер модуля I2C, [125](#)
- I2C\_Abort\_SlaveArbitr\_Lost
  - Драйвер модуля I2C, [125](#)
- I2C\_Abort\_SlaveDataCmd\_Error
  - Драйвер модуля I2C, [125](#)
- I2C\_Abort\_StartByte\_Ack
  - Драйвер модуля I2C, [124](#)
- I2C\_Abort\_Tx\_FlushCnt
  - Драйвер модуля I2C, [125](#)
- I2C\_Abort\_TxData\_Nack
  - Драйвер модуля I2C, [124](#)
- i2c\_addr\_size
  - i2c\_slave\_config\_t, [423](#)
- i2c\_addr\_size\_t
  - Драйвер модуля I2C, [125](#)
- I2C\_Address10Bit
  - Драйвер модуля I2C, [125](#)
- I2C\_Address7Bit
  - Драйвер модуля I2C, [125](#)
- I2C\_CFG\_MASK

- Драйвер модуля I2C, [122](#)
- I2C\_ClearAllInterrupts
  - Драйвер модуля I2C, [130](#)
- I2C\_ClearInterrupt
  - Драйвер модуля I2C, [130](#)
- i2c\_direction\_t
  - Драйвер модуля I2C, [125](#)
- I2C\_DisableInterrupts
  - Драйвер модуля I2C, [131](#)
- I2C\_DMADescriptorInitRX
  - Драйвер модуля I2C, [131](#)
- I2C\_DMADescriptorInitTX
  - Драйвер модуля I2C, [131](#)
- I2C\_Enable
  - Драйвер модуля I2C, [132](#)
- I2C\_EnableInterrupts
  - Драйвер модуля I2C, [133](#)
- I2C\_FastSpeedMode
  - Драйвер модуля I2C, [129](#)
- I2C\_GetEnabledInterrupts
  - Драйвер модуля I2C, [133](#)
- I2C\_GetInstance
  - Драйвер модуля I2C, [133](#)
- I2C\_HighSpeedMode
  - Драйвер модуля I2C, [129](#)
- i2c\_interrupt
  - Драйвер модуля I2C, [125](#)
- I2C\_IRQ\_Activity
  - Драйвер модуля I2C, [127](#)
- I2C\_IRQ\_GenCall
  - Драйвер модуля I2C, [127](#)
- I2C\_IRQ\_RdReq
  - Драйвер модуля I2C, [126](#)
- I2C\_IRQ\_RxDone
  - Драйвер модуля I2C, [126](#)
- I2C\_IRQ\_RxFull
  - Драйвер модуля I2C, [126](#)
- I2C\_IRQ\_RxOver
  - Драйвер модуля I2C, [126](#)
- I2C\_IRQ\_RxUnder
  - Драйвер модуля I2C, [126](#)
- I2C\_IRQ\_StartDet
  - Драйвер модуля I2C, [127](#)
- I2C\_IRQ\_StopDet
  - Драйвер модуля I2C, [127](#)
- I2C\_IRQ\_TxAbrt
  - Драйвер модуля I2C, [126](#)
- I2C\_IRQ\_TxEmpty
  - Драйвер модуля I2C, [126](#)
- I2C\_IRQ\_TxOver
  - Драйвер модуля I2C, [126](#)
- I2C\_IsEnable
  - Драйвер модуля I2C, [135](#)
- i2c\_master\_config\_t, [419](#)
  - baudrate\_bps, [420](#)
  - enable\_master, [420](#)
  - sda\_hold, [420](#)
  - sda\_setup, [420](#)
- i2c\_master\_transfer\_callback\_t
  - Драйвер модуля I2C, [123](#)
- i2c\_master\_transfer\_flags\_t
  - Драйвер модуля I2C, [127](#)
- i2c\_master\_transfer\_states
  - Драйвер модуля I2C, [127](#)
- i2c\_master\_transfer\_t, [421](#)
  - addr\_size, [421](#)
  - data, [421](#)
  - data\_size, [422](#)
  - direction, [422](#)
  - flags, [422](#)
  - slave\_address, [422](#)
  - subaddress, [422](#)
  - subaddress\_size, [422](#)
- I2C\_MASTER\_TRANSMIT\_IGNORE\_LAST\_NACK
  - Драйвер модуля I2C, [122](#)
- I2C\_MasterAddrSet
  - Драйвер модуля I2C, [135](#)
- I2C\_MasterDeinit
  - Драйвер модуля I2C, [135](#)
- I2C\_MasterGetBusActiveState
  - Драйвер модуля I2C, [136](#)
- I2C\_MasterGetDefaultConfig
  - Драйвер модуля I2C, [136](#)
- I2C\_MasterInit
  - Драйвер модуля I2C, [136](#)
- I2C\_MasterReadBlocking
  - Драйвер модуля I2C, [137](#)
- I2C\_MasterSetBaudRate
  - Драйвер модуля I2C, [138](#)
- I2C\_MasterTransferAbort
  - Драйвер модуля I2C, [139](#)
- I2C\_MasterTransferAbortDMA
  - Драйвер модуля I2C, [139](#)
- I2C\_MasterTransferBlocking
  - Драйвер модуля I2C, [139](#)
- I2C\_MasterTransferCreateHandle
  - Драйвер модуля I2C, [140](#)
- I2C\_MasterTransferCreateHandleDMA
  - Драйвер модуля I2C, [141](#)
- I2C\_MasterTransferDMA
  - Драйвер модуля I2C, [141](#)
- I2C\_MasterTransferGetCount
  - Драйвер модуля I2C, [142](#)
- I2C\_MasterTransferHandleIRQ
  - Драйвер модуля I2C, [142](#)
- I2C\_MasterTransferNonBlocking
  - Драйвер модуля I2C, [142](#)
- I2C\_MasterWriteBlocking
  - Драйвер модуля I2C, [143](#)
- I2C\_MTS\_Idle
  - Драйвер модуля I2C, [127](#)
- I2C\_MTS\_ReceiveData
  - Драйвер модуля I2C, [127](#)
- I2C\_MTS\_ReceiveDataBegin
  - Драйвер модуля I2C, [127](#)
- I2C\_MTS\_ReceiveLastData

- Драйвер модуля I2C, [127](#)
- I2C\_MTS\_Start
  - Драйвер модуля I2C, [127](#)
- I2C\_MTS\_Stop
  - Драйвер модуля I2C, [127](#)
- I2C\_MTS\_TransmitData
  - Драйвер модуля I2C, [127](#)
- I2C\_MTS\_TransmitSubaddr
  - Драйвер модуля I2C, [127](#)
- I2C\_MTS\_WaitForCompletion
  - Драйвер модуля I2C, [127](#)
- I2C\_Read
  - Драйвер модуля I2C, [125](#)
- I2C\_Reset
  - Драйвер модуля I2C, [144](#)
- I2C\_RETRY\_TIMES
  - Драйвер модуля I2C, [122](#)
- i2c\_slave\_config\_t, [423](#)
  - ack\_gen\_call, [423](#)
  - address, [423](#)
  - enable\_slave, [423](#)
  - i2c\_addr\_size, [423](#)
- i2c\_slave\_event\_transfer\_t
  - Драйвер модуля I2C, [127](#)
- i2c\_slave\_fsm\_t
  - Драйвер модуля I2C, [128](#)
- i2c\_slave\_transfer\_callback\_t
  - Драйвер модуля I2C, [124](#)
- i2c\_slave\_transfer\_t, [424](#)
  - completion\_status, [424](#)
  - event, [424](#)
  - event\_mask, [424](#)
  - handle, [424](#)
  - rx\_data, [424](#)
  - rx\_size, [425](#)
  - transferred\_count, [425](#)
  - tx\_data, [425](#)
  - tx\_size, [425](#)
- I2C\_SlaveDeinit
  - hal\_i2c.h, [535](#)
- I2C\_SlaveEnable
  - hal\_i2c.h, [536](#)
- I2C\_SlaveEvent\_Completion
  - Драйвер модуля I2C, [128](#)
- I2C\_SlaveEvent\_MatchAddrGen
  - Драйвер модуля I2C, [128](#)
- I2C\_SlaveEvent\_MatchAddrSlave
  - Драйвер модуля I2C, [128](#)
- I2C\_SlaveEvent\_Receive
  - Драйвер модуля I2C, [128](#)
- I2C\_SlaveEvent\_Transmit
  - Драйвер модуля I2C, [128](#)
- I2C\_SlaveEvents\_All
  - Драйвер модуля I2C, [128](#)
- I2C\_SlaveEvents\_Ordinary
  - Драйвер модуля I2C, [128](#)
- I2C\_SlaveFsm\_Idle
  - Драйвер модуля I2C, [128](#)
- I2C\_SlaveFsm\_Init
  - Драйвер модуля I2C, [128](#)
- I2C\_SlaveFsm\_Receive
  - Драйвер модуля I2C, [128](#)
- I2C\_SlaveFsm\_Stop
  - Драйвер модуля I2C, [128](#)
- I2C\_SlaveFsm\_Transmit
  - Драйвер модуля I2C, [128](#)
- I2C\_SlaveGetDefaultConfig
  - hal\_i2c.h, [536](#)
- I2C\_SlaveInit
  - hal\_i2c.h, [536](#)
- I2C\_SlaveReadBlocking
  - hal\_i2c.h, [537](#)
- I2C\_SlaveSetAddress
  - hal\_i2c.h, [537](#)
- I2C\_SlaveSetReceiveBuffer
  - hal\_i2c.h, [538](#)
- I2C\_SlaveSetSendBuffer
  - hal\_i2c.h, [538](#)
- I2C\_SlaveTransferAbort
  - hal\_i2c.h, [539](#)
- I2C\_SlaveTransferCreateHandle
  - hal\_i2c.h, [539](#)
- I2C\_SlaveTransferGetCount
  - hal\_i2c.h, [540](#)
- I2C\_SlaveTransferHandleIRQ
  - hal\_i2c.h, [540](#)
- I2C\_SlaveTransferNonBlocking
  - hal\_i2c.h, [541](#)
- I2C\_SlaveWriteBlocking
  - hal\_i2c.h, [541](#)
- i2c\_speed\_mode\_t
  - Драйвер модуля I2C, [128](#)
- I2C\_StandardSpeedMode
  - Драйвер модуля I2C, [129](#)
- I2C\_Stat\_Active
  - Драйвер модуля I2C, [129](#)
- I2C\_Stat\_Master\_Active
  - Драйвер модуля I2C, [129](#)
- I2C\_STAT\_MSTCODE\_IDLE
  - Драйвер модуля I2C, [122](#)
- I2C\_STAT\_MSTCODE\_NACKADR
  - Драйвер модуля I2C, [123](#)
- I2C\_STAT\_MSTCODE\_NACKDAT
  - Драйвер модуля I2C, [123](#)
- I2C\_STAT\_MSTCODE\_RXREADY
  - Драйвер модуля I2C, [123](#)
- I2C\_STAT\_MSTCODE\_TXREADY
  - Драйвер модуля I2C, [123](#)
- I2C\_Stat\_RxFifo\_Full
  - Драйвер модуля I2C, [129](#)
- I2C\_Stat\_RxFifo\_NotEmpty
  - Драйвер модуля I2C, [129](#)
- I2C\_Stat\_Slave\_Active
  - Драйвер модуля I2C, [129](#)
- I2C\_STAT\_SLVST\_ADDR
  - Драйвер модуля I2C, [123](#)

- I2C\_STAT\_SLVST\_RX  
Драйвер модуля I2C, [123](#)
- I2C\_STAT\_SLVST\_TX  
Драйвер модуля I2C, [123](#)
- I2C\_Stat\_TxFifo\_Empty  
Драйвер модуля I2C, [129](#)
- I2C\_Stat\_TxFifo\_NotFull  
Драйвер модуля I2C, [129](#)
- I2C\_Status\_AddrNack  
Драйвер модуля I2C, [129](#)
- I2C\_Status\_ArbitrationLost  
Драйвер модуля I2C, [129](#)
- I2C\_Status\_BreakTransfer  
Драйвер модуля I2C, [129](#)
- I2C\_Status\_Busy  
Драйвер модуля I2C, [129](#)
- I2C\_Status\_DmaRequestFail  
Драйвер модуля I2C, [129](#)
- i2c\_status\_flags  
Драйвер модуля I2C, [129](#)
- I2C\_Status\_HsCodeError  
Драйвер модуля I2C, [130](#)
- I2C\_Status\_HwError  
Драйвер модуля I2C, [129](#)
- I2C\_Status\_Idle  
Драйвер модуля I2C, [129](#)
- I2C\_Status\_InvalidParameter  
Драйвер модуля I2C, [129](#)
- I2C\_Status\_Nack  
Драйвер модуля I2C, [129](#)
- I2C\_Status\_NoTransferInProgress  
Драйвер модуля I2C, [129](#)
- I2C\_Status\_Ok  
Драйвер модуля I2C, [129](#)
- I2C\_Status\_SetStartError  
Драйвер модуля I2C, [129](#)
- i2c\_status\_t  
Драйвер модуля I2C, [129](#)
- I2C\_Status\_Timeout  
Драйвер модуля I2C, [129](#)
- I2C\_Status\_UnDef  
Драйвер модуля I2C, [130](#)
- I2C\_Status\_UnexpectedState  
Драйвер модуля I2C, [130](#)
- I2C\_Status\_UnexpectedState1  
Драйвер модуля I2C, [130](#)
- I2C\_Status\_UnexpectedState2  
Драйвер модуля I2C, [130](#)
- I2C\_Status\_UnexpectedState3  
Драйвер модуля I2C, [130](#)
- I2C\_Status\_UnexpectedState4  
Драйвер модуля I2C, [130](#)
- I2C\_Status\_UnexpectedState5  
Драйвер модуля I2C, [130](#)
- I2C\_Status\_UnexpectedState6  
Драйвер модуля I2C, [130](#)
- I2C\_Status\_UnexpectedState7  
Драйвер модуля I2C, [130](#)
- I2C\_Status\_UnexpectedState8  
Драйвер модуля I2C, [130](#)
- I2C\_Status\_UnexpectedState9  
Драйвер модуля I2C, [130](#)
- I2C\_Status\_UserError  
Драйвер модуля I2C, [129](#)
- I2C\_TransferDataFlag  
Драйвер модуля I2C, [127](#)
- I2C\_TransferReStartFlag  
Драйвер модуля I2C, [127](#)
- I2C\_TransferStartFlag  
Драйвер модуля I2C, [127](#)
- I2C\_TransferStopFlag  
Драйвер модуля I2C, [127](#)
- I2C\_UndefinedSpeedMode  
Драйвер модуля I2C, [129](#)
- I2C\_Write  
Драйвер модуля I2C, [125](#)
- I2S\_AbortTx  
Драйвер модуля I2S, [150](#)
- I2S\_ClearDataOverrunFlag  
Драйвер модуля I2S, [150](#)
- I2S\_Deinit  
Драйвер модуля I2S, [150](#)
- I2S\_DisableInterrupt  
Драйвер модуля I2S, [150](#)
- I2S\_DisableInterruptMask  
Драйвер модуля I2S, [151](#)
- I2S\_DisableTx  
Драйвер модуля I2S, [151](#)
- I2S\_EnableInterrupt  
Драйвер модуля I2S, [151](#)
- I2S\_EnableInterruptMask  
Драйвер модуля I2S, [151](#)
- I2S\_EnableTx  
Драйвер модуля I2S, [152](#)
- I2S\_FlagTxFifoEmpty  
Драйвер модуля I2S, [148](#)
- I2S\_FlagTxFifoOverrun  
Драйвер модуля I2S, [148](#)
- I2S\_GetDefaultConfig  
Драйвер модуля I2S, [152](#)
- I2S\_GetEnabledInterruptMask  
Драйвер модуля I2S, [152](#)
- I2S\_GetInterruptStatus  
Драйвер модуля I2S, [152](#)
- I2S\_GetInterruptStatusMask  
Драйвер модуля I2S, [153](#)
- I2S\_Init  
Драйвер модуля I2S, [153](#)
- I2S\_InterruptTriggerLevel\_1  
Драйвер модуля I2S, [148](#)
- I2S\_InterruptTriggerLevel\_10  
Драйвер модуля I2S, [148](#)
- I2S\_InterruptTriggerLevel\_11  
Драйвер модуля I2S, [148](#)
- I2S\_InterruptTriggerLevel\_12  
Драйвер модуля I2S, [148](#)

- I2S\_InterruptTriggerLevel\_13  
Драйвер модуля I2S, [148](#)
- I2S\_InterruptTriggerLevel\_14  
Драйвер модуля I2S, [148](#)
- I2S\_InterruptTriggerLevel\_15  
Драйвер модуля I2S, [148](#)
- I2S\_InterruptTriggerLevel\_16  
Драйвер модуля I2S, [148](#)
- I2S\_InterruptTriggerLevel\_2  
Драйвер модуля I2S, [148](#)
- I2S\_InterruptTriggerLevel\_3  
Драйвер модуля I2S, [148](#)
- I2S\_InterruptTriggerLevel\_4  
Драйвер модуля I2S, [148](#)
- I2S\_InterruptTriggerLevel\_5  
Драйвер модуля I2S, [148](#)
- I2S\_InterruptTriggerLevel\_6  
Драйвер модуля I2S, [148](#)
- I2S\_InterruptTriggerLevel\_7  
Драйвер модуля I2S, [148](#)
- I2S\_InterruptTriggerLevel\_8  
Драйвер модуля I2S, [148](#)
- I2S\_InterruptTriggerLevel\_9  
Драйвер модуля I2S, [148](#)
- I2S\_IsInterruptEnabled  
Драйвер модуля I2S, [153](#)
- I2S\_NoSclkGating  
Драйвер модуля I2S, [149](#)
- I2S\_Resolution\_12  
Драйвер модуля I2S, [149](#)
- I2S\_Resolution\_16  
Драйвер модуля I2S, [149](#)
- I2S\_Resolution\_20  
Драйвер модуля I2S, [149](#)
- I2S\_Resolution\_24  
Драйвер модуля I2S, [149](#)
- I2S\_Resolution\_32  
Драйвер модуля I2S, [149](#)
- I2S\_ResolutionDefault  
Драйвер модуля I2S, [149](#)
- i2s\_sclk\_gating\_t  
Драйвер модуля I2S, [147](#)
- I2S\_SclkCycles\_16  
Драйвер модуля I2S, [149](#)
- I2S\_SclkCycles\_24  
Драйвер модуля I2S, [149](#)
- I2S\_SclkCycles\_32  
Драйвер модуля I2S, [149](#)
- I2S\_SclkGatingCycles\_12  
Драйвер модуля I2S, [149](#)
- I2S\_SclkGatingCycles\_16  
Драйвер модуля I2S, [149](#)
- I2S\_SclkGatingCycles\_20  
Драйвер модуля I2S, [149](#)
- I2S\_SclkGatingCycles\_24  
Драйвер модуля I2S, [149](#)
- I2S\_Status\_Fail  
Драйвер модуля I2S, [150](#)
- I2S\_Status\_InvalidArgument  
Драйвер модуля I2S, [150](#)
- I2S\_Status\_Ok  
Драйвер модуля I2S, [150](#)
- I2S\_Status\_TxBusy  
Драйвер модуля I2S, [150](#)
- I2S\_Status\_UnsupportedBitRate  
Драйвер модуля I2S, [150](#)
- i2s\_transfer\_callback\_t  
Драйвер модуля I2S, [147](#)
- I2S\_TransferAbort  
Драйвер модуля I2S, [154](#)
- I2S\_TransferCreateHandle  
Драйвер модуля I2S, [154](#)
- I2S\_TransferHandleIRQ  
Драйвер модуля I2S, [154](#)
- I2S\_TransferNonBlocking  
Драйвер модуля I2S, [155](#)
- I2S\_WriteLeftFifo  
Драйвер модуля I2S, [155](#)
- I2S\_WriteRightFifo  
Драйвер модуля I2S, [155](#)
- id  
\_can\_frame\_filter, [360](#)  
\_can\_rx\_buffer\_frame, [365](#)  
\_can\_tx\_buffer\_frame, [371](#)
- ide  
\_can\_rx\_buffer\_frame, [365](#)  
\_can\_tx\_buffer\_frame, [371](#)
- inhibit\_din  
\_qspi\_config, [400](#)
- inhibit\_dout  
\_qspi\_config, [400](#)
- inputs\_mask\_mult  
pwm\_channel\_config, [439](#)
- inputs\_mask\_one  
pwm\_channel\_config, [439](#)
- instance  
spi\_handle, [468](#)
- int\_ctrl  
dualtimer\_hardware\_config, [419](#)
- int\_en  
\_dma\_multiblock\_config, [381](#)  
\_pwm\_handle, [397](#)  
dma\_channel\_ctl\_cfg, [417](#)
- int\_freq  
\_pwm\_handle, [398](#)
- int\_source  
\_pwm\_handle, [398](#)
- inten  
wdt\_config, [481](#)
- intermediate\_len  
\_qspi\_nor\_handle, [403](#)
- interrupt\_control  
vtu\_config, [480](#)
- interrupt\_enable  
timer\_hardware\_config, [472](#)
- interrupt\_level



- `_i2s_config`, 388
- IOIM\_ClearIRQHandler
  - Драйвер менеджера прерываний устройств и DMA каналов, 157
- IOIM\_ClearIRQHandler\_DMA
  - Драйвер менеджера прерываний устройств и DMA каналов, 157
- IOIM\_GetIRQNumber
  - Драйвер менеджера прерываний устройств и DMA каналов, 158
- IOIM\_NA\_IRQ\_NUM
  - Драйвер менеджера прерываний устройств и DMA каналов, 157
- IOIM\_SetIRQHandler
  - Драйвер менеджера прерываний устройств и DMA каналов, 158
- IOIM\_SetIRQHandler\_DMA
  - Драйвер менеджера прерываний устройств и DMA каналов, 158
- IOIM\_Status\_NullHandler
  - Драйвер менеджера прерываний устройств и DMA каналов, 157
- IOIM\_Status\_Ok
  - Драйвер менеджера прерываний устройств и DMA каналов, 157
- ioim\_status\_t
  - Драйвер менеджера прерываний устройств и DMA каналов, 157
- IOIM\_Status\_UnknownBase
  - Драйвер менеджера прерываний устройств и DMA каналов, 157
- irq\_num
  - `_i2c_slave_handle`, 387
- is\_dst\_periph
  - `_dma_channel_config`, 373
- is\_quad\_need\_enable
  - `_qspi_nor_config`, 402
- is\_src\_periph
  - `_dma_channel_config`, 373
- jtm\_callback\_t
  - Драйвер модуля JTM, 160
- jtm\_config\_t, 425
  - `tcal`, 426
  - `wcal`, 426
  - `wtcalconf`, 426
  - `wtconf`, 426
- JTM\_CreateHandle
  - Драйвер модуля JTM, 161
- JTM\_GetParameterValue
  - Драйвер модуля JTM, 161
- JTM\_GetParameterValueNonBlocking
  - Драйвер модуля JTM, 162
- JTM\_Init
  - Драйвер модуля JTM, 162
- jtm\_parameter\_t
  - Драйвер модуля JTM, 160
- JTM\_Status\_BadParameter
  - Драйвер модуля JTM, 161
- JTM\_Status\_Busy
  - Драйвер модуля JTM, 161
- IO JTM\_Status\_Fail
  - Драйвер модуля JTM, 161
- JTM\_Status\_Ok
  - Драйвер модуля JTM, 161
- IO jtm\_status\_t
  - Драйвер модуля JTM, 161
- IO JTM\_Temperature
  - Драйвер модуля JTM, 161
- JTM\_Vcasn
  - Драйвер модуля JTM, 161
- IO JTM\_Vcore
  - Драйвер модуля JTM, 161
- IO koer
  - `_can_rx_buffer_frame`, 365
- IO LBBLEndian
  - Общие определения для библиотеки HAL, 83
- IO ldamode
  - `pwm_channel_config`, 439
- ldbmde
  - `pwm_channel_config`, 439
- IO ldcswrf
  - `pwm_channel_config`, 439
- IO le32
  - `_qspi_xip_config`, 406
- left\_samples
  - `_i2s_handle`, 389
  - `_i2s_transfer`, 390
- IO lfe\_bypass
  - `rcw_config`, 445
  - `rcw_trim_reg`, 452
- LLP
  - `_dma_descriptor`, 378
- llp\_dst\_en
  - `dma_channel_ctl_cfg`, 417
- LLP\_HI
  - `_dma_channel_reg`, 376
- LLP\_LO
  - `_dma_channel_reg`, 376
- llp\_src\_en
  - `dma_channel_ctl_cfg`, 417
- load
  - `dualtimer_hardware_config`, 419
  - `wdt_config`, 481
- loadprd
  - `pwm_channel_config`, 440
- lock
  - `clkctr_pll_cfg`, 415
  - `sdmcc_card_t`, 458
- loopback\_enable
  - `spi_config_t`, 465
- man
  - `clkctr_pll_cfg`, 415
- mask
  - `_can_frame_filter`, 360

- master
  - spi\_config\_t, 465
- MAX
  - Общие определения для библиотеки HAL, 81
- max\_chip\_erase\_time
  - \_nor\_handle, 395
- max\_current
  - sdmmc\_cfg\_pin\_map, 461
- max\_page\_program\_time
  - \_nor\_handle, 395
- max\_sector\_erase\_time
  - \_nor\_handle, 395
- mem\_control\_config
  - \_nor\_config, 393
- memory\_size\_bytes
  - \_nor\_handle, 395
- MHU\_CPU0\_ClearInt
  - Драйвер модуля программных прерываний MHU, 163
- MHU\_CPU0\_SetInt
  - Драйвер модуля программных прерываний MHU, 164
- MHU\_CPU0\_StatInt
  - Драйвер модуля программных прерываний MHU, 164
- MHU\_CPU1\_ClearInt
  - Драйвер модуля программных прерываний MHU, 164
- MHU\_CPU1\_SetInt
  - Драйвер модуля программных прерываний MHU, 164
- MHU\_CPU1\_StatInt
  - Драйвер модуля программных прерываний MHU, 165
- microwire\_busy\_ready\_check\_t
  - Драйвер модуля SPI, 274
- microwire\_cfg
  - spi\_config\_t, 465
- microwire\_ctrlword\_len\_t
  - Драйвер модуля SPI, 274
- microwire\_single\_serial\_t
  - Драйвер модуля SPI, 275
- microwire\_tx\_rx\_t
  - Драйвер модуля SPI, 275
- MIN
  - Общие определения для библиотеки HAL, 82
- minute
  - \_rtc\_datetime, 406
- mode
  - \_qspi\_config, 400
  - dualtimer\_hardware\_config, 419
  - spi\_handle, 468
  - vtu\_config, 480
- month
  - \_rtc\_datetime, 407
- motorola\_cfg
  - spi\_config\_t, 465
- msb
  - \_qspi\_config, 401
- nb\_filters\_used
  - \_can\_frame\_filter\_config, 360
- nb\_frames
  - \_can\_rx\_transfer, 366
  - \_can\_tx\_transfer, 372
- nb\_samples
  - \_i2s\_handle, 389
  - \_i2s\_transfer, 390
- need\_1V8en
  - sdmmc\_card\_t, 458
- next\_desc
  - \_dma\_channel\_config, 373
- nf\_man
  - clkctr\_pll\_cfg, 415
- NOR\_CmdEraseChip
  - Драйвер модуля QSPI, 214
- NOR\_CmdEraseChipNor
  - Драйвер модуля QSPI, 214
- NOR\_CmdErasePage
  - Драйвер модуля QSPI, 214
- NOR\_CmdEraseSector
  - Драйвер модуля QSPI, 214
- NOR\_CmdEraseSector32KB
  - Драйвер модуля QSPI, 214
- NOR\_CmdEraseSector4KB
  - Драйвер модуля QSPI, 214
- NOR\_CmdEraseSector4KBA32
  - Драйвер модуля QSPI, 214
- NOR\_CmdEraseSectorA32
  - Драйвер модуля QSPI, 214
- NOR\_CmdInvalid
  - Драйвер модуля QSPI, 213
- NOR\_CmdReadMemory
  - Драйвер модуля QSPI, 214
- NOR\_CmdReadMemoryA32
  - Драйвер модуля QSPI, 214
- NOR\_CmdReadMemorySDR\_1\_1\_1
  - Драйвер модуля QSPI, 214
- NOR\_CmdReadMemorySDR\_1\_1\_2
  - Драйвер модуля QSPI, 214
- NOR\_CmdReadMemorySDR\_1\_1\_4
  - Драйвер модуля QSPI, 214
- NOR\_CmdReadMemorySDR\_1\_1\_4\_A32
  - Драйвер модуля QSPI, 214
- NOR\_CmdReadMemorySDR\_1\_2\_2
  - Драйвер модуля QSPI, 214
- NOR\_CmdReadMemorySDR\_1\_4\_4
  - Драйвер модуля QSPI, 214
- NOR\_CmdReadMemorySDR\_1\_4\_4\_A32
  - Драйвер модуля QSPI, 214
- NOR\_CmdReadSecStatus\_35
  - Драйвер модуля QSPI, 213
- NOR\_CmdReadSecStatus\_3F
  - Драйвер модуля QSPI, 213
- NOR\_CmdReadStatus
  - Драйвер модуля QSPI, 213
- NOR\_CmdWriteDisable



- Драйвер модуля QSPI, [213](#)
- NOR\_CmdWriteEnable
  - Драйвер модуля QSPI, [213](#)
- NOR\_CmdWriteMemory
  - Драйвер модуля QSPI, [213](#)
- NOR\_CmdWriteMemoryA32
  - Драйвер модуля QSPI, [213](#)
- NOR\_CmdWriteSecStatus\_31
  - Драйвер модуля QSPI, [213](#)
- NOR\_CmdWriteSecStatus\_3E
  - Драйвер модуля QSPI, [213](#)
- NOR\_CmdWriteStatus
  - Драйвер модуля QSPI, [213](#)
- NOR\_FlashEraseBlock
  - Драйвер модуля QSPI, [215](#)
- NOR\_FlashEraseChip
  - Драйвер модуля QSPI, [215](#)
- NOR\_FlashPageProgram
  - Драйвер модуля QSPI, [215](#)
- NOR\_FlashProgramBlock
  - Драйвер модуля QSPI, [216](#)
- NOR\_FlashRead
  - Драйвер модуля QSPI, [216](#)
- NOR\_FlashReadXIP
  - Драйвер модуля QSPI, [217](#)
- NOR\_QuadModeNotConfig
  - Драйвер модуля QSPI, [212](#)
- NOR\_QuadModeStatusReg1\_Bit6
  - Драйвер модуля QSPI, [212](#)
- NOR\_QuadModeStatusReg2\_Bit1
  - Драйвер модуля QSPI, [212](#)
- NOR\_QuadModeStatusReg2\_Bit1\_0x31
  - Драйвер модуля QSPI, [212](#)
- NOR\_QuadModeStatusReg2\_Bit7
  - Драйвер модуля QSPI, [212](#)
- NOR\_Status\_Fail
  - Драйвер модуля QSPI, [214](#)
- NOR\_Status\_InvalidArgument
  - Драйвер модуля QSPI, [214](#)
- NOR\_Status\_Success
  - Драйвер модуля QSPI, [214](#)
- nor\_status\_t
  - Драйвер модуля QSPI, [214](#)
- NOR\_Status\_Timeout
  - Драйвер модуля QSPI, [214](#)
- nr\_man
  - clkctr\_pll\_cfg, [415](#)
- od\_man
  - clkctr\_pll\_cfg, [415](#)
- osc\_sel
  - rcw\_config, [445](#)
  - rcw\_config\_reg, [448](#)
- out
  - \_pwm\_handle, [398](#)
- page\_size\_bytes
  - \_nor\_handle, [395](#)
- page\_write\_memory\_cmd
  - \_nor\_command\_set, [392](#)
- parameter
  - \_jtm\_handle, [391](#)
- parity\_manual
  - uart\_config, [474](#)
- parity\_mode
  - uart\_config, [475](#)
- period
  - pwm\_channel\_config, [440](#)
  - vtu\_config, [480](#)
- period\_us
  - \_pwm\_handle, [398](#)
- pin\_map
  - sdmmc\_port\_cfg\_t, [462](#)
- pl
  - rcw\_config, [445](#)
  - rcw\_config\_reg, [448](#)
- PLL\_MAX\_MULTIPLIER
  - Драйвер модуля CLKCTR, [46](#)
- PLL\_MIN\_MULTIPLIER
  - Драйвер модуля CLKCTR, [46](#)
- power\_callback\_t
  - Драйвер модуля POWER, [167](#)
- POWER\_ClearInterrupts
  - Драйвер модуля POWER, [171](#)
- power\_config, [426](#)
  - dcdc\_enable, [427](#)
  - flash\_low\_voltage\_read\_enable, [427](#)
  - run\_configuration, [427](#)
  - standby\_configuration, [427](#)
  - trim\_configuration, [427](#)
  - vlevel0, [427](#)
  - vlevel1, [427](#)
  - vlevel2, [427](#)
- POWER\_CreateHandle
  - Драйвер модуля POWER, [171](#)
- power\_dcdc\_mode
  - Драйвер модуля POWER, [167](#)
- power\_dcdc\_threshold
  - Драйвер модуля POWER, [168](#)
- power\_dcdc\_vlevel
  - Драйвер модуля POWER, [168](#)
- power\_dcdc\_vlevel\_value
  - Драйвер модуля POWER, [168](#)
- POWER\_DcdcModeAuto
  - Драйвер модуля POWER, [168](#)
- POWER\_DcdcModePfm
  - Драйвер модуля POWER, [168](#)
- POWER\_DcdcModePwm
  - Драйвер модуля POWER, [168](#)
- POWER\_DcdcModePwmFccm
  - Драйвер модуля POWER, [168](#)
- POWER\_DcdcThreshold\_0\_60V
  - Драйвер модуля POWER, [168](#)
- POWER\_DcdcThreshold\_0\_62V
  - Драйвер модуля POWER, [168](#)
- POWER\_DcdcThreshold\_0\_64V
  - Драйвер модуля POWER, [168](#)

- POWER\_DcdcThreshold\_0\_66V  
Драйвер модуля POWER, 168
- POWER\_DcdcThreshold\_0\_68V  
Драйвер модуля POWER, 168
- POWER\_DcdcThreshold\_0\_70V  
Драйвер модуля POWER, 168
- POWER\_DcdcThreshold\_0\_72V  
Драйвер модуля POWER, 168
- POWER\_DcdcThreshold\_0\_74V  
Драйвер модуля POWER, 168
- POWER\_DcdcThreshold\_0\_76V  
Драйвер модуля POWER, 168
- POWER\_DcdcThreshold\_0\_78V  
Драйвер модуля POWER, 168
- POWER\_DcdcThreshold\_0\_80V  
Драйвер модуля POWER, 168
- POWER\_DcdcThreshold\_0\_82V  
Драйвер модуля POWER, 168
- POWER\_DcdcThreshold\_0\_84V  
Драйвер модуля POWER, 168
- POWER\_DcdcThreshold\_0\_86V  
Драйвер модуля POWER, 168
- POWER\_DcdcThreshold\_0\_88V  
Драйвер модуля POWER, 168
- POWER\_DcdcThreshold\_0\_90V  
Драйвер модуля POWER, 168
- POWER\_DcdcVLevel0  
Драйвер модуля POWER, 168
- POWER\_DcdcVLevel1  
Драйвер модуля POWER, 168
- POWER\_DcdcVLevel2  
Драйвер модуля POWER, 168
- POWER\_DcdcVLevel\_0\_85V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_0\_86V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_0\_87V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_0\_88V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_0\_89V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_0\_90V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_0\_91V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_0\_92V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_0\_93V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_0\_94V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_0\_95V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_0\_96V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_0\_97V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_0\_98V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_0\_99V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_1\_00V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_1\_01V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_1\_02V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_1\_03V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_1\_04V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_1\_05V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_1\_06V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_1\_07V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_1\_08V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_1\_09V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_1\_10V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_1\_11V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_1\_12V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_1\_13V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_1\_14V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_1\_15V  
Драйвер модуля POWER, 169
- POWER\_DcdcVLevel\_1\_16V  
Драйвер модуля POWER, 169
- POWER\_DisableInterrupt  
Драйвер модуля POWER, 171
- POWER\_DisableInterruptMask  
Драйвер модуля POWER, 172
- power\_eco\_mode  
Драйвер модуля POWER, 169
- POWER\_EcoDcdc  
Драйвер модуля POWER, 170
- POWER\_EcoDcdcAndApc  
Драйвер модуля POWER, 170
- POWER\_EcoOff  
Драйвер модуля POWER, 170
- POWER\_EcoReserved  
Драйвер модуля POWER, 170
- POWER\_EnableInterrupt  
Драйвер модуля POWER, 172
- POWER\_EnableInterruptMask  
Драйвер модуля POWER, 172
- power\_flash\_mode  
Драйвер модуля POWER, 170

- POWER\_FlashModeNormal
  - Драйвер модуля POWER, [170](#)
- POWER\_FlashModePowerDown
  - Драйвер модуля POWER, [170](#)
- POWER\_FlashModeSleep
  - Драйвер модуля POWER, [170](#)
- POWER\_GetCurrentConfig
  - Драйвер модуля POWER, [172](#)
- POWER\_GetEnabledInterruptMask
  - Драйвер модуля POWER, [173](#)
- POWER\_GetInterruptStatus
  - Драйвер модуля POWER, [173](#)
- POWER\_GetInterruptStatusMask
  - Драйвер модуля POWER, [173](#)
- POWER\_GetStatus
  - Драйвер модуля POWER, [174](#)
- power\_handle, [428](#)
  - callback, [428](#)
  - user\_data, [428](#)
- power\_interrupt
  - Драйвер модуля POWER, [170](#)
- POWER\_IsInterruptEnabled
  - Драйвер модуля POWER, [174](#)
- power\_mode\_config, [429](#)
  - apc\_eco\_threshold, [429](#)
  - apc\_low\_clk\_enable, [429](#)
  - dcdc\_ccm\_enable, [429](#)
  - dcdc\_level, [429](#)
  - dcdc\_low\_consumption\_enable, [429](#)
  - dcdc\_mode, [430](#)
  - dcdc\_sink\_enable, [430](#)
  - dcdc\_swdrv, [430](#)
  - eco\_mode, [430](#)
  - flash\_power\_mode, [430](#)
- POWER\_SetConfig
  - Драйвер модуля POWER, [174](#)
- POWER\_Shutdown
  - Драйвер модуля POWER, [175](#)
- POWER\_Standby
  - Драйвер модуля POWER, [175](#)
- POWER\_StartTestMode
  - Драйвер модуля POWER, [175](#)
- power\_state, [430](#)
  - dcdc\_is\_ready, [431](#)
  - flash\_low\_voltage\_read\_enabled, [431](#)
  - flash\_power\_mode, [431](#)
  - vdda\_is\_lower\_threshold, [431](#)
- power\_status
  - Драйвер модуля POWER, [170](#)
- POWER\_Status\_Fail
  - Драйвер модуля POWER, [170](#)
- POWER\_Status\_Ok
  - Драйвер модуля POWER, [170](#)
- POWER\_StopTestMode
  - Драйвер модуля POWER, [176](#)
- power\_test\_block
  - Драйвер модуля POWER, [170](#)
- POWER\_TestBlockApc
  - Драйвер модуля POWER, [171](#)
- POWER\_TestBlockDcdc
  - Драйвер модуля POWER, [171](#)
- POWER\_TestBlockJtm
  - Драйвер модуля POWER, [171](#)
- POWER\_TestBlockRwc
  - Драйвер модуля POWER, [171](#)
- power\_trim\_config, [431](#)
  - apc\_force\_trim, [432](#)
  - apc\_vref\_it, [432](#)
  - apc\_vref\_tt, [432](#)
  - apc\_vref\_vt, [432](#)
  - dcdc\_imax, [432](#)
  - dcdc\_imin, [432](#)
  - dcdc\_trimlc, [433](#)
- POWER\_VmonFalling
  - Драйвер модуля POWER, [170](#)
- POWER\_VmonRising
  - Драйвер модуля POWER, [170](#)
- ppu\_add\_event\_name
  - Драйвер модуля PPU, [178](#)
- PPU\_AddEventNameAll
  - Драйвер модуля PPU, [178](#)
- PPU\_ClrIRQMask
  - Драйвер модуля PPU, [182](#)
- PPU\_ClrIRQStatus
  - Драйвер модуля PPU, [182](#)
- ppu\_config, [433](#)
  - reqForCPU, [433](#)
- ppu\_domain\_index
  - Драйвер модуля PPU, [178](#)
- PPU\_DomainCPU0
  - Драйвер модуля PPU, [179](#)
- PPU\_DomainCPU1
  - Драйвер модуля PPU, [179](#)
- PPU\_DomainCRYPTO
  - Драйвер модуля PPU, [179](#)
- PPU\_DomainDEBUG
  - Драйвер модуля PPU, [179](#)
- PPU\_DomainGMS
  - Драйвер модуля PPU, [179](#)
- PPU\_DomainGNSS
  - Драйвер модуля PPU, [179](#)
- PPU\_DomainMax
  - Драйвер модуля PPU, [179](#)
- PPU\_DomainSRAM0
  - Драйвер модуля PPU, [179](#)
- PPU\_DomainSRAM1
  - Драйвер модуля PPU, [179](#)
- PPU\_DomainSRAM2
  - Драйвер модуля PPU, [179](#)
- PPU\_DomainSRAM3
  - Драйвер модуля PPU, [179](#)
- PPU\_DomainSYS
  - Драйвер модуля PPU, [179](#)
- PPU\_DYN\_FULL\_RET\_SPT
  - Драйвер модуля PPU, [179](#)
- PPU\_DYN\_FUNC\_RET\_SPT
  - Драйвер модуля PPU, [179](#)

- Драйвер модуля PPU, [179](#)
- PPU\_DYN\_LGC\_RET\_SPT
  - Драйвер модуля PPU, [179](#)
- PPU\_DYN\_MEM\_OFF\_SPT
  - Драйвер модуля PPU, [179](#)
- PPU\_DYN\_MEM\_RET\_EMU\_SPT
  - Драйвер модуля PPU, [179](#)
- PPU\_DYN\_MEM\_RET\_SPT
  - Драйвер модуля PPU, [180](#)
- PPU\_DYN\_OFF\_EMU\_SPT
  - Драйвер модуля PPU, [180](#)
- PPU\_DYN\_OFF\_SPT
  - Драйвер модуля PPU, [180](#)
- PPU\_DYN\_ON\_SPT
  - Драйвер модуля PPU, [179](#)
- PPU\_DYN\_WRM\_RST\_SPT
  - Драйвер модуля PPU, [179](#)
- PPU\_DynAccept
  - Драйвер модуля PPU, [178](#)
- PPU\_DynDeny
  - Драйвер модуля PPU, [178](#)
- PPU\_EmuAccept
  - Драйвер модуля PPU, [179](#)
- PPU\_EmuDeny
  - Драйвер модуля PPU, [179](#)
- ppu\_event\_name
  - Драйвер модуля PPU, [179](#)
- PPU\_EventNameAll
  - Драйвер модуля PPU, [179](#)
- PPU\_FULL\_RET\_RAM\_REG
  - Драйвер модуля PPU, [180](#)
- PPU\_FUNC\_RET\_RAM\_REG
  - Драйвер модуля PPU, [180](#)
- PPU\_GetIRQMask
  - Драйвер модуля PPU, [183](#)
- PPU\_GetIRQStatus
  - Драйвер модуля PPU, [183](#)
- PPU\_GetLastAPIStatus
  - Драйвер модуля PPU, [183](#)
- PPU\_GetPDxSenseFromPDy
  - Драйвер модуля PPU, [184](#)
- PPU\_GetPowerState
  - Драйвер модуля PPU, [184](#)
- PPU\_Init
  - Драйвер модуля PPU, [184](#)
- PPU\_LOCK\_SPT
  - Драйвер модуля PPU, [180](#)
- PPU\_Locked
  - Драйвер модуля PPU, [179](#)
- PPU\_MEM\_RET\_RAM\_REG
  - Драйвер модуля PPU, [180](#)
- PPU\_OFF\_MEM\_RET\_TRANS
  - Драйвер модуля PPU, [180](#)
- PPU\_OP\_ACTIVE
  - Драйвер модуля PPU, [180](#)
- ppu\_opportunities\_idr0
  - Драйвер модуля PPU, [179](#)
- ppu\_opportunities\_idr1
  - Драйвер модуля PPU, [180](#)
- ppu\_power\_mode
  - Драйвер модуля PPU, [180](#)
- PPU\_PowerModeDbgRecov
  - Драйвер модуля PPU, [181](#)
- PPU\_PowerModeFullRet
  - Драйвер модуля PPU, [181](#)
- PPU\_PowerModeFuncRet
  - Драйвер модуля PPU, [181](#)
- PPU\_PowerModeLogicRet
  - Драйвер модуля PPU, [181](#)
- PPU\_PowerModeMax
  - Драйвер модуля PPU, [181](#)
- PPU\_PowerModeMemOff
  - Драйвер модуля PPU, [181](#)
- PPU\_PowerModeMemRet
  - Драйвер модуля PPU, [181](#)
- PPU\_PowerModeMemRetEmu
  - Драйвер модуля PPU, [181](#)
- PPU\_PowerModeOff
  - Драйвер модуля PPU, [181](#)
- PPU\_PowerModeOffEmu
  - Драйвер модуля PPU, [181](#)
- PPU\_PowerModeOn
  - Драйвер модуля PPU, [181](#)
- PPU\_PowerModeWarmRst
  - Драйвер модуля PPU, [181](#)
- PPU\_PWR\_MODE\_ENTRY\_DEL\_SPT
  - Драйвер модуля PPU, [180](#)
- ppu\_sense\_index
  - Драйвер модуля PPU, [181](#)
- PPU\_SenseCPU0
  - Драйвер модуля PPU, [181](#)
- PPU\_SenseCPU1
  - Драйвер модуля PPU, [181](#)
- PPU\_SenseCRYPTO
  - Драйвер модуля PPU, [181](#)
- PPU\_SenseGMS
  - Драйвер модуля PPU, [181](#)
- PPU\_SenseGNSS
  - Драйвер модуля PPU, [181](#)
- PPU\_SenseSRAM0
  - Драйвер модуля PPU, [181](#)
- PPU\_SenseSRAM1
  - Драйвер модуля PPU, [181](#)
- PPU\_SenseSRAM2
  - Драйвер модуля PPU, [181](#)
- PPU\_SenseSRAM3
  - Драйвер модуля PPU, [181](#)
- PPU\_SenseSYS
  - Драйвер модуля PPU, [181](#)
- PPU\_SetIRQMask
  - Драйвер модуля PPU, [185](#)
- PPU\_SetIRQStatus
  - Драйвер модуля PPU, [185](#)
- PPU\_SetPDCMPPUSense
  - Драйвер модуля PPU, [186](#)
- PPU\_SetState

- Драйвер модуля PPU, [186](#)
- PPU\_SetStateDynamic
  - Драйвер модуля PPU, [187](#)
- PPU\_STA\_DBG\_RECOV\_SPT
  - Драйвер модуля PPU, [180](#)
- PPU\_STA\_FULL\_RET\_SPT
  - Драйвер модуля PPU, [180](#)
- PPU\_STA\_FUNC\_RET\_SPT
  - Драйвер модуля PPU, [180](#)
- PPU\_STA\_LGC\_RET\_SPT
  - Драйвер модуля PPU, [180](#)
- PPU\_STA\_MEM\_OFF\_SPT
  - Драйвер модуля PPU, [180](#)
- PPU\_STA\_MEM\_RET\_EMU\_SPT
  - Драйвер модуля PPU, [180](#)
- PPU\_STA\_MEM\_RET\_SPT
  - Драйвер модуля PPU, [180](#)
- PPU\_STA\_OFF\_EMU\_SPT
  - Драйвер модуля PPU, [180](#)
- PPU\_STA\_OFF\_SPT
  - Драйвер модуля PPU, [180](#)
- PPU\_STA\_ON\_SPT
  - Драйвер модуля PPU, [180](#)
- PPU\_STA\_POLICY\_OP\_IRQ\_SPT
  - Драйвер модуля PPU, [180](#)
- PPU\_STA\_POLICY\_PWR\_IRQ\_SPT
  - Драйвер модуля PPU, [180](#)
- PPU\_STA\_PolicyPwr
  - Драйвер модуля PPU, [178](#)
- PPU\_STA\_WRM\_RST\_SPT
  - Драйвер модуля PPU, [180](#)
- PPU\_StaAccept
  - Драйвер модуля PPU, [179](#)
- PPU\_StaDeny
  - Драйвер модуля PPU, [179](#)
- PPU\_StaPolicyOp
  - Драйвер модуля PPU, [178](#)
- PPU\_StaPolicyTrn
  - Драйвер модуля PPU, [179](#)
- PPU\_StateOffRequestHandler
  - Драйвер модуля PPU, [187](#)
- ppu\_status
  - Драйвер модуля PPU, [181](#)
- PPU\_Status\_ConfigError
  - Драйвер модуля PPU, [182](#)
- PPU\_Status\_DriverError
  - Драйвер модуля PPU, [182](#)
- PPU\_Status\_FeatureNotSupport
  - Драйвер модуля PPU, [182](#)
- PPU\_Status\_InvalidArgument
  - Драйвер модуля PPU, [182](#)
- PPU\_Status\_Ok
  - Драйвер модуля PPU, [182](#)
- PPU\_SW\_DEV\_DEL\_SPT
  - Драйвер модуля PPU, [180](#)
- PPU\_UnsptPolicy
  - Драйвер модуля PPU, [178](#)
- presc
  - rcw\_config, [445](#)
- prescale
  - dualtimer\_hardware\_config, [419](#)
- prescaler
  - \_can\_timing\_config, [369](#)
  - \_ttcan\_config, [410](#)
  - pwm\_channel\_config, [440](#)
  - vtu\_config, [480](#)
- prescaler\_cmd
  - pwm\_channel\_config, [440](#)
- prescaler\_divmux
  - pwm\_channel\_config, [440](#)
- prescaler\_mode
  - pwm\_channel\_config, [440](#)
- prescaler\_syncrst
  - pwm\_channel\_config, [440](#)
- prohibit\_overflow
  - \_can\_rxb\_config, [367](#)
- ptb\_config
  - \_can\_config, [358](#)
- pull\_en
  - sdmmc\_cfg\_pin\_map, [461](#)
- PWM\_ApplyLongSoftOuts
  - Драйвер модуля PWM, [203](#)
- PWM\_ApplySoftOuts
  - Драйвер модуля PWM, [203](#)
- pwm\_callback
  - \_pwm\_handle, [398](#)
- pwm\_channel\_config, [434](#)
  - channel, [435](#)
  - chopper\_duty, [435](#)
  - chopper\_first\_width, [435](#)
  - chopper\_freq, [435](#)
  - chopper\_work, [435](#)
  - cmd, [435](#)
  - cmpa, [436](#)
  - cmpb, [436](#)
  - cnt\_eq\_cmpa\_dec\_outa, [436](#)
  - cnt\_eq\_cmpa\_dec\_outb, [436](#)
  - cnt\_eq\_cmpa\_inc\_outa, [436](#)
  - cnt\_eq\_cmpa\_inc\_outb, [436](#)
  - cnt\_eq\_cmpb\_dec\_outa, [436](#)
  - cnt\_eq\_cmpb\_dec\_outb, [436](#)
  - cnt\_eq\_cmpb\_inc\_outa, [437](#)
  - cnt\_eq\_cmpb\_inc\_outb, [437](#)
  - cnt\_eq\_prd\_outa, [437](#)
  - cnt\_eq\_prd\_outb, [437](#)
  - cnt\_eq\_zero\_outa, [437](#)
  - cnt\_eq\_zero\_outb, [437](#)
  - cntmode, [437](#)
  - counter, [437](#)
  - ctrphs, [438](#)
  - dz\_falling\_edge\_delay\_clk, [438](#)
  - dz\_falling\_edge\_outb\_inv, [438](#)
  - dz\_falling\_edge\_source, [438](#)
  - dz\_outa\_enable, [438](#)
  - dz\_outb\_enable, [438](#)
  - dz\_rising\_edge\_delay\_clk, [438](#)



- dz\_rising\_edge\_outa\_inv, [438](#)
- dz\_rising\_edge\_source, [439](#)
- eventprd, [439](#)
- inputs\_mask\_mult, [439](#)
- inputs\_mask\_one, [439](#)
- ldamode, [439](#)
- ldbmode, [439](#)
- ldcswrf, [439](#)
- loadprd, [440](#)
- period, [440](#)
- prescaler, [440](#)
- prescaler\_cmd, [440](#)
- prescaler\_divmux, [440](#)
- prescaler\_mode, [440](#)
- prescaler\_syncrst, [440](#)
- pwm\_int\_enable, [441](#)
- pwm\_int\_source, [441](#)
- pwm\_tu\_int\_mult, [441](#)
- pwm\_tu\_int\_one, [441](#)
- scmpamode, [441](#)
- scmpbmode, [441](#)
- sw\_forced\_long\_outa, [441](#)
- sw\_forced\_long\_outb, [441](#)
- sw\_forced\_outa, [442](#)
- sw\_forced\_outb, [442](#)
- syncphsen, [442](#)
- trip\_unit\_action\_outa, [442](#)
- trip\_unit\_action\_outb, [442](#)
- pwm\_chopper\_duty
  - Драйвер модуля PWM, [191](#), [192](#)
- pwm\_chopper\_first\_width
  - Драйвер модуля PWM, [192](#), [193](#)
- pwm\_chopper\_freq
  - Драйвер модуля PWM, [194](#)
- pwm\_chopper\_work
  - Драйвер модуля PWM, [194](#)
- PWM\_ChopperDuty\_1\_8
  - Драйвер модуля PWM, [192](#)
- pwm\_ChopperDuty\_1\_8
  - Драйвер модуля PWM, [192](#)
- PWM\_ChopperDuty\_2\_8
  - Драйвер модуля PWM, [192](#)
- pwm\_ChopperDuty\_2\_8
  - Драйвер модуля PWM, [192](#)
- PWM\_ChopperDuty\_3\_8
  - Драйвер модуля PWM, [192](#)
- pwm\_ChopperDuty\_3\_8
  - Драйвер модуля PWM, [192](#)
- PWM\_ChopperDuty\_4\_8
  - Драйвер модуля PWM, [192](#)
- pwm\_ChopperDuty\_4\_8
  - Драйвер модуля PWM, [192](#)
- PWM\_ChopperDuty\_5\_8
  - Драйвер модуля PWM, [192](#)
- pwm\_ChopperDuty\_5\_8
  - Драйвер модуля PWM, [192](#)
- PWM\_ChopperDuty\_6\_8
  - Драйвер модуля PWM, [192](#)
- pwm\_ChopperDuty\_6\_8
  - Драйвер модуля PWM, [192](#)
- PWM\_ChopperFirstWidth\_0\_8
  - Драйвер модуля PWM, [193](#)
- pwm\_ChopperFirstWidth\_0\_8
  - Драйвер модуля PWM, [193](#)
- PWM\_ChopperFirstWidth\_10\_8
  - Драйвер модуля PWM, [193](#)
- pwm\_ChopperFirstWidth\_10\_8
  - Драйвер модуля PWM, [193](#)
- PWM\_ChopperFirstWidth\_11\_8
  - Драйвер модуля PWM, [193](#)
- pwm\_ChopperFirstWidth\_11\_8
  - Драйвер модуля PWM, [193](#)
- PWM\_ChopperFirstWidth\_12\_8
  - Драйвер модуля PWM, [194](#)
- pwm\_ChopperFirstWidth\_12\_8
  - Драйвер модуля PWM, [193](#)
- PWM\_ChopperFirstWidth\_13\_8
  - Драйвер модуля PWM, [194](#)
- pwm\_ChopperFirstWidth\_13\_8
  - Драйвер модуля PWM, [193](#)
- PWM\_ChopperFirstWidth\_14\_8
  - Драйвер модуля PWM, [194](#)
- pwm\_ChopperFirstWidth\_14\_8
  - Драйвер модуля PWM, [193](#)
- PWM\_ChopperFirstWidth\_15\_8
  - Драйвер модуля PWM, [194](#)
- pwm\_ChopperFirstWidth\_15\_8
  - Драйвер модуля PWM, [193](#)
- PWM\_ChopperFirstWidth\_1\_8
  - Драйвер модуля PWM, [193](#)
- pwm\_ChopperFirstWidth\_1\_8
  - Драйвер модуля PWM, [193](#)
- PWM\_ChopperFirstWidth\_2\_8
  - Драйвер модуля PWM, [193](#)
- pwm\_ChopperFirstWidth\_2\_8
  - Драйвер модуля PWM, [193](#)
- PWM\_ChopperFirstWidth\_3\_8
  - Драйвер модуля PWM, [193](#)
- pwm\_ChopperFirstWidth\_3\_8
  - Драйвер модуля PWM, [193](#)
- PWM\_ChopperFirstWidth\_4\_8
  - Драйвер модуля PWM, [193](#)
- pwm\_ChopperFirstWidth\_4\_8
  - Драйвер модуля PWM, [193](#)
- PWM\_ChopperFirstWidth\_5\_8
  - Драйвер модуля PWM, [193](#)
- pwm\_ChopperFirstWidth\_5\_8
  - Драйвер модуля PWM, [193](#)
- PWM\_ChopperFirstWidth\_6\_8
  - Драйвер модуля PWM, [193](#)
- pwm\_ChopperFirstWidth\_6\_8
  - Драйвер модуля PWM, [193](#)

- PWM\_ChopperFirstWidth\_7\_8
  - Драйвер модуля PWM, [193](#)
- pwm\_ChopperFirstWidth\_7\_8
  - Драйвер модуля PWM, [193](#)
- PWM\_ChopperFirstWidth\_8\_8
  - Драйвер модуля PWM, [193](#)
- pwm\_ChopperFirstWidth\_8\_8
  - Драйвер модуля PWM, [193](#)
- PWM\_ChopperFirstWidth\_9\_8
  - Драйвер модуля PWM, [193](#)
- pwm\_ChopperFirstWidth\_9\_8
  - Драйвер модуля PWM, [193](#)
- PWM\_ChopperFreqClk\_16
  - Драйвер модуля PWM, [194](#)
- pwm\_ChopperFreqClk\_16
  - Драйвер модуля PWM, [194](#)
- PWM\_ChopperFreqClk\_24
  - Драйвер модуля PWM, [194](#)
- pwm\_ChopperFreqClk\_24
  - Драйвер модуля PWM, [194](#)
- PWM\_ChopperFreqClk\_32
  - Драйвер модуля PWM, [194](#)
- pwm\_ChopperFreqClk\_32
  - Драйвер модуля PWM, [194](#)
- PWM\_ChopperFreqClk\_40
  - Драйвер модуля PWM, [194](#)
- pwm\_ChopperFreqClk\_40
  - Драйвер модуля PWM, [194](#)
- PWM\_ChopperFreqClk\_48
  - Драйвер модуля PWM, [194](#)
- pwm\_ChopperFreqClk\_48
  - Драйвер модуля PWM, [194](#)
- PWM\_ChopperFreqClk\_56
  - Драйвер модуля PWM, [194](#)
- pwm\_ChopperFreqClk\_56
  - Драйвер модуля PWM, [194](#)
- PWM\_ChopperFreqClk\_64
  - Драйвер модуля PWM, [194](#)
- pwm\_ChopperFreqClk\_64
  - Драйвер модуля PWM, [194](#)
- PWM\_ChopperFreqClk\_8
  - Драйвер модуля PWM, [194](#)
- pwm\_ChopperFreqClk\_8
  - Драйвер модуля PWM, [194](#)
- pwm\_ChopperWorkOff
  - Драйвер модуля PWM, [195](#)
- pwm\_ChopperWorkOn
  - Драйвер модуля PWM, [195](#)
- PWM\_CmdForAllChannels
  - Драйвер модуля PWM, [204](#)
- pwm\_cntmode
  - Драйвер модуля PWM, [195](#)
- pwm\_CntModeDown
  - Драйвер модуля PWM, [195](#)
- pwm\_CntModeOff
  - Драйвер модуля PWM, [195](#)
- pwm\_CntModeUp
  - Драйвер модуля PWM, [195](#)
- pwm\_CntModeUpDown
  - Драйвер модуля PWM, [195](#)
- PWM\_COUNT
  - Драйвер модуля PWM, [191](#)
- PWM\_Deinit
  - Драйвер модуля PWM, [204](#)
- pwm\_dirsync
  - Драйвер модуля PWM, [195](#)
- pwm\_DirSyncDown
  - Драйвер модуля PWM, [195](#)
- pwm\_DirSyncUp
  - Драйвер модуля PWM, [195](#)
- pwm\_dz\_mode
  - Драйвер модуля PWM, [195](#), [196](#)
- pwm\_dz\_outx\_inv
  - Драйвер модуля PWM, [196](#)
- pwm\_dz\_signal
  - Драйвер модуля PWM, [196](#)
- PWM\_DzModeOff
  - Драйвер модуля PWM, [196](#)
- pwm\_DzModeOff
  - Драйвер модуля PWM, [196](#)
- PWM\_DzModeOn
  - Драйвер модуля PWM, [196](#)
- pwm\_DzModeOn
  - Драйвер модуля PWM, [196](#)
- pwm\_DzSignalOutA
  - Драйвер модуля PWM, [197](#)
- pwm\_DzSignalOutB
  - Драйвер модуля PWM, [197](#)
- PWM\_DzSignalOutxInvOff
  - Драйвер модуля PWM, [196](#)
- pwm\_DzSignalOutxInvOff
  - Драйвер модуля PWM, [196](#)
- PWM\_DzSignalOutxInvOn
  - Драйвер модуля PWM, [196](#)
- pwm\_DzSignalOutxInvOn
  - Драйвер модуля PWM, [196](#)
- PWM\_Enable
  - Драйвер модуля PWM, [204](#), [205](#)
- pwm\_eventprd
  - Драйвер модуля PWM, [197](#)
- PWM\_EventPrdNo
  - Драйвер модуля PWM, [197](#)
- pwm\_EventPrdNo
  - Драйвер модуля PWM, [197](#)
- PWM\_EventPrdOne
  - Драйвер модуля PWM, [197](#)
- pwm\_EventPrdOne
  - Драйвер модуля PWM, [197](#)
- PWM\_EventPrdThree
  - Драйвер модуля PWM, [197](#)
- pwm\_EventPrdThree
  - Драйвер модуля PWM, [197](#)
- PWM\_EventPrdTwo
  - Драйвер модуля PWM, [197](#)
- pwm\_EventPrdTwo
  - Драйвер модуля PWM, [197](#)

- PWM\_GetChannelDefaultConfig  
     Драйвер модуля PWM, 205  
 PWM\_GetCntStat  
     Драйвер модуля PWM, 205  
 PWM\_Init  
     Драйвер модуля PWM, 206  
 PWM\_InitChannel  
     Драйвер модуля PWM, 206  
 pwm\_int\_en  
     Драйвер модуля PWM, 197  
 pwm\_int\_enable  
     pwm\_channel\_config, 441  
 pwm\_int\_source  
     pwm\_channel\_config, 441  
     Драйвер модуля PWM, 197, 198  
 PWM\_IntCallback  
     Драйвер модуля PWM, 206  
 pwm\_IntEnNo  
     Драйвер модуля PWM, 197  
 pwm\_IntEnYes  
     Драйвер модуля PWM, 197  
 PWM\_IntSourceCtrcntEquCmpADec  
     Драйвер модуля PWM, 198  
 pwm\_IntSourceCtrcntEquCmpADec  
     Драйвер модуля PWM, 198  
 PWM\_IntSourceCtrcntEquCmpAInc  
     Драйвер модуля PWM, 198  
 pwm\_IntSourceCtrcntEquCmpAInc  
     Драйвер модуля PWM, 198  
 PWM\_IntSourceCtrcntEquCmpBDec  
     Драйвер модуля PWM, 198  
 pwm\_IntSourceCtrcntEquCmpBDec  
     Драйвер модуля PWM, 198  
 PWM\_IntSourceCtrcntEquCmpBInc  
     Драйвер модуля PWM, 198  
 pwm\_IntSourceCtrcntEquCmpBInc  
     Драйвер модуля PWM, 198  
 PWM\_IntSourceCtrcntEquCtrprd  
     Драйвер модуля PWM, 198  
 pwm\_IntSourceCtrcntEquCtrprd  
     Драйвер модуля PWM, 198  
 PWM\_IntSourceCtrcntEquZero  
     Драйвер модуля PWM, 198  
 pwm\_IntSourceCtrcntEquZero  
     Драйвер модуля PWM, 198  
 pwm\_IntSourceNo  
     Драйвер модуля PWM, 198  
 pwm\_ldcswrf  
     Драйвер модуля PWM, 198  
 pwm\_LdcswrfCtrcntEquCtrprd  
     Драйвер модуля PWM, 198  
 pwm\_LdcswrfCtrcntZero  
     Драйвер модуля PWM, 198  
 pwm\_LdcswrfCtrcntZeroEquCtrprd  
     Драйвер модуля PWM, 198  
 pwm\_LdcswrfDirect  
     Драйвер модуля PWM, 198  
 pwm\_ldxmode  
     Драйвер модуля PWM, 198  
 pwm\_LdxModeCtrcntEquCtrprd  
     Драйвер модуля PWM, 199  
 pwm\_LdxModeCtrcntZero  
     Драйвер модуля PWM, 199  
 pwm\_LdxModeCtrcntZeroOrEquCtrprd  
     Драйвер модуля PWM, 199  
 pwm\_LdxModeNoLoad  
     Драйвер модуля PWM, 199  
 pwm\_loadprd  
     Драйвер модуля PWM, 199  
 pwm\_LoadPrdDirect  
     Драйвер модуля PWM, 199  
 pwm\_LoadPrdRegister  
     Драйвер модуля PWM, 199  
 PWM\_OutA  
     Драйвер модуля PWM, 191  
 PWM\_OutB  
     Драйвер модуля PWM, 191  
 pwm\_outx\_cmd  
     Драйвер модуля PWM, 199  
 pwm\_OutxCmdClear  
     Драйвер модуля PWM, 199  
 pwm\_OutxCmdNo  
     Драйвер модуля PWM, 199  
 pwm\_OutxCmdSet  
     Драйвер модуля PWM, 199  
 pwm\_OutxCmdToggle  
     Драйвер модуля PWM, 199  
 pwm\_polarity  
     vtu\_config, 481  
 pwm\_polarity2  
     vtu\_config, 481  
 pwm\_prescaler\_cmd  
     Драйвер модуля PWM, 199  
 pwm\_prescaler\_divmux  
     Драйвер модуля PWM, 200  
 pwm\_prescaler\_mode  
     Драйвер модуля PWM, 200  
 pwm\_prescaler\_syncrst  
     Драйвер модуля PWM, 200  
 pwm\_PrescalerDivMux1  
     Драйвер модуля PWM, 200  
 pwm\_PrescalerDivMux16  
     Драйвер модуля PWM, 200  
 pwm\_PrescalerDivMux2  
     Драйвер модуля PWM, 200  
 pwm\_PrescalerDivMux4  
     Драйвер модуля PWM, 200  
 pwm\_PrescalerDivMux8  
     Драйвер модуля PWM, 200  
 pwm\_PrescalerDivMuxPWMClk  
     Драйвер модуля PWM, 200  
 pwm\_PrescalerSyncRstDis  
     Драйвер модуля PWM, 200  
 pwm\_PrescalerSyncRstEn  
     Драйвер модуля PWM, 200  
 pwm\_PrescCmdReset



- Драйвер модуля PWM, [200](#)
- pwm\_PrescCmdSave
  - Драйвер модуля PWM, [200](#)
- pwm\_PrescModeAlways
  - Драйвер модуля PWM, [200](#)
- pwm\_PrescModeTimerIsRun
  - Драйвер модуля PWM, [200](#)
- pwm\_run\_command
  - Драйвер модуля PWM, [200](#)
- pwm\_RunCmdRun
  - Драйвер модуля PWM, [201](#)
- pwm\_RunCmdStop
  - Драйвер модуля PWM, [201](#)
- pwm\_RunCmdStopEvent
  - Драйвер модуля PWM, [201](#)
- pwm\_scmpxmode
  - Драйвер модуля PWM, [201](#)
- pwm\_SCmpxModeDirect
  - Драйвер модуля PWM, [201](#)
- pwm\_SCmpxModeReg
  - Драйвер модуля PWM, [201](#)
- PWM\_SetPeriod
  - Драйвер модуля PWM, [207](#)
- pwm\_status
  - Драйвер модуля PWM, [201](#)
- PWM\_Status\_BadConfigure
  - Драйвер модуля PWM, [201](#)
- PWM\_Status\_InvalidArgument
  - Драйвер модуля PWM, [201](#)
- PWM\_Status\_Ok
  - Драйвер модуля PWM, [201](#)
- pwm\_syncosel
  - Драйвер модуля PWM, [201](#)
- pwm\_SyncoSelCtrntEquCmpb
  - Драйвер модуля PWM, [201](#)
- pwm\_SyncoSelCtrntZero
  - Драйвер модуля PWM, [201](#)
- pwm\_SyncoSelOff
  - Драйвер модуля PWM, [201](#)
- pwm\_SyncoSelSynci
  - Драйвер модуля PWM, [201](#)
- pwm\_syncphsen
  - Драйвер модуля PWM, [201](#)
- pwm\_SyncPhsEnDis
  - Драйвер модуля PWM, [202](#)
- pwm\_SyncPhsEnEn
  - Драйвер модуля PWM, [202](#)
- pwm\_trip\_unit\_action
  - Драйвер модуля PWM, [202](#)
- pwm\_trip\_unit\_signal
  - Драйвер модуля PWM, [202](#)
- PWM\_TripUnitActionHigh
  - Драйвер модуля PWM, [202](#)
- pwm\_TripUnitActionHigh
  - Драйвер модуля PWM, [202](#)
- PWM\_TripUnitActionNo
  - Драйвер модуля PWM, [202](#)
- pwm\_TripUnitActionNo
  - Драйвер модуля PWM, [202](#)
- PWM\_TripUnitActionOne
  - Драйвер модуля PWM, [202](#)
- pwm\_TripUnitActionOne
  - Драйвер модуля PWM, [202](#)
- PWM\_TripUnitActionZero
  - Драйвер модуля PWM, [202](#)
- pwm\_TripUnitActionZero
  - Драйвер модуля PWM, [202](#)
- pwm\_TripUnitSignalNotUsed
  - Драйвер модуля PWM, [202](#)
- pwm\_TripUnitSignalUsed
  - Драйвер модуля PWM, [202](#)
- pwmtu\_int\_mult
  - pwm\_channel\_config, [441](#)
- pwmtu\_int\_one
  - pwm\_channel\_config, [441](#)
- pz
  - rcw\_config, [445](#)
  - rcw\_config\_reg, [448](#)
- QSPI\_ClearInterrupt
  - Драйвер модуля QSPI, [217](#)
- QSPI\_CommandAllSerial
  - Драйвер модуля QSPI, [212](#)
- QSPI\_CommandDataQuad
  - Драйвер модуля QSPI, [212](#)
- QSPI\_CommandNoOpcodeAddrFourBytes
  - Драйвер модуля QSPI, [213](#)
- QSPI\_CommandNoOpcodeAddrThreeBytes
  - Драйвер модуля QSPI, [213](#)
- QSPI\_CommandOpcodeAddrFourBytes
  - Драйвер модуля QSPI, [213](#)
- QSPI\_CommandOpcodeAddrOneByte
  - Драйвер модуля QSPI, [213](#)
- QSPI\_CommandOpcodeAddrThreeBytes
  - Драйвер модуля QSPI, [213](#)
- QSPI\_CommandOpcodeAddrTwoBytes
  - Драйвер модуля QSPI, [213](#)
- QSPI\_CommandOpcodeOnly
  - Драйвер модуля QSPI, [213](#)
- QSPI\_CommandOpcodeSerial
  - Драйвер модуля QSPI, [212](#)
- qspi\_config
  - \_nor\_handle, [395](#)
- qspi\_config\_t
  - Драйвер модуля QSPI, [212](#)
- QSPI\_DeInit
  - Драйвер модуля QSPI, [218](#)
- QSPI\_DisableDMA
  - Драйвер модуля QSPI, [218](#)
- QSPI\_DisableInterrupt
  - Драйвер модуля QSPI, [218](#)
- QSPI\_DisableXIP
  - Драйвер модуля QSPI, [218](#)
- qspi\_dma\_handle\_t, [442](#)
- QSPI\_DMA\_Status\_Fail
  - Драйвер модуля QSPI, [215](#)
- QSPI\_DMA\_Status\_InvalidArgument

- Драйвер модуля QSPI, 215
- QSPI\_DMA\_Status\_Success
  - Драйвер модуля QSPI, 215
- qspi\_dma\_status\_t
  - Драйвер модуля QSPI, 214
- QSPI\_DMADescriptorInitRX
  - Драйвер модуля QSPI, 219
- QSPI\_DMADescriptorInitTX
  - Драйвер модуля QSPI, 219
- QSPI\_DMAREadDescriptorInitRX
  - Драйвер модуля QSPI, 219
- QSPI\_DualSPI
  - Драйвер модуля QSPI, 213
- QSPI\_Enable
  - Драйвер модуля QSPI, 220
- QSPI\_EnableDMA
  - Драйвер модуля QSPI, 220
- QSPI\_EnableInterrupt
  - Драйвер модуля QSPI, 220
- QSPI\_EnableXIP
  - Драйвер модуля QSPI, 221
- QSPI\_GetDefaultCommandSet
  - Драйвер модуля QSPI, 221
- QSPI\_GetDefaultConfig
  - Драйвер модуля QSPI, 221
- QSPI\_GetDefaultConfigXIP
  - Драйвер модуля QSPI, 221
- QSPI\_GetDummyDMADescriptorsCount
  - Драйвер модуля QSPI, 222
- QSPI\_GetInstance
  - Драйвер модуля QSPI, 222
- QSPI\_GetReadDMADescriptorsCount
  - Драйвер модуля QSPI, 222
- QSPI\_GetRXLVL
  - Драйвер модуля QSPI, 222
- QSPI\_GetStatusFlag
  - Драйвер модуля QSPI, 224
- QSPI\_GetTXLVL
  - Драйвер модуля QSPI, 224
- QSPI\_Init
  - Драйвер модуля QSPI, 224
- QSPI\_NorFlashInit
  - Драйвер модуля QSPI, 225
- QSPI\_NormalSPI
  - Драйвер модуля QSPI, 213
- QSPI\_QuadSPI
  - Драйвер модуля QSPI, 213
- QSPI\_ReadData
  - Драйвер модуля QSPI, 225
- QSPI\_ReadDataByte
  - Драйвер модуля QSPI, 225
- QSPI\_ReadDataDMA
  - Драйвер модуля QSPI, 226
- QSPI\_SetBitSize
  - Драйвер модуля QSPI, 226
- QSPI\_SetInhibitDin
  - Драйвер модуля QSPI, 226
- QSPI\_SetInhibitDout
  - Драйвер модуля QSPI, 227
- QSPI\_SetQMode
  - Драйвер модуля QSPI, 227
- QSPI\_SetSlaveSelect
  - Драйвер модуля QSPI, 227
- QSPI\_TransferCreateHandleDMA
  - Драйвер модуля QSPI, 227
- QSPI\_WriteData
  - Драйвер модуля QSPI, 228
- QSPI\_WriteDataByte
  - Драйвер модуля QSPI, 228
- QSPI\_WriteDataDMA
  - Драйвер модуля QSPI, 228
- quad\_control\_config
  - \_nor\_config, 394
- quad\_enable\_bit\_shift
  - \_qspi\_nor\_config, 402
- quad\_enable\_command
  - \_qspi\_nor\_config, 402
- quad\_mode\_setting
  - \_qspi\_nor\_init\_config, 404
- quad\_read\_command
  - \_qspi\_nor\_config, 402
- rca
  - sdmmc\_card\_t, 458
- read\_cmd\_format
  - \_qspi\_nor\_handle, 403
- read\_memory\_command
  - \_nor\_command\_set, 392
- read\_status\_cmd
  - \_nor\_command\_set, 392
- ref\_clk
  - \_pwm\_handle, 398
- reference\_id
  - \_ttcan\_config, 410
- reference\_ide
  - \_ttcan\_config, 410
- reg\_base
  - sdmmc\_port\_cfg\_t, 462
- reg\_value
  - rwc\_union\_reg, 454
- regs
  - sdmmc\_card\_t, 458
- reload\_value
  - timer\_hardware\_config, 472
- remaining\_bytes
  - \_i2c\_master\_handle, 385
- remaining\_bytes\_DMA
  - \_i2c\_master\_dma\_handle, 383
- remaining\_subaddr
  - \_i2c\_master\_handle, 385
- reqForCPU
  - ppu\_config, 433
- ReqOffForCPU
  - Драйвер модуля PPU, 178
- resen
  - wdt\_config, 481
- reset\_en

- rcw\_config, 445
  - rcw\_config\_reg, 449
- resolution
  - \_i2s\_config, 388
- right\_samples
  - \_i2s\_handle, 389
  - \_i2s\_transfer, 390
- rtr
  - \_can\_rx\_buffer\_frame, 365
  - \_can\_tx\_buffer\_frame, 371
- rts
  - \_can\_rx\_buffer\_frame, 365
- run\_configuration
  - power\_config, 427
- RWC\_Alarm
  - Драйвер модуля RWC, 234
- rcw\_alarm\_enable
  - Драйвер модуля RWC, 233
- rcw\_alarm\_reg, 443
  - alarm, 443
- RWC\_AlarmDisable
  - Драйвер модуля RWC, 234
- RWC\_AlarmEnable
  - Драйвер модуля RWC, 234
- rcw\_cmd
  - Драйвер модуля RWC, 234
- RWC\_CmdRead
  - Драйвер модуля RWC, 234
- RWC\_CmdWait
  - Драйвер модуля RWC, 234
- RWC\_CmdWrite
  - Драйвер модуля RWC, 234
- RWC\_CMOSSignal
  - Драйвер модуля RWC, 235
- RWC\_Config
  - Драйвер модуля RWC, 234
- rcw\_config, 444
  - alarm\_en, 444
  - alarm\_time, 444
  - clkdiv, 444
  - general, 445
  - lfe\_bypass, 445
  - osc\_sel, 445
  - pl, 445
  - presc, 445
  - pz, 445
  - reset\_en, 445
  - shutdown\_force, 445
  - time, 446
  - time\_clk\_sel, 446
  - trim\_lfe, 446
  - trim\_lfi, 446
  - trimload, 446
  - wake\_en, 446
  - wake\_in\_en, 446
  - wake\_pol, 446
  - wake\_stat1, 447
- rcw\_config\_reg, 447
  - \_\_pad0\_\_, 448
  - \_\_pad1\_\_, 448
  - \_\_pad2\_\_, 448
  - alarm\_en, 448
  - clk\_div, 448
  - osc\_sel, 448
  - pl, 448
  - pz, 448
  - reset\_en, 449
  - shutdown\_force, 449
  - time\_clk\_sel, 449
  - wake\_in\_en, 449
  - wake\_stat1, 449
- RWC\_ConvertDatetimeToSeconds
  - hal\_rcw.h, 589
- RWC\_Deinit
  - Драйвер модуля RWC, 238
- RWC\_Div1
  - Драйвер модуля RWC, 235
- RWC\_Div1024
  - Драйвер модуля RWC, 235
- RWC\_Div128
  - Драйвер модуля RWC, 235
- RWC\_Div16
  - Драйвер модуля RWC, 235
- RWC\_Div16384
  - Драйвер модуля RWC, 235
- RWC\_Div2
  - Драйвер модуля RWC, 235
- RWC\_Div2048
  - Драйвер модуля RWC, 235
- RWC\_Div256
  - Драйвер модуля RWC, 235
- RWC\_Div32
  - Драйвер модуля RWC, 235
- RWC\_Div32768
  - Драйвер модуля RWC, 235
- RWC\_Div4
  - Драйвер модуля RWC, 235
- RWC\_Div4096
  - Драйвер модуля RWC, 235
- RWC\_Div512
  - Драйвер модуля RWC, 235
- RWC\_Div64
  - Драйвер модуля RWC, 235
- RWC\_Div8
  - Драйвер модуля RWC, 235
- RWC\_Div8192
  - Драйвер модуля RWC, 235
- RWC\_DivMax
  - Драйвер модуля RWC, 235
- RWC\_EnableAlarmTimerInterruptFromDPD
  - Драйвер модуля RWC, 238
- RWC\_EnableWakeUpTimerInterruptFromDPD
  - Драйвер модуля RWC, 239
- rcw\_freq\_serial
  - Драйвер модуля RWC, 234
- RWC\_FS125kHz

- Драйвер модуля RWC, [234](#)
- RWC\_FS1MHz
  - Драйвер модуля RWC, [234](#)
- RWC\_FS250kHz
  - Драйвер модуля RWC, [234](#)
- RWC\_FS2MHz
  - Драйвер модуля RWC, [234](#)
- RWC\_FS4MHz
  - Драйвер модуля RWC, [234](#)
- RWC\_FS500kHz
  - Драйвер модуля RWC, [234](#)
- RWC\_FS625kHz
  - Драйвер модуля RWC, [234](#)
- RWC\_FS8MHz
  - Драйвер модуля RWC, [234](#)
- rcw\_general\_reg, [449](#)
  - time, [450](#)
- RWC\_GeneralReg
  - Драйвер модуля RWC, [234](#)
- RWC\_GetAlarm
  - Драйвер модуля RWC, [239](#)
- RWC\_GetDatetime
  - Драйвер модуля RWC, [239](#)
- RWC\_GetDefaultConfig
  - Драйвер модуля RWC, [240](#)
- RWC\_GetInternalRegister
  - Драйвер модуля RWC, [240](#)
- RWC\_GetLastAPIStatus
  - Драйвер модуля RWC, [241](#)
- RWC\_GetRTCClkParam
  - Драйвер модуля RWC, [241](#)
- RWC\_GetSecondsTimerCount
  - Драйвер модуля RWC, [241](#)
- RWC\_GetSecondsTimerMatch
  - Драйвер модуля RWC, [242](#)
- RWC\_GetStatusFlags
  - Драйвер модуля RWC, [242](#)
- RWC\_GetTime
  - Драйвер модуля RWC, [242](#)
- RWC\_Init
  - Драйвер модуля RWC, [243](#)
- rcw\_internal\_register
  - Драйвер модуля RWC, [234](#)
- RWC\_InterruptClear
  - Драйвер модуля RWC, [243](#)
- RWC\_IRQWkUpDisable
  - Драйвер модуля RWC, [237](#)
- RWC\_IRQWkUpEnable
  - Драйвер модуля RWC, [237](#)
- rcw\_lfe\_bypass
  - Драйвер модуля RWC, [234](#)
- RWC\_QuartzResonator
  - Драйвер модуля RWC, [235](#)
- rcw\_reset\_type
  - Драйвер модуля RWC, [235](#)
- RWC\_ResetAllExpectedGeneralReg
  - Драйвер модуля RWC, [235](#)
- RWC\_ResetOnlyWakeConfigAndConfig
  - Драйвер модуля RWC, [235](#)
- rcw\_rtcclk\_divisor
  - Драйвер модуля RWC, [235](#)
- rcw\_rtcclk\_type
  - Драйвер модуля RWC, [235](#)
- RWC\_RTCClkTypeLFE
  - Драйвер модуля RWC, [236](#)
- RWC\_RTCClkTypeLFI
  - Драйвер модуля RWC, [236](#)
- RWC\_SetAlarm
  - Драйвер модуля RWC, [243](#)
- RWC\_SetDatetime
  - Драйвер модуля RWC, [244](#)
- RWC\_SetInternalRegister
  - Драйвер модуля RWC, [244](#)
- RWC\_SetLFEbypass
  - Драйвер модуля RWC, [245](#)
- RWC\_SetLFx
  - Драйвер модуля RWC, [245](#)
- RWC\_SetResetCtrl
  - Драйвер модуля RWC, [246](#)
- RWC\_SetRTCClkParam
  - Драйвер модуля RWC, [246](#)
- RWC\_SetSecondsTimerCount
  - Драйвер модуля RWC, [247](#)
- RWC\_SetSecondsTimerMatch
  - Драйвер модуля RWC, [247](#)
- RWC\_SetWakeUpActiveLow
  - Драйвер модуля RWC, [247](#)
- RWC\_SetWakeUpEnable
  - Драйвер модуля RWC, [248](#)
- rcw\_shutdown\_force
  - Драйвер модуля RWC, [236](#)
- RWC\_ShutdownNoSet
  - Драйвер модуля RWC, [236](#)
- RWC\_ShutdownSet
  - Драйвер модуля RWC, [236](#)
- rcw\_status
  - Драйвер модуля RWC, [236](#)
- RWC\_Status\_CheckError
  - Драйвер модуля RWC, [236](#)
- RWC\_Status\_ConfigureError
  - Драйвер модуля RWC, [236](#)
- RWC\_Status\_HardwareBusy
  - Драйвер модуля RWC, [236](#)
- RWC\_Status\_InvalidArgument
  - Драйвер модуля RWC, [236](#)
- RWC\_Status\_Ok
  - Драйвер модуля RWC, [236](#)
- RWC\_Status\_Timeout
  - Драйвер модуля RWC, [236](#)
- RWC\_Status\_VerifyError
  - Драйвер модуля RWC, [236](#)
- RWC\_SYNC\_RETRY\_TIMES
  - Драйвер модуля RWC, [232](#)
- RWC\_Time
  - Драйвер модуля RWC, [234](#)
- rcw\_time\_clk\_sel

- Драйвер модуля RWC, 236
- rcw\_time\_reg, 450
  - time, 451
- RWC\_TimeClock1Hz
  - Драйвер модуля RWC, 237
- RWC\_TimeClock32kHz
  - Драйвер модуля RWC, 237
- rcw\_timer\_status
  - Драйвер модуля RWC, 237
- RWC\_Trim
  - Драйвер модуля RWC, 234
- rcw\_trim\_reg, 451
  - \_\_pad0\_\_, 451
  - \_\_pad1\_\_, 451
  - lfe\_bypass, 452
  - trim\_lfe, 452
  - trim\_lfi, 452
  - wake\_stat2, 452
  - wake\_stat3, 452
- RWC\_TrimLoad
  - Драйвер модуля RWC, 234
- rcw\_trimload\_reg, 452
  - \_\_pad0\_\_, 453
  - trimload, 453
- rcw\_union\_reg, 453
  - alarm, 454
  - config, 454
  - general, 454
  - reg\_value, 454
  - time, 454
  - trim, 454
  - trimload, 454
  - wake\_config, 455
- rcw\_wake\_config\_reg, 455
  - \_\_pad0\_\_, 455
  - \_\_pad1\_\_, 455
  - wake\_en, 456
  - wake\_pol, 456
- rcw\_wake\_up\_irq\_enable
  - Драйвер модуля RWC, 237
- rcw\_wake\_up\_polarity
  - Драйвер модуля RWC, 237
- RWC\_WakeConfig
  - Драйвер модуля RWC, 234
- RWC\_WakeStat1
  - Драйвер модуля RWC, 237
- RWC\_WakeStat2
  - Драйвер модуля RWC, 237
- RWC\_WakeStat3
  - Драйвер модуля RWC, 237
- rcw\_wkup\_enable
  - Драйвер модуля RWC, 237
- RWC\_WkUpDisable
  - Драйвер модуля RWC, 238
- RWC\_WkUpEnable
  - Драйвер модуля RWC, 238
- RWC\_WkUpPolarityHigh
  - Драйвер модуля RWC, 237
- RWC\_WkUpPolarityLow
  - Драйвер модуля RWC, 237
- rx\_data
  - i2c\_slave\_transfer\_t, 424
  - spi\_half\_duplex\_transfer\_t, 466
  - spi\_handle, 468
  - spi\_transfer\_t, 471
  - uart\_handle, 476
  - uart\_transfer, 479
- rx\_data\_size
  - spi\_half\_duplex\_transfer\_t, 466
  - uart\_handle, 476
- rx\_data\_size\_all
  - \_uart\_dma\_handle, 412
  - uart\_handle, 476
- rx\_desc
  - \_i2c\_master\_dma\_handle, 383
  - \_spi\_dma\_handle, 408
- rx\_dma
  - \_i2c\_master\_dma\_handle, 384
- rx\_dma\_handle
  - \_uart\_dma\_handle, 412
- rx\_frames
  - \_can\_handle, 361
- rx\_handle
  - \_spi\_dma\_handle, 408
- rx\_in\_progress
  - \_spi\_dma\_handle, 408
- rx\_nb\_frames\_all
  - \_can\_handle, 361
- rx\_nb\_frames\_rest
  - \_can\_handle, 362
- rx\_remaining\_bytes
  - spi\_handle, 468
- rx\_ring\_buffer
  - uart\_handle, 476
- rx\_ring\_buffer\_head
  - uart\_handle, 476
- rx\_ring\_buffer\_size
  - uart\_handle, 476
- rx\_ring\_buffer\_tail
  - uart\_handle, 476
- rx\_size
  - i2c\_slave\_transfer\_t, 425
- rx\_state
  - \_uart\_dma\_handle, 412
  - uart\_handle, 477
- rxb\_config
  - \_can\_config, 358
- sample\_rate
  - \_i2s\_config, 388
- SAR
  - \_dma\_descriptor, 379
- SAR\_HI
  - \_dma\_channel\_reg, 376
- SAR\_LO
  - \_dma\_channel\_reg, 376
- scatter\_en

- `_dma_multiblock_config`, 382
  - `dma_channel_ctl_cfg`, 417
- `sclk_gating`
  - `_i2s_config`, 388
- `sclk_per_sample`
  - `_i2s_config`, 388
- `scmpamode`
  - `pwm_channel_config`, 441
- `scmpbmode`
  - `pwm_channel_config`, 441
- `sd_uhs_mode`
  - `sdmmc_port_cfg_t`, 462
- `sda_hold`
  - `i2c_master_config_t`, 420
- `sda_setup`
  - `i2c_master_config_t`, 420
- `sdhc_mode`
  - `sdmmc_card_t`, 458
- `SDMMC_1v8`
  - Драйвер модуля SDMMC, 256
- `SDMMC_3v3`
  - Драйвер модуля SDMMC, 256
- `SDMMC_3v3To1v8`
  - Драйвер модуля SDMMC, 256
- `SDMMC_CALC_DIVIDER`
  - Драйвер модуля SDMMC, 251
- `SDMMC_CalcMemorySpace`
  - Драйвер модуля SDMMC, 256
- `sdmmc_card_t`, 456
  - `cfg`, 457
  - `cid`, 457
  - `csd`, 457
  - `ddr_mode`, 457
  - `dev_num`, 457
  - `freq_divider`, 457
  - `freq_input`, 457
  - `gpio_vol`, 457
  - `hs_mode`, 458
  - `lock`, 458
  - `need_1V8en`, 458
  - `rca`, 458
  - `regs`, 458
  - `sdhc_mode`, 458
  - `total_size`, 458
  - `type`, 458
  - `version`, 459
- `sdmmc_cfg_pin_map`, 459
  - `CD`, 459
  - `CK`, 459
  - `CMD`, 460
  - `D0`, 460
  - `D1`, 460
  - `D2`, 460
  - `D3`, 460
  - `D4`, 460
  - `D5`, 460
  - `D6`, 460
  - `D7`, 461
- `max_current`, 461
  - `pull_en`, 461
- `SDMMC_CORECFG0_SLOTTYPE_EMBEDDED`
  - Драйвер модуля SDMMC, 251
- `SDMMC_CORECFG0_SLOTTYPE_REMOVABLE`
  - Драйвер модуля SDMMC, 252
- `SDMMC_CORECFG0_SLOTTYPE_SHARED_BUS`
  - Драйвер модуля SDMMC, 252
- `SDMMC_DataBusTransferWidth_1Bit`
  - Драйвер модуля SDMMC, 255
- `SDMMC_DataBusTransferWidth_4bit`
  - Драйвер модуля SDMMC, 255
- `SDMMC_DisableCard`
  - Драйвер модуля SDMMC, 256
- `SDMMC_ExtDataBusTransferWidth_8bit`
  - Драйвер модуля SDMMC, 255
- `SDMMC_HostPWR_1V8`
  - Драйвер модуля SDMMC, 254
- `SDMMC_HostPWR_3V0`
  - Драйвер модуля SDMMC, 254
- `SDMMC_HostPWR_3V3`
  - Драйвер модуля SDMMC, 254
- `SDMMC_InitCard`
  - Драйвер модуля SDMMC, 257
- `SDMMC_IS_MMC`
  - Драйвер модуля SDMMC, 252
- `SDMMC_IS_SD`
  - Драйвер модуля SDMMC, 252
- `SDMMC_MMC_RCA_ADDR`
  - Драйвер модуля SDMMC, 252
- `SDMMC_NoResponce`
  - Драйвер модуля SDMMC, 255
- `sdmmc_port_cfg_t`, 461
  - `delay_us`, 462
  - `emmc_8bit_en`, 462
  - `freq_out`, 462
  - `freq_out_init`, 462
  - `hs_en`, 462
  - `pin_map`, 462
  - `reg_base`, 462
  - `sd_uhs_mode`, 462
  - `slot_count`, 463
  - `slot_type`, 463
  - `timeout_cd`, 463
  - `timeout_init`, 463
- `SDMMC_Read`
  - Драйвер модуля SDMMC, 257
- `SDMMC_ReadAsync`
  - Драйвер модуля SDMMC, 258
- `SDMMC_ReadWait`
  - Драйвер модуля SDMMC, 258
- `SDMMC_ResponceLength136`
  - Драйвер модуля SDMMC, 255
- `SDMMC_ResponceLength48`
  - Драйвер модуля SDMMC, 255
- `SDMMC_ResponceLength48_Check`
  - Драйвер модуля SDMMC, 255
- `SDMMC_SD_OCR_INIT_VALUE`



- Драйвер модуля SDMMC, [252](#)
- SDMMC\_SD\_SEND\_IF\_COND\_PATTERN
  - Драйвер модуля SDMMC, [252](#)
- SDMMC\_SD\_UHS\_MODE\_DDR50
  - Драйвер модуля SDMMC, [253](#)
- SDMMC\_SD\_UHS\_MODE\_DEFAULT\_SDR12
  - Драйвер модуля SDMMC, [253](#)
- SDMMC\_SD\_UHS\_MODE\_HIGHSPEED\_SDR25
  - Драйвер модуля SDMMC, [253](#)
- SDMMC\_SD\_UHS\_MODE\_SDR104
  - Драйвер модуля SDMMC, [253](#)
- SDMMC\_SD\_UHS\_MODE\_SDR50
  - Драйвер модуля SDMMC, [253](#)
- SDMMC\_SDHC\_SECTOR\_SIZE
  - Драйвер модуля SDMMC, [253](#)
- SDMMC\_SDMA\_ALIGN
  - Драйвер модуля SDMMC, [253](#)
- SDMMC\_SDMA\_BLOCK\_ALIGN
  - Драйвер модуля SDMMC, [253](#)
- SDMMC\_SDMA\_BLOCK\_SIZE
  - Драйвер модуля SDMMC, [254](#)
- SDMMC\_SDMA\_IS\_BLOCK\_ALIGN\_ADDR
  - Драйвер модуля SDMMC, [254](#)
- SDMMC\_SDMA\_TransferRead
  - Драйвер модуля SDMMC, [255](#)
- SDMMC\_SDMA\_TransferWrite
  - Драйвер модуля SDMMC, [255](#)
- SDMMC\_Status\_Err
  - Драйвер модуля SDMMC, [255](#)
- SDMMC\_Status\_Ok
  - Драйвер модуля SDMMC, [255](#)
- sdmmc\_status\_t
  - Драйвер модуля SDMMC, [255](#)
- SDMMC\_TIMEOUTCONTROL\_MAX\_VALUE
  - Драйвер модуля SDMMC, [254](#)
- SDMMC\_TypeMMC
  - Драйвер модуля SDMMC, [254](#)
- SDMMC\_TypeSD
  - Драйвер модуля SDMMC, [254](#)
- sdmmc\_voltage\_t
  - Драйвер модуля SDMMC, [255](#)
- SDMMC\_Write
  - Драйвер модуля SDMMC, [258](#)
- SDMMC\_WriteAsync
  - Драйвер модуля SDMMC, [259](#)
- SDMMC\_WriteWait
  - Драйвер модуля SDMMC, [260](#)
- second
  - \_rtc\_datetime, [407](#)
- secondary\_sample\_point\_offset
  - \_can\_timing\_config, [369](#)
- SECONDS\_IN\_A\_DAY
  - Драйвер модуля RWC, [233](#)
- SECONDS\_IN\_A\_HOUR
  - Драйвер модуля RWC, [233](#)
- SECONDS\_IN\_A\_MINUTE
  - Драйвер модуля RWC, [233](#)
- sector\_size\_bytes
  - \_nor\_handle, [395](#)
- seg1
  - \_can\_timing\_config, [369](#)
- seg2
  - \_can\_timing\_config, [370](#)
- sel
  - clkctr\_pll\_cfg, [415](#)
- self\_acknowledge
  - \_can\_rxb\_config, [367](#)
- SGR\_HI
  - \_dma\_channel\_reg, [377](#)
- SGR\_LO
  - \_dma\_channel\_reg, [377](#)
- shift\_dir
  - spi\_config\_internal\_t, [464](#)
- shutdown\_force
  - rwconfig, [445](#)
  - rwconfig\_reg, [449](#)
- single\_serial
  - spi\_microwire\_cfg\_t, [470](#)
- size
  - dualtimer\_hardware\_config, [419](#)
- sjw
  - \_can\_timing\_config, [370](#)
- slave\_address
  - i2c\_master\_transfer\_t, [422](#)
- slave\_fsm
  - \_i2c\_slave\_handle, [387](#)
- slave\_pol
  - \_qspi\_config, [401](#)
- slave\_select
  - \_qspi\_config, [401](#)
- slot\_count
  - sdmmc\_port\_cfg\_t, [463](#)
- slot\_type
  - sdmmc\_port\_cfg\_t, [463](#)
- smc\_adv
  - Драйвер модуля SMC, [262](#)
- SMC\_AdvLcd
  - Драйвер модуля SMC, [262](#)
- SMC\_AdvMemory
  - Драйвер модуля SMC, [262](#)
- SMC\_AdvNotUsed
  - Драйвер модуля SMC, [262](#)
- SMC\_AdvUsed
  - Драйвер модуля SMC, [262](#)
- smc\_bit\_depth
  - Драйвер модуля SMC, [262](#)
- SMC\_BitDepth16
  - Драйвер модуля SMC, [262](#)
- SMC\_BitDepth24
  - Драйвер модуля SMC, [262](#)
- SMC\_BitDepth32
  - Драйвер модуля SMC, [262](#)
- SMC\_BitDepth8
  - Драйвер модуля SMC, [262](#)
- smc\_bls
  - Драйвер модуля SMC, [262](#)

- SMC\_BlsAsNcs
  - Драйвер модуля SMC, [262](#)
- SMC\_BlsAsNwe
  - Драйвер модуля SMC, [262](#)
- smc\_burst\_align
  - Драйвер модуля SMC, [262](#)
- SMC\_BurstAlign128
  - Драйвер модуля SMC, [263](#)
- SMC\_BurstAlign256
  - Драйвер модуля SMC, [263](#)
- SMC\_BurstAlign32
  - Драйвер модуля SMC, [263](#)
- SMC\_BurstAlign64
  - Драйвер модуля SMC, [263](#)
- SMC\_BurstAlignNo
  - Драйвер модуля SMC, [263](#)
- SMC\_CheckConfigure
  - Драйвер модуля SMC, [264](#)
- smc\_cmd\_type
  - Драйвер модуля SMC, [263](#)
- SMC\_CmdTypeModeReg
  - Драйвер модуля SMC, [263](#)
- SMC\_CmdTypeModeRegAndUpdateRegs
  - Драйвер модуля SMC, [263](#)
- SMC\_CmdTypeUpdateRegs
  - Драйвер модуля SMC, [263](#)
- SMC\_CmdTypeUpdateRegsAndAHBCommand
  - Драйвер модуля SMC, [263](#)
- smc\_cre
  - Драйвер модуля SMC, [263](#)
- SMC\_CRE0
  - Драйвер модуля SMC, [263](#)
- SMC\_CRE1
  - Драйвер модуля SMC, [263](#)
- SMC\_DirectCmd
  - Драйвер модуля SMC, [265](#)
- smc\_incr\_to\_incr4
  - Драйвер модуля SMC, [263](#)
- SMC\_IncrToIncr4Disable
  - Драйвер модуля SMC, [263](#)
- SMC\_IncrToIncr4Enable
  - Драйвер модуля SMC, [263](#)
- smc\_packet\_lenght
  - Драйвер модуля SMC, [263](#)
- SMC\_PacketLenght1
  - Драйвер модуля SMC, [264](#)
- SMC\_PacketLenght4
  - Драйвер модуля SMC, [264](#)
- SMC\_PacketLenght8
  - Драйвер модуля SMC, [264](#)
- SMC\_PacketLenghtEndless
  - Драйвер модуля SMC, [264](#)
- SMC\_PowerSaveOff
  - Драйвер модуля SMC, [265](#)
- SMC\_PowerSaveOn
  - Драйвер модуля SMC, [266](#)
- smc\_rd\_wr\_type
  - Драйвер модуля SMC, [264](#)
- SMC\_RdWrAsync
  - Драйвер модуля SMC, [264](#)
- SMC\_RdWrSync
  - Драйвер модуля SMC, [264](#)
- SMC\_RefreshPeriod
  - Драйвер модуля SMC, [266](#)
- SMC\_SetCycles
  - Драйвер модуля SMC, [267](#)
- SMC\_SetOpmode
  - Драйвер модуля SMC, [267](#)
- smc\_status
  - Драйвер модуля SMC, [264](#)
- SMC\_Status\_InvalidArgument
  - Драйвер модуля SMC, [264](#)
- SMC\_Status\_Ok
  - Драйвер модуля SMC, [264](#)
- SMC\_UserConfig
  - Драйвер модуля SMC, [268](#)
- spi\_config\_internal\_t, [463](#)
  - frame\_width\_bits, [464](#)
  - frame\_width\_bytes, [464](#)
  - shift\_dir, [464](#)
- spi\_config\_t, [464](#)
  - baud\_rate\_bps, [465](#)
  - data\_width\_bits, [465](#)
  - direction, [465](#)
  - frame\_format, [465](#)
  - loopback\_enable, [465](#)
  - master, [465](#)
  - microwire\_cfg, [465](#)
  - motorola\_cfg, [465](#)
- SPI\_CurrentStatusInterrupts
  - Драйвер модуля SPI, [281](#)
- SPI\_Data10Bits
  - Драйвер модуля SPI, [276](#)
- SPI\_Data11Bits
  - Драйвер модуля SPI, [276](#)
- SPI\_Data12Bits
  - Драйвер модуля SPI, [277](#)
- SPI\_Data13Bits
  - Драйвер модуля SPI, [277](#)
- SPI\_Data14Bits
  - Драйвер модуля SPI, [277](#)
- SPI\_Data15Bits
  - Драйвер модуля SPI, [277](#)
- SPI\_Data16Bits
  - Драйвер модуля SPI, [277](#)
- SPI\_Data17Bits
  - Драйвер модуля SPI, [277](#)
- SPI\_Data18Bits
  - Драйвер модуля SPI, [277](#)
- SPI\_Data19Bits
  - Драйвер модуля SPI, [277](#)
- SPI\_Data20Bits
  - Драйвер модуля SPI, [277](#)
- SPI\_Data21Bits
  - Драйвер модуля SPI, [277](#)
- SPI\_Data22Bits



- Драйвер модуля SPI, [277](#)
- SPI\_Data23Bits
  - Драйвер модуля SPI, [277](#)
- SPI\_Data24Bits
  - Драйвер модуля SPI, [277](#)
- SPI\_Data25Bits
  - Драйвер модуля SPI, [277](#)
- SPI\_Data26Bits
  - Драйвер модуля SPI, [277](#)
- SPI\_Data27Bits
  - Драйвер модуля SPI, [277](#)
- SPI\_Data28Bits
  - Драйвер модуля SPI, [277](#)
- SPI\_Data29Bits
  - Драйвер модуля SPI, [277](#)
- SPI\_Data30Bits
  - Драйвер модуля SPI, [277](#)
- SPI\_Data31Bits
  - Драйвер модуля SPI, [277](#)
- SPI\_Data32Bits
  - Драйвер модуля SPI, [277](#)
- SPI\_Data4Bits
  - Драйвер модуля SPI, [276](#)
- SPI\_Data5Bits
  - Драйвер модуля SPI, [276](#)
- SPI\_Data6Bits
  - Драйвер модуля SPI, [276](#)
- SPI\_Data7Bits
  - Драйвер модуля SPI, [276](#)
- SPI\_Data8Bits
  - Драйвер модуля SPI, [276](#)
- SPI\_Data9Bits
  - Драйвер модуля SPI, [276](#)
- SPI\_Deinit
  - Драйвер модуля SPI, [281](#)
- SPI\_DisableInterrupts
  - Драйвер модуля SPI, [282](#)
- SPI\_DMADescriptorInitRX
  - Драйвер модуля SPI, [282](#)
- SPI\_DMADescriptorInitTX
  - Драйвер модуля SPI, [282](#)
- SPI\_DUMMYDATA
  - Драйвер модуля SPI, [274](#)
- SPI\_Enable
  - Драйвер модуля SPI, [283](#)
- SPI\_EnableInterrupts
  - Драйвер модуля SPI, [283](#)
- SPI\_EnableRxDMA
  - Драйвер модуля SPI, [284](#)
- SPI\_EnableTxDMA
  - Драйвер модуля SPI, [284](#)
- SPI\_FfMicrowire
  - Драйвер модуля SPI, [276](#)
- SPI\_FfMotorola
  - Драйвер модуля SPI, [276](#)
- SPI\_FfTexas
  - Драйвер модуля SPI, [276](#)
- spi\_frame\_format\_t
  - Драйвер модуля SPI, [275](#)
- spi\_frame\_width\_t
  - Драйвер модуля SPI, [276](#)
- SPI\_GetConfig
  - Драйвер модуля SPI, [284](#)
- SPI\_GetInstance
  - Драйвер модуля SPI, [284](#)
- SPI\_GetStatusFlags
  - Драйвер модуля SPI, [285](#)
- spi\_half\_duplex\_transfer\_t, [466](#)
  - rx\_data, [466](#)
  - rx\_data\_size, [466](#)
  - tx\_data, [466](#)
  - tx\_data\_size, [466](#)
- spi\_handle, [467](#)
  - callback, [467](#)
  - frame\_width\_bits, [467](#)
  - frame\_width\_bytes, [468](#)
  - instance, [468](#)
  - mode, [468](#)
  - rx\_data, [468](#)
  - rx\_remaining\_bytes, [468](#)
  - state, [468](#)
  - total\_byte\_to\_transfer, [468](#)
  - tx\_data, [468](#)
  - tx\_remaining\_bytes, [469](#)
  - user\_data, [469](#)
- spi\_interrupt\_enable
  - Драйвер модуля SPI, [277](#)
- SPI\_IRQ\_All
  - Драйвер модуля SPI, [278](#)
- SPI\_IRQ\_MultiMaster
  - Драйвер модуля SPI, [277](#)
- SPI\_IRQ\_RxFifoOverflow
  - Драйвер модуля SPI, [277](#)
- SPI\_IRQ\_RxFifoTrigger
  - Драйвер модуля SPI, [277](#)
- SPI\_IRQ\_RxFifoUnderflow
  - Драйвер модуля SPI, [277](#)
- SPI\_IRQ\_RxOnly
  - Драйвер модуля SPI, [278](#)
- SPI\_IRQ\_TxFifoOverflow
  - Драйвер модуля SPI, [277](#)
- SPI\_IRQ\_TxFifoTrigger
  - Драйвер модуля SPI, [277](#)
- SPI\_IRQ\_TxOnly
  - Драйвер модуля SPI, [278](#)
- SPI\_IsEnable
  - Драйвер модуля SPI, [285](#)
- SPI\_MasterGetDefaultConfig
  - Драйвер модуля SPI, [285](#)
- SPI\_MasterHalfDuplexTransferBlocking
  - Драйвер модуля SPI, [286](#)
- SPI\_MasterHalfDuplexTransferDMA
  - Драйвер модуля SPI, [286](#)
- SPI\_MasterHalfDuplexTransferNonBlocking
  - Драйвер модуля SPI, [287](#)
- SPI\_MasterInit

- Драйвер модуля SPI, [287](#)
- SPI\_MasterSetBaud
  - Драйвер модуля SPI, [288](#)
- SPI\_MasterTransferAbort
  - Драйвер модуля SPI, [288](#)
- SPI\_MasterTransferAbortDMA
  - Драйвер модуля SPI, [289](#)
- SPI\_MasterTransferBlocking
  - Драйвер модуля SPI, [289](#)
- SPI\_MasterTransferCreateHandle
  - Драйвер модуля SPI, [290](#)
- SPI\_MasterTransferCreateHandleDMA
  - Драйвер модуля SPI, [290](#)
- SPI\_MasterTransferDMA
  - Драйвер модуля SPI, [291](#)
- SPI\_MasterTransferGetByte
  - Драйвер модуля SPI, [291](#)
- SPI\_MasterTransferGetRemainingByte
  - Драйвер модуля SPI, [291](#)
- SPI\_MasterTransferGetTotalByte
  - Драйвер модуля SPI, [293](#)
- SPI\_MasterTransferHandleIRQ
  - Драйвер модуля SPI, [293](#)
- SPI\_MasterTransferNonBlocking
  - Драйвер модуля SPI, [293](#)
- spi\_microwire\_cfg\_t, [469](#)
  - busy\_ready\_check, [469](#)
  - ctrl\_word\_len, [469](#)
  - single\_serial, [470](#)
  - tx\_rx, [470](#)
- SPI\_MicrowireBusyReadyCheckDisable
  - Драйвер модуля SPI, [274](#)
- SPI\_MicrowireBusyReadyCheckEnable
  - Драйвер модуля SPI, [274](#)
- SPI\_MicrowireCtrlWordLen10Bit
  - Драйвер модуля SPI, [275](#)
- SPI\_MicrowireCtrlWordLen11Bit
  - Драйвер модуля SPI, [275](#)
- SPI\_MicrowireCtrlWordLen12Bit
  - Драйвер модуля SPI, [275](#)
- SPI\_MicrowireCtrlWordLen13Bit
  - Драйвер модуля SPI, [275](#)
- SPI\_MicrowireCtrlWordLen14Bit
  - Драйвер модуля SPI, [275](#)
- SPI\_MicrowireCtrlWordLen15Bit
  - Драйвер модуля SPI, [275](#)
- SPI\_MicrowireCtrlWordLen16Bit
  - Драйвер модуля SPI, [275](#)
- SPI\_MicrowireCtrlWordLen1Bit
  - Драйвер модуля SPI, [275](#)
- SPI\_MicrowireCtrlWordLen2Bit
  - Драйвер модуля SPI, [275](#)
- SPI\_MicrowireCtrlWordLen3Bit
  - Драйвер модуля SPI, [275](#)
- SPI\_MicrowireCtrlWordLen4Bit
  - Драйвер модуля SPI, [275](#)
- SPI\_MicrowireCtrlWordLen5Bit
  - Драйвер модуля SPI, [275](#)
- SPI\_MicrowireCtrlWordLen6Bit
  - Драйвер модуля SPI, [275](#)
- SPI\_MicrowireCtrlWordLen7Bit
  - Драйвер модуля SPI, [275](#)
- SPI\_MicrowireCtrlWordLen8Bit
  - Драйвер модуля SPI, [275](#)
- SPI\_MicrowireCtrlWordLen9Bit
  - Драйвер модуля SPI, [275](#)
- SPI\_MicrowireRx
  - Драйвер модуля SPI, [275](#)
- SPI\_MicrowireSerial
  - Драйвер модуля SPI, [275](#)
- SPI\_MicrowireSingle
  - Драйвер модуля SPI, [275](#)
- SPI\_MicrowireTx
  - Драйвер модуля SPI, [275](#)
- spi\_mode
  - \_qspi\_config, [401](#)
- spi\_mode\_t
  - Драйвер модуля SPI, [278](#)
- SPI\_ModeDuplex
  - Драйвер модуля SPI, [278](#)
- SPI\_ModeHalfDuplex
  - Драйвер модуля SPI, [278](#)
- SPI\_ModeSimplexRx
  - Драйвер модуля SPI, [278](#)
- SPI\_ModeSimplexTx
  - Драйвер модуля SPI, [278](#)
- spi\_motorola\_cap\_data\_t
  - Драйвер модуля SPI, [278](#)
- spi\_motorola\_cfg\_t, [470](#)
  - cap\_data, [470](#)
  - clk\_pol, [470](#)
- spi\_motorola\_clk\_pol\_t
  - Драйвер модуля SPI, [278](#)
- SPI\_MotorolaCapDataFalling
  - Драйвер модуля SPI, [278](#)
- SPI\_MotorolaCapDataRising
  - Драйвер модуля SPI, [278](#)
- SPI\_MotorolaClkPolHi
  - Драйвер модуля SPI, [279](#)
- SPI\_MotorolaClkPolLow
  - Драйвер модуля SPI, [279](#)
- SPI\_ReadData
  - Драйвер модуля SPI, [294](#)
- SPI\_Reset
  - Драйвер модуля SPI, [294](#)
- SPI\_RETRY\_TIMES
  - Драйвер модуля SPI, [274](#)
- spi\_rxfifo\_watermark\_t
  - Драйвер модуля SPI, [279](#)
- SPI\_RxFifoWatermark1
  - Драйвер модуля SPI, [279](#)
- SPI\_RxFifoWatermark2
  - Драйвер модуля SPI, [279](#)
- SPI\_RxFifoWatermark3
  - Драйвер модуля SPI, [279](#)
- SPI\_RxFifoWatermark4

- Драйвер модуля SPI, 279
- SPI\_RxFifoWatermark5
  - Драйвер модуля SPI, 279
- SPI\_RxFifoWatermark6
  - Драйвер модуля SPI, 279
- SPI\_RxFifoWatermark7
  - Драйвер модуля SPI, 279
- SPI\_RxFifoWatermark8
  - Драйвер модуля SPI, 279
- SPI\_RxFullFlag
  - Драйвер модуля SPI, 280
- SPI\_RxNotEmptyFlag
  - Драйвер модуля SPI, 280
- SPI\_SetDummyData
  - Драйвер модуля SPI, 296
- spi\_shift\_direction\_t
  - Драйвер модуля SPI, 279
- SPI\_ShiftDirLsbFirst
  - Драйвер модуля SPI, 279
- SPI\_ShiftDirMsbFirst
  - Драйвер модуля SPI, 279
- SPI\_SlaveGetDefaultConfig
  - Драйвер модуля SPI, 296
- SPI\_SlaveInit
  - Драйвер модуля SPI, 296
- SPI\_SlaveTransferAbort
  - Драйвер модуля SPI, 297
- SPI\_SlaveTransferAbortDMA
  - Драйвер модуля SPI, 297
- SPI\_SlaveTransferCreateHandle
  - Драйвер модуля SPI, 298
- SPI\_SlaveTransferCreateHandleDMA
  - Драйвер модуля SPI, 298
- SPI\_SlaveTransferDMA
  - Драйвер модуля SPI, 299
- SPI\_SlaveTransferGetByte
  - Драйвер модуля SPI, 299
- SPI\_SlaveTransferHandleIRQ
  - Драйвер модуля SPI, 299
- SPI\_SlaveTransferNonBlocking
  - Драйвер модуля SPI, 301
- spi\_status
  - Драйвер модуля SPI, 279
- SPI\_Status\_BaudrateNotSupport
  - Драйвер модуля SPI, 280
- SPI\_Status\_Busy
  - Драйвер модуля SPI, 280
- SPI\_Status\_Fail
  - Драйвер модуля SPI, 279
- spi\_status\_flags
  - Драйвер модуля SPI, 280
- SPI\_Status\_Idle
  - Драйвер модуля SPI, 280
- SPI\_Status\_IncorrectCall
  - Драйвер модуля SPI, 280
- SPI\_Status\_InvalidArgument
  - Драйвер модуля SPI, 280
- SPI\_Status\_NoTransferInProgress
  - Драйвер модуля SPI, 280
- SPI\_Status\_Ok
  - Драйвер модуля SPI, 279
- SPI\_Status\_ReadOnly
  - Драйвер модуля SPI, 279
- SPI\_Status\_RxError
  - Драйвер модуля SPI, 280
- SPI\_Status\_RxFifoBufferOverrun
  - Драйвер модуля SPI, 280
- SPI\_Status\_RxRingBufferOverrun
  - Драйвер модуля SPI, 280
- SPI\_Status\_Timeout
  - Драйвер модуля SPI, 280
- SPI\_Status\_TxError
  - Драйвер модуля SPI, 280
- SPI\_Status\_UnexpectedState
  - Драйвер модуля SPI, 280
- SPI\_TakeDownInterrupts
  - Драйвер модуля SPI, 301
- spi\_trans\_status
  - Драйвер модуля SPI, 280
- spi\_transfer\_t, 471
  - data\_size, 471
  - rx\_data, 471
  - tx\_data, 471
- SPI\_TransStatus\_Busy
  - Драйвер модуля SPI, 280
- SPI\_TransStatus\_Error
  - Драйвер модуля SPI, 280
- SPI\_TransStatus\_Error\_1
  - Драйвер модуля SPI, 280
- SPI\_TransStatus\_Error\_2
  - Драйвер модуля SPI, 280
- SPI\_TransStatus\_Error\_3
  - Драйвер модуля SPI, 280
- SPI\_TransStatus\_Error\_4
  - Драйвер модуля SPI, 280
- SPI\_TransStatus\_Idle
  - Драйвер модуля SPI, 280
- SPI\_TxEmptyFlag
  - Драйвер модуля SPI, 280
- spi\_txfifo\_watermark\_t
  - Драйвер модуля SPI, 280
- SPI\_TxFifoWatermark0
  - Драйвер модуля SPI, 281
- SPI\_TxFifoWatermark1
  - Драйвер модуля SPI, 281
- SPI\_TxFifoWatermark2
  - Драйвер модуля SPI, 281
- SPI\_TxFifoWatermark3
  - Драйвер модуля SPI, 281
- SPI\_TxFifoWatermark4
  - Драйвер модуля SPI, 281
- SPI\_TxFifoWatermark5
  - Драйвер модуля SPI, 281
- SPI\_TxFifoWatermark6
  - Драйвер модуля SPI, 281
- SPI\_TxFifoWatermark7

- Драйвер модуля SPI, 281
- SPI\_TxNotFullFlag
  - Драйвер модуля SPI, 280
- SPI\_WriteData
  - Драйвер модуля SPI, 301
- src\_addr
  - \_dma\_channel\_config, 374
  - \_dma\_multiblock\_config, 382
- src\_burst\_size
  - \_dma\_multiblock\_config, 382
  - dma\_channel\_ctl\_cfg, 417
- src\_data\_width
  - \_dma\_multiblock\_config, 382
- src\_incr
  - \_dma\_multiblock\_config, 382
  - dma\_channel\_ctl\_cfg, 417
- src\_tr\_width
  - dma\_channel\_ctl\_cfg, 417
- SSTAT
  - \_dma\_descriptor, 379
- SSTAT\_HI
  - \_dma\_channel\_reg, 377
- SSTAT\_LO
  - \_dma\_channel\_reg, 377
- SSTATAR\_HI
  - \_dma\_channel\_reg, 377
- SSTATAR\_LO
  - \_dma\_channel\_reg, 377
- standby\_configuration
  - power\_config, 427
- start\_enable
  - timer\_hardware\_config, 472
- start\_value
  - timer\_hardware\_config, 472
- state
  - \_i2c\_master\_dma\_handle, 384
  - \_i2c\_master\_handle, 385
  - \_spi\_dma\_handle, 408
  - spi\_handle, 468
- stb\_config
  - \_can\_config, 358
- stop\_bit\_count
  - uart\_config, 475
- subaddr\_buf
  - \_i2c\_master\_handle, 386
- subaddress
  - i2c\_master\_transfer\_t, 422
- subaddress\_size
  - i2c\_master\_transfer\_t, 422
- SUPPRESS\_FALL\_THROUGH\_WARNING
  - Общие определения для библиотеки HAL, 82
- sw\_forced\_long\_outa
  - pwm\_channel\_config, 441
- sw\_forced\_long\_outb
  - pwm\_channel\_config, 441
- sw\_forced\_outa
  - pwm\_channel\_config, 442
- sw\_forced\_outb
  - pwm\_channel\_config, 442
- syncphsen
  - pwm\_channel\_config, 442
- tcal
  - jtm\_config\_t, 426
- time
  - rcw\_config, 446
  - rcw\_general\_reg, 450
  - rcw\_time\_reg, 451
  - rcw\_union\_reg, 454
- time\_clk\_sel
  - rcw\_config, 446
  - rcw\_config\_reg, 449
- timeout\_cd
  - sdmmc\_port\_cfg\_t, 463
- timeout\_init
  - sdmmc\_port\_cfg\_t, 463
- TIMER\_COUNT
  - Драйвер модуля TIM, 303
- TIMER\_Debug
  - Драйвер модуля TIM, 304
- TIMER\_Deinit
  - Драйвер модуля TIM, 305
- TIMER\_GetAPIStatus
  - Драйвер модуля TIM, 305
- TIMER\_GetTicks
  - Драйвер модуля TIM, 305
- TIMER\_GetTimerHardwareValue
  - Драйвер модуля TIM, 306
- TIMER\_Hardware
  - Драйвер модуля TIM, 305
- timer\_hardware\_config, 472
  - interrupt\_enable, 472
  - reload\_value, 472
  - start\_enable, 472
  - start\_value, 472
  - work\_type, 473
- TIMER\_HARDWARE\_FIELD\_MAX
  - Драйвер модуля TIM, 303
- TIMER\_Init
  - Драйвер модуля TIM, 306
- TIMER\_IRQClear
  - Драйвер модуля TIM, 307
- TIMER\_IRQDisable
  - Драйвер модуля TIM, 307
- TIMER\_IRQEnable
  - Драйвер модуля TIM, 307
- TIMER\_IRQGetStatus
  - Драйвер модуля TIM, 308
- timer\_num\_mode
  - Драйвер модуля VTU, 338
- TIMER\_Reset
  - Драйвер модуля TIM, 308
- TIMER\_Run
  - Драйвер модуля TIM, 308
- TIMER\_SetConfig
  - Драйвер модуля TIM, 309
- TIMER\_SetTick

- Драйвер модуля TIM, [309](#)
- TIMER\_Software
  - Драйвер модуля TIM, [305](#)
- TIMER\_SOFTWARE\_FIELD\_HIGH\_OFFSET
  - Драйвер модуля TIM, [304](#)
- TIMER\_SOFTWARE\_FIELD\_MAX
  - Драйвер модуля TIM, [304](#)
- timer\_status
  - Драйвер модуля TIM, [304](#)
- TIMER\_Status\_BadConfigure
  - Драйвер модуля TIM, [304](#)
- TIMER\_Status\_InvalidArgument
  - Драйвер модуля TIM, [304](#)
- TIMER\_Status\_NotIni
  - Драйвер модуля TIM, [304](#)
- TIMER\_Status\_NotSupport
  - Драйвер модуля TIM, [304](#)
- TIMER\_Status\_Ok
  - Драйвер модуля TIM, [304](#)
- TIMER\_Status\_TimerBusy
  - Драйвер модуля TIM, [304](#)
- TIMER\_Stop
  - Драйвер модуля TIM, [310](#)
- timer\_type\_of\_counting
  - Драйвер модуля TIM, [304](#)
- TIMER\_Work
  - Драйвер модуля TIM, [304](#)
- timer\_work\_mode
  - Драйвер модуля TIM, [304](#)
- timing\_config
  - \_can\_config, [358](#)
- total\_byte\_to\_transfer
  - spi\_handle, [468](#)
- total\_size
  - sdmmc\_card\_t, [458](#)
- transfer
  - \_i2c\_master\_handle, [386](#)
  - \_i2c\_slave\_handle, [387](#)
- transfer\_count
  - \_i2c\_master\_handle, [386](#)
- transfer\_size
  - \_spi\_dma\_handle, [408](#)
- transfer\_type
  - \_dma\_multiblock\_config, [382](#)
  - dma\_channel\_ctl\_cfg, [417](#)
- transferred\_count
  - i2c\_slave\_transfer\_t, [425](#)
- transmit\_enable\_window
  - \_ttcan\_config, [410](#)
- transmit\_trigger\_pointer
  - \_ttcan\_config, [410](#)
- trigger\_time0
  - \_ttcan\_config, [410](#)
- trigger\_time1
  - \_ttcan\_config, [410](#)
- trigger\_type
  - \_ttcan\_config, [410](#)
- trim
  - rcw\_union\_reg, [454](#)
- trim\_configuration
  - power\_config, [427](#)
- trim\_lfe
  - rcw\_config, [446](#)
  - rcw\_trim\_reg, [452](#)
- trim\_lfi
  - rcw\_config, [446](#)
  - rcw\_trim\_reg, [452](#)
- trimload
  - rcw\_config, [446](#)
  - rcw\_trimload\_reg, [453](#)
  - rcw\_union\_reg, [454](#)
- trip\_unit\_action\_outa
  - pwm\_channel\_config, [442](#)
- trip\_unit\_action\_outb
  - pwm\_channel\_config, [442](#)
- ttsen
  - \_can\_tx\_buffer\_frame, [372](#)
- tx
  - \_can\_rx\_buffer\_frame, [366](#)
- tx\_data
  - i2c\_slave\_transfer\_t, [425](#)
  - spi\_half\_duplex\_transfer\_t, [466](#)
  - spi\_handle, [468](#)
  - spi\_transfer\_t, [471](#)
  - uart\_handle, [477](#)
  - uart\_transfer, [479](#)
- tx\_data\_size
  - spi\_half\_duplex\_transfer\_t, [466](#)
  - uart\_handle, [477](#)
- tx\_data\_size\_all
  - \_uart\_dma\_handle, [412](#)
  - uart\_handle, [477](#)
- tx\_desc
  - \_i2c\_master\_dma\_handle, [384](#)
  - \_spi\_dma\_handle, [409](#)
- tx\_discipline
  - \_can\_stb\_config, [368](#)
- tx\_dma
  - \_i2c\_master\_dma\_handle, [384](#)
- tx\_dma\_handle
  - \_uart\_dma\_handle, [412](#)
- tx\_frames\_prim
  - \_can\_handle, [362](#)
- tx\_frames\_sec
  - \_can\_handle, [362](#)
- tx\_handle
  - \_spi\_dma\_handle, [409](#)
- tx\_in\_progress
  - \_spi\_dma\_handle, [409](#)
- tx\_nb\_frames\_all\_prim
  - \_can\_handle, [362](#)
- tx\_nb\_frames\_all\_sec
  - \_can\_handle, [362](#)
- tx\_nb\_frames\_rest\_prim
  - \_can\_handle, [362](#)
- tx\_nb\_frames\_rest\_sec





- Драйвер модуля UART, 316
- UART\_ParityOdd
  - Драйвер модуля UART, 316
- UART\_ReadBlocking
  - Драйвер модуля UART, 322
- UART\_ReadByte
  - Драйвер модуля UART, 323
- UART\_ReadByteWait
  - Драйвер модуля UART, 323
- uart\_rs485\_active\_state
  - Драйвер модуля UART, 316
- UART\_RS485\_ActiveStateHigh
  - Драйвер модуля UART, 316
- UART\_RS485\_ActiveStateLow
  - Драйвер модуля UART, 316
- uart\_rs485\_mode
  - Драйвер модуля UART, 316
- UART\_RS485\_ModeFullDuplex
  - Драйвер модуля UART, 316
- UART\_RS485\_ModeHalfDuplexAuto
  - Драйвер модуля UART, 316
- UART\_RS485\_ModeHalfDuplexManual
  - Драйвер модуля UART, 316
- UART\_Rs485Mode
  - Драйвер модуля UART, 323
- uart\_rxfifo\_watermark
  - Драйвер модуля UART, 316
- UART\_RxFifoHalfFull
  - Драйвер модуля UART, 317
- UART\_RxFifoOneChar
  - Драйвер модуля UART, 317
- UART\_RxFifoQuarterFull
  - Драйвер модуля UART, 317
- UART\_RxFifoTwoToFull
  - Драйвер модуля UART, 317
- UART\_RxInterruptEnable
  - Драйвер модуля UART, 315
- UART\_RxLineInterruptEnable
  - Драйвер модуля UART, 315
- UART\_SetBaudRate
  - Драйвер модуля UART, 324
- UART\_SetIr
  - Драйвер модуля UART, 324
- UART\_SetLoopback
  - Драйвер модуля UART, 325
- UART\_SetRs485
  - Драйвер модуля UART, 325
- UART\_SetRxFifoWatermark
  - Драйвер модуля UART, 325
- UART\_SetTxFifoWatermark
  - Драйвер модуля UART, 325
- uart\_status
  - Драйвер модуля UART, 317
- UART\_Status\_BaudrateNotSupport
  - Драйвер модуля UART, 317
- UART\_Status\_BreakLineError
  - Драйвер модуля UART, 317
- UART\_Status\_Fail
  - Драйвер модуля UART, 317
- UART\_Status\_FramingError
  - Драйвер модуля UART, 317
- UART\_Status\_InvalidArgument
  - Драйвер модуля UART, 317
- UART\_Status\_NoTransferInProgress
  - Драйвер модуля UART, 317
- UART\_Status\_Ok
  - Драйвер модуля UART, 317
- UART\_Status\_ParityError
  - Драйвер модуля UART, 317
- UART\_Status\_ReadOnly
  - Драйвер модуля UART, 317
- UART\_Status\_RxBusy
  - Драйвер модуля UART, 317
- UART\_Status\_RxError
  - Драйвер модуля UART, 317
- UART\_Status\_RxFifoBufferOverflow
  - Драйвер модуля UART, 317
- UART\_Status\_RxIdle
  - Драйвер модуля UART, 317
- UART\_Status\_RxRingBufferOverflow
  - Драйвер модуля UART, 317
- UART\_Status\_Timeout
  - Драйвер модуля UART, 317
- UART\_Status\_TxBusy
  - Драйвер модуля UART, 317
- UART\_Status\_TxError
  - Драйвер модуля UART, 317
- UART\_Status\_TxIdle
  - Драйвер модуля UART, 317
- UART\_Status\_TxRingBufferNull
  - Драйвер модуля UART, 317
- uart\_stop\_bit\_count
  - Драйвер модуля UART, 317
- UART\_ThresholdInterruptEnable
  - Драйвер модуля UART, 315
- uart\_transfer, 478
  - data\_size, 479
  - rx\_data, 479
  - tx\_data, 479
- UART\_TransferAbortReceive
  - Драйвер модуля UART, 326
- UART\_TransferAbortReceiveDMA
  - Драйвер модуля UART, 326
- UART\_TransferAbortSend
  - Драйвер модуля UART, 326
- UART\_TransferAbortSendDMA
  - Драйвер модуля UART, 327
- UART\_TransferCreateHandle
  - Драйвер модуля UART, 327
- UART\_TransferCreateHandleDMA
  - Драйвер модуля UART, 327
- UART\_TransferGetReceiveCount
  - Драйвер модуля UART, 328
- UART\_TransferGetRxRingBufferLength
  - Драйвер модуля UART, 328
- UART\_TransferGetSendCount
  - Драйвер модуля UART, 328

- Драйвер модуля UART, 329
- UART\_TransferGetTxRingBufferLength
  - Драйвер модуля UART, 329
- UART\_TransferHandleIRQ
  - Драйвер модуля UART, 329
- UART\_TransferReceiveDMA
  - Драйвер модуля UART, 330
- UART\_TransferReceiveNonBlocking
  - Драйвер модуля UART, 330
- UART\_TransferSendDMA
  - Драйвер модуля UART, 331
- UART\_TransferSendNonBlocking
  - Драйвер модуля UART, 331
- UART\_TransferStartRingBuffer
  - Драйвер модуля UART, 332
- UART\_TransferStartTxRingBuffer
  - Драйвер модуля UART, 332
- UART\_TransferStopRingBuffer
  - Драйвер модуля UART, 333
- UART\_TransferStopTxRingBuffer
  - Драйвер модуля UART, 333
- UART\_TwoOrOneAndHalfStopBit
  - Драйвер модуля UART, 318
- uart\_txfifo\_watermark
  - Драйвер модуля UART, 318
- UART\_TxFifoEmpty
  - Драйвер модуля UART, 318
- UART\_TxFifoHalfFull
  - Драйвер модуля UART, 318
- UART\_TxFifoQuarterFull
  - Драйвер модуля UART, 318
- UART\_TxFifoTwoChars
  - Драйвер модуля UART, 318
- UART\_TxInterruptEnable
  - Драйвер модуля UART, 315
- UART\_WaitWhileActive
  - Драйвер модуля UART, 334
- UART\_WriteBlocking
  - Драйвер модуля UART, 334
- UART\_WriteByte
  - Драйвер модуля UART, 334
- UART\_WriteByteWait
  - Драйвер модуля UART, 335
- UART\_WriteTxRing
  - Драйвер модуля UART, 335
- UINT16\_MAX
  - Общие определения для библиотеки HAL, 82
- UINT32\_MAX
  - Общие определения для библиотеки HAL, 82
- UNUSED
  - Общие определения для библиотеки HAL, 82
- user\_data
  - \_can\_handle, 362
  - \_dma\_handle, 380
  - \_i2c\_master\_dma\_handle, 384
  - \_i2c\_master\_handle, 386
  - \_i2c\_slave\_handle, 387
  - \_i2s\_handle, 389
  - \_jtm\_handle, 391
  - \_spi\_dma\_handle, 409
  - \_uart\_dma\_handle, 413
  - power\_handle, 428
  - spi\_handle, 469
  - uart\_handle, 478
- vdda\_is\_lower\_threshold
  - power\_state, 431
- version
  - sdmcc\_card\_t, 459
- vlevel0
  - power\_config, 427
- vlevel1
  - power\_config, 427
- vlevel2
  - power\_config, 427
- VTU\_Capture
  - Драйвер модуля VTU, 339
- vtu\_capture\_edge\_control
  - Драйвер модуля VTU, 338
- VTU\_CaptureBothEdgeResetBothEdge
  - Драйвер модуля VTU, 338
- VTU\_CaptureBothEdgeResetFallingEdge
  - Драйвер модуля VTU, 338
- VTU\_CaptureBothEdgeResetNo
  - Драйвер модуля VTU, 338
- VTU\_CaptureBothEdgeResetRisingEdge
  - Драйвер модуля VTU, 338
- VTU\_CaptureFallingEdgeResetNo
  - Драйвер модуля VTU, 338
- VTU\_CaptureFallingEdgeResetYes
  - Драйвер модуля VTU, 338
- VTU\_CaptureRisingEdgeResetNo
  - Драйвер модуля VTU, 338
- VTU\_CaptureRisingEdgeResetYes
  - Драйвер модуля VTU, 338
- VTU\_CaptureToDTYCAPx
  - Драйвер модуля VTU, 339
- VTU\_CaptureToPERCAPx
  - Драйвер модуля VTU, 339
- VTU\_ClearTimerIRQ
  - Драйвер модуля VTU, 340
- vtu\_config, 479
  - capture\_edge\_control1, 480
  - capture\_edge\_control2, 480
  - counter, 480
  - duty\_cycle\_capture, 480
  - interrupt\_control, 480
  - mode, 480
  - period, 480
  - prescaler, 480
  - pwm\_polarity, 481
  - pwm\_polarity2, 481
- VTU\_CounterOverflow
  - Драйвер модуля VTU, 339
- VTU\_Deinit
  - Драйвер модуля VTU, 340
- VTU\_DutyCycleMatch



- Драйвер модуля VTU, [339](#)
- VTU\_EnableTimer
  - Драйвер модуля VTU, [341](#)
- VTU\_EnableTimerIRQ
  - Драйвер модуля VTU, [341](#)
- VTU\_GetCaptureEdgeCtrl
  - Драйвер модуля VTU, [341](#)
- VTU\_GetCounter
  - Драйвер модуля VTU, [342](#)
- VTU\_GetDefaultConfig
  - Драйвер модуля VTU, [342](#)
- VTU\_GetDutyCycleCapture
  - Драйвер модуля VTU, [343](#)
- VTU\_GetLastAPIStatus
  - Драйвер модуля VTU, [343](#)
- VTU\_GetPeriodCapture
  - Драйвер модуля VTU, [343](#)
- VTU\_GetPrescaler
  - Драйвер модуля VTU, [344](#)
- VTU\_GetPWMPolarity
  - Драйвер модуля VTU, [344](#)
- VTU\_GetTimerIRQ
  - Драйвер модуля VTU, [344](#)
- VTU\_HighByteDutyCycleMatch
  - Драйвер модуля VTU, [339](#)
- VTU\_HighBytePeriodMatch
  - Драйвер модуля VTU, [339](#)
- VTU\_Init
  - Драйвер модуля VTU, [345](#)
- vtu\_interrupt\_control
  - Драйвер модуля VTU, [338](#)
- VTU\_LowByteDutyCycleMatch
  - Драйвер модуля VTU, [339](#)
- VTU\_LowBytePeriodMatch
  - Драйвер модуля VTU, [339](#)
- VTU\_LowPower
  - Драйвер модуля VTU, [339](#)
- vtu\_mode
  - Драйвер модуля VTU, [339](#)
- VTU\_NoInterrupt
  - Драйвер модуля VTU, [339](#)
- VTU\_One
  - Драйвер модуля VTU, [339](#)
- VTU\_PeriodMatch
  - Драйвер модуля VTU, [339](#)
- VTU\_PWM16Bit
  - Драйвер модуля VTU, [339](#)
- vtu\_pwm\_polarity
  - Драйвер модуля VTU, [339](#)
- VTU\_PWMDual8Bit
  - Драйвер модуля VTU, [339](#)
- VTU\_SetCallback
  - Драйвер модуля VTU, [345](#)
- VTU\_SetCaptureEdgeCtrl
  - Драйвер модуля VTU, [346](#)
- VTU\_SetCounter
  - Драйвер модуля VTU, [346](#)
- VTU\_SetDutyCycleCapture
  - Драйвер модуля VTU, [346](#)
- VTU\_SetPeriodCapture
  - Драйвер модуля VTU, [347](#)
- VTU\_SetPrescaler
  - Драйвер модуля VTU, [347](#)
- VTU\_SetPWMPolarity
  - Драйвер модуля VTU, [348](#)
- vtu\_status
  - Драйвер модуля VTU, [339](#)
- VTU\_Status\_BadConfigure
  - Драйвер модуля VTU, [340](#)
- VTU\_Status\_DriverError
  - Драйвер модуля VTU, [340](#)
- VTU\_Status\_DualTimerNotCanRun
  - Драйвер модуля VTU, [340](#)
- VTU\_Status\_InvalidArgument
  - Драйвер модуля VTU, [340](#)
- VTU\_Status\_Ok
  - Драйвер модуля VTU, [340](#)
- VTU\_Status\_TimerBusy
  - Драйвер модуля VTU, [340](#)
- VTU\_Status\_TimerNotInit
  - Драйвер модуля VTU, [340](#)
- VTU\_Timer0Mode16bit
  - Драйвер модуля VTU, [338](#)
- VTU\_Timer0Mode8bit
  - Драйвер модуля VTU, [338](#)
- VTU\_Timer1Mode8bit
  - Драйвер модуля VTU, [338](#)
- VTU\_Timer2Mode16bit
  - Драйвер модуля VTU, [338](#)
- VTU\_Timer2Mode8bit
  - Драйвер модуля VTU, [338](#)
- VTU\_Timer3Mode8bit
  - Драйвер модуля VTU, [338](#)
- VTU\_Zero
  - Драйвер модуля VTU, [339](#)
- wake\_config
  - rcw\_union\_reg, [455](#)
- wake\_en
  - rcw\_config, [446](#)
  - rcw\_wake\_config\_reg, [456](#)
- wake\_in\_en
  - rcw\_config, [446](#)
  - rcw\_config\_reg, [449](#)
- wake\_pol
  - rcw\_config, [446](#)
  - rcw\_wake\_config\_reg, [456](#)
- wake\_stat1
  - rcw\_config, [447](#)
  - rcw\_config\_reg, [449](#)
- wake\_stat2
  - rcw\_trim\_reg, [452](#)
- wake\_stat3
  - rcw\_trim\_reg, [452](#)
- watch\_trigger\_time0
  - \_ttcan\_config, [411](#)
- watch\_trigger\_time1

- `_ttcan_config`, 411
- `wcal`
  - `jtm_config_t`, 426
- `WDT_ClearStatusFlags`
  - Драйвер модуля WDT, 350
- `wdt_config`, 481
  - `inten`, 481
  - `load`, 481
  - `resen`, 481
- `WDT_Deinit`
  - Драйвер модуля WDT, 351
- `WDT_Disable`
  - Драйвер модуля WDT, 351
- `WDT_Enable`
  - Драйвер модуля WDT, 351
- `WDT_GetDefaultConfig`
  - Драйвер модуля WDT, 352
- `WDT_GetLastAPIStatus`
  - Драйвер модуля WDT, 352
- `WDT_GetStatusFlagsMsk`
  - Драйвер модуля WDT, 352
- `WDT_GetStatusFlagsRaw`
  - Драйвер модуля WDT, 353
- `WDT_GetWarningValue`
  - Драйвер модуля WDT, 353
- `WDT_Init`
  - Драйвер модуля WDT, 353
- `wdt_inten_type`
  - Драйвер модуля WDT, 350
- `WDT_IntenDisable`
  - Драйвер модуля WDT, 350
- `WDT_IntenEnable`
  - Драйвер модуля WDT, 350
- `WDT_NUMBER_OF_TIMERS`
  - Драйвер модуля WDT, 350
- `WDT_Refresh`
  - Драйвер модуля WDT, 354
- `wdt_resen_type`
  - Драйвер модуля WDT, 350
- `WDT_ResenDisable`
  - Драйвер модуля WDT, 350
- `WDT_ResenEnable`
  - Драйвер модуля WDT, 350
- `WDT_SetTimeoutValue`
  - Драйвер модуля WDT, 354
- `WDT_SetWarningValue`
  - Драйвер модуля WDT, 354
- `wdt_status`
  - Драйвер модуля WDT, 350
- `WDT_Status_BadConfigure`
  - Драйвер модуля WDT, 350
- `WDT_Status_InvalidArgument`
  - Драйвер модуля WDT, 350
- `WDT_Status_Ok`
  - Драйвер модуля WDT, 350
- `WDT_Status_TimerBusy`
  - Драйвер модуля WDT, 350
- `work_type`
  - `timer_hardware_config`, 473
- `write_disable_cmd`
  - `_nor_command_set`, 392
- `write_enable_cmd`
  - `_nor_command_set`, 393
- `write_status_cmd`
  - `_nor_command_set`, 393
- `write_two_status_bytes`
  - `_qspi_nor_config`, 402
- `wtcalconf`
  - `jtm_config_t`, 426
- `wtconf`
  - `jtm_config_t`, 426
- `xip_config`
  - `_nor_handle`, 395
- `year`
  - `_rtc_datetime`, 407
- `YEAR_RANGE_END`
  - Драйвер модуля RWC, 233
- `YEAR_RANGE_START`
  - Драйвер модуля RWC, 233
- Драйвер менеджера прерываний IO устройств и DMA каналов, 156
  - `IOIM_ClearIRQHandler`, 157
  - `IOIM_ClearIRQHandler_DMA`, 157
  - `IOIM_GetIRQNumber`, 158
  - `IOIM_NA_IRQ_NUM`, 157
  - `IOIM_SetIRQHandler`, 158
  - `IOIM_SetIRQHandler_DMA`, 158
  - `IOIM_Status_NullHandler`, 157
  - `IOIM_Status_Ok`, 157
  - `ioim_status_t`, 157
  - `IOIM_Status_UnknownBase`, 157
- Драйвер модуля CAN, 9
  - `_can_bytes_in_datafield`, 14
  - `_can_flag`, 14
  - `_can_kind_of_error`, 15
  - `_can_status`, 16
  - `_can_stb_discipline`, 16
  - `_canfd_mode`, 16
  - `_ttcan_timer_prescaler`, 17
  - `_ttcan_trigger_type`, 17
  - `CAN_0ByteDatafield`, 14
  - `CAN_12ByteDatafield`, 14
  - `CAN_16ByteDatafield`, 14
  - `CAN_1ByteDatafield`, 14
  - `CAN_20ByteDatafield`, 14
  - `CAN_24ByteDatafield`, 14
  - `CAN_2ByteDatafield`, 14
  - `CAN_32ByteDatafield`, 14
  - `CAN_3ByteDatafield`, 14
  - `CAN_48ByteDatafield`, 14
  - `CAN_4ByteDatafield`, 14
  - `CAN_5ByteDatafield`, 14
  - `CAN_64ByteDatafield`, 14
  - `CAN_6ByteDatafield`, 14

- CAN\_7ByteDatafield, [14](#)
- CAN\_8ByteDatafield, [14](#)
- CAN\_AbortPrimaryTxBuffer, [17](#)
- CAN\_AbortSecondaryTxBuffer, [18](#)
- CAN\_BoschFd, [17](#)
- CAN\_CalculateImprovedTimingValues, [18](#)
- CAN\_ClearStatusFlag, [18](#)
- CAN\_ClearStatusFlagMask, [19](#)
- CAN\_Deinit, [19](#)
- CAN\_DisableInterrupt, [19](#)
- CAN\_DisableInterruptMask, [20](#)
- CAN\_EnableInterrupt, [20](#)
- CAN\_EnableInterruptMask, [20](#)
- CAN\_EnterNormalMode, [21](#)
- CAN\_EnterStandbyMode, [21](#)
- CAN\_ErrorAckError, [15](#)
- CAN\_ErrorBitError, [15](#)
- CAN\_ErrorCrcError, [16](#)
- CAN\_ErrorFormError, [15](#)
- CAN\_ErrorNone, [15](#)
- CAN\_ErrorOtherError, [16](#)
- CAN\_ErrorReserved, [16](#)
- CAN\_ErrorStuffError, [15](#)
- CAN\_Fifo, [16](#)
- can\_flag\_t, [14](#)
- CAN\_FlagAbortState, [15](#)
- CAN\_FlagArbitrationLost, [15](#)
- CAN\_FlagBusError, [15](#)
- CAN\_FlagError, [15](#)
- CAN\_FlagErrorPassiveInterrupt, [15](#)
- CAN\_FlagErrorPassiveState, [15](#)
- CAN\_FlagErrorWarningState, [15](#)
- CAN\_FlagRBAAlmostFull, [15](#)
- CAN\_FlagRBFfull, [15](#)
- CAN\_FlagRBOverrun, [15](#)
- CAN\_FlagReceive, [15](#)
- CAN\_FlagsNumber, [15](#)
- CAN\_FlagTimeTriggered, [15](#)
- CAN\_FlagTransmissionPrimary, [15](#)
- CAN\_FlagTransmissionSecondary, [15](#)
- CAN\_FlagTransmitBufferFull, [15](#)
- CAN\_FlagTriggerError, [15](#)
- CAN\_FlagWatchTriggerError, [15](#)
- CAN\_GetDefaultConfig, [21](#)
- CAN\_GetEnabledInterruptMask, [21](#)
- CAN\_GetStatusFlag, [22](#)
- CAN\_GetStatusFlagMask, [22](#)
- CAN\_Init, [22](#)
- CAN\_IsInterruptEnabled, [23](#)
- CAN\_IsoFd, [17](#)
- CAN\_IsPrimaryTransmitRequestPending, [23](#)
- CAN\_IsRxBufferAlmostFull, [23](#)
- CAN\_IsRxBufferEmpty, [24](#)
- CAN\_IsRxBufferFull, [24](#)
- CAN\_IsSecondaryTransmitRequestPending, [24](#)
- CAN\_IsSecondaryTxBufferEmpty, [25](#)
- CAN\_IsSecondaryTxBufferFull, [25](#)
- CAN\_IsSecondaryTxBufferMoreThanHalfFull, [25](#)
- can\_kind\_of\_error\_t, [14](#)
- CAN\_Priority, [16](#)
- CAN\_ReadRxBuffer, [26](#)
- CAN\_SetArbitrationTimingConfig, [26](#)
- CAN\_SetFilterConfig, [26](#)
- CAN\_SetPrimaryTxBufferConfig, [27](#)
- CAN\_SetRxBufferConfig, [27](#)
- CAN\_SetSecondaryTxBufferConfig, [27](#)
- CAN\_SetTTCANConfig, [27](#)
- CAN\_Status\_Fail, [16](#)
- CAN\_Status\_InvalidArgument, [16](#)
- CAN\_Status\_Ok, [16](#)
- CAN\_Status\_RxBusy, [16](#)
- CAN\_Status\_RxEmpty, [16](#)
- CAN\_Status\_RxIdle, [16](#)
- CAN\_Status\_TxBusy, [16](#)
- CAN\_Status\_TxIdle, [16](#)
- CAN\_TransferAbortReceive, [28](#)
- CAN\_TransferAbortSendPrimary, [28](#)
- CAN\_TransferAbortSendSecondary, [28](#)
- CAN\_TransferCreateHandle, [28](#)
- CAN\_TransferGetReceivedCount, [29](#)
- CAN\_TransferGetSentPrimaryCount, [29](#)
- CAN\_TransferGetSentSecondaryCount, [30](#)
- CAN\_TransferHandleIRQ, [30](#)
- CAN\_TransferReceiveFifoBlocking, [30](#)
- CAN\_TransferReceiveFifoNonBlocking, [31](#)
- CAN\_TransferSendPrimaryBlocking, [31](#)
- CAN\_TransferSendPrimaryNonBlocking, [32](#)
- CAN\_TransferSendSecondaryBlocking, [32](#)
- CAN\_TransferSendSecondaryNonBlocking, [33](#)
- CAN\_TriggerImmediate, [17](#)
- CAN\_TriggerSingleShotTransmit, [17](#)
- CAN\_TriggerTime, [17](#)
- CAN\_TriggerTransmitStart, [17](#)
- CAN\_TriggerTransmitStop, [17](#)
- CAN\_TTCANDiv1, [17](#)
- CAN\_TTCANDiv2, [17](#)
- CAN\_TTCANDiv4, [17](#)
- CAN\_TTCANDiv8, [17](#)
- CAN\_WritePrimaryTxBuffer, [33](#)
- CAN\_WriteSecondaryTxBuffer, [34](#)
- Драйвер модуля CLKCTR, [34](#)
- CLKCTR\_BasePikClkForce, [47](#)
- clkctr\_clk\_force\_type, [46](#)
- CLKCTR\_ClkForceTypeDynamic, [47](#)
- CLKCTR\_ClkForceTypeForce, [47](#)
- CLKCTR\_CPUDBGPikClkForce, [47](#)
- CLKCTR\_CPUFClkForce, [47](#)
- CLKCTR\_CPUSysClkForce, [47](#)
- CLKCTR\_CryptoSysClkForce, [47](#)
- clkctr\_device\_clk\_force, [47](#)
- clkctr\_extern\_freq, [47](#)
- CLKCTR\_ExternFreqHFI, [47](#)
- CLKCTR\_ExternFreqI2SExtClk, [47](#)

- CLKCTR\_ExternFreqLFI, 47
- CLKCTR\_ExternFreqXTI, 47
- CLKCTR\_ExternFreqXTI32, 47
- CLKCTR\_FCLK\_MAX, 40
- CLKCTR\_FCLK\_MIN, 40
- CLKCTR\_FREQ\_NOT\_SET, 40
- CLKCTR\_GetAllDiv, 50
- CLKCTR\_GetClk, 51
- CLKCTR\_GetClkForce, 51
- CLKCTR\_GetDivClk, 51
- CLKCTR\_GetFClk, 52
- CLKCTR\_GetFClkDiv, 52
- CLKCTR\_GetFClkInt, 52
- CLKCTR\_GetFrequency, 53
- CLKCTR\_GetGNSSClk, 53
- CLKCTR\_GetGNSSClkDiv, 53
- CLKCTR\_GetHFI, 54
- CLKCTR\_GetHFIClk, 54
- CLKCTR\_GetHFIClkForMainClk, 54
- CLKCTR\_GetHFITrim, 55
- CLKCTR\_GetI2SClk, 55
- CLKCTR\_GetI2SClkDiv, 55
- CLKCTR\_GetI2SExtClk, 56
- CLKCTR\_GetLFI, 56
- CLKCTR\_GetLPClk, 56
- CLKCTR\_GetMainClk, 57
- CLKCTR\_GetMCOClk, 57
- CLKCTR\_GetMCODiv, 57
- CLKCTR\_GetMCOEn, 57
- CLKCTR\_GetPLLClk, 58
- CLKCTR\_GetPLLConfig, 58
- CLKCTR\_GetPLLRef, 58
- CLKCTR\_GetQSPIClk, 59
- CLKCTR\_GetQSPIClkDiv, 59
- CLKCTR\_GetRTCClk, 59
- CLKCTR\_GetSwitchClk, 60
- CLKCTR\_GetSwitchI2SClk, 60
- CLKCTR\_GetSwitchLPClk, 60
- CLKCTR\_GetSwitchMainClk, 61
- CLKCTR\_GetSwitchMCOClk, 61
- CLKCTR\_GetSwitchPLLRef, 61
- CLKCTR\_GetSwitchPMUDIS, 62
- CLKCTR\_GetSwitchRTCClk, 62
- CLKCTR\_GetSwitchUSBClk, 62
- CLKCTR\_GetSysClk, 63
- CLKCTR\_GetSysClkDiv, 63
- CLKCTR\_GetUSBClk, 63
- CLKCTR\_GetXTI, 64
- CLKCTR\_GetXTI32, 64
- CLKCTR\_GetXTIClk, 64
- CLKCTR\_GMSSysClkForce, 47
- CLKCTR\_GNSSCLK\_MAX, 40
- CLKCTR\_GNSSCLK\_MIN, 41
- clkctr\_hfi\_for\_main\_type, 47
- CLKCTR\_HFI\_MAX, 41
- CLKCTR\_HFI\_MIN, 41
- CLKCTR\_HFIForMainTypeHFI, 48
- CLKCTR\_HFIForMainTypeXTI, 48
- CLKCTR\_I2S\_EXTCLK\_MAX, 41
- CLKCTR\_I2S\_EXTCLK\_MIN, 41
- clkctr\_i2sclk, 48
- CLKCTR\_I2SCLK\_MAX, 41
- CLKCTR\_I2SCLK\_MIN, 41
- CLKCTR\_I2SClkTypeI2SClk, 48
- CLKCTR\_I2SClkTypeSysClk, 48
- clkctr\_int\_freq, 48
- CLKCTR\_IntFreqFClk, 48
- CLKCTR\_IntFreqFClkInt, 48
- CLKCTR\_IntFreqGNSSClk, 48
- CLKCTR\_IntFreqHFIClk, 48
- CLKCTR\_IntFreqHFIForMain, 48
- CLKCTR\_IntFreqI2SClk, 48
- CLKCTR\_IntFreqLPClk, 48
- CLKCTR\_IntFreqMainClk, 48
- CLKCTR\_IntFreqMCOClk, 48
- CLKCTR\_IntFreqPLLClk, 48
- CLKCTR\_IntFreqPLLRefClk, 48
- CLKCTR\_IntFreqQSPIClk, 48
- CLKCTR\_IntFreqRTCClk, 48
- CLKCTR\_IntFreqSysClk, 48
- CLKCTR\_IntFreqUSBClk, 48
- CLKCTR\_IntFreqXTIClk, 48
- CLKCTR\_LFI\_MAX, 41
- CLKCTR\_LFI\_MIN, 42
- clkctr\_lpcclk, 48
- CLKCTR\_LPClkType500, 49
- CLKCTR\_LPClkTypeRTCClk, 49
- clkctr\_mainclk, 49
- CLKCTR\_MainClkTypeHFIClk, 49
- CLKCTR\_MainClkTypeMax, 49
- CLKCTR\_MainClkTypePLLClk, 49
- CLKCTR\_MainClkTypeXTIClk, 49
- CLKCTR\_MAN\_MAX, 42
- CLKCTR\_MAX\_FCLK\_DIV, 42
- CLKCTR\_MAX\_GNSSCLK\_DIV, 42
- CLKCTR\_MAX\_I2SCLK\_DIV, 42
- CLKCTR\_MAX\_MCOCLK\_DIV, 42
- CLKCTR\_MAX\_QSPICLK\_DIV, 42
- CLKCTR\_MAX\_SYSCLK\_DIV, 43
- clkctr\_mcoclk, 49
- CLKCTR\_MCOClkTypeFCIk, 49
- CLKCTR\_MCOClkTypeFCIkInt, 49
- CLKCTR\_MCOClkTypeHFIClk, 49
- CLKCTR\_MCOClkTypeLPClk, 49
- CLKCTR\_MCOClkTypeMainClk, 49
- CLKCTR\_MCOClkTypePLLClk, 49
- CLKCTR\_MCOClkTypeRTCClk, 49
- CLKCTR\_MCOClkTypeSysClk, 49
- CLKCTR\_MIN\_FCLK\_DIV, 43
- CLKCTR\_MIN\_GNSSCLK\_DIV, 43
- CLKCTR\_MIN\_I2SCLK\_DIV, 43
- CLKCTR\_MIN\_MCOCLK\_DIV, 43
- CLKCTR\_MIN\_QSPICLK\_DIV, 43
- CLKCTR\_MIN\_SYSCLK\_DIV, 43
- CLKCTR\_NF\_MAN\_MAX, 43
- CLKCTR\_NR\_MAN\_MAX, 44

- CLKCTR\_OD\_MAN\_MAX, 44
- CLKCTR\_PLLCLK\_MAX, 44
- CLKCTR\_PLLCLK\_MIN, 44
- CLKCTR\_PLLCLK\_OD\_MAX, 44
- CLKCTR\_PLLCLK\_OD\_MIN, 44
- clkctr\_pllref, 49
- CLKCTR\_PLLREF\_MAX, 45
- CLKCTR\_PLLREF\_MIN, 45
- CLKCTR\_PLLRefTypeHFCIk, 49
- CLKCTR\_PLLRefTypeXTICIk, 49
- CLKCTR\_QSPICLK\_MAX, 45
- CLKCTR\_QSPICLK\_MIN, 45
- clkctr\_rtccIk, 49
- CLKCTR\_RTCCIkTypeLFE, 50
- CLKCTR\_RTCCIkTypeLFI, 50
- CLKCTR\_SEL\_MAX, 45
- CLKCTR\_SetClkForce, 65
- CLKCTR\_SetDivClk, 65
- CLKCTR\_SetFCIkDiv, 65
- CLKCTR\_SetFrequency, 66
- CLKCTR\_SetGNSSClkDiv, 66
- CLKCTR\_SetHFI, 67
- CLKCTR\_SetHFITrim, 67
- CLKCTR\_SetI2SCLkDiv, 68
- CLKCTR\_SetI2SExtClk, 68
- CLKCTR\_SetLFI, 69
- CLKCTR\_SetMCOdiv, 69
- CLKCTR\_SetMCOEn, 70
- CLKCTR\_SetPll, 70
- CLKCTR\_SetPLLConfig, 70
- CLKCTR\_SetPllMan, 71
- CLKCTR\_SetQSPIClkDiv, 72
- CLKCTR\_SetSwitchClk, 72
- CLKCTR\_SetSwitchI2SCLk, 72
- CLKCTR\_SetSwitchLPCLk, 73
- CLKCTR\_SetSwitchMainClk, 73
- CLKCTR\_SetSwitchMCOClk, 74
- CLKCTR\_SetSwitchPLLRef, 74
- CLKCTR\_SetSwitchPMUDIS, 75
- CLKCTR\_SetSwitchRTCCIk, 76
- CLKCTR\_SetSwitchUSBCIk, 76
- CLKCTR\_SetSysClkDiv, 77
- CLKCTR\_SetSysDiv, 77
- CLKCTR\_SetXTI, 77
- CLKCTR\_SetXTI32, 78
- CLKCTR\_SMCCIkForce, 47
- CLKCTR\_SRAMFCIkForce, 47
- CLKCTR\_SRAMSysClkForce, 47
- clkctr\_status, 50
- CLKCTR\_Status\_CheckError, 50
- CLKCTR\_Status\_ConfigureError, 50
- CLKCTR\_Status\_InvalidArgument, 50
- CLKCTR\_Status\_Ok, 50
- CLKCTR\_Status\_VerifyError, 50
- CLKCTR\_SYSCLK\_MAX, 45
- CLKCTR\_SYSCLK\_MIN, 45
- CLKCTR\_SysFCIkForce, 47
- CLKCTR\_SysSysClkForce, 47
- clkctr\_usbcIk, 50
- CLKCTR\_USBCIkTypeHFCIk, 50
- CLKCTR\_USBCIkTypeXTICIk, 50
- CLKCTR\_XTI32\_MAX, 46
- CLKCTR\_XTI32\_MIN, 46
- CLKCTR\_XTI\_MAX, 46
- CLKCTR\_XTI\_MIN, 46
- HFI\_FREQUENCY, 46
- PLL\_MAX\_MULTIPLIER, 46
- PLL\_MIN\_MULTIPLIER, 46
- Драйвер модуля DMA, 83
  - \_dma\_int, 89
  - \_dma\_priority, 89
  - \_dma\_status, 90
  - \_dma\_transfer\_type, 90
  - DMA\_AbortTransfer, 91
  - DMA\_AllIRQ, 89
  - DMA\_BurstSize1, 89
  - DMA\_BurstSize128, 89
  - DMA\_BurstSize16, 89
  - DMA\_BurstSize256, 89
  - DMA\_BurstSize32, 89
  - DMA\_BurstSize4, 89
  - DMA\_BurstSize64, 89
  - DMA\_BurstSize8, 89
  - DMA\_CHANNEL\_CTL, 87
  - DMA\_CHANNEL\_CTL\_BLOCKSIZE\_MASK, 88
  - DMA\_CHANNEL\_CTL\_BLOCKSIZE\_SHIFT, 88
  - DMA\_ChannelIRQHandle, 91
  - DMA\_ChannelIsActive, 91
  - DMA\_ChannelPriority0, 90
  - DMA\_ChannelPriority1, 90
  - DMA\_ChannelPriority2, 90
  - DMA\_ChannelPriority3, 90
  - DMA\_ChannelPriority4, 90
  - DMA\_ChannelPriority5, 90
  - DMA\_ChannelPriority6, 90
  - DMA\_ChannelPriority7, 90
  - DMA\_CreateHandle, 91
  - DMA\_Decr, 88
  - DMA\_Deinit, 92
  - DMA\_DisableChannel, 92
  - DMA\_DisableChannelInterrupt, 92
  - DMA\_EnableChannel, 92
  - DMA\_EnableChannelInterrupt, 93
  - DMA\_EnableChannelPeriphRq, 93
  - DMA\_GetChannelPriority, 93
  - DMA\_GetCTLCfgMask, 94
  - DMA\_GetDescriptorCount, 94
  - DMA\_HardwareHandshakeEnable, 94
  - DMA\_Incr, 88
  - DMA\_Init, 95
  - DMA\_InitMultiblockDescriptor, 95
  - DMA\_IntBlock, 89
  - DMA\_IntDstTran, 89
  - DMA\_IntError, 89



- DMA\_IntSrcTran, 89
- DMA\_IntTfr, 89
- DMA\_MemoryToMemory\_DMA, 90
- DMA\_MemoryToPeripheral\_DMA, 90
- DMA\_MemoryToPeripheral\_Peripheral, 90
- DMA\_NoChange, 88
- DMA\_PeripheralToMemory\_DMA, 90
- DMA\_PeripheralToMemory\_Peripheral, 90
- DMA\_PeripheralToPeripheral\_DMA, 90
- DMA\_PeripheralToPeripheral\_DST, 90
- DMA\_PeripheralToPeripheral\_SRC, 90
- DMA\_PrepareChannelTransfer, 95
- DMA\_ScatterGatherEnable, 96
- DMA\_SetCallback, 96
- DMA\_SetChannelPriority, 96
- DMA\_SetupDescriptor, 97
- DMA\_SetupDstScatter, 97
- DMA\_SetupSrcGather, 97
- DMA\_StartTransfer, 98
- DMA\_Status\_Busy, 90
- DMA\_Status\_Fail, 90
- DMA\_Status\_NoBase, 90
- DMA\_Status\_Success, 90
- DMA\_SubmitChannelDescriptor, 98
- DMA\_SubmitChannelTransfer, 98
- DMA\_SubmitChannelTransferParameter, 99
- DMA\_Transfer128BitWidth, 89
- DMA\_Transfer16BitWidth, 89
- DMA\_Transfer256BitWidth, 89
- DMA\_Transfer32BitWidth, 89
- DMA\_Transfer64BitWidth, 89
- DMA\_Transfer8BitWidth, 89
- Драйвер модуля DTIM, 100
  - DUALTIMER\_Deinit, 103
  - DUALTIMER\_Disable, 103
  - DUALTIMER\_Enable, 103
  - DUALTIMER\_FreeRunning, 102
  - DUALTIMER\_GetAPIStatus, 104
  - DUALTIMER\_GetDefaultConfig, 104
  - DUALTIMER\_GetRawStatus, 104
  - DUALTIMER\_GetStatus, 105
  - DUALTIMER\_GetTick, 105
  - DUALTIMER\_Init, 105
  - dualtimer\_interrupt\_control, 101
  - DUALTIMER\_InterruptDisable, 102
  - DUALTIMER\_InterruptEnable, 102
  - DUALTIMER\_IrqClr, 106
  - DUALTIMER\_MAX\_INDEX, 101
  - dualtimer\_mode, 102
  - DUALTIMER\_NUMBER\_OF\_DUALTIMERS, 101
  - dualtimer\_number\_of\_repetitions, 102
  - DUALTIMER\_OneShot, 102
  - DUALTIMER\_Periodic, 102
  - dualtimer\_prescale, 102
  - DUALTIMER\_Prescale1, 102
  - DUALTIMER\_Prescale16, 102
  - DUALTIMER\_Prescale256, 102
  - DUALTIMER\_Reload, 106
  - DUALTIMER\_Run, 107
  - dualtimer\_status, 102
  - DUALTIMER\_Status\_BadConfigure, 103
  - DUALTIMER\_Status\_InvalidArgument, 103
  - DUALTIMER\_Status\_Ok, 103
  - DUALTIMER\_Status\_TimerBusy, 103
  - DUALTIMER\_Stop, 107
  - dualtimer\_timer\_size, 103
  - DUALTIMER\_TimerSize16, 103
  - DUALTIMER\_TimerSize32, 103
  - dualtimer\_work\_enable, 103
  - DUALTIMER\_WrappingMode, 102
- Драйвер модуля FLASH., 107
  - FCTR\_CMD\_ERASE, 109
  - FCTR\_CMD\_MASS\_ERASE, 109
  - FCTR\_CMD\_READ, 109
  - FCTR\_CMD\_ROW\_WRITE, 109
  - FCTR\_CMD\_WRITE, 109
  - FCTR\_IRQ\_STS\_CLR\_SUCCESS\_FLAGS, 109
  - FCTR\_IRQ\_STS\_SET\_RESULT\_FLAGS, 109
  - FLASH\_Erase, 111
  - FLASH\_Init, 111
  - FLASH\_MainRegion, 110
  - FLASH\_MassErase, 112
  - FLASH\_Program, 112
  - FLASH\_Read, 113
  - flash\_region, 110
  - flash\_status, 110
  - FLASH\_Status\_AddressAlignmentError, 111
  - FLASH\_Status\_AddressOutOfRange, 111
  - FLASH\_Status\_CheckError, 111
  - FLASH\_Status\_ConfigureError, 111
  - FLASH\_Status\_InvalidArgument, 111
  - FLASH\_Status\_Ok, 111
  - FLASH\_Status\_VerifyError, 111
  - FLASH\_SystemRegion, 110
  - FLASH\_TEST\_ADDRESSES, 110
  - FLASH\_VerifyErase, 113
  - FLASH\_VerifyProgram, 114
  - FLASH\_WriteWord, 115
- Драйвер модуля GPIO, 116
  - GPIO\_ALT\_FUNC\_CAN\_GNSS\_USB, 118
  - GPIO\_ALT\_FUNC\_I2C\_I2S, 118
  - GPIO\_ALT\_FUNC\_PWM\_VTU, 118
  - GPIO\_ALT\_FUNC\_QSPI\_SPI2, 118
  - GPIO\_ALT\_FUNC\_SDMMC\_SMC, 118
  - GPIO\_ALT\_FUNC\_SPI0\_SPI1, 118
  - GPIO\_ALT\_FUNC\_TRACE\_JTAG\_FBIST, 118
  - GPIO\_ALT\_FUNC\_UART, 118
  - GPIO\_MODE\_AF, 118
  - GPIO\_MODE\_GPIO, 118
  - GPIO\_MODE\_HI\_Z, 118
  - GPIO\_MODE\_INVALID, 118
  - gpio\_mode\_t, 117

- gpio\_pin\_function\_t, 118
- GPIO\_PORTA, 116
- GPIO\_PORTB, 116
- GPIO\_PORTC, 116
- GPIO\_PORTD, 117
- GPIO\_PORTPIN, 117
- GPIO\_PORTPIN\_GET\_MASK, 117
- GPIO\_PORTPIN\_GET\_PIN\_NUM, 117
- GPIO\_PORTPIN\_GET\_PORT\_NUM, 117
- Драйвер модуля I2C, 118
  - I2C\_Abort\_10B\_Addr1\_Nack, 124
  - I2C\_Abort\_10B\_Addr2\_Nack, 124
  - I2C\_Abort\_7B\_Addr\_Nack, 124
  - I2C\_Abort\_Arbitr\_Lost, 125
  - i2c\_abort\_flags, 124
  - I2C\_Abort\_GenCall\_Nack, 124
  - I2C\_Abort\_GenCall\_Read, 124
  - I2C\_Abort\_HS\_RStart\_Dis, 125
  - I2C\_Abort\_HsCode\_Ack, 124
  - I2C\_Abort\_Master\_Dis, 125
  - I2C\_Abort\_MasterDetect, 125
  - I2C\_Abort\_Read\_RStart\_Dis, 125
  - I2C\_Abort\_RStart\_Dis, 125
  - I2C\_Abort\_RxFifo\_NotEmpty, 125
  - I2C\_Abort\_SlaveArbitr\_Lost, 125
  - I2C\_Abort\_SlaveDataCmd\_Error, 125
  - I2C\_Abort\_StartByte\_Ack, 124
  - I2C\_Abort\_Tx\_FlushCnt, 125
  - I2C\_Abort\_TxData\_Nack, 124
  - i2c\_addr\_size\_t, 125
  - I2C\_Address10Bit, 125
  - I2C\_Address7Bit, 125
  - I2C\_CFG\_MASK, 122
  - I2C\_ClearAllInterrupts, 130
  - I2C\_ClearInterrupt, 130
  - i2c\_direction\_t, 125
  - I2C\_DisableInterrupts, 131
  - I2C\_DMADescriptorInitRX, 131
  - I2C\_DMADescriptorInitTX, 131
  - I2C\_Enable, 132
  - I2C\_EnableInterrupts, 133
  - I2C\_FastSpeedMode, 129
  - I2C\_GetEnabledInterrupts, 133
  - I2C\_GetInstance, 133
  - I2C\_HighSpeedMode, 129
  - i2c\_interrupt, 125
  - I2C\_IRQ\_Activity, 127
  - I2C\_IRQ\_GenCall, 127
  - I2C\_IRQ\_RdReq, 126
  - I2C\_IRQ\_RxDone, 126
  - I2C\_IRQ\_RxFull, 126
  - I2C\_IRQ\_RxOver, 126
  - I2C\_IRQ\_RxUnder, 126
  - I2C\_IRQ\_StartDet, 127
  - I2C\_IRQ\_StopDet, 127
  - I2C\_IRQ\_TxAbrt, 126
  - I2C\_IRQ\_TxEmpty, 126
  - I2C\_IRQ\_TxOver, 126
  - I2C\_IsEnable, 135
  - i2c\_master\_transfer\_callback\_t, 123
  - i2c\_master\_transfer\_flags\_t, 127
  - i2c\_master\_transfer\_states, 127
  - I2C\_MASTER\_TRANSMIT\_IGNORE\_LAST\_NACK, 122
  - I2C\_MasterAddrSet, 135
  - I2C\_MasterDeinit, 135
  - I2C\_MasterGetBusActiveState, 136
  - I2C\_MasterGetDefaultConfig, 136
  - I2C\_MasterInit, 136
  - I2C\_MasterReadBlocking, 137
  - I2C\_MasterSetBaudRate, 138
  - I2C\_MasterTransferAbort, 139
  - I2C\_MasterTransferAbortDMA, 139
  - I2C\_MasterTransferBlocking, 139
  - I2C\_MasterTransferCreateHandle, 140
  - I2C\_MasterTransferCreateHandleDMA, 141
  - I2C\_MasterTransferDMA, 141
  - I2C\_MasterTransferGetCount, 142
  - I2C\_MasterTransferHandleIRQ, 142
  - I2C\_MasterTransferNonBlocking, 142
  - I2C\_MasterWriteBlocking, 143
  - I2C\_MTS\_Idle, 127
  - I2C\_MTS\_ReceiveData, 127
  - I2C\_MTS\_ReceiveDataBegin, 127
  - I2C\_MTS\_ReceiveLastData, 127
  - I2C\_MTS\_Start, 127
  - I2C\_MTS\_Stop, 127
  - I2C\_MTS\_TransmitData, 127
  - I2C\_MTS\_TransmitSubaddr, 127
  - I2C\_MTS\_WaitForCompletion, 127
  - I2C\_Read, 125
  - I2C\_Reset, 144
  - I2C\_RETRY\_TIMES, 122
  - i2c\_slave\_event\_transfer\_t, 127
  - i2c\_slave\_fsm\_t, 128
  - i2c\_slave\_transfer\_callback\_t, 124
  - I2C\_SlaveEvent\_Completion, 128
  - I2C\_SlaveEvent\_MatchAddrGen, 128
  - I2C\_SlaveEvent\_MatchAddrSlave, 128
  - I2C\_SlaveEvent\_Receive, 128
  - I2C\_SlaveEvent\_Transmit, 128
  - I2C\_SlaveEvents\_All, 128
  - I2C\_SlaveEvents\_Ordinary, 128
  - I2C\_SlaveFsm\_Idle, 128
  - I2C\_SlaveFsm\_Init, 128
  - I2C\_SlaveFsm\_Receive, 128
  - I2C\_SlaveFsm\_Stop, 128
  - I2C\_SlaveFsm\_Transmit, 128
  - i2c\_speed\_mode\_t, 128
  - I2C\_StandardSpeedMode, 129
  - I2C\_Stat\_Active, 129
  - I2C\_Stat\_Master\_Active, 129
  - I2C\_STAT\_MSTCODE\_IDLE, 122
  - I2C\_STAT\_MSTCODE\_NACKADR, 123
  - I2C\_STAT\_MSTCODE\_NACKDAT, 123
  - I2C\_STAT\_MSTCODE\_RXREADY, 123

- I2C\_STAT\_MSTCODE\_TXREADY, 123
- I2C\_Stat\_RxFifo\_Full, 129
- I2C\_Stat\_RxFifo\_NotEmpty, 129
- I2C\_Stat\_Slave\_Active, 129
- I2C\_STAT\_SLVST\_ADDR, 123
- I2C\_STAT\_SLVST\_RX, 123
- I2C\_STAT\_SLVST\_TX, 123
- I2C\_Stat\_TxFifo\_Empty, 129
- I2C\_Stat\_TxFifo\_NotFull, 129
- I2C\_Status\_AddrNack, 129
- I2C\_Status\_ArbitrationLost, 129
- I2C\_Status\_BreakTransfer, 129
- I2C\_Status\_Busy, 129
- I2C\_Status\_DmaRequestFail, 129
- i2c\_status\_flags, 129
- I2C\_Status\_HsCodeError, 130
- I2C\_Status\_HwError, 129
- I2C\_Status\_Idle, 129
- I2C\_Status\_InvalidParameter, 129
- I2C\_Status\_Nack, 129
- I2C\_Status\_NoTransferInProgress, 129
- I2C\_Status\_Ok, 129
- I2C\_Status\_SetStartError, 129
- i2c\_status\_t, 129
- I2C\_Status\_Timeout, 129
- I2C\_Status\_UnDef, 130
- I2C\_Status\_UnexpectedState, 130
- I2C\_Status\_UnexpectedState1, 130
- I2C\_Status\_UnexpectedState2, 130
- I2C\_Status\_UnexpectedState3, 130
- I2C\_Status\_UnexpectedState4, 130
- I2C\_Status\_UnexpectedState5, 130
- I2C\_Status\_UnexpectedState6, 130
- I2C\_Status\_UnexpectedState7, 130
- I2C\_Status\_UnexpectedState8, 130
- I2C\_Status\_UnexpectedState9, 130
- I2C\_Status\_UserError, 129
- I2C\_TransferDataFlag, 127
- I2C\_TransferReStartFlag, 127
- I2C\_TransferStartFlag, 127
- I2C\_TransferStopFlag, 127
- I2C\_UndefinedSpeedMode, 129
- I2C\_Write, 125
- Драйвер модуля I2S, 144
  - \_i2s\_flag, 148
  - \_i2s\_interrupt\_level, 148
  - \_i2s\_resolution, 148
  - \_i2s\_sclk\_gating, 149
  - \_i2s\_sclk\_per\_sample, 149
  - \_i2s\_status, 149
- I2S\_AbortTx, 150
- I2S\_ClearDataOverrunFlag, 150
- I2S\_Deinit, 150
- I2S\_DisableInterrupt, 150
- I2S\_DisableInterruptMask, 151
- I2S\_DisableTx, 151
- I2S\_EnableInterrupt, 151
- I2S\_EnableInterruptMask, 151
- I2S\_EnableTx, 152
- I2S\_FlagTxFifoEmpty, 148
- I2S\_FlagTxFifoOverrun, 148
- I2S\_GetDefaultConfig, 152
- I2S\_GetEnabledInterruptMask, 152
- I2S\_GetInterruptStatus, 152
- I2S\_GetInterruptStatusMask, 153
- I2S\_Init, 153
- I2S\_InterruptTriggerLevel\_1, 148
- I2S\_InterruptTriggerLevel\_10, 148
- I2S\_InterruptTriggerLevel\_11, 148
- I2S\_InterruptTriggerLevel\_12, 148
- I2S\_InterruptTriggerLevel\_13, 148
- I2S\_InterruptTriggerLevel\_14, 148
- I2S\_InterruptTriggerLevel\_15, 148
- I2S\_InterruptTriggerLevel\_16, 148
- I2S\_InterruptTriggerLevel\_2, 148
- I2S\_InterruptTriggerLevel\_3, 148
- I2S\_InterruptTriggerLevel\_4, 148
- I2S\_InterruptTriggerLevel\_5, 148
- I2S\_InterruptTriggerLevel\_6, 148
- I2S\_InterruptTriggerLevel\_7, 148
- I2S\_InterruptTriggerLevel\_8, 148
- I2S\_InterruptTriggerLevel\_9, 148
- I2S\_IsInterruptEnabled, 153
- I2S\_NoSclkGating, 149
- I2S\_Resolution\_12, 149
- I2S\_Resolution\_16, 149
- I2S\_Resolution\_20, 149
- I2S\_Resolution\_24, 149
- I2S\_Resolution\_32, 149
- I2S\_ResolutionDefault, 149
- i2s\_sclk\_gating\_t, 147
- I2S\_SclkCycles\_16, 149
- I2S\_SclkCycles\_24, 149
- I2S\_SclkCycles\_32, 149
- I2S\_SclkGatingCycles\_12, 149
- I2S\_SclkGatingCycles\_16, 149
- I2S\_SclkGatingCycles\_20, 149
- I2S\_SclkGatingCycles\_24, 149
- I2S\_Status\_Fail, 150
- I2S\_Status\_InvalidArgument, 150
- I2S\_Status\_Ok, 150
- I2S\_Status\_TxBusy, 150
- I2S\_Status\_UnsupportedBitRate, 150
- i2s\_transfer\_callback\_t, 147
- I2S\_TransferAbort, 154
- I2S\_TransferCreateHandle, 154
- I2S\_TransferHandleIRQ, 154
- I2S\_TransferNonBlocking, 155
- I2S\_WriteLeftFifo, 155
- I2S\_WriteRightFifo, 155
- Драйвер модуля JTM, 159
  - jtm\_callback\_t, 160
  - JTM\_CreateHandle, 161
  - JTM\_GetParameterValue, 161
  - JTM\_GetParameterValueNonBlocking, 162
  - JTM\_Init, 162



- jtm\_parameter\_t, 160
- JTM\_Status\_BadParameter, 161
- JTM\_Status\_Busy, 161
- JTM\_Status\_Fail, 161
- JTM\_Status\_Ok, 161
- jtm\_status\_t, 161
- JTM\_Temperature, 161
- JTM\_Vcasn, 161
- JTM\_Vcore, 161
- Драйвер модуля POWER, 165
  - power\_callback\_t, 167
  - POWER\_ClearInterrupts, 171
  - POWER\_CreateHandle, 171
  - power\_dcdc\_mode, 167
  - power\_dcdc\_threshold, 168
  - power\_dcdc\_vlevel, 168
  - power\_dcdc\_vlevel\_value, 168
  - POWER\_DcdcModeAuto, 168
  - POWER\_DcdcModePfm, 168
  - POWER\_DcdcModePwm, 168
  - POWER\_DcdcModePwmFcm, 168
  - POWER\_DcdcThreshold\_0\_60V, 168
  - POWER\_DcdcThreshold\_0\_62V, 168
  - POWER\_DcdcThreshold\_0\_64V, 168
  - POWER\_DcdcThreshold\_0\_66V, 168
  - POWER\_DcdcThreshold\_0\_68V, 168
  - POWER\_DcdcThreshold\_0\_70V, 168
  - POWER\_DcdcThreshold\_0\_72V, 168
  - POWER\_DcdcThreshold\_0\_74V, 168
  - POWER\_DcdcThreshold\_0\_76V, 168
  - POWER\_DcdcThreshold\_0\_78V, 168
  - POWER\_DcdcThreshold\_0\_80V, 168
  - POWER\_DcdcThreshold\_0\_82V, 168
  - POWER\_DcdcThreshold\_0\_84V, 168
  - POWER\_DcdcThreshold\_0\_86V, 168
  - POWER\_DcdcThreshold\_0\_88V, 168
  - POWER\_DcdcThreshold\_0\_90V, 168
  - POWER\_DcdcVLevel0, 168
  - POWER\_DcdcVLevel1, 168
  - POWER\_DcdcVLevel2, 168
  - POWER\_DcdcVLevel\_0\_85V, 169
  - POWER\_DcdcVLevel\_0\_86V, 169
  - POWER\_DcdcVLevel\_0\_87V, 169
  - POWER\_DcdcVLevel\_0\_88V, 169
  - POWER\_DcdcVLevel\_0\_89V, 169
  - POWER\_DcdcVLevel\_0\_90V, 169
  - POWER\_DcdcVLevel\_0\_91V, 169
  - POWER\_DcdcVLevel\_0\_92V, 169
  - POWER\_DcdcVLevel\_0\_93V, 169
  - POWER\_DcdcVLevel\_0\_94V, 169
  - POWER\_DcdcVLevel\_0\_95V, 169
  - POWER\_DcdcVLevel\_0\_96V, 169
  - POWER\_DcdcVLevel\_0\_97V, 169
  - POWER\_DcdcVLevel\_0\_98V, 169
  - POWER\_DcdcVLevel\_0\_99V, 169
  - POWER\_DcdcVLevel\_1\_00V, 169
  - POWER\_DcdcVLevel\_1\_01V, 169
  - POWER\_DcdcVLevel\_1\_02V, 169
  - POWER\_DcdcVLevel\_1\_03V, 169
  - POWER\_DcdcVLevel\_1\_04V, 169
  - POWER\_DcdcVLevel\_1\_05V, 169
  - POWER\_DcdcVLevel\_1\_06V, 169
  - POWER\_DcdcVLevel\_1\_07V, 169
  - POWER\_DcdcVLevel\_1\_08V, 169
  - POWER\_DcdcVLevel\_1\_09V, 169
  - POWER\_DcdcVLevel\_1\_10V, 169
  - POWER\_DcdcVLevel\_1\_11V, 169
  - POWER\_DcdcVLevel\_1\_12V, 169
  - POWER\_DcdcVLevel\_1\_13V, 169
  - POWER\_DcdcVLevel\_1\_14V, 169
  - POWER\_DcdcVLevel\_1\_15V, 169
  - POWER\_DcdcVLevel\_1\_16V, 169
  - POWER\_DisableInterrupt, 171
  - POWER\_DisableInterruptMask, 172
  - power\_eco\_mode, 169
  - POWER\_EcoDcdc, 170
  - POWER\_EcoDcdcAndApc, 170
  - POWER\_EcoOff, 170
  - POWER\_EcoReserved, 170
  - POWER\_EnableInterrupt, 172
  - POWER\_EnableInterruptMask, 172
  - power\_flash\_mode, 170
  - POWER\_FlashModeNormal, 170
  - POWER\_FlashModePowerDown, 170
  - POWER\_FlashModeSleep, 170
  - POWER\_GetCurrentConfig, 172
  - POWER\_GetEnabledInterruptMask, 173
  - POWER\_GetInterruptStatus, 173
  - POWER\_GetInterruptStatusMask, 173
  - POWER\_GetStatus, 174
  - power\_interrupt, 170
  - POWER\_IsInterruptEnabled, 174
  - POWER\_SetConfig, 174
  - POWER\_Shutdown, 175
  - POWER\_Standby, 175
  - POWER\_StartTestMode, 175
  - power\_status, 170
  - POWER\_Status\_Fail, 170
  - POWER\_Status\_Ok, 170
  - POWER\_StopTestMode, 176
  - power\_test\_block, 170
  - POWER\_TestBlockApc, 171
  - POWER\_TestBlockDcdc, 171
  - POWER\_TestBlockJtm, 171
  - POWER\_TestBlockRwc, 171
  - POWER\_VmonFalling, 170
  - POWER\_VmonRising, 170
- Драйвер модуля PPU, 176
  - ppu\_add\_event\_name, 178
  - PPU\_AddEventNameAll, 178
  - PPU\_ClrIRQMask, 182
  - PPU\_ClrIRQStatus, 182
  - ppu\_domain\_index, 178
  - PPU\_DomainCPU0, 179
  - PPU\_DomainCPU1, 179
  - PPU\_DomainCRYPTO, 179

- PPU\_DomainDEBUG, 179
- PPU\_DomainGMS, 179
- PPU\_DomainGNSS, 179
- PPU\_DomainMax, 179
- PPU\_DomainSRAM0, 179
- PPU\_DomainSRAM1, 179
- PPU\_DomainSRAM2, 179
- PPU\_DomainSRAM3, 179
- PPU\_DomainSYS, 179
- PPU\_DYN\_FULL\_RET\_SPT, 179
- PPU\_DYN\_FUNC\_RET\_SPT, 179
- PPU\_DYN\_LGC\_RET\_SPT, 179
- PPU\_DYN\_MEM\_OFF\_SPT, 179
- PPU\_DYN\_MEM\_RET\_EMU\_SPT, 179
- PPU\_DYN\_MEM\_RET\_SPT, 180
- PPU\_DYN\_OFF\_EMU\_SPT, 180
- PPU\_DYN\_OFF\_SPT, 180
- PPU\_DYN\_ON\_SPT, 179
- PPU\_DYN\_WRM\_RST\_SPT, 179
- PPU\_DynAccept, 178
- PPU\_DynDeny, 178
- PPU\_EmuAccept, 179
- PPU\_EmuDeny, 179
- ppu\_event\_name, 179
- PPU\_EventNameAll, 179
- PPU\_FULL\_RET\_RAM\_REG, 180
- PPU\_FUNC\_RET\_RAM\_REG, 180
- PPU\_GetIRQMask, 183
- PPU\_GetIRQStatus, 183
- PPU\_GetLastAPIStatus, 183
- PPU\_GetPDxSenseFromPDy, 184
- PPU\_GetPowerState, 184
- PPU\_Init, 184
- PPU\_LOCK\_SPT, 180
- PPU\_Locked, 179
- PPU\_MEM\_RET\_RAM\_REG, 180
- PPU\_OFF\_MEM\_RET\_TRANS, 180
- PPU\_OP\_ACTIVE, 180
- ppu\_opportunities\_idr0, 179
- ppu\_opportunities\_idr1, 180
- ppu\_power\_mode, 180
- PPU\_PowerModeDbgRecov, 181
- PPU\_PowerModeFullRet, 181
- PPU\_PowerModeFuncRet, 181
- PPU\_PowerModeLogicRet, 181
- PPU\_PowerModeMax, 181
- PPU\_PowerModeMemOff, 181
- PPU\_PowerModeMemRet, 181
- PPU\_PowerModeMemRetEmu, 181
- PPU\_PowerModeOff, 181
- PPU\_PowerModeOffEmu, 181
- PPU\_PowerModeOn, 181
- PPU\_PowerModeWarmRst, 181
- PPU\_PWR\_MODE\_ENTRY\_DEL\_SPT, 180
- ppu\_sense\_index, 181
- PPU\_SenseCPU0, 181
- PPU\_SenseCPU1, 181
- PPU\_SenseCRYPTO, 181
- PPU\_SenseGMS, 181
- PPU\_SenseGNSS, 181
- PPU\_SenseSRAM0, 181
- PPU\_SenseSRAM1, 181
- PPU\_SenseSRAM2, 181
- PPU\_SenseSRAM3, 181
- PPU\_SenseSYS, 181
- PPU\_SetIRQMask, 185
- PPU\_SetIRQStatus, 185
- PPU\_SetPDCMPPUSense, 186
- PPU\_SetState, 186
- PPU\_SetStateDynamic, 187
- PPU\_STA\_DBG\_RECOV\_SPT, 180
- PPU\_STA\_FULL\_RET\_SPT, 180
- PPU\_STA\_FUNC\_RET\_SPT, 180
- PPU\_STA\_LGC\_RET\_SPT, 180
- PPU\_STA\_MEM\_OFF\_SPT, 180
- PPU\_STA\_MEM\_RET\_EMU\_SPT, 180
- PPU\_STA\_MEM\_RET\_SPT, 180
- PPU\_STA\_OFF\_EMU\_SPT, 180
- PPU\_STA\_OFF\_SPT, 180
- PPU\_STA\_ON\_SPT, 180
- PPU\_STA\_POLICY\_OP\_IRQ\_SPT, 180
- PPU\_STA\_POLICY\_PWR\_IRQ\_SPT, 180
- PPU\_STA\_PolicyPwr, 178
- PPU\_STA\_WRM\_RST\_SPT, 180
- PPU\_StaAccept, 179
- PPU\_StaDeny, 179
- PPU\_StaPolicyOp, 178
- PPU\_StaPolicyTrn, 179
- PPU\_StateOffRequestHandler, 187
- ppu\_status, 181
- PPU\_Status\_ConfigError, 182
- PPU\_Status\_DriverError, 182
- PPU\_Status\_FeatureNotSupport, 182
- PPU\_Status\_InvalidArgument, 182
- PPU\_Status\_Ok, 182
- PPU\_SW\_DEV\_DEL\_SPT, 180
- PPU\_UnsptPolicy, 178
- ReqOffForCPU, 178
- Драйвер модуля PWM, 188
  - PWM\_ApplyLongSoftOuts, 203
  - PWM\_ApplySoftOuts, 203
  - pwm\_chopper\_duty, 191, 192
  - pwm\_chopper\_first\_width, 192, 193
  - pwm\_chopper\_freq, 194
  - pwm\_chopper\_work, 194
  - PWM\_ChopperDuty\_1\_8, 192
  - pwm\_ChopperDuty\_1\_8, 192
  - PWM\_ChopperDuty\_2\_8, 192
  - pwm\_ChopperDuty\_2\_8, 192
  - PWM\_ChopperDuty\_3\_8, 192
  - pwm\_ChopperDuty\_3\_8, 192
  - PWM\_ChopperDuty\_4\_8, 192
  - pwm\_ChopperDuty\_4\_8, 192
  - PWM\_ChopperDuty\_5\_8, 192
  - pwm\_ChopperDuty\_5\_8, 192

- PWM\_ChopperDuty\_6\_8, 192
- pwm\_ChopperDuty\_6\_8, 192
- PWM\_ChopperDuty\_7\_8, 192
- pwm\_ChopperDuty\_7\_8, 192
- PWM\_ChopperFirstWidth\_0\_8, 193
- pwm\_ChopperFirstWidth\_0\_8, 193
- PWM\_ChopperFirstWidth\_10\_8, 193
- pwm\_ChopperFirstWidth\_10\_8, 193
- PWM\_ChopperFirstWidth\_11\_8, 193
- pwm\_ChopperFirstWidth\_11\_8, 193
- PWM\_ChopperFirstWidth\_12\_8, 194
- pwm\_ChopperFirstWidth\_12\_8, 193
- PWM\_ChopperFirstWidth\_13\_8, 194
- pwm\_ChopperFirstWidth\_13\_8, 193
- PWM\_ChopperFirstWidth\_14\_8, 194
- pwm\_ChopperFirstWidth\_14\_8, 193
- PWM\_ChopperFirstWidth\_15\_8, 194
- pwm\_ChopperFirstWidth\_15\_8, 193
- PWM\_ChopperFirstWidth\_1\_8, 193
- pwm\_ChopperFirstWidth\_1\_8, 193
- PWM\_ChopperFirstWidth\_2\_8, 193
- pwm\_ChopperFirstWidth\_2\_8, 193
- PWM\_ChopperFirstWidth\_3\_8, 193
- pwm\_ChopperFirstWidth\_3\_8, 193
- PWM\_ChopperFirstWidth\_4\_8, 193
- pwm\_ChopperFirstWidth\_4\_8, 193
- PWM\_ChopperFirstWidth\_5\_8, 193
- pwm\_ChopperFirstWidth\_5\_8, 193
- PWM\_ChopperFirstWidth\_6\_8, 193
- pwm\_ChopperFirstWidth\_6\_8, 193
- PWM\_ChopperFirstWidth\_7\_8, 193
- pwm\_ChopperFirstWidth\_7\_8, 193
- PWM\_ChopperFirstWidth\_8\_8, 193
- pwm\_ChopperFirstWidth\_8\_8, 193
- PWM\_ChopperFirstWidth\_9\_8, 193
- pwm\_ChopperFirstWidth\_9\_8, 193
- PWM\_ChopperFreqClk\_16, 194
- pwm\_ChopperFreqClk\_16, 194
- PWM\_ChopperFreqClk\_24, 194
- pwm\_ChopperFreqClk\_24, 194
- PWM\_ChopperFreqClk\_32, 194
- pwm\_ChopperFreqClk\_32, 194
- PWM\_ChopperFreqClk\_40, 194
- pwm\_ChopperFreqClk\_40, 194
- PWM\_ChopperFreqClk\_48, 194
- pwm\_ChopperFreqClk\_48, 194
- PWM\_ChopperFreqClk\_56, 194
- pwm\_ChopperFreqClk\_56, 194
- PWM\_ChopperFreqClk\_64, 194
- pwm\_ChopperFreqClk\_64, 194
- PWM\_ChopperFreqClk\_8, 194
- pwm\_ChopperFreqClk\_8, 194
- pwm\_ChopperWorkOff, 195
- pwm\_ChopperWorkOn, 195
- PWM\_CmdForAllChannels, 204
- pwm\_cntmode, 195
- pwm\_CntModeDown, 195
- pwm\_CntModeOff, 195
- pwm\_CntModeUp, 195
- pwm\_CntModeUpDown, 195
- PWM\_COUNT, 191
- PWM\_Deinit, 204
- pwm\_dirsync, 195
- pwm\_DirSyncDown, 195
- pwm\_DirSyncUp, 195
- pwm\_dz\_mode, 195, 196
- pwm\_dz\_outx\_inv, 196
- pwm\_dz\_signal, 196
- PWM\_DzModeOff, 196
- pwm\_DzModeOff, 196
- PWM\_DzModeOn, 196
- pwm\_DzModeOn, 196
- pwm\_DzSignalOutA, 197
- pwm\_DzSignalOutB, 197
- PWM\_DzSignalOutxInvOff, 196
- pwm\_DzSignalOutxInvOff, 196
- PWM\_DzSignalOutxInvOn, 196
- pwm\_DzSignalOutxInvOn, 196
- PWM\_Enable, 204, 205
- pwm\_eventprd, 197
- PWM\_EventPrdNo, 197
- pwm\_EventPrdNo, 197
- PWM\_EventPrdOne, 197
- pwm\_EventPrdOne, 197
- PWM\_EventPrdThree, 197
- pwm\_EventPrdThree, 197
- PWM\_EventPrdTwo, 197
- pwm\_EventPrdTwo, 197
- PWM\_GetChannelDefaultConfig, 205
- PWM\_GetCntStat, 205
- PWM\_Init, 206
- PWM\_InitChannel, 206
- pwm\_int\_en, 197
- pwm\_int\_source, 197, 198
- PWM\_IntCallback, 206
- pwm\_IntEnNo, 197
- pwm\_IntEnYes, 197
- PWM\_IntSourceCtrcntEquCmpADec, 198
- pwm\_IntSourceCtrcntEquCmpADec, 198
- PWM\_IntSourceCtrcntEquCmpAInc, 198
- pwm\_IntSourceCtrcntEquCmpAInc, 198
- PWM\_IntSourceCtrcntEquCmpBDec, 198
- pwm\_IntSourceCtrcntEquCmpBDec, 198
- PWM\_IntSourceCtrcntEquCmpBInc, 198
- pwm\_IntSourceCtrcntEquCmpBInc, 198
- PWM\_IntSourceCtrcntEquCtrprd, 198
- pwm\_IntSourceCtrcntEquCtrprd, 198
- PWM\_IntSourceCtrcntEquZero, 198
- pwm\_IntSourceCtrcntEquZero, 198
- pwm\_IntSourceNo, 198
- pwm\_ldcswrf, 198
- pwm\_LdcswrfCtrcntEquCtrprd, 198
- pwm\_LdcswrfCtrcntZero, 198
- pwm\_LdcswrfCtrcntZeroEquCtrprd, 198
- pwm\_LdcswrfDirect, 198
- pwm\_ldxmode, 198

- pwm\_LdxModeCtrcntEquCtrprd, 199
- pwm\_LdxModeCtrcntZero, 199
- pwm\_LdxModeCtrcntZeroOrEquCtrprd, 199
- pwm\_LdxModeNoLoad, 199
- pwm\_loadprd, 199
- pwm\_LoadPrdDirect, 199
- pwm\_LoadPrdRegister, 199
- PWM\_OutA, 191
- PWM\_OutB, 191
- pwm\_outx\_cmd, 199
- pwm\_OutxCmdClear, 199
- pwm\_OutxCmdNo, 199
- pwm\_OutxCmdSet, 199
- pwm\_OutxCmdToggle, 199
- pwm\_prescaler\_cmd, 199
- pwm\_prescaler\_divmux, 200
- pwm\_prescaler\_mode, 200
- pwm\_prescaler\_syncrst, 200
- pwm\_PrescalerDivMux1, 200
- pwm\_PrescalerDivMux16, 200
- pwm\_PrescalerDivMux2, 200
- pwm\_PrescalerDivMux4, 200
- pwm\_PrescalerDivMux8, 200
- pwm\_PrescalerDivMuxPWMClk, 200
- pwm\_PrescalerSyncRstDis, 200
- pwm\_PrescalerSyncRstEn, 200
- pwm\_PrescCmdReset, 200
- pwm\_PrescCmdSave, 200
- pwm\_PrescModeAlways, 200
- pwm\_PrescModeTimerIsRun, 200
- pwm\_run\_command, 200
- pwm\_RunCmdRun, 201
- pwm\_RunCmdStop, 201
- pwm\_RunCmdStopEvent, 201
- pwm\_scmpxmode, 201
- pwm\_SCmpxModeDirect, 201
- pwm\_SCmpxModeReg, 201
- PWM\_SetPeriod, 207
- pwm\_status, 201
- PWM\_Status\_BadConfigure, 201
- PWM\_Status\_InvalidArgument, 201
- PWM\_Status\_Ok, 201
- pwm\_syncosel, 201
- pwm\_SyncoSelCtrcntEquCmpb, 201
- pwm\_SyncoSelCtrcntZero, 201
- pwm\_SyncoSelOff, 201
- pwm\_SyncoSelSynci, 201
- pwm\_syncphsen, 201
- pwm\_SyncPhsEnDis, 202
- pwm\_SyncPhsEnEn, 202
- pwm\_trip\_unit\_action, 202
- pwm\_trip\_unit\_signal, 202
- PWM\_TripUnitActionHigh, 202
- pwm\_TripUnitActionHigh, 202
- PWM\_TripUnitActionNo, 202
- pwm\_TripUnitActionNo, 202
- PWM\_TripUnitActionOne, 202
- pwm\_TripUnitActionOne, 202
- PWM\_TripUnitActionZero, 202
- pwm\_TripUnitActionZero, 202
- pwm\_TripUnitSignalNotUsed, 202
- pwm\_TripUnitSignalUsed, 202
- Драйвер модуля QSPI, 207
  - \_qspi\_command\_format, 212
  - \_qspi\_command\_type, 212
  - \_qspi\_qmode, 213
  - \_serial\_nor\_command, 213
- DUMMY\_BYTE, 211
- FLASH\_STAT\_BUSY, 211
- FLASH\_STAT\_WEL, 211
- NOR\_CmdEraseChip, 214
- NOR\_CmdEraseChipNor, 214
- NOR\_CmdErasePage, 214
- NOR\_CmdEraseSector, 214
- NOR\_CmdEraseSector32KB, 214
- NOR\_CmdEraseSector4KB, 214
- NOR\_CmdEraseSector4KBA32, 214
- NOR\_CmdEraseSectorA32, 214
- NOR\_CmdInvalid, 213
- NOR\_CmdReadMemory, 214
- NOR\_CmdReadMemoryA32, 214
- NOR\_CmdReadMemorySDR\_1\_1\_1, 214
- NOR\_CmdReadMemorySDR\_1\_1\_2, 214
- NOR\_CmdReadMemorySDR\_1\_1\_4, 214
- NOR\_CmdReadMemorySDR\_1\_1\_4\_A32, 214
- NOR\_CmdReadMemorySDR\_1\_2\_2, 214
- NOR\_CmdReadMemorySDR\_1\_4\_4, 214
- NOR\_CmdReadMemorySDR\_1\_4\_4\_A32, 214
- NOR\_CmdReadSecStatus\_35, 213
- NOR\_CmdReadSecStatus\_3F, 213
- NOR\_CmdReadStatus, 213
- NOR\_CmdWriteDisable, 213
- NOR\_CmdWriteEnable, 213
- NOR\_CmdWriteMemory, 213
- NOR\_CmdWriteMemoryA32, 213
- NOR\_CmdWriteSecStatus\_31, 213
- NOR\_CmdWriteSecStatus\_3E, 213
- NOR\_CmdWriteStatus, 213
- NOR\_FlashEraseBlock, 215
- NOR\_FlashEraseChip, 215
- NOR\_FlashPageProgram, 215
- NOR\_FlashProgramBlock, 216
- NOR\_FlashRead, 216
- NOR\_FlashReadXIP, 217
- NOR\_QuadModeNotConfig, 212
- NOR\_QuadModeStatusReg1\_Bit6, 212
- NOR\_QuadModeStatusReg2\_Bit1, 212
- NOR\_QuadModeStatusReg2\_Bit1\_0x31, 212
- NOR\_QuadModeStatusReg2\_Bit7, 212
- NOR\_Status\_Fail, 214
- NOR\_Status\_InvalidArgument, 214
- NOR\_Status\_Success, 214
- nor\_status\_t, 214

- NOR\_Status\_Timeout, [214](#)
- QSPI\_ClearInterrupt, [217](#)
- QSPI\_CommandAllSerial, [212](#)
- QSPI\_CommandDataQuad, [212](#)
- QSPI\_CommandNoOpcodeAddrFourBytes, [213](#)
- QSPI\_CommandNoOpcodeAddrThreeBytes, [213](#)
- QSPI\_CommandOpcodeAddrFourBytes, [213](#)
- QSPI\_CommandOpcodeAddrOneByte, [213](#)
- QSPI\_CommandOpcodeAddrThreeBytes, [213](#)
- QSPI\_CommandOpcodeAddrTwoBytes, [213](#)
- QSPI\_CommandOpcodeOnly, [213](#)
- QSPI\_CommandOpcodeSerial, [212](#)
- qspi\_config\_t, [212](#)
- QSPI\_DeInit, [218](#)
- QSPI\_DisableDMA, [218](#)
- QSPI\_DisableInterrupt, [218](#)
- QSPI\_DisableXIP, [218](#)
- QSPI\_DMA\_Status\_Fail, [215](#)
- QSPI\_DMA\_Status\_InvalidArgument, [215](#)
- QSPI\_DMA\_Status\_Success, [215](#)
- qspi\_dma\_status\_t, [214](#)
- QSPI\_DMADescriptorInitRX, [219](#)
- QSPI\_DMADescriptorInitTX, [219](#)
- QSPI\_DMAReadDescriptorInitRX, [219](#)
- QSPI\_DualSPI, [213](#)
- QSPI\_Enable, [220](#)
- QSPI\_EnableDMA, [220](#)
- QSPI\_EnableInterrupt, [220](#)
- QSPI\_EnableXIP, [221](#)
- QSPI\_GetDefaultCommandSet, [221](#)
- QSPI\_GetDefaultConfig, [221](#)
- QSPI\_GetDefaultConfigXIP, [221](#)
- QSPI\_GetDummyDMADescriptorsCount, [222](#)
- QSPI\_GetInstance, [222](#)
- QSPI\_GetReadDMADescriptorsCount, [222](#)
- QSPI\_GetRXLVL, [222](#)
- QSPI\_GetStatusFlag, [224](#)
- QSPI\_GetTXLVL, [224](#)
- QSPI\_Init, [224](#)
- QSPI\_NorFlashInit, [225](#)
- QSPI\_NormalSPI, [213](#)
- QSPI\_QuadSPI, [213](#)
- QSPI\_ReadData, [225](#)
- QSPI\_ReadDataByte, [225](#)
- QSPI\_ReadDataDMA, [226](#)
- QSPI\_SetBitSize, [226](#)
- QSPI\_SetInhibitDin, [226](#)
- QSPI\_SetInhibitDout, [227](#)
- QSPI\_SetQMode, [227](#)
- QSPI\_SetSlaveSelect, [227](#)
- QSPI\_TransferCreateHandleDMA, [227](#)
- QSPI\_WriteData, [228](#)
- QSPI\_WriteDataByte, [228](#)
- QSPI\_WriteDataDMA, [228](#)
- Драйвер модуля RWC, [229](#)
- DAYS\_IN\_A\_YEAR, [232](#)
- RWC\_Alarm, [234](#)
- rcw\_alarm\_enable, [233](#)
- RWC\_AlarmDisable, [234](#)
- RWC\_AlarmEnable, [234](#)
- rcw\_cmd, [234](#)
- RWC\_CmdRead, [234](#)
- RWC\_CmdWait, [234](#)
- RWC\_CmdWrite, [234](#)
- RWC\_CMOSSignal, [235](#)
- RWC\_Config, [234](#)
- RWC\_Deinit, [238](#)
- RWC\_Div1, [235](#)
- RWC\_Div1024, [235](#)
- RWC\_Div128, [235](#)
- RWC\_Div16, [235](#)
- RWC\_Div16384, [235](#)
- RWC\_Div2, [235](#)
- RWC\_Div2048, [235](#)
- RWC\_Div256, [235](#)
- RWC\_Div32, [235](#)
- RWC\_Div32768, [235](#)
- RWC\_Div4, [235](#)
- RWC\_Div4096, [235](#)
- RWC\_Div512, [235](#)
- RWC\_Div64, [235](#)
- RWC\_Div8, [235](#)
- RWC\_Div8192, [235](#)
- RWC\_DivMax, [235](#)
- RWC\_EnableAlarmTimerInterruptFromDPD, [238](#)
- RWC\_EnableWakeUpTimerInterruptFromDPD, [239](#)
- rcw\_freq\_serial, [234](#)
- RWC\_FS125kHz, [234](#)
- RWC\_FS1MHz, [234](#)
- RWC\_FS250kHz, [234](#)
- RWC\_FS2MHz, [234](#)
- RWC\_FS4MHz, [234](#)
- RWC\_FS500kHz, [234](#)
- RWC\_FS625kHz, [234](#)
- RWC\_FS8MHz, [234](#)
- RWC\_GeneralReg, [234](#)
- RWC\_GetAlarm, [239](#)
- RWC\_GetDatetime, [239](#)
- RWC\_GetDefaultConfig, [240](#)
- RWC\_GetInternalRegister, [240](#)
- RWC\_GetLastAPIStatus, [241](#)
- RWC\_GetRTCClkParam, [241](#)
- RWC\_GetSecondsTimerCount, [241](#)
- RWC\_GetSecondsTimerMatch, [242](#)
- RWC\_GetStatusFlags, [242](#)
- RWC\_GetTime, [242](#)
- RWC\_Init, [243](#)
- rcw\_internal\_register, [234](#)
- RWC\_InterruptClear, [243](#)
- RWC\_IRQWkUpDisable, [237](#)
- RWC\_IRQWkUpEnable, [237](#)



- rcw\_lfe\_bypass, 234
- RWC\_QuartzResonator, 235
- rcw\_reset\_type, 235
- RWC\_ResetAllExpectedGeneralReg, 235
- RWC\_ResetOnlyWakeConfigAndConfig, 235
- rcw\_rtccclk\_divisor, 235
- rcw\_rtccclk\_type, 235
- RWC\_RTCClkTypeLFE, 236
- RWC\_RTCClkTypeLFI, 236
- RWC\_SetAlarm, 243
- RWC\_SetDatetime, 244
- RWC\_SetInternalRegister, 244
- RWC\_SetLFEBypass, 245
- RWC\_SetLFX, 245
- RWC\_SetResetCtrl, 246
- RWC\_SetRTCClkParam, 246
- RWC\_SetSecondsTimerCount, 247
- RWC\_SetSecondsTimerMatch, 247
- RWC\_SetWakeUpActiveLewel, 247
- RWC\_SetWakeUpEnable, 248
- rcw\_shutdown\_force, 236
- RWC\_ShutdownNoSet, 236
- RWC\_ShutdownSet, 236
- rcw\_status, 236
- RWC\_Status\_CheckError, 236
- RWC\_Status\_ConfigureError, 236
- RWC\_Status\_HardwareBusy, 236
- RWC\_Status\_InvalidArgument, 236
- RWC\_Status\_Ok, 236
- RWC\_Status\_Timeout, 236
- RWC\_Status\_VerifyError, 236
- RWC\_SYNC\_RETRY\_TIMES, 232
- RWC\_Time, 234
- rcw\_time\_clk\_sel, 236
- RWC\_TimeClock1Hz, 237
- RWC\_TimeClock32kHz, 237
- rcw\_timer\_status, 237
- RWC\_Trim, 234
- RWC\_TrimLoad, 234
- rcw\_wake\_up\_irq\_enable, 237
- rcw\_wake\_up\_polarity, 237
- RWC\_WakeConfig, 234
- RWC\_WakeStat1, 237
- RWC\_WakeStat2, 237
- RWC\_WakeStat3, 237
- rcw\_wkup\_enable, 237
- RWC\_WkUpDisable, 238
- RWC\_WkUpEnable, 238
- RWC\_WkUpPolarityHigh, 237
- RWC\_WkUpPolarityLow, 237
- SECONDS\_IN\_A\_DAY, 233
- SECONDS\_IN\_A\_HOUR, 233
- SECONDS\_IN\_A\_MINUTE, 233
- YEAR\_RANGE\_END, 233
- YEAR\_RANGE\_START, 233
- Драйвер модуля SDMMC, 248
  - SDMMC\_1v8, 256
  - SDMMC\_3v3, 256
  - SDMMC\_3v3To1v8, 256
  - SDMMC\_CALC\_DIVIDER, 251
  - SDMMC\_CalcMemorySpace, 256
  - SDMMC\_CORECFG0\_SLOTTYPE\_EMBEDDED, 251
  - SDMMC\_CORECFG0\_SLOTTYPE\_REMOVABLE, 252
  - SDMMC\_CORECFG0\_SLOTTYPE\_SHARED\_BUS, 252
  - SDMMC\_DataBusTransferWidth\_1Bit, 255
  - SDMMC\_DataBusTransferWidth\_4bit, 255
  - SDMMC\_DisableCard, 256
  - SDMMC\_ExtDataBusTransferWidth\_8bit, 255
  - SDMMC\_HostPWR\_1V8, 254
  - SDMMC\_HostPWR\_3V0, 254
  - SDMMC\_HostPWR\_3V3, 254
  - SDMMC\_InitCard, 257
  - SDMMC\_IS\_MMC, 252
  - SDMMC\_IS\_SD, 252
  - SDMMC\_MMC\_RCA\_ADDR, 252
  - SDMMC\_NoResponse, 255
  - SDMMC\_Read, 257
  - SDMMC\_ReadAsync, 258
  - SDMMC\_ReadWait, 258
  - SDMMC\_ResponseLength136, 255
  - SDMMC\_ResponseLength48, 255
  - SDMMC\_ResponseLength48\_Check, 255
  - SDMMC\_SD\_OCR\_INIT\_VALUE, 252
  - SDMMC\_SD\_SEND\_IF\_COND\_PATTERN, 252
  - SDMMC\_SD\_UHS\_MODE\_DDR50, 253
  - SDMMC\_SD\_UHS\_MODE\_DEFAULT\_SDR12, 253
  - SDMMC\_SD\_UHS\_MODE\_HIGHSPEED\_SDR25, 253
  - SDMMC\_SD\_UHS\_MODE\_SDR104, 253
  - SDMMC\_SD\_UHS\_MODE\_SDR50, 253
  - SDMMC\_SDHC\_SECTOR\_SIZE, 253
  - SDMMC\_SDMA\_ALIGN, 253
  - SDMMC\_SDMA\_BLOCK\_ALIGN, 253
  - SDMMC\_SDMA\_BLOCK\_SIZE, 254
  - SDMMC\_SDMA\_IS\_BLOCK\_ALIGN\_ADDR, 254
  - SDMMC\_SDMA\_TransferRead, 255
  - SDMMC\_SDMA\_TransferWrite, 255
  - SDMMC\_Status\_Err, 255
  - SDMMC\_Status\_Ok, 255
  - sdmmc\_status\_t, 255
  - SDMMC\_TIMEOUTCONTROL\_MAX\_VALUE, 254
  - SDMMC\_TypeMMC, 254
  - SDMMC\_TypeSD, 254
  - sdmmc\_voltage\_t, 255
  - SDMMC\_Write, 258
  - SDMMC\_WriteAsync, 259
  - SDMMC\_WriteWait, 260
- Драйвер модуля SMC, 260

- smc\_adv, 262
- SMC\_AdvLcd, 262
- SMC\_AdvMemory, 262
- SMC\_AdvNotUsed, 262
- SMC\_AdvUsed, 262
- smc\_bit\_depth, 262
- SMC\_BitDepth16, 262
- SMC\_BitDepth24, 262
- SMC\_BitDepth32, 262
- SMC\_BitDepth8, 262
- smc\_bls, 262
- SMC\_BlsAsNcs, 262
- SMC\_BlsAsNwe, 262
- smc\_burst\_align, 262
- SMC\_BurstAlign128, 263
- SMC\_BurstAlign256, 263
- SMC\_BurstAlign32, 263
- SMC\_BurstAlign64, 263
- SMC\_BurstAlignNo, 263
- SMC\_CheckConfigure, 264
- smc\_cmd\_type, 263
- SMC\_CmdTypeModeReg, 263
- SMC\_CmdTypeModeRegAndUpdateRegs, 263
- SMC\_CmdTypeUpdateRegs, 263
- SMC\_CmdTypeUpdateRegsAndAHBCommand, 263
- smc\_cre, 263
- SMC\_CRE0, 263
- SMC\_CRE1, 263
- SMC\_DirectCmd, 265
- smc\_incr\_to\_incr4, 263
- SMC\_IncrToIncr4Disable, 263
- SMC\_IncrToIncr4Enable, 263
- smc\_packet\_lenght, 263
- SMC\_PacketLenght1, 264
- SMC\_PacketLenght4, 264
- SMC\_PacketLenght8, 264
- SMC\_PacketLenghtEndless, 264
- SMC\_PowerSaveOff, 265
- SMC\_PowerSaveOn, 266
- smc\_rd\_wr\_type, 264
- SMC\_RdWrAsync, 264
- SMC\_RdWrSync, 264
- SMC\_RefreshPeriod, 266
- SMC\_SetCycles, 267
- SMC\_SetOpmode, 267
- smc\_status, 264
- SMC\_Status\_InvalidArgument, 264
- SMC\_Status\_Ok, 264
- SMC\_UserConfig, 268
- Драйвер модуля SPI, 269
  - microwire\_busy\_ready\_check\_t, 274
  - microwire\_ctrlword\_len\_t, 274
  - microwire\_single\_serial\_t, 275
  - microwire\_tx\_rx\_t, 275
  - SPI\_CurrentStatusInterrupts, 281
  - SPI\_Data10Bits, 276
  - SPI\_Data11Bits, 276
  - SPI\_Data12Bits, 277
  - SPI\_Data13Bits, 277
  - SPI\_Data14Bits, 277
  - SPI\_Data15Bits, 277
  - SPI\_Data16Bits, 277
  - SPI\_Data17Bits, 277
  - SPI\_Data18Bits, 277
  - SPI\_Data19Bits, 277
  - SPI\_Data20Bits, 277
  - SPI\_Data21Bits, 277
  - SPI\_Data22Bits, 277
  - SPI\_Data23Bits, 277
  - SPI\_Data24Bits, 277
  - SPI\_Data25Bits, 277
  - SPI\_Data26Bits, 277
  - SPI\_Data27Bits, 277
  - SPI\_Data28Bits, 277
  - SPI\_Data29Bits, 277
  - SPI\_Data30Bits, 277
  - SPI\_Data31Bits, 277
  - SPI\_Data32Bits, 277
  - SPI\_Data4Bits, 276
  - SPI\_Data5Bits, 276
  - SPI\_Data6Bits, 276
  - SPI\_Data7Bits, 276
  - SPI\_Data8Bits, 276
  - SPI\_Data9Bits, 276
  - SPI\_Deinit, 281
  - SPI\_DisableInterrupts, 282
  - SPI\_DMADescriptorInitRX, 282
  - SPI\_DMADescriptorInitTX, 282
  - SPI\_DUMMYDATA, 274
  - SPI\_Enable, 283
  - SPI\_EnableInterrupts, 283
  - SPI\_EnableRxDMA, 284
  - SPI\_EnableTxDMA, 284
  - SPI\_FfMicrowire, 276
  - SPI\_FfMotorola, 276
  - SPI\_FfTexas, 276
  - spi\_frame\_format\_t, 275
  - spi\_frame\_width\_t, 276
  - SPI\_GetConfig, 284
  - SPI\_GetInstance, 284
  - SPI\_GetStatusFlags, 285
  - spi\_interrupt\_enable, 277
  - SPI\_IRQ\_All, 278
  - SPI\_IRQ\_MultiMaster, 277
  - SPI\_IRQ\_RxFifoOverflow, 277
  - SPI\_IRQ\_RxFifoTrigger, 277
  - SPI\_IRQ\_RxFifoUnderflow, 277
  - SPI\_IRQ\_RxOnly, 278
  - SPI\_IRQ\_TxFifoOverflow, 277
  - SPI\_IRQ\_TxFifoTrigger, 277
  - SPI\_IRQ\_TxOnly, 278
  - SPI\_IsEnable, 285
  - SPI\_MasterGetDefaultConfig, 285
  - SPI\_MasterHalfDuplexTransferBlocking, 286

- SPI\_MasterHalfDuplexTransferDMA, 286
- SPI\_MasterHalfDuplexTransferNonBlocking, 287
- SPI\_MasterInit, 287
- SPI\_MasterSetBaud, 288
- SPI\_MasterTransferAbort, 288
- SPI\_MasterTransferAbortDMA, 289
- SPI\_MasterTransferBlocking, 289
- SPI\_MasterTransferCreateHandle, 290
- SPI\_MasterTransferCreateHandleDMA, 290
- SPI\_MasterTransferDMA, 291
- SPI\_MasterTransferGetByte, 291
- SPI\_MasterTransferGetRemainingByte, 291
- SPI\_MasterTransferGetTotalByte, 293
- SPI\_MasterTransferHandleIRQ, 293
- SPI\_MasterTransferNonBlocking, 293
- SPI\_MicrowireBusyReadyCheckDisable, 274
- SPI\_MicrowireBusyReadyCheckEnable, 274
- SPI\_MicrowireCtrlWordLen10Bit, 275
- SPI\_MicrowireCtrlWordLen11Bit, 275
- SPI\_MicrowireCtrlWordLen12Bit, 275
- SPI\_MicrowireCtrlWordLen13Bit, 275
- SPI\_MicrowireCtrlWordLen14Bit, 275
- SPI\_MicrowireCtrlWordLen15Bit, 275
- SPI\_MicrowireCtrlWordLen16Bit, 275
- SPI\_MicrowireCtrlWordLen1Bit, 275
- SPI\_MicrowireCtrlWordLen2Bit, 275
- SPI\_MicrowireCtrlWordLen3Bit, 275
- SPI\_MicrowireCtrlWordLen4Bit, 275
- SPI\_MicrowireCtrlWordLen5Bit, 275
- SPI\_MicrowireCtrlWordLen6Bit, 275
- SPI\_MicrowireCtrlWordLen7Bit, 275
- SPI\_MicrowireCtrlWordLen8Bit, 275
- SPI\_MicrowireCtrlWordLen9Bit, 275
- SPI\_MicrowireRx, 275
- SPI\_MicrowireSerial, 275
- SPI\_MicrowireSingle, 275
- SPI\_MicrowireTx, 275
- spi\_mode\_t, 278
- SPI\_ModeDuplex, 278
- SPI\_ModeHalfDuplex, 278
- SPI\_ModeSimplexRx, 278
- SPI\_ModeSimplexTx, 278
- spi\_motorola\_cap\_data\_t, 278
- spi\_motorola\_clk\_pol\_t, 278
- SPI\_MotorolaCapDataFalling, 278
- SPI\_MotorolaCapDataRising, 278
- SPI\_MotorolaClkPolHi, 279
- SPI\_MotorolaClkPolLow, 279
- SPI\_ReadData, 294
- SPI\_Reset, 294
- SPI\_RETRY\_TIMES, 274
- spi\_rxfifo\_watermark\_t, 279
- SPI\_RxFifoWatermark1, 279
- SPI\_RxFifoWatermark2, 279
- SPI\_RxFifoWatermark3, 279
- SPI\_RxFifoWatermark4, 279
- SPI\_RxFifoWatermark5, 279
- SPI\_RxFifoWatermark6, 279
- SPI\_RxFifoWatermark7, 279
- SPI\_RxFifoWatermark8, 279
- SPI\_RxFullFlag, 280
- SPI\_RxNotEmptyFlag, 280
- SPI\_SetDummyData, 296
- spi\_shift\_direction\_t, 279
- SPI\_ShiftDirLsbFirst, 279
- SPI\_ShiftDirMsbFirst, 279
- SPI\_SlaveGetDefaultConfig, 296
- SPI\_SlaveInit, 296
- SPI\_SlaveTransferAbort, 297
- SPI\_SlaveTransferAbortDMA, 297
- SPI\_SlaveTransferCreateHandle, 298
- SPI\_SlaveTransferCreateHandleDMA, 298
- SPI\_SlaveTransferDMA, 299
- SPI\_SlaveTransferGetByte, 299
- SPI\_SlaveTransferHandleIRQ, 299
- SPI\_SlaveTransferNonBlocking, 301
- spi\_status, 279
- SPI\_Status\_BaudrateNotSupport, 280
- SPI\_Status\_Busy, 280
- SPI\_Status\_Fail, 279
- spi\_status\_flags, 280
- SPI\_Status\_Idle, 280
- SPI\_Status\_IncorrectCall, 280
- SPI\_Status\_InvalidArgument, 280
- SPI\_Status\_NoTransferInProgress, 280
- SPI\_Status\_Ok, 279
- SPI\_Status\_ReadOnly, 279
- SPI\_Status\_RxError, 280
- SPI\_Status\_RxFifoBufferOverflow, 280
- SPI\_Status\_RxRingBufferOverflow, 280
- SPI\_Status\_Timeout, 280
- SPI\_Status\_TxError, 280
- SPI\_Status\_UnexpectedState, 280
- SPI\_TakeDownInterrupts, 301
- spi\_trans\_status, 280
- SPI\_TransStatus\_Busy, 280
- SPI\_TransStatus\_Error, 280
- SPI\_TransStatus\_Error\_1, 280
- SPI\_TransStatus\_Error\_2, 280
- SPI\_TransStatus\_Error\_3, 280
- SPI\_TransStatus\_Error\_4, 280
- SPI\_TransStatus\_Idle, 280
- SPI\_TxEmptyFlag, 280
- spi\_txfifo\_watermark\_t, 280
- SPI\_TxFifoWatermark0, 281
- SPI\_TxFifoWatermark1, 281
- SPI\_TxFifoWatermark2, 281
- SPI\_TxFifoWatermark3, 281
- SPI\_TxFifoWatermark4, 281
- SPI\_TxFifoWatermark5, 281
- SPI\_TxFifoWatermark6, 281
- SPI\_TxFifoWatermark7, 281
- SPI\_TxNotFullFlag, 280
- SPI\_WriteData, 301
- Драйвер модуля TIM, 302



- TIMER\_COUNT, 303
- TIMER\_Debug, 304
- TIMER\_Deinit, 305
- TIMER\_GetAPIStatus, 305
- TIMER\_GetTicks, 305
- TIMER\_GetTimerHardwareValue, 306
- TIMER\_Hardware, 305
- TIMER\_HARDWARE\_FIELD\_MAX, 303
- TIMER\_Init, 306
- TIMER\_IRQClear, 307
- TIMER\_IRQDisable, 307
- TIMER\_IRQEnable, 307
- TIMER\_IRQGetStatus, 308
- TIMER\_Reset, 308
- TIMER\_Run, 308
- TIMER\_SetConfig, 309
- TIMER\_SetTick, 309
- TIMER\_Software, 305
- TIMER\_SOFTWARE\_FIELD\_HIGH\_OFFSET, 304
- TIMER\_SOFTWARE\_FIELD\_MAX, 304
- timer\_status, 304
- TIMER\_Status\_BadConfigure, 304
- TIMER\_Status\_InvalidArgument, 304
- TIMER\_Status\_NotIni, 304
- TIMER\_Status\_NotSupport, 304
- TIMER\_Status\_Ok, 304
- TIMER\_Status\_TimerBusy, 304
- TIMER\_Stop, 310
- timer\_type\_of\_counting, 304
- TIMER\_Work, 304
- timer\_work\_mode, 304
- Драйвер модуля UART, 310
- UART\_5BitsPerChar, 315
- UART\_6BitsPerChar, 315
- UART\_7BitsPerChar, 315
- UART\_8BitsPerChar, 315
- UART\_AllInterruptsEnable, 315
- uart\_data\_len, 315
- UART\_Deinit, 318
- UART\_DisableInterrupts, 318
- UART\_DMADescriptorInitRX, 319
- UART\_DMADescriptorInitTX, 319
- UART\_EnableInterrupts, 319
- UART\_GetDefaultConfig, 320
- UART\_GetEnabledInterrupts, 320
- UART\_GetRxFifoCount, 321
- UART\_GetStatusFlags, 321
- UART\_GetTxFifoCount, 321
- UART\_Init, 322
- uart\_interrupt\_enable, 315
- UART\_LSR\_FlagRxError, 316
- UART\_LSR\_FlagRxFrameError, 316
- UART\_LSR\_FlagRxLinebreakError, 316
- UART\_LSR\_FlagRxOverflowError, 316
- UART\_LSR\_FlagRxParityError, 316
- UART\_LSR\_FlagRxReady, 316
- uart\_lsr\_flags, 315
- UART\_LSR\_FlagTxHwEmpty, 316
- UART\_LSR\_FlagTxSwEmpty, 316
- UART\_ModemInterruptEnable, 315
- UART\_OneStopBit, 318
- uart\_parity\_mode, 316
- UART\_ParityEven, 316
- UART\_ParityOdd, 316
- UART\_ReadBlocking, 322
- UART\_ReadByte, 323
- UART\_ReadByteWait, 323
- uart\_rs485\_active\_state, 316
- UART\_RS485\_ActiveStateHigh, 316
- UART\_RS485\_ActiveStateLow, 316
- uart\_rs485\_mode, 316
- UART\_RS485\_ModeFullDuplex, 316
- UART\_RS485\_ModeHalfDuplexAuto, 316
- UART\_RS485\_ModeHalfDuplexManual, 316
- UART\_Rs485Mode, 323
- uart\_rxfifo\_watermark, 316
- UART\_RxFifoHalfFull, 317
- UART\_RxFifoOneChar, 317
- UART\_RxFifoQuarterFull, 317
- UART\_RxFifoTwoToFull, 317
- UART\_RxInterruptEnable, 315
- UART\_RxLineInterruptEnable, 315
- UART\_SetBaudRate, 324
- UART\_SetIr, 324
- UART\_SetLoopback, 325
- UART\_SetRs485, 325
- UART\_SetRxFifoWatermark, 325
- UART\_SetTxFifoWatermark, 325
- uart\_status, 317
- UART\_Status\_BaudrateNotSupport, 317
- UART\_Status\_BreakLineError, 317
- UART\_Status\_Fail, 317
- UART\_Status\_FramingError, 317
- UART\_Status\_InvalidArgument, 317
- UART\_Status\_NoTransferInProgress, 317
- UART\_Status\_Ok, 317
- UART\_Status\_ParityError, 317
- UART\_Status\_ReadOnly, 317
- UART\_Status\_RxBusy, 317
- UART\_Status\_RxError, 317
- UART\_Status\_RxFifoBufferOverrun, 317
- UART\_Status\_RxIdle, 317
- UART\_Status\_RxRingBufferOverrun, 317
- UART\_Status\_Timeout, 317
- UART\_Status\_TxBusy, 317
- UART\_Status\_TxError, 317
- UART\_Status\_TxIdle, 317
- UART\_Status\_TxRingBufferNull, 317
- uart\_stop\_bit\_count, 317
- UART\_ThresholdInterruptEnable, 315
- UART\_TransferAbortReceive, 326
- UART\_TransferAbortReceiveDMA, 326
- UART\_TransferAbortSend, 326
- UART\_TransferAbortSendDMA, 327
- UART\_TransferCreateHandle, 327

- UART\_TransferCreateHandleDMA, 327
- UART\_TransferGetReceiveCount, 328
- UART\_TransferGetRxRingBufferLength, 328
- UART\_TransferGetSendCount, 329
- UART\_TransferGetTxRingBufferLength, 329
- UART\_TransferHandleIRQ, 329
- UART\_TransferReceiveDMA, 330
- UART\_TransferReceiveNonBlocking, 330
- UART\_TransferSendDMA, 331
- UART\_TransferSendNonBlocking, 331
- UART\_TransferStartRingBuffer, 332
- UART\_TransferStartTxRingBuffer, 332
- UART\_TransferStopRingBuffer, 333
- UART\_TransferStopTxRingBuffer, 333
- UART\_TwoOrOneAndHalfStopBit, 318
- uart\_txfifo\_watermark, 318
- UART\_TxFifoEmpty, 318
- UART\_TxFifoHalfFull, 318
- UART\_TxFifoQuarterFull, 318
- UART\_TxFifoTwoChars, 318
- UART\_TxInterruptEnable, 315
- UART\_WaitWhileActive, 334
- UART\_WriteBlocking, 334
- UART\_WriteByte, 334
- UART\_WriteByteWait, 335
- UART\_WriteTxRing, 335
- Драйвер модуля VTU, 336
  - timer\_num\_mode, 338
  - VTU\_Capture, 339
  - vtu\_capture\_edge\_control, 338
  - VTU\_CaptureBothEdgeResetBothEdge, 338
  - VTU\_CaptureBothEdgeResetFallingEdge, 338
  - VTU\_CaptureBothEdgeResetNo, 338
  - VTU\_CaptureBothEdgeResetRisingEdge, 338
  - VTU\_CaptureFallingEdgeResetNo, 338
  - VTU\_CaptureFallingEdgeResetYes, 338
  - VTU\_CaptureRisingEdgeResetNo, 338
  - VTU\_CaptureRisingEdgeResetYes, 338
  - VTU\_CaptureToDTYCAPx, 339
  - VTU\_CaptureToPERCAPx, 339
  - VTU\_ClearTimerIRQ, 340
  - VTU\_CounterOverflow, 339
  - VTU\_Deinit, 340
  - VTU\_DutyCycleMatch, 339
  - VTU\_EnableTimer, 341
  - VTU\_EnableTimerIRQ, 341
  - VTU\_GetCaptureEdgeCtrl, 341
  - VTU\_GetCounter, 342
  - VTU\_GetDefaultConfig, 342
  - VTU\_GetDutyCycleCapture, 343
  - VTU\_GetLastAPIStatus, 343
  - VTU\_GetPeriodCapture, 343
  - VTU\_GetPrescaler, 344
  - VTU\_GetPWMPolarity, 344
  - VTU\_GetTimerIRQ, 344
  - VTU\_HighByteDutyCycleMatch, 339
  - VTU\_HighBytePeriodMatch, 339
  - VTU\_Init, 345
  - vtu\_interrupt\_control, 338
  - VTU\_LowByteDutyCycleMatch, 339
  - VTU\_LowBytePeriodMatch, 339
  - VTU\_LowPower, 339
  - vtu\_mode, 339
  - VTU\_NoInterrupt, 339
  - VTU\_One, 339
  - VTU\_PeriodMatch, 339
  - VTU\_PWM16Bit, 339
  - vtu\_pwm\_polarity, 339
  - VTU\_PWMDual8Bit, 339
  - VTU\_SetCallback, 345
  - VTU\_SetCaptureEdgeCtrl, 346
  - VTU\_SetCounter, 346
  - VTU\_SetDutyCycleCapture, 346
  - VTU\_SetPeriodCapture, 347
  - VTU\_SetPrescaler, 347
  - VTU\_SetPWMPolarity, 348
  - vtu\_status, 339
  - VTU\_Status\_BadConfigure, 340
  - VTU\_Status\_DriverError, 340
  - VTU\_Status\_DualTimerNotCanRun, 340
  - VTU\_Status\_InvalidArgument, 340
  - VTU\_Status\_Ok, 340
  - VTU\_Status\_TimerBusy, 340
  - VTU\_Status\_TimerNotInit, 340
  - VTU\_Timer0Mode16bit, 338
  - VTU\_Timer0Mode8bit, 338
  - VTU\_Timer1Mode8bit, 338
  - VTU\_Timer2Mode16bit, 338
  - VTU\_Timer2Mode8bit, 338
  - VTU\_Timer3Mode8bit, 338
  - VTU\_Zero, 339
- Драйвер модуля WDT, 348
  - WDT\_ClearStatusFlags, 350
  - WDT\_Deinit, 351
  - WDT\_Disable, 351
  - WDT\_Enable, 351
  - WDT\_GetDefaultConfig, 352
  - WDT\_GetLastAPIStatus, 352
  - WDT\_GetStatusFlagsMsk, 352
  - WDT\_GetStatusFlagsRaw, 353
  - WDT\_GetWarningValue, 353
  - WDT\_Init, 353
  - wdt\_inten\_type, 350
  - WDT\_IntenDisable, 350
  - WDT\_IntenEnable, 350
  - WDT\_NUMBER\_OF\_TIMERS, 350
  - WDT\_Refresh, 354
  - wdt\_resen\_type, 350
  - WDT\_ResenDisable, 350
  - WDT\_ResenEnable, 350
  - WDT\_SetTimeoutValue, 354
  - WDT\_SetWarningValue, 354
  - wdt\_status, 350
  - WDT\_Status\_BadConfigure, 350
  - WDT\_Status\_InvalidArgument, 350

WDT\_Status\_Ok, [350](#)  
WDT\_Status\_TimerBusy, [350](#)  
Драйвер модуля программных прерываний  
MHU, [163](#)  
MHU\_CPU0\_ClearInt, [163](#)  
MHU\_CPU0\_SetInt, [164](#)  
MHU\_CPU0\_StatInt, [164](#)  
MHU\_CPU1\_ClearInt, [164](#)  
MHU\_CPU1\_SetInt, [164](#)  
MHU\_CPU1\_StatInt, [165](#)  
Общие определения для библиотеки HAL, [78](#)  
BE\_TO\_LE16, [80](#)  
BE\_TO\_LE24, [80](#)  
BE\_TO\_LE32, [80](#)  
DIM, [81](#)  
FIELD\_BIT, [81](#)  
LBBLEndian, [83](#)  
MAX, [81](#)  
MIN, [82](#)  
SUPPRESS\_FALL\_THROUGH\_WARNING,  
[82](#)  
UINT16\_MAX, [82](#)  
UINT32\_MAX, [82](#)  
UNUSED, [82](#)