Отладчик gdb.

Руководство системного программиста

ОАО НПЦ
«ЭЛВИС»

support@elvees.com, www.multicore.ru

СОДЕРЖАНИЕ

1. ОБЩИЕ СВЕДЕНИЯ О ПРОГРАММЕ

GDB (GNU Debugger) – стандартный отладчик для GNU операционных систем (ОС). В данном документе описываются только функциональность, имеющая отношение к отладке программ, выполняемых на чипах серии Multicore. Основная документация GDB находится по адресу http://www.gnu.org/software/GDB/documentation.

1.1 Функции программы

GDB позволяет производить символьную отладку программ, выполняемых как на эмуляторе, так и на симуляторе.

1.2 Условия выполнения программы

GDB распространяется под операционные системы семейства Windows NT и дистрибутива CentOS 7.

1.2.1 Требования к аппаратной части

Для обеспечения работоспособности необходимо

1) ПЭВМ с процессором типа Intel Core 2 Duo, либо AMD Phenom. Оперативная память и память магнитного жёсткого диска должны обеспечивать работу установленной ОС.
2) Комплект соответствующего отладочного модуля.

1.2.2 Требования к программному обеспечению

Для отладки программ, выполняемых на плате, необходим установленный сервис отладки MJTAGSERVER, устанавливается программой MJTAG_server_setup_6_6.exe. В случае использования GDB с поддержкой расширений на языке python должен быть установлен интерпретатор python 2.7.

2. СТРУКТУРА ПРОГРАММЫ

2.1 Модель отладки программ, выполняемых без операционной системы

Отладочная цель (target) - это среда выполнения, занятая отлаживаемой программой. Для отладки программ на чипах серии Multicore существуют две отладочные цели:

1) **multicore-em** - для отладки на эмуляторе

2) **multicore-sim** - для отладки на симуляторе.

В независимости от выбора отладочной цели отлаживаемая программа представлена в отладчике GDB как процесс, где каждый поток выполнения соответствует ядру платы.

Список ядер (потоков) можно получить, выполнив команду **info threads**. Текущее ядро выбирается посредством команды **thread num**, где **num** – номер ядра, полученный через команду **info threads**.

Регистры текущего ядра (потока) доступны через стандартную команду GDB **info all-registers**. Регистры выбранных периферийных устройств также доступны через команду **info all-registers** при любом текущем ядре. Выбрать периферийные устройства можно с помощью команды **multicore-add-peripheral-device**.

3. НАСТРОЙКА ПРОГРАММЫ

Предусмотрен следующий порядок действий:

1) подключить плату к ПВЭМ;
2) запустить MJTAGServer;
3) запустить GDB;
4) указать текущую архитектуру;
5) выполнить, если необходимо, предварительную настройку командами **multicore-mdb-command**, **multicore-add-peripheral-device**, **multicore-remove-register-description, multicore-sim-trace.**
6) выбрать тип отладочной цели через команду **target** с аргументами **multicore-em** или **multicore-sim**;
7) настроить, если необходимо, эмулятор, используя команду **monitor.**

Текущая архитектура указывается через переменную **architecture**. Пример.

```
set architecture mips
```

Команды настройки и выбора отладочной цели можно занести в инициализационный скрипт, который передается GDB при старте, например:

```
gdb –x gdbinit,
```

где gdbinit - инициализационный скрипт.

3.1 Особенности отладки программ, выполняемых на dsp-ядрах архитектуры elcore30

1.2.3 Опции сборки

При сборке исходных файлов DSP нужно использовать ключ ассемблера -mcx7b.

Пример:

```
clang dsp_main.c -target elcore -g -O0 -Wa,-mcx7b,-Wignore-weak-parse -
Qunused-arguments -g -Tmulticore-elcore30.ld -o dspu
```

1.2.4 Скрипт линковки

Также в скрипте линковки должна быть указана архитектура elcore32:

```
SEARCH_DIR(.)
OUTPUT_ARCH(elcore32)
SECTIONS {
```

1.2.5 После сборочные действия

Если объектный файл DSP будет линковаться в конечную программу для ядра RISC, то нужно вырезать отладочную информацию во избежание конфликтов. Пример:

```
elcopy -L xxx --prefix-symbols=dspu --strip-debug  dspu dspu.o
```

Предварительно нужно сохранить копию слинкованного файла DSP для дальнейшего использования в отладке. Пример:

```
cp dspu dspu.dbg
```

1.2.6 Скрипт инициализации GDB

В скрипте инициализации нужно прописать регионы памяти всех ядер DSP. Пример:

```
elcore add-pram-and-xyram 0xb8440000 0xb8400000
elcore add-pram-and-xyram 0xb8840000 0xb8800000
```

Команда add-pram-and-xyram принимает первым аргументом адрес PRAM, вторым - адрес XYRAM. Регионы должны добавляться в порядке номеров ядер, к которым они относятся.

1.2.7 Совместная отладка с архитектурой mips

В случае совместной отладки с архитектурой mips следует добавить следующую команду в скрипт инициализации:

```
set elcore sign-extend-address on
```

Режим sign-extend-address дополняет адреса DSP аналогичным образом дополнению адресов RISC. Данная команда должна выполняться в самом начале скрипта инициализации.

1.2.8 Добавление символьной информации

Символьная информация добавляется с помощью команды **elcore add-symbol-file**.

Эта команда первым аргументом принимает файл DSP с отладочной информацией, вторым – номер ядра, на котором данный файл будет выполняться. Нумерация ядер DSP начинается с нуля.

Пример:

```
elcore add-symbol-file dspu0.dbg 0
elcore add-symbol-file dspu1.dbg 1
```

Команда elcore add-symbol-file реализована на основе пакета языка Python pyelftools. Этот пакет можно установить через менеджер пакетов pip:

```
pip install pyelftools
```

1.2.9 Удаление символьной информации

Удалить символьный файл можно при помощи команды **remove-symbol-file**. Пример.

```
remove-symbol-file -a 0xb84000000
```

4. ПРОВЕРКА ПРОГРАММЫ

Для проверки программы необходимо сделать следующее:

1) запустить GDB;
2) выбрать отладочную цель, выполнив команду **target multicore-em** для эмулятора.

Если никаких ошибок не произошло, то GDB успешно создал отладочную цель и может начать отладку.

Пример вывода gdb после успешной инициализации для платы MCom02:

```
(gdb) target multicore-em
Successfully connected to /tmp/mdb.sock.
List of suitable devices:
 0. MCom-02 on ARM-USB-TINY-H0
Opening device: ARM-USB-TINY-H0.
```

5. ДОПОЛНИТЕЛЬНЫЕ ВОЗМОЖНОСТИ

Дополнительные возможности доступны через выполнение нижеперечисленных команд или посредством установки значений переменных GDB.

Отобразить значение переменной можно, выполнив команду show var_name, где var_name - имя переменной.

Задать значение переменной можно, выполнив команду set var_name value, где var_name - имя переменной и value – строка, принадлежащая к допустимому множеству значений. Переменные, управляющие включением и выключением какого-либо режима, имеют только два значения: **on** – когда режим включен, и **off** – когда режим выключен.

Отдельный вид команд – команды удаленного монитора, которые доступны только после создания отладочной цели.

5.1 Дополнительные переменные GDB

5.1.1. multicore-do-not-add-prefix-to-register-name

Если значение переменной выставлено в **on**, тогда к имени периферийного регистра будет добавлен префикс в виде имени устройства, к которому принадлежит этот регистр.

По умолчанию значение переменной равно **on**. Чтобы выставление значения переменной оказывало действие, нужно указывать значение до создания отладочной цели.

5.1.2. region-to-core-auto-mapping

Если значение переменной выставлено в **on**, тогда соответствующие области памяти, найденные в описании платы, привязываются к ядрам. Значение по умолчанию - **on**.

5.1.3. multicore-debug

Переменная управляет включением и выключением отладочного вывода модуля GDB multicore. Значение по умолчанию – **off**.

5.1.4. multicore-monitor

Переменная позволяет включать и отключать обработку системных вызовов write и exit на уровне GDB. Значение по умолчанию: **on** – для симулятора, **off** – для эмулятора.

5.1.6 multicore-use-only-hbreaks

Если переменная выставлена в **on**, тогда точки останова, устанавливаемые любой командой GDB, будут аппаратными. Значение по умолчанию: **off**.

5.1.7 multicore-skip-all-peripheral-devices

Если переменная выставлена в **on**, тогда ни один регистр периферийных устройств не будет отображен командой info all-registers. Значение по умолчанию: **on**. Переменную необходимо устанавливать перед созданием отладочной цели.

5.1.8 multicore-use-target-gdbarch-in-list-register-names

Если переменная выставлена в **on**, при выполнении запроса list-register-names протокола GDB/mi будет использована архитектура по умолчанию, а не архитектура текущего потока. Значение по умолчанию: **on**.

5.1.9 multicore-skip-executable-loading

Если значение переменной выставлено в **on**, то при выполнении команды run загрузка исполняемого файла выполняться не будет. Значение по умолчанию: **on**.

5.1.10 multicore-em-skip-default-initialization

Инициализация по умолчанию mdb эквивалентна следующему набору mdb команд: listdevs, opendev device_number.

Если значение этой переменной выставлено в **on**, тогда инициализация по умолчанию не будет выполняться. В этом случае корректная инициализация эмулятора должна быть обеспечена выполнением серии команд multicore-mdb-command. Значение по умолчанию – **off**.

5.1.11 elcore-breakpoint-adjustment

Данная переменная управляет включением и выключением режима смещения адреса точек останова для архитектуры elcore. Смещение адреса точки останова необходимо, когда очевидно, что по данному адресу точка останова не сработает. Значение по умолчание при отладке на плате - **on**, при отладке на симуляторе - **off**.

5.1.12 elcore-debug

Установка значения в **on** включает отладочную печать для elcore модуля. Значение по умолчанию – **off**.

5.1.13 architecture

Переменная позволяет задать текущую архитектуру. Допустимые значения: **mips**, **arm**, **elcore**, **elcore32**.

5.1.14 sign-extend-address

Переменная **sign-extend-address** позволяет включить режим дополнения адрес DSP, необходимый для совместной отладки с ядрами архитектуры mips.

Этот режим должен включаться в самом начале скрипта инициализации GDB. Значение по умолчанию - off.

5.2 Дополнительные команды GDB

5.2.6 Создание отладочной цели эмулятора

Синтаксис: target multicore-em **[**device_number**]**

Описание: команда создает отладочную цель, работающую через эмулятор. Если не указывать номер устройства, то будет выбрано нулевое устройство.

5.2.7 . Создание отладочной цели симулятора

Синтаксис: target multicore-sim config_file

Описание: команда создает отладочную цель, работающую с симулятором. Аргумент c**on**fig_file представляет собой путь к конфигурационному файлу симулятора. Для того чтобы указывать только имя конфигурационного файла нужно следующее:

1) конфигурационные файлы находятся в директории, которая имеет имя mcdevice;
2) директория mcdevice находится в одной папке с исполняемым файлов GDB, или определена переменная окружения SIM3X_CONFIG_PATH, в которой указан путь к mcdevice.

5.2.8 multicore-add-peripheral-device

Синтаксис: multicore-add-peripheral-device device_name

Описание: добавить периферийное устройство с именем device_name в список устройств, чьи регистры будут отображаться командой info all-registers. Команда должна выполняться до создания отладочной цели. Список имен устройств можно получить через команду **multicore-print-peripheral-devices**.

5.2.9 multicore-remove-register-description

Синтаксис: multicore-remove-register-description device_name register_name
Описание: после выполнения команды регистр с именем register_name, относящийся к периферийному устройству device_name, не будет отображаться командой info all-registers. Команда должна выполняться до создания отладочной цели.

5.2.10 multicore-print-peripheral-devices

Синтаксис: multicore-print-peripheral-devices
Описание: команда выводит список всех периферийных устройств. Команда должна выполняться после создания отладочной цели.

5.2.11 multicore_map_region_to_core

Синтаксис: multicore_map_region_to_core start_address end_addre core_number region_type.

Описание: start_address и end_address - начальный и конечный адреса региона, core_number - номер ядра, к которому привязывается регион, и regi**on**_type - тип региона, для региона с исполняемыми инструкциями - text, для региона с данными - data. Данная команда привязывает регион памяти к соответствующему ядру. Привязанный регион памяти используется для определения того, на которое ядро ставить точку останова, и для трансляции внутренних dsp адресов во внешние адреса. Команда должна выполняться до создания отладочной цели.

5.2.12 multicore-print-mapped-regions

Синтаксис: multicore-print-mapped-regions

Описание: команда выводит список регионов памяти, привязанных к ядрам.

5.2.13 multicore-sim-trace

Синтаксис: multicore-sim-trace trace_command

Описание: Данная команда позволяет передавать симулятору команду трассировки. Команда должна выполняться до создания отладочной цели.

5.2.14 multicore-mdb-command

Синтаксис: multicore-mdb-command cmd

Описание: данная команда позволяет выполнить команды mdb до создания отладочной цели.

5.2.15 multicore-clear-mdb-command-list

Синтаксис: multicore-clear-mdb-command-list

Описание: данная команда очищает список команд инициализации mdb, создаваемый командой multicore-mdb-command.

5.2.16 multicore-platform-description

Синтаксис: multicore-platform-description path_to_platform_description

Описание: данная команда позволяет указать путь к файлу описания платы для эмулятора. Данная команда должна выполняться до создания отладочной цели.

5.2.17 elcore add-pram-and-xyram

Синтаксис: elcore add-pram-and-xyram pram_address xyram_address

Где **pram_address** - начальный адрес региона PRAM, **xyram_address** - начальный адрес региона XYRAM.

Описание: указать информацию о регионах памяти ядер DSP.

Добавлять регионы следует в порядке ядер, к которым они принадлежат.

5.2.18 elcore print-memory-map

Синтаксис: elcore print-memory-map

Описание: вывести список регионов dsp, добавленный ранее командой add-pram-and-xyram.

5.2.19 elcore clear-regions

Синтаксис: elcore clear-regions

Описание: очистить заданный ранее список регионов памяти.

5.2.20 elcore add-symbol-file

Синтаксис: elcore add-symbol-file filename core_number

Где **filename** - путь к файлу dsp с отладочной информацией, **core_number** - номер ядра DSP, где данный объектный файл будет исполняться. Нумерация ядер DSP начинается с нуля. Для использования данной команды нужно иметь установленный в системе пакета языка Python peelftools.

5.3 Команды удаленного монитора

5.3.6 Команды mdb

Синтаксис: monitor mdb_command

Описание: данная команда позволяет выполнять команды mdb после создания отладочной цели multicore-em.

5.3.7 clock-count

Синтаксис: monitor clock-count index

Описание: данная команда выводит число тиков симулятора после запуска модели. Если параметр index равен 0, тогда выводится время в наносекундах, если index равен 1, тогда выводится число тиков risc-ядра, если index равен 0x1000 + i, тогда выводится число тиков на i-м dsp-ядре.

5.3.8 show backend

Синтаксис: monitor show backend

Описание: выводит название нижележащего уровня: emulator – для отладочной цели multicore-em или simulator – для multicore-sim.

5.3.9 show chipname

Синтаксис: monitor show chipname

Описание: выводит название отладочной платы.

5.3.10 initddr

Синтаксис: monitor initddr [frequency] [memory_ports]

Описание: настройка контроллеров памяти ddr. Параметр frequency позволяет задать частоту памяти в МГц. Параметр memory_ports позволяет указать настраиваемые

порты: 0x1 - первый порт, 0x2 – второй. В случае вызова команды **initddr** без аргументов производится настройка обоих портов с частотой 528 МГц.

6 СООБЩЕНИЯ СИСТЕМНОМУ ПРОГРАММИСТУ

Сообщения, которые могут выдаваться при создании отладочной цели:

1) **Couldn't open sim model.** - не получилось создать модель симулятора, потому что был задан неправильный путь к конфигурационному файлу или конфигурационный файл некорректен.
2) **Couldn't open target: error message** - не получилось открыть устройство, возможные причины: не запущен mjtagserver, неправильно заданный номер устройства, устройства не подключено к ПЭВМ.
3) **Executable loading failed** - не удалось загрузить elf файл в память устройства, либо по причине инвалидности содержимого исполняемого файла, либо из-за того, что память, в которую загружается elf файл, недоступна.
4) **Ddr initialization failed** – не удалось настроить DDR память. Нужно проверить, что до запуска GDB не выполнялись на плате программы, которые делающие какую-либо настройку, как-то загрузчики, операционные системы и т.п.

Версия документа	
1.00 (4.05.2016)	Начальная версия.