Микроконтроллер BE-U1000. Рекомендации по использованию

Отладка по интерфейсу JTAG

Содержание

1 Введение

В настоящем документе даны примеры подготовки и отладки программ на микроконтроллере BE-U1000 (далее – МК).

В документе показана отладка при помощи интерфейса JTAG, отладочного сервера OpenOCD и программного отладчика GDB. OpenOCD и GDB входят в состав SDK.

Ознакомиться с командами и возможностями отладки с помощью GDB можно в руководстве пользователя GDB.

Общая информация о МК дана в *«BE-U1000. Краткое описание»* и *«BE-U1000. Руководство пользователя»*.

Ознакомиться с характеристиками МК можно в *«BE-U1000. Техническая спецификация»* и *«BE-U1000. Reference Manual»*.

2 Условные обозначения, термины и сокращения

В документе используются условные обозначения, приведенные в таблице 1.

Таблица 1 – Условные обозначения

Обозначение	Расшифровка
`[Текст]`	Переменные данные в командах. Примеры: `[путь к файлу]`, `[имя образа]`. Данные вводятся без символов `[ ]`.

В документе применяются сокращения и термины, указанные в таблице 2.

Таблица 2 – Сокращения и термины

Обозначение	Расшифровка
ИС	Интегральная схема.
МК	Микроконтроллер BE-U1000.
Страповый вывод	Вывод МК, предназначенный для выбора режима работы при начальной загрузке.
BootROM	Программа начального загрузчика, расположенная в ROM0.
Точка останова (Breakpoint)	Преднамеренное прерывание выполнения программы. Оно позволяет изучить состояние МК, проверить значения переменных и пошагово выполнить код программы.
eFlash	Встроенная флэш-память МК.
GDB	GNU Debugger – кроссплатформенный программный отладчик.
JTAG	Joint Test Action Group – стандартизированный аппаратный интерфейс (IEEE 1149.1), предназначенный для тестирования и отладки электронных устройств.
OpenOCD	Open On-Chip Debugger – отладочный сервер, предназначенный для взаимодействия с JTAG-интерфейсом.
QSPI	Quad SPI – расширение классического SPI, позволяющее передавать данные по четырём линиям данных параллельно вместо одной, что значительно увеличивает пропускную способность последовательного интерфейса.
QSPI Flash	Тип флэш-памяти, которая использует интерфейс QSPI для передачи данных.
SPI	Serial Peripheral Interface (последовательный периферийный интерфейс) – это синхронный последовательный интерфейс для обмена данными между микроконтроллерами и периферией (датчики, дисплеи, SD-карты).
TCM	Tightly Coupled Memory (тесно связанная память) — память, предназначенная для хранения критически важных с точки зрения производительности данных и кода.
UART	Universal Asynchronous Receiver-transmitter (универсальный асинхронный приемопередатчик) – это протокол последовательной связи, используемый для обмена данными между электронными устройствами без синхронизирующего тактового сигнала, с настройкой общей битовой скорости и добавлением служебных битов (стартовый, стоповый) для кадрирования данных.
XIP	Execute-In-Place – способ исполнения кода непосредственно из QSPI Flash, без предварительной загрузки в TCM.

3 Окружение

- **ОС Windows 10 / 11 (x86)**
- **SDK v. 2.2.0**
  Процесс развертывания SDK под ОС Windows приведен в документе «Микроконтроллер BE-U1000. Руководство пользователя». В настоящем документе предполагается, что SDK расположен в папке `C:\SDK`.
- Отладочная плата **EVU-BA-2.5**

  Описание платы приведено в документах «Отладочная плата EVU-BA-2.5, техническое описание», «Руководство по быстрому старту на платах серий EVU-BA и EVU-LI для ОС Windows 10/11».
- **Драйвер `WinUSB`**
  Рекомендуется устанавливать драйвер при помощи утилиты zadig. Для этого в интерфейсе утилиты во вкладке `Options` нажмите на пункт `List All Devices`, после чего замените драйвер платы на `WinUSB`.
- **`PuTTY`** или аналогичная программа удаленного терминала для вывода информации в UART.

4 Общий алгоритм отладки

Ниже приведен перечень действий, необходимых для подготовки и запуска отладки на МК. Рекомендуется по порядку выполнять все пункты настоящего раздела.

4.1 Сборка образа программы

Сборку проекта рекомендуется проводить со следующими параметрами:

- `BUILD_TYPE=debug` – сборка образа, оптимизированного компилятором для отладки.
- `OPT=0` – отключение оптимизации кода отладчиком.
- `DEBUG=gdb3` – сборка образа, оптимизированного компилятором для отладки в GDB.

Указанные параметры специфичны для SDK BE-U1000. Более подробное описание параметров сборки см. в `[SDK_path]\Tools\Makefile_template`.

Подробная информация по сборке образов для различных типов памяти приведена в документе «*Микроконтроллер BE-U1000. Руководство пользователя*».

Пример сборки образа для TCM:

1. Открыть окно командной строки (запустить `cmd.exe`)
2. Перейти в папку с программой

```
cd C:\MCU
```

3. Ввести команду для сборки

```
make SDK_DIR = $(realpath $(CURDIR)/../sdk)BOARD = EVU_BA_2_5 BUILD_TYPE=debug
MEM_REG_ROM = TCMA OPT=0 DEBUG=gdb3 CSRC += $(SDK_DIR)/Drivers/HAL/Src/bmcu_cru.c
CSRC += $(SDK_DIR)/Drivers/HAL/Src/bmcu_uart.c include
$(SDK_DIR)/Tools/build/common.mk
```

Эти параметры можно задать в сборочном скрипте `makefile`, в таком случае достаточно команды `make`.

В результате в папке `output\debug` будут сгенерированы файлы, скомпилированные из исходного текста программы.

Для очистки результатов сборки необходимо ввести команду `make clean`.

4.2 Включение режима отладки

4.2.1 Включение программно эмулируемого адаптера JTAG (JTAG Int)

Встроенный программно эмулируемый адаптер JTAG исполняется на ядре Core 2 и используется при отладке программного кода, выполняемого на ядрах Core 0 и Core 1.

Core 2 выполняет при этом функции адаптера USB-JTAG и недоступен для исполнения программ, а МК определяется по USB как составное CDC-устройство.

Для включения программно эмулируемого адаптера JTAG необходимо перевести МК в режим «JTAG_INT», для чего на выводы конфигурации PC[8], PC[4], PB[10] и PA[10] нужно подать напряжения, соответствующие логическим уровням согласно таблице 3 и перезагрузить МК.

При использовании отладочной платы EVU-BA необходимо подключить USB-кабель к разъему XS1 (USB).

4.2.2 Включение внешнего аппаратного адаптера JTAG (JTAG Ext)

Этот режим предназначен для отладки программ, запускаемых на ядрах Core 0, Core 1 и Core 2, с использованием внешнего аппаратного адаптера JTAG. Аппаратный адаптер обеспечивает связь между интерфейсом JTAG МК и ПК, на котором работает отладочный сервер OpenOCD.

Выводы интерфейса JTAG на МК являются выводами портов PA и PC, переключенными в режим альтернативной функции JTAG. Подробнее см. в документе «*BE-U1000. Техническая спецификация*».

Для использования внешнего аппаратного адаптера JTAG необходимо перевести МК в режим работы «JTAG Ext». Для этого установите на выводах конфигурации PC[8], PC[4], PB[10] и PA[10] логические уровни согласно таблице 3, после чего выполните сброс (перезагрузку) МК.

Таблица 3 – Логические уровни на выводах конфигурации МК для переключения в режимы «JTAG INT» и «JTAG EXT»

Наименование вывода МК	Наименование вывода на EVU-BA	Логический уровень на выводе для переключения в режим отладки	
		JTAG INT	JTAG EXT
PC[8]	DBG	1	0
PC[4]	M2	1	1
PB[10]	M1	0	0
PA[10]	M0	0	0

Если Вы используете отладочную плату EVU-BA, то у Вас есть возможность использовать встроенный аппаратный адаптер JTAG, реализованный на ИС FTDI (FT2232HL). Эта ИС выполняет функции преобразователя интерфейсов USB-JTAG.

Для удобства отладки при помощи внешнего аппаратного отладчика с использованием платы EVU-BA, сигнал *TRST* может быть сконфигурирован на выводе 3 разъема XP9 (на МК вывод PA[4]).

Подключение к плате EVU-BA:

1. Выбрать в качестве источника питания разъем USB, установив перемычку на разъеме XP11 (PWR SEL) в положение «USB», или внешний источник питания в соответствии с документом «Отладочная плата EVU-BA-2.5, техническое описание».
2. Выбрать режим загрузки «JTAG Ext», установив перемычки на разъеме XP1 в соответствии с таблицей 3.
3. Подключить кабель USB Type-C к разъему XS2 (JTAG).
4. Подключить второй конец кабеля к ПК. После подключения на плате загорится светодиод LD2 (PWR).

Более подробную информацию о подключении можно найти в документе «Отладочная плата EVU-BA-2.5, техническое описание».

4.3 Запуск отладочного сервера и программного отладчика

Для доступа к отладке необходимо запустить отладочный сервер OpenOCD, указав необходимые конфигурационные файлы, и запустить программный отладчик GDB из состава SDK.

4.3.1 Работа с программно эмулируемым адаптером JTAG (JTAG Int)

4.3.1.1 Запуск отладочного сервера OpenOCD

Шаг 1: Открыть окно командной строки

Запустить `cmd.exe`.

Шаг 2: Перейти в папку с OpenOCD

```
cd C:\SDK\Tools\OpenOCD\bin
```

Шаг 3: Запустить OpenOCD

```
.\openocd.exe -f interface\baikal\core2-jtag.cfg -f board\baikal\evu_ba_2_5.cfg
```

Запуск программного отладчика GDB

Шаг 1: Открыть новое окно командной строки

Шаг 2: Перейти в папку с GDB

```
cd C:\SDK\Tools\toolchain\riscv32-none-elf\bin
```

Шаг 3: Запустить GDB

```
.\riscv32-none-elf-gdb.exe -x C:\SDK\Tools\scripts\bmcu.gdb
```

4.3.2 Работа с внешним аппаратным адаптером JTAG

В качестве внешнего аппаратного адаптера возможно использование готовых решений из числа поддерживаемых:

- SEGGER J-Link (v 9.3 и выше);
- OLIMEX-ARM-USB-OCD-H;
- Платы на основе FT2232H.

Для каждого из перечисленных выше аппаратных адаптеров в SDK подготовлен файл конфигурации:

- SEGGER J-Link – файл `jlink-jtag.cfg`;
- OLIMEX-ARM-USB-OCD-H – файл `olimex-jtag.cfg`;
- Платы на основе FT2232H – файл `ft2232h-jtag.cfg`.

Перечисленные выше файлы расположены в папке SDK `Tools\OpenOCD\share\openocd\scripts\interface\baikal`.

4.3.2.1 Запуск отладочного сервера OpenOCD

Шаг 1: Открыть окно командной строки

Запустить `cmd.exe`.

Шаг 2: Перейти в папку с OpenOCD

```
cd C:\SDK\Tools\OpenOCD\bin
```

Шаг 3: Запустить OpenOCD

```
.\openocd.exe -f interface\baikal\evu-ba_ftdi.cfg -f board\baikal\evu_ba_2_5.cfg
```

4.3.2.2 Запуск программного отладчика GDB

Шаг 1: Открыть новое окно командной строки

Шаг 2: Перейти в папку с GDB

```
cd C:\SDK\Tools\toolchain\riscv32-none-elf\bin
```

Шаг 3: Запустить GDB

```
.\riscv32-none-elf-gdb.exe -x C:\SDK\Tools\scripts\bmcu.gdb
```

4.4 Загрузка образа программы в МК

Для загрузки образа программы для МК в память МК можно воспользоваться любым из способов, представленных в документе «*Микроконтроллер BE-U1000. Руководство пользователя*».

Ниже приведены примеры загрузки через интерфейс JTAG (выполняется с помощью GDB).

Для загрузки необходимо предварительно выполнить требования разделов 4.1, 4.2, 4.3 настоящего документа.

Примеры загрузки приведены с учетом расположения образа программы в папке `C:\MCU\output\debug`.

4.4.1 Загрузка в TCM

В окне командной строки с GDB ввести команду на запись образа в память МК:

```
restore C:\MCU\output\debug\MCU.bin binary 0x40010000
```

4.4.2 Загрузка в eFlash

1. В окне командной строки с GDB ввести команду на очистку eFlash:

```
monitor eFlash erase
```

2. Перезагрузить МК:

```
monitor reset
```

> В случае отладки при помощи JTAG Int необходимо перезагрузить МК снятием с него питания, после чего заново запустить OpenOCD и GDB.

3. Записать образ программы в память МК:

```
restore C:\MCU\output\debug\MCU.bin binary 0xa0000000
```

4. Перезагрузить МК:

```
monitor reset
```

4.4.3 Загрузка в QSPI Flash

1. В окне командной строки с GDB ввести команды на инициализацию QSPI Flash:

```
monitor QSPI_clk_init
```

```
monitor QSPI1 init
```

2. Очистить QSPI Flash (если требуется):

```
monitor QSPI1 erase
```

3. Записать образ программы в память МК:

```
monitor flash write_bank 1 C:\MCU\output\debug\MCU.bin 0x90000000
monitor reset
```

В случае отладки при помощи JTAG Int необходимо перезагрузить МК снятием с него питания, после чего заново запустить OpenOCD и GDB.

Показана запись в QSPI1. Для записи в QSPI0 необходимо использовать адрес `0x80000000` и `QSPI0` в качестве параметра команды `monitor`.

Указанные команды актуальны для микросхем памяти, соответствующих стандарту *JEDEC JESD216*.

4.5 Подготовка к отладке

4.5.1 Настройка памяти

Для отладки необходимо произвести настройки по инициализации памяти и переводу ее в режим «read only».

Для исполнения из TCM дополнительных действий не потребуется.

eFlash

```
enable mem 3
```

Перед повторной загрузкой программы в eFlash в одной и той же отладочной сессии необходимо выполнить команду `disable mem 3`.

QSPI Flash

```
monitor QSPI_clk_init
monitor QSPI1 init
monitor QSPI1 XIP on
```

Указанные команды актуальны для микросхем памяти, соответствующих стандарту *JEDEC JESD216*.

Для повторной загрузки программы в QSPI Flash необходимо перезагрузить МК и память.

4.5.2 Загрузка отладочных символов

Для удобства отладки рекомендуется использовать файл `.elf` с отладочными символами, генерируемый при сборке образа.

Указание в GDB пути к файлу `.elf`:

```
file "C:\MCU\output\debug\MCU.elf"
```

4.6 Отладка

4.6.1 Запуск программы

Шаг 1: Переход на адрес начала программы

```
monitor jump TCMA
```

Приведен пример перехода к адресу TCMA. Допускается использование как адресов в шестнадцатеричном представлении (например, `0x40012000`), так и имен разделов памяти (`TCMA`, `TCMB`, `QSPI0`, `QSPI1`, `eFlash`).

Шаг 2: Запустить программу

```
continue
```

Не следует использовать команду `run`, так как это приведет к некорректной перезагрузке микроконтроллера.

4.6.2 Установка точек останова

Синтаксис команды для установки точки останова:

```
break [путь к файлу]\[имя файла]:[номер строки]\[имя функции]
```

Для отладки программы из eFlash или QSPI Flash доступны только три аппаратных точки останова. При отладке из eFlash и QSPI Flash по умолчанию будут установлены аппаратные точки останова, при отладке из TCM – программные.

Вывод информации о точках останова:

```
info breakpoints
```

Добавление точки останова на строке 237:

```
break main.c:237
Breakpoint 2 at 0x40012ef6: file main.c, line 237.
```

Удаление точки останова:

```
delete breakpoint 2
```

4.6.3 Отслеживание значений переменных

Включение отслеживания переменной `i` при шагах или на точках останова:

```
display [имя переменной]
```

Пример включения отслеживания:

```
display fib_arr
```

Пример отключения отслеживания:

```
undisplay fib_arr
```

4.6.4 Отображение локальных переменных

Отображение всех локальных переменных текущего кадра стека и их значений:

```
info locals
j = 0
i = 0
arr = {{0, 2, 3}, {4, 5, 6}, {7, 8, 9}}
```

4.6.5 Отображение переменных без остановки исполнения

В некоторых случаях может потребоваться отлаживать программу без остановки ее исполнения. Для этого в качестве отладчика необходимо использовать OpenOCD.

Предварительно должен быть запущен отладочный сервер (п. 4.3.1.1), в память загружена отлаживаемая программа (п. 4.4).

Подключение к локальному отладочному серверу:

```
telnet 127.0.0.1 4445
```

Показано подключение к ядру Core 0 (порт 4445). Для отладки на ядрах Core 1 или Core 2 используйте порты 4446 или 4447.

Переход к области памяти, в который записана программа (в качестве примера указан адрес eFlash):

```
jump 0xa0000000
```

Запуск исполнения программы:

```
resume
```

Просмотр значения переменной:

```
mdw phys [адрес переменной в памяти]
```

Адрес переменной в памяти можно узнать из файла `[XXX].map`, генерируемого при сборке программы.

4.6.6 Пошаговая отладка

Выполнение текущей строки программы с переходом на следующую:

```
next
```

Пример пошаговой отладки после точки останова:

```
Breakpoint 1, main () at main.c:34
32              fib_arr[i] = sum(fib_arr[i - 1], fib_arr[i - 2]);
1: very_important_data = {42, 73, 69}
2: fib_arr = {0, 1, 0, 0, 0, 0, 0}

(gdb) next
31          for (size_t i = 2; i <= FIBNUM; ++i) {
1: very_important_data = {42, 73, 69}
2: fib_arr = {0, 1, 1, 0, 0, 0, 0}

(gdb) next
31          for (size_t i = 2; i <= FIBNUM; ++i) {
1: very_important_data = {42, 73, 69}
2: fib_arr = {0, 1, 1, 2, 0, 0, 0}
```

5 Пример отладки программы

Показан пример отладки программы в режиме «JTAG Int» (встроенный программно эмулируемый адаптер). Образ программы записывается в TCM. Отладка проводилась с окружением в соответствии с разделом 3 настоящего документа.

5.1 Подготовка проекта программы

Приведен пример программы, реализующей последовательность Фибоначчи для заполнения массива.

Настоящий документ снабжен примером программы. Для начала работы необходимо загрузить и распаковать его, как показано на рисунке ниже.

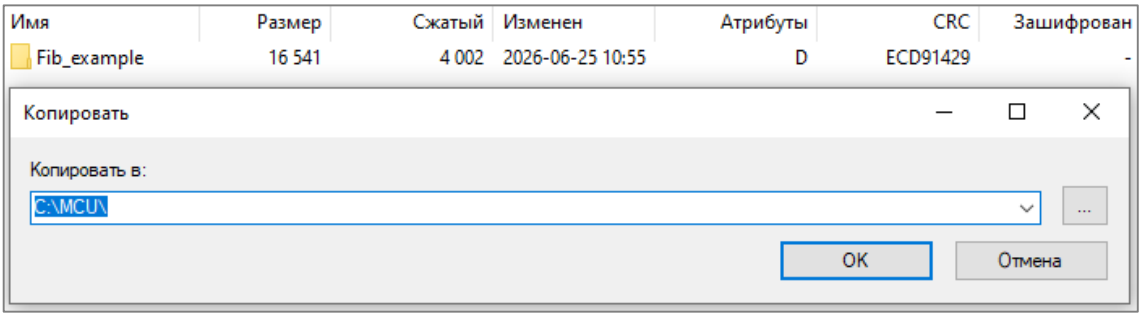

Рисунок 1 – Распаковка архива в папку `C:\MCU`

Проект состоит из файлов с исходным кодом на языке СИ и сборочного скрипта (makefile). Основная функция (main) содержит следующий код:

```
1   #include <stdio.h>
2   #include <stdint-gcc.h>
3   #include "bsp.h"
4   #define FIBNUM 7
5
6   int
7   __io_putchar (int ch) {
8       return bsp_serial_putchar(ch);
9   }
10
11  uint32_t
12  sum (uint32_t a, uint32_t b) {
12      return a + b;
14  }
15
16  int
17  main (void) {
        // Important data array
18      uint32_t very_important_data[3] = {42, 73, 69};
        // Fibonacci array:
19      uint32_t fib_arr[FIBNUM] = {0, 1};
20
21      bsp_serial_init();
22
23      // Print important data
24      printf("Very important data content (before Fibonacci): ");
25      for (size_t i = 0; i < 3; ++i) {
26          printf("%ld ", very_important_data[i]);
27      }
28      printf("\r\n");
29
```

```
30      // Filling the fib_arr array with Fibonacci numbers
31      for (size_t i = 2; i <= FIBNUM; ++i) {
32          fib_arr[i] = sum(fib_arr[i - 1], fib_arr[i - 2]);
33      }
34
35      // Print Fibonacci sequence
36      for (size_t i = 0; i < FIBNUM; ++i) {
37          printf("Fibonacci element %d: %ld\r\n", i + 1, fib_arr[i]);
38      }
39
40      // Print important data again
41      printf("Very important data content (after Fibonacci): ");
42      for (size_t i = 0; i < 3; ++i) {
43          printf("%ld ", very_important_data[i]);
44      }
45      printf("\r\n");
46
47      return 0;
48  }
```

В основной функции `main` объявлено два массива – `very_important_data` и `fib_arr`. Содержимое массива `very_important_data` не должно изменяться в ходе выполнения программы. В массив `fib_arr` программа записывает числа из ряда Фибоначчи.

Ход выполнения программы:

- Вывод содержимого массива `very_important_data`
- Заполнение массива `fib_arr`. Заполнение происходит с ошибкой, заключающейся в неверном количестве итераций обращения к элементам массива. Массив имеет 7 элементов (с нулевого по шестой), а последняя итерация заполнения массива обращается к восьмому элементу. Т.к. восьмого элемента не существует, происходит изменение данных в массиве `very_important_data`.
- Повторный вывод массива `very_important_data`. Содержимое массива, вследствие допущенной ранее ошибки, будет изменено.

5.2 Сборка образа программы

Для сборки образа должен быть загружен и подготовлен SDK. Процесс подготовки SDK под ОС Windows приведен в документе «Микроконтроллер BE-U1000. Руководство пользователя».

Шаг 1: Запуск командной строки и переход в папку с программой

Открыть окно командной строки (запустить `cmd.exe`).

Ввести команду:

```
cd C:\MCU
```

Шаг 2: Сборка образа

```
make
```

Результаты сборки будут находиться в папке `output\debug`.

5.3 Загрузка и запуск программы

Показана загрузка в TCMA в режиме «JTAG Int» и вывод информации через UART.

Шаг 1: Подготовка

1. Подключение USB-кабелей к разъемам XS1 и XS2.
2. Запуск МК в режиме «JTAG Int». Для этого необходимо установить на выводах конфигурации PC[8], PC[4], PB[10] и PA[10] логические уровни согласно таблице 3, после чего выполнить сброс (перезагрузку) МК.

Наименование вывода МК	Наименование вывода EVU-BA	Логический уровень на выводе
PC[8]	DBG	0
PC[4]	M2	0
PB[10]	M1	0
PA[10]	M0	1

Шаг 2: Подключение к интерфейсу МК «UART0»

1. Подключение к МК через виртуальный COM-порт.
2. Запуск `PuTTY` и подключение к интерфейсу МК «UART0».

Установка параметров подключения к интерфейсу:

Скорость передачи (Baud Rate)	115200
Управление потоком (Flow Control)	Нет
Биты четности (Parity)	Нет
Биты данных (Data Bits)	8
Стоп-биты (Stop Bits)	1

Установка параметров для корректного отображения конца строки:

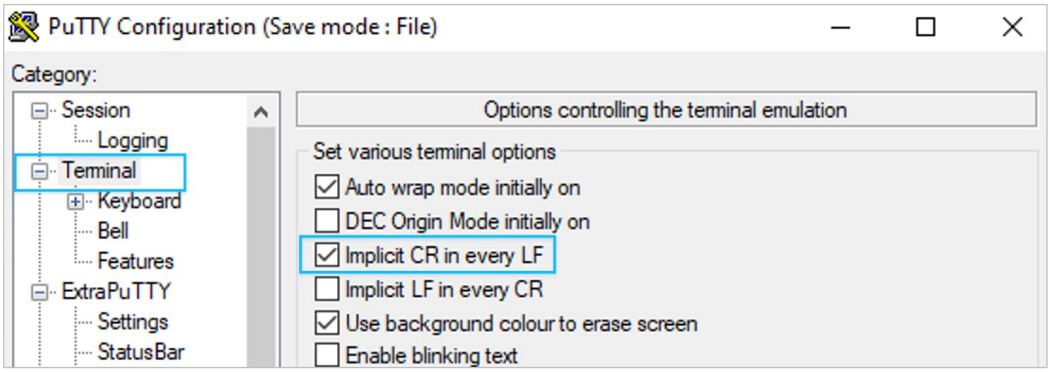

Рисунок 2 – Настройка `Putty`

Подключение:

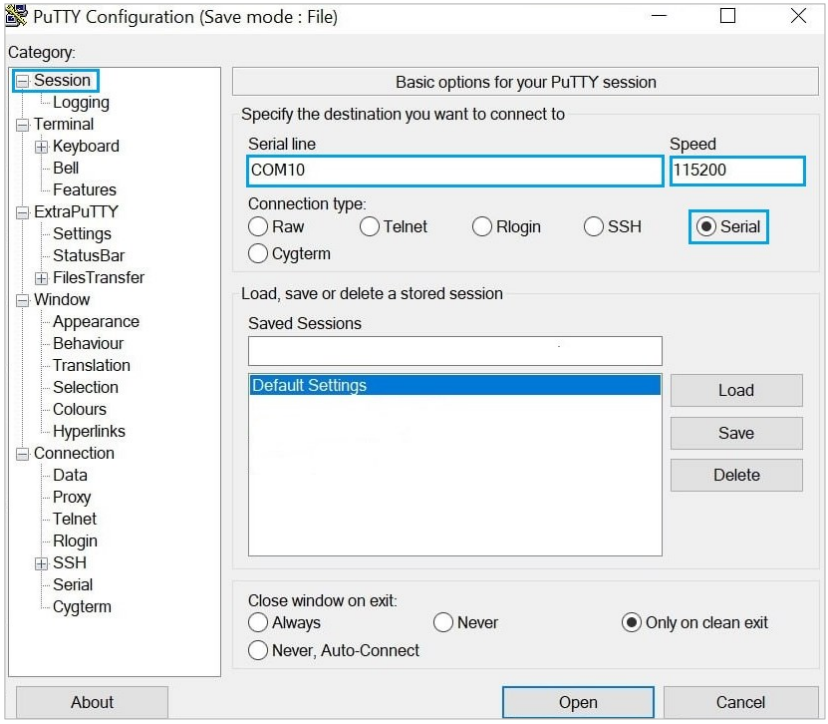

Рисунок 3 – Подключение к МК

Шаг 3: Подключение к адаптеру JTAG на ядре Core2

1. Запуск отладочного сервера OpenOCD:

1.1 Открытие окна командной строки (запуск `cmd.exe`)

1.2 Переход в папку с OpenOCD

```
cd C:\SDK\Tools\OpenOCD\bin
```

1.3 Запуск OpenOCD

```
.\openocd.exe -f interface\baikal\core2-jtag.cfg -f
board\baikal\evu_ba_2_5.cfg
```

2. Запуск программного отладчика GDB

Открытие нового окна командной строки и запуск GDB:

```
C:\SDK\Tools\toolchain\riscv32-none-elf\bin\riscv32-none-elf-gdb.exe -x
C:\SDK\Tools\scripts\bmcu.gdb
```

Шаг 4: Загрузка образа программы в память МК

Ввод в окне командной строки с GDB команды на запись образа в ТСМА:

```
restore C:\MCU\output\debug\MCU.bin binary 0x40010000
```

Шаг 5: Запуск программы

Переход на адрес начала программы:

```
monitor jump 0x40010000
```

Запуск программы:

```
continue
```

Информация в UART-консоли:

```
Very important data content (before Fibonacci): 42 73 69
Fibonacci element 1: 0
Fibonacci element 2: 1
Fibonacci element 3: 1
Fibonacci element 4: 2
Fibonacci element 5: 3
Fibonacci element 6: 5
Fibonacci element 7: 8
Very important data contentt (after Fibonacci): 13 73 69
```

Из вывода видно, что были изменены данные в массиве `very_important_data`, которые изменяться не должны. Это свидетельствует об ошибке в цикле заполнения массива `fib_arr` (строки 31 -33). Ошибку необходимо обнаружить и устранить в ходе отладки.

5.4 Подготовка к отладке

Ввод в окне командной строки с GDB команды на запись образа в память МК:

```
restore C:\MCU\output\debug\MCU.bin binary 0x40010000
```

Для удобства отладки будем использовать файл `.elf` с отладочными символами, генерируемый при сборке образа.

Указание в GDB пути к файлу `.elf`:

```
file "C:\MCU\output\debug\MCU.elf"
```

5.5 Отладка

Для поиска ошибки в программе необходимо проверить цикл заполнения массива `fib_arr` (строки 31 - 33). Для этого установим точку останова на строку 31, и добавим вывод используемых в этом цикле переменных – индекс цикла (`i`) и массивы `fib_arr` и `very_important_data`.

Установка точки останова

Установка точки останова на строке 31 для проверки значений, возвращаемых функцией `fib_arr[i] = sum(fib_arr[i - 1], fib_arr[i - 2])`:

```
break main.c:31
```

Настройка отслеживания значений переменных

Для проверки состояния программы при выполнении функции устанавливаем отслеживание значений ячеек массива:

Настройка отслеживания состояния массива `fib_arr`:

```
display fib_arr
```

Настройка отслеживания состояния массива `very_important_data`:

```
display very_important_data
```

Настройка отслеживания номера итерации:

```
display i
```

Если используются разные кодировки в файле `main.c` и в командной строке, при вводе имен функций или переменных это может привести к ошибкам вида:

```
No symbol in current context.
```

```
warning: could not convert from the host encoding (CP1252) to UTF-32.
```

Для их устранения можно вывести список локальных переменных (команда `info locals`) и скопировать необходимое имя из консоли.

Запуск программы

Переход на адрес начала программы:

```
monitor jump 0x40010000
```

Команда на запуск:

```
continue
```

Вывод отладочной информации

Остановка на строке 31:

```
Breakpoint 1, main () at main.c:31
31          for (size_t i = 2; i <= FIBNUM; ++i) {
1: i = 2
2: very_important_data = {42, 73, 69}
3: fib_arr = {0, 1, 0, 0, 0, 0, 0}
```

Дальнейшая пошаговая отладка:

```
(gdb) next
32              fib_arr[i] = sum(fib_arr[i - 1], fib_arr[i - 2]);
1: i = 2
2: very_important_data = {42, 73, 69}
3: fib_arr = {0, 1, 0, 0, 0, 0, 0}

(gdb) next
31          for (size_t i = 2; i <= FIBNUM; ++i) {
1: i = 2
2: very_important_data = {42, 73, 69}
3: fib_arr = {0, 1, 1, 0, 0, 0, 0}
```

```
(gdb) next
32              fib_arr[i] = sum(fib_arr[i - 1], fib_arr[i - 2]);
1: i = 3
2: very_important_data = {42, 73, 69}
3: fib_arr = {0, 1, 1, 0, 0, 0, 0}

(gdb) next
31          for (size_t i = 2; i <= FIBNUM; ++i) {
1: i = 3
2: very_important_data = {42, 73, 69}
3: fib_arr = {0, 1, 1, 2, 0, 0, 0}

(gdb) next
32              fib_arr[i] = sum(fib_arr[i - 1], fib_arr[i - 2]);
1: i = 4
2: very_important_data = {42, 73, 69}
3: fib_arr = {0, 1, 1, 2, 0, 0, 0}

(gdb) next
31          for (size_t i = 2; i <= FIBNUM; ++i) {
1: i = 4
2: very_important_data = {42, 73, 69}
3: fib_arr = {0, 1, 1, 2, 3, 0, 0}

(gdb) next
32              fib_arr[i] = sum(fib_arr[i - 1], fib_arr[i - 2]);
1: i = 5
2: very_important_data = {42, 73, 69}
3: fib_arr = {0, 1, 1, 2, 3, 0, 0}

(gdb) next
31          for (size_t i = 2; i <= FIBNUM; ++i) {
1: i = 5
2: very_important_data = {42, 73, 69}
3: fib_arr = {0, 1, 1, 2, 3, 5, 0}

(gdb) next
32              fib_arr[i] = sum(fib_arr[i - 1], fib_arr[i - 2]);
1: i = 6
2: very_important_data = {42, 73, 69}
3: fib_arr = {0, 1, 1, 2, 3, 5, 0}

(gdb) next
31          for (size_t i = 2; i <= FIBNUM; ++i) {
1: i = 6
2: very_important_data = {42, 73, 69}
3: fib_arr = {0, 1, 1, 2, 3, 5, 8}

(gdb) next
32              fib_arr[i] = sum(fib_arr[i - 1], fib_arr[i - 2]);
1: i = 7
2: very_important_data = {42, 73, 69}
3: fib_arr = {0, 1, 1, 2, 3, 5, 8}
```

```
(gdb) next
31          for (size_t i = 2; i <= FIBNUM; ++i) {
1: i = 7
2: very_important_data = {13, 73, 69}
3: fib_arr = {0, 1, 1, 2, 3, 5, 8}
```

Из отладочной информации видно, что после записи последнего элемента массива `fib_arr`, в цикле была выполнена лишняя итерация, в результате чего был изменен первый элемент массива `very_important_data`.

Лишняя итерация выполняется вследствие условия `i <= FIBNUM` (меньше либо равно) в строке 31. Программа делает лишнюю (седьмую) итерацию заполнения массива, вследствие чего затрагивается массив `very_important_data`.

5.6 Исправление ошибки и повторный запуск

Исправление ошибки в коде:

```
31      for (size_t i = 2; i <= FIBNUM; ++i) {
```

Из 31 строки удален символ `=`. Это устанавливает оператор «меньше» вместо «меньше либо равно».

Повторный запуск программы:

```
@ Go 0x40010000(0x22800, 0x0, 0x3)
Very important data content (before Fibonacci): 42 73 69
Fibonacci element 1: 0
Fibonacci element 2: 1
Fibonacci element 3: 1
Fibonacci element 4: 2
Fibonacci element 5: 3
Fibonacci element 6: 5
Fibonacci element 7: 8
Very important data contentt (after Fibonacci): 42 73 69
```

Из вывода видно, что массив `very_important_data` не затронут, ошибка устранена.

6 История изменений

Версия	Дата	Описание
1	30.06.2026	Начальная версия
2	02.07.2026	Исправления и уточнения
3	04.08.2026	Добавлен пункт 4.6.5 – «Отображение переменных без остановки исполнения»

7 Контактные данные

Официальный сайт
www.baikalelectronics.ru

Главный офис
www.baikalelectronics.ru/contacts

Электронная почта
info@baikalelectronics.ru

Телефон
+7 495 221-39-47