Микроконтроллер BE-U1000

Рекомендации по использованию

Работа с памятью

АО «БАЙКАЛ ЭЛЕКТРОНИКС», ИНН: 7707767484

АО «БАЙКАЛ ЭЛЕКТРОНИКС» оставляет за собой право вносить
любые изменения в настоящий документ без дополнительного уведомления

Содержание

1 ВВЕДЕНИЕ 3
2 УСЛОВНЫЕ ОБОЗНАЧЕНИЯ, ТЕРМИНЫ И СОКРАЩЕНИЯ 3
3 ОБЩАЯ ИНФОРМАЦИЯ 4
4 TCM 6
4.1 TCM Core 0/1 6
4.1.1 Чтение 7
4.1.2 Запись 7
4.1.3 Исполнение 7
4.1.4 Доступ к памяти 8
4.1.4.1 Фронтальный порт 8
4.1.4.2 Атомарные модификации 9
4.2 TCM Core 2 10
4.2.1 Чтение 11
4.2.2 Запись 11
4.2.3 Исполнение 12
5 EFLASH 13
5.1 Способы доступа 14
5.1.1 Регистры блока CRU для управления eFlash 14
5.1.2 Регистры контроллера eFlash 15
5.1.2.1 Настройка временных параметров 15
5.2 Чтение 18
5.2.1 Регистровый доступ 18
5.2.2 Непосредственный доступ 19
5.3 Стирание 20
5.4 Запись 21
5.4.1 Регистровый доступ 21

5.4.2 Непосредственный доступ 22
5.5 Исполнение 22
6 SRAM 23
6.1 Чтение 23
6.2 Запись 24
7 QSPI FLASH 24
8 ROM 26
9 ПРОИЗВОДИТЕЛЬНОСТЬ 27
ИСТОРИЯ ИЗМЕНЕНИЙ 28
КОНТАКТНЫЕ ДАННЫЕ 28

1 Введение

В данном документе представлена подробная информация, примеры и разъяснения по работе с различными типами памяти микроконтроллера BE-U1000 (далее – МК).

Документ снабжен примерами кода на языке Си.

Примеры кода в настоящем документе разработаны с использованием SDK v2.2.0 и проверены на плате EVU-BA v2.5.

Для ознакомления с общей информацией о МК см. документы «BE-U1000. Краткое описание» *и* «BE-U1000. Руководство пользователя».

Для ознакомления с характеристиками МК см. документы «BE-U1000. Техническая спецификация» и «BE-U1000. Reference Manual».

2 Условные обозначения, термины и сокращения

В документе используются сокращения и термины, указанные в таблице ниже.

Таблица 2-1 – Сокращения и термины

Обозначение	Расшифровка
МК	Микроконтроллер BE-U1000.
BootROM	Программа начального загрузчика, расположенная в ROM0.
Core 0/1/2	Ядро 0/1/2 микроконтроллера BE-U1000.
CRU	Control Registers Unit – блок МК, отвечающий за общие настройки: тактирование, сброс, выбор режимов работы ядер Core 1 и Core 2, GPIO и т.д.
DMA	Direct Memory Access – прямой доступ к памяти.
eFlash	Встроенная флэш-память МК. Содержит две области – Main и NVR.
eFlash Main	Основная область eFlash, предназначенная для хранения программ или данных пользователя.
eFlash NVR	Сегмент eFlash, предназначенный для хранения конфигурационных параметров МК и BootROM.
PLL	Phase-Locked Loop (блок фазовой автоподстройки частоты, блок ФАПЧ, блок PLL) – устройство, используемое для формирования тактовых сигналов с заданной частотой.
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. В настоящем документе данный термин обозначает любую SPI-память, подключенную к МК.
ROM	Read-Only Memory (постоянная память, ПЗУ) — тип памяти, которая хранит неизменяемые данные и инструкции.
SPI	Serial Peripheral Interface (последовательный периферийный интерфейс) – это синхронный последовательный интерфейс для обмена данными между микроконтроллерами и периферией (датчики, дисплеи, SD-карты).
SRAM	Внутренняя оперативная память МК, предназначенная для хранения временных данных.
TCM	Tightly Coupled Memory (тесно связанная память) — память, предназначенная для хранения критически важных с точки зрения производительности данных и кода.

Обозначение	Расшифровка
WC	Wait Cycle – количество тактовых сигналов, которое пропускается перед началом операции.
XIP	eXecute-In-Place – способ исполнения кода непосредственно с QSPI Flash, без предварительной загрузки в TCM.

3 Общая информация

Микроконтроллер BE-U1000 в ходе работы может использовать различные виды памяти. Каждый вид памяти обладает набором характеристик, которые определяют способы эффективного использования конкретного вида памяти.

В данном документе приведены основные характеристики тех видов памяти, которые доступны при работе с МК, затронуты темы настройки, типовых сценариев использования, режимов работы, доступных операций и скоростных характеристик различных видов памяти.

Пользователю доступны следующие типы памяти МК:

- **TCM**
  Встроенная быстрая оперативная память, предназначенная для хранения данных и кода, время доступа к которым имеет критическое значение для производительности.
- **SRAM**
  Встроенная оперативная память, предназначенная для хранения временных данных, необходимых для работы ПО.
- **eFlash**
  Встроенная энергонезависимая память, предназначенная для хранения данных, а также хранения и исполнения кода.
- **ROM**
  Неперезаписываемая память, содержимое которой определяется на производстве. Содержит несколько регионов, в которых хранятся служебный код и данные.
- **QSPI Flash**
  Общее название для внешней постоянной SPI-памяти. Поддерживается использование Standard SPI, Dual SPI и Quad SPI. Предназначена для хранения больших объемов данных. В МК реализована возможность подключения двух QSPI Flash через порты QSPI0 и QSPI1.

Встроенная память (TCM, SRAM, eFlash, ROM) является частью МК.

Внешняя память (QSPI Flash) подключается через внешние порты МК.

Схема взаимодействия и доступа к различным типам памяти представлена на рисунке 3-1.

Характеристики перечисленных типов памяти приведены в таблице 3-1.

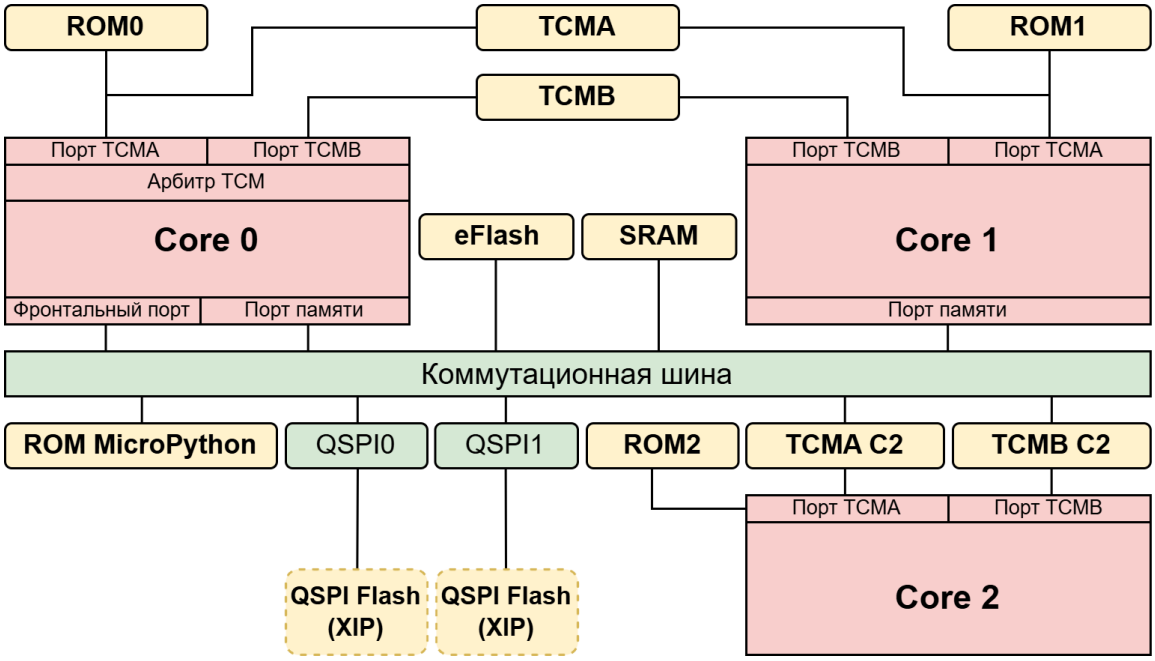


Рисунок 3-1 – Схема взаимодействия и доступа

Таблица 3-1 – Характеристики памяти

Тип памяти	Объем, КБ	Диапазон адресов	Доступ для ядер	Допустимые операции[3]
TCMA	64	0x4001 0000 – 0x4001 FFFF	Core 0 Core 1	RWX
TCMB	96	0x4002 0000 – 0x4003 7FFF	Core 0 Core 1	RWX
TCMA C2	2	0x4000 7800 – 0x4000 7FFF[1] 0x7000 F800 – 0x7000 FFFF[2]	Core 0 Core 1 Core 2	RWX
TCMB C2	8	0x4000 8000 – 0x4000 9FFF[1] 0x7001 0000 – 0x7001 1FFF[2]	Core 0 Core 1 Core 2	RWX
SRAM	32	0x7000 0000 – 0x7000 7FFF	Core 0 Core 1	RW
eFlash	256	0xA000 0000 – 0xA003 FFFF	Core 0 Core 1	RWX
NVR	4	0xA000 0000 – 0xA000 1000	Core 0 Core 1	RW
ROM0	64	0x4000 0000 – 0x4000 FFFF	Core 0	RX
ROM1	32	0x4000 8000 – 0x4000 FFFF	Core 1	RX
ROM2	16	0x40000000 – 0x40003FFF	Core 2	RX
ROM MicroPython	192	0xA080 0000 – 0xA082 FFFF	Core 0 Core 1	RX
QSPI0	до 16384	0x8000 0000 – 0x80FF FFFF	Core 0 Core 1	RWX[4]
QSPI1	до 16384	0x9000 0000 – 0x90FF FFFF	Core 0 Core 1	RWX[4]

[1] При прямом доступе из ядра Core 2
[2] При доступе из ядер Core 0 / 1
[3] R – чтение, W – запись, X - исполнение
[4] Операция записи (W) доступна, если память работает в стандартном (не XIP) режиме.

4 TCM

Tightly Coupled Memory (тесно связанная память) – встроенная энергозависимая оперативная память, предназначенная для хранения данных и кода, время доступа к которым имеет критическое значение для производительности. Доступ к ней осуществляется за один такт (при условии отсутствия необходимости арбитража).

Поддерживается исполнение кода.

Таблица 4-1 – Характеристики TCM

Тип памяти	Объем, КБ	Диапазон адресов	Доступ для ядер
TCMA	64	0x4001 0000 – 0x4001 FFFF	Core 0 Core 1
TCMB	96	0x4002 0000 – 0x4003 7FFF	Core 0 Core 1
TCMA C2	2	0x4000 7800 – 0x4000 7FFF[1] 0x7000 F800 – 0x7000 FFFF[2]	Core 0 Core 1 Core 2
TCMB C2	8	0x4000 8000 – 0x4000 9FFF[1] 0x7001 0000 – 0x7001 1FFF[2]	Core 0 Core 1 Core 2

[1] При прямом доступе из ядра Core 2
[2] При доступе из ядер Core 0 / 1

TCM разделена на **два региона**:

- TCMA
- TCMB

В МК предусмотрено **две TCM**:

- TCM **ядер Core 0 и 1** (далее TCM Core 0/1). Подключена напрямую к этим ядрам.
- TCM **ядра Core 2** (далее TCM Core 2). Подключена к ядру Core 2 и к коммутационной шине.

TCM является **двухпортовой** памятью, что при отсутствии конфликта адресов позволяет одновременно получать к ней доступ из разных ядер.

4.1 TCM Core 0/1

Каждое из ядер Core 0 и Core 1 связано с TCMA и TCMB отдельными шинами. Это позволяет ускорить операции, проводимые с разными регионами. Например:

- Исполнение программ из TCMA, чтение/запись данных из TCMB
- Чтение данных из TCMA, запись в память TCMB

Поддерживается одновременный доступ к TCMA или TCMB с обоих ядер при условии отсутствия конфликта адресов.

4.1.1 Чтение

Пример чтения данных из TCMA по явному адресу:

```
#define TCMA_BASEADDR (0x40010000U) ①
uint32_t AddressOffset = 0x400; ②
uint32_t a = *(uint32_t *)(TCMA_BASEADDR + AddressOffset); ③
```

① Установка базового адреса TCMA
② Установка смещения адреса чтения относительно базового адреса
③ Чтение хранящегося в TCMA значения по смещению `0x400` и сохранение в переменную `a`

4.1.2 Запись

Пример записи данных в TCMB по явному адресу:

```
#define TCMB_BASEADDR (0x40020000U) ①
uint32_t AddressOffset = 0x400; ②
uint32_t Data = 451; ③
*(uint32_t *)(TCMB_BASEADDR + AddressOffset) = Data; ④
```

① Установка базового адреса TCMB
② Установка смещения адреса записи относительно базового адреса
③ Запись значения `451` в переменную `Data`
④ Запись значения переменной `Data` в TCMB со смещением `0x400`

4.1.3 Исполнение

Возможны следующие сценарии исполнения кода из TCM:

- Образ программы был загружен непосредственно в TCM по DFU, через UART либо через JTAG.
- Образ программы находится в QSPI Flash, а МК запущен в режиме загрузки «QSPI». При старте МК Boot ROM перезапишет образ программы из QSPI Flash в TCM и передаст ему управление.
- Начальный старт программы осуществляется в режиме загрузки eFlash, после чего программа копирует код в TCM для исполнения.

Возможны гибридные варианты использования TCM. В ней могут храниться данные и код, требующие малого времени доступа. Например, основная часть программы исполняется из eFlash, при этом часть функций вызывается из TCM.

Для исполнения из TCM необходимо:

- Подготовить образ программы
- Записать его любым подходящим способом в память МК
- Загрузить МК в режиме исполнения кода из того типа памяти, в который загружен образ.

Подробнее о подготовке образа, его записи в память и выборе режима загрузки МК см. в документе «BE-U1000. Руководство пользователя».



Адрес в TCM, из которого будет запущено исполнение кода, должен быть выровнен (кратен 4 байтам).

Пример исполнения кода из TCMA:

```
__attribute__ ((section(".tcma_section")))
int32_t tcma_function(void) {
    return 69;                              ①
}

int32_t (*function)(void) = tcma_function;
int32_t a = function();                     ②
```

① Функция `tcma_function` расположена в секции `tcma_section`, которая находится в TCMA.
② Переменная `a` получает значение, которое возвращает функция `tcma_function`, исполняемая из TCM.



Предполагается, что код программы исполняется из региона памяти, отличного от TCM, и что при компоновке выделена секция `tcma_section`, которая размещается в TCM.

4.1.4 Доступ к памяти

4.1.4.1 Фронтальный порт

Как видно на рисунке 3-1, ядро Core 0 содержит следующие блоки:

- Фронтальный порт – позволяет DMA и USB обращаться к памяти TCMA и TCMB, а также используется для выполнения атомарных операций Core 0 и Core 1.
- Арбитр TCM – координирует запросы к TCMA и TCMB от фронтального порта и ядра Core 0.

Для доступа к TCMA и TCMB через фронтальный порт следует использовать смещение `0x10000000` относительно их базовых адресов.

Пример чтения из TCMA через фронтальный порт:

```
#define TCMA_BASEADDR (0x40010000U) ①
#define FRONTPORT_OFFSET (0x10000000U) ②
uint32_t AddressOffset = 0x400; ③

uint32_t a = *(uint32_t *)(TCMA_BASEADDR + FRONTPORT_OFFSET + AddressOffset); ④
```

① Установка базового адреса TCMA
② Установка значения смещения для доступа через фронтальный порт
③ Установка смещения адреса чтения
④ Чтение через фронтальный порт значения, хранящегося в TCMA по смещению `0x400` и сохранение значения в переменную `a`.

4.1.4.2 Атомарные модификации

Атомарная модификация — это неделимая операция чтения-модификации-записи, выполняемая без прерываний или вмешательства других потоков.

При обращениях к TCMA или TCMB через фронтальный порт запросы на чтение/запись обрабатываются арбитром TCM. Это позволяет обеспечить атомарность модификации в TCMA или TCMB при одновременном обращении к одному и тому же региону памяти двумя ядрами одновременно.

Для осуществления атомарной модификации используются специальные инструкции RISC-V:

- Эксклюзивного чтения (`lr` – load-reserved)
- Записи (`sc` – store-conditional).

Общий алгоритм атомарной модификации элемента TCMA при исполнении программы на ядре Core 0 и Core 1:

1. Чтение ячейки памяти TCMA с помощью инструкции эксклюзивного чтения `lr`, используя фронтальный порт, для обработки обращения арбитром TCM. Арбитр сохранит адрес чтения для последующего контроля атомарности.
2. Модификация значения переменной, которая хранит считанное на шаге 1 значение.
3. Запись модифицированного значения при помощи инструкции `sc`.
4. Проверка выполнения инструкции эксклюзивной записи.

Успешность выполнения операции зависит от того, производились ли другие операции модификации по данному адресу в промежутке между операциями эксклюзивного чтения и эксклюзивной записи.

Более подробную информацию об ограничениях, накладываемых на цикл атомарной модификации данных в TCM (общая длина цикла, допустимые инструкции и т.д.), можно получить по ссылке.

Пример атомарной модификации для переменной в памяти TCMB со смещением `0x20`:

```
// TCMB_BASE = 0x40020000U
// FRONTPORT_OFFSET = 0x10000000U
// VAR_TCMB_OFFSET = 0x20U

// Variable_address = TCMB_BASE + FRONTPORT_OFFSET + VAR_TCMB_OFFSET
//                  = 0x40020000 + 0x10000000 + 0x20


__asm__ volatile (" li t0, 0x50020020  \n"   ①
                  "loop:               \n"   ②
                  " lr.w t1, (t0)      \n"   ③
                  " addi t1, t1, 42    \n"   ④
                  " sc.w t2, t1, (t0)  \n"   ⑤
                  " bne t2, zero, loop \n"); ⑥
```

① Запись адреса переменной в регистр t0
② Начало цикла атомарной модификации
③ Эксклюзивное чтение значения по адресу, который содержится в регистре t0. Считанное значение

сохраняется в регистр t1.
④ Модификация значения в регистре t1. В данном случае – прибавление значения 42.
⑤ Операция эксклюзивной записи значения из регистра t1 в переменную, адрес которой содержится в регистре t0. Результат выполнения операции будет записан в регистр t2.
⑥ Проверка результата выполнения операции эксклюзивной записи. В случае, если модификация переменной не была проведена, в регистре t2 будет храниться ненулевое значение и поток исполнения перейдет на метку `loop`. В случае успешной атомарной модификации поток исполнения продолжится дальше.

4.2 TCM Core 2

МК содержит два региона TCM, предназначенные для работы ядра Core 2: TCMA C2 и TCMB C2. TCMA C2 и TCMB C2 может использоваться ядром Core 2 как для хранения исполняемого кода, так и для хранения временных данных.

TCMA C2 и TCMB C2 также соединены с коммутационной шиной, обеспечивающей ядрам Core 0 и Core 1 доступ к ним.

Поддерживается одновременный доступ к TCMA C2 или TCMB C2 с ядер Core 0/1 и Core 2 при условии отсутствия конфликта адресов.

Регионы TCMA C2 и TCMB C2 имеют разную адресацию со стороны ядра Core 2 (внутренняя адресация) и со стороны коммутационной шины (внешняя адресация).

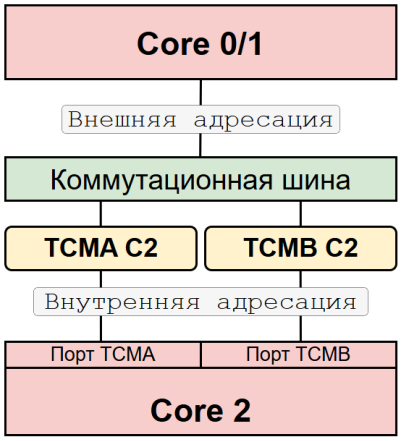


Рисунок 4-1 – Схема взаимодействия и доступа

Пример использования TCM C2:

- Код, исполняемый на ядре Core 2 (далее – «код Core 2»), помещается в память, доступную для ядра Core 0 / 1.
- Код Core 0 / 1 копирует код Core 2 в TCM C2.
- Ядро Core 0 / 1 подготавливает к запуску ядро Core 2. Адрес расположения кода Core 2 должен быть задан исходя из внутренней адресации.
- Ядро Core 0 / 1 запускает ядро Core 2, которое исполняет код, находящийся в TCM C2.

4.2.1 Чтение

Пример чтения памяти TCMA C2 при помощи кода Core 0 / 1 (используется внешний адрес):

```
#define CORE2_TCMA_SYS_BASE (0x7000F800U) ①
uint32_t AddressOffset = 0x400; ②
uint32_t a = *(uint32_t *)(CORE2_TCMA_SYS_BASE + AddressOffset); ③
```

① Установка внешнего адреса TCMA C2
② Установка смещения адреса чтения относительно базового адреса
③ Чтение хранящегося в TCMA C2 значения по внешнему адресу со смещением `0x400` и сохранение в переменную `a`.

Пример чтения памяти TCMB C2 при помощи кода Core 2 (используется внутренний адрес):

```
#define CORE2_TCMB_BASE (0x40008000U) ①
uint32_t AddressOffset = 0x400; ②
uint32_t a = *(uint32_t *)(CORE2_TCMB_BASE + AddressOffset); ③
```

① Установка внутреннего адреса TCMB C2
② Установка смещения адреса чтения относительно базового адреса
③ Чтение хранящегося в TCMB C2 значения по внутреннему адресу со смещением `0x400` и сохранение в переменную `a`.

4.2.2 Запись

Пример записи данных в TCMA C2 при помощи кода Core 0 / 1 (используется внешний адрес):

```
#define CORE2_TCMA_SYS_BASE (0x7000F800U) ①
uint32_t AddressOffset = 0x400; ②
uint32_t Data = 451; ③
*(uint32_t *)(CORE2_TCMA_SYS_BASE + AddressOffset) = Data; ④
```

① Установка внешнего адреса TCMA C2
② Установка смещения адреса записи относительно базового адреса
③ Запись значения `451` в переменную `Data`
④ Запись значения переменной `Data` по внешнему адресу TCMA C2 со смещением

Пример записи данных в TCMB C2 при помощи кода Core 2 (используется внутренний адрес):

```
#define CORE2_TCMB_BASE (0x40008000U) ①
uint32_t AddressOffset = 0x400; ②
uint32_t Data = 451; ③
*(uint32_t *)(CORE2_TCMB_BASE + AddressOffset) = Data; ④
```

① Установка внутреннего адреса TCMB C2
② Установка смещения адреса записи относительно базового адреса
③ Запись значения `451` в переменную `Data`
④ Запись значения переменной `Data` по внешнему адресу TCMA C2 со смещением

4.2.3 Исполнение

Пример запуска кода на ядре Core 2 из TCMA C2:

```
// ..... ①
/* Core2 code
1. Assembler code ②
40007800: 0001 c.nop // Dummy operation
40007802: BFFD c.j -2 // Jump up to the c.nop operation
2. Binary code (little endian): ③
0100 FDBF
Length: 8 bytes
*/
uint8_t core2_code[] = {0x01, 0x00, 0xFD, 0xBF}; ④
// Reset the Core2 itself, Core2 processor complex and TCM Arbitrator
CRU->SYSCR0 &= ~(CRU_SYSCR0_CORE2CXRSTN | CRU_SYSCR0_CORE2FPRSTN |
CRU_SYSCR0_CORE2RSTN); ⑤
// Releases Core2 processor complex reset
CRU->SYSCR0 |= CRU_SYSCR0_CORE2CXRSTN; ⑥
// Releases Core2 TCM Arbitrator (Front Port) reset
CRU->SYSCR0 |= CRU_SYSCR0_CORE2FPRSTN; ⑦
// Copy Core2 binary code to be executed to the Core2's TCMA memory area
for (size_t i = 0U; i < sizeof(core2_code); ++i) { ⑧
    *(__IO uint8_t *)(0x7000F800 + i) = core2_code[i];
}
// Set the Core2 code execution start address (TCMA memory region)
CRU->SYSCR2 = 0x40007800; ⑨
// Release Core2 reset
CRU->SYSCR0 |= CRU_SYSCR0_CORE2RSTN; ⑩
// ..... ⑪
```

① Код, исполняющийся на ядре Core 0 или Core 1.

② Код, который будет запущен на ядре Core 2 из TCMA C2, в виде ассемблерного кода представляет собой цикл из пустой операции `nop` и инструкции безусловного перехода на адрес `<текущий адрес>-2`. Используются инструкции из расширения набора команд `C`, которое поддерживается ядром Core 2.

③ Бинарное представление кода, который будет запущен на ядре Core 2.

④ Массив значений, подготовленный для копирования в TCMA C2. Содержит код, исполняемый на Core 2.

⑤ Установка сигнала сброса процессорного комплекса, фронтального порта и ядра Core 2.

⑥ Снятие сигнала сброса процессорного комплекса Core 2.

⑦ Снятие сигнала сброса фронтального порта ядра Core 2.

⑧ Ядро Core 0 или Core 1 копирует исполняемый код для Core 2 из подготовленного массива в память TCMA C2. Для доступа к памяти TCMA C2 используется внешняя адресация.

⑨ Установка стартового адреса для ядра Core 2. Используется внутренняя адресация.

⑩ Снятие сигнала сброса ядра Core 2. На этом шаге ядро Core начинает исполнение кода с заданного на предыдущем шаге адреса (в данном случае - с начального адреса региона TCMA C2).

⑪ Продолжение кода ПО, исполняющегося на ядре Core 0 или Core 1.



TCM C2 может быть использована ядрами Core 0 и Core 1 в качестве оперативной памяти.

5 eFlash

Память eFlash предназначена для хранения и исполнения кода, а также для пользовательских данных, требующих хранения в энергонезависимой памяти (настройки, журналы событий, медиаданные и т.д.)

Ядра Core 0 или Core 1 имеют доступ к eFlash по чтению, записи и исполнению. Ядро Core 2 не имеет доступа к eFlash.

eFlash разделена на два области:

- Main – основная область eFlash. Предназначена для пользовательских данных и исполняемого кода.
- NVR – служебная область, предназначенная для хранения системных настроек МК и служебной информации. Часть памяти NVR может быть использована для хранения пользовательских настроек. NVR имеет ту же адресацию, что и первые 4 КБ eFlash Main, доступ осуществляется при помощи установки в 1 битового поля `NVR` регистра `SYSCR0`.

Таблица 5-1 – Распределение eFlash

Область eFlash	Объем, КБ	Диапазон адресов	SYSCR0.NVR	Доступ для ядер	Допустимые операции
Main	256	0xA000 0000 – 0xA003 FFFF	0	Core 0 Core 1	RWX
NVR	4	0xA000 0000 – 0xA000 1000	1	Core 0 Core 1	RW

Память eFlash имеет иерархическую структуру:

- eFlash Main делится на 256 секторов по 1 КБ. Также для поблочного стирания сектора группируются в блоки по 8 секторов, всего 32 блока (Рисунок 5-1).
- eFlash NVR делится на 4 сектора по 1 КБ (Рисунок 5-2). Сектора NVR используются:
  - NVR1 – Служебный сектор eFlash для хранения заводских настроек. Пользователь не имеет возможности записи и стирания этого сектора.
  - NVR2 – Служебный сектор eFlash для хранения параметров BootROM. Пользователь может ограничить запись и стирание этого сектора.
  - NVR3 – Служебный сектор eFlash для хранения параметров BootROM и параметров пользователя
  - NVR4 – Служебный сектор eFlash для хранения параметров BootROM и параметров пользователя.



В связи с различными подходами к нумерации секторов NVR (с нуля или с единицы) в различных источниках, при именовании и индексировании секторов NVR могут возникать разночтения, например, сектор NVR2 может иметь индекс 1 при нумерации с нуля.

Количество циклов стирания/записи памяти eFlash: не менее 100 000.

Время хранения информации: не менее 10 лет.



Рисунок 5-1 – структура eFlash Main

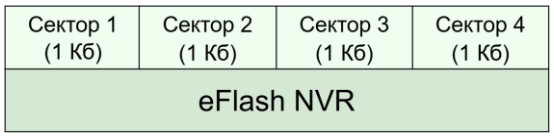


Рисунок 5-2 – структура eFlash NVR

5.1 Способы доступа

- **Регистровый доступ** – обращение к eFlash с помощью управляющих регистров контроллера eFlash. Позволяет осуществлять такие операции, как:
  - Конфигурация
  - Стирание (сектор, блок, вся память)
  - Запись 32-битного слова
  - Чтение 32-битного слова
  - Включение и выключение режима пониженного энергопотребления
- **Непосредственный доступ**: работа напрямую с массивом памяти eFlash через адресное пространство. Основной способ доступа в ходе исполнения ПО. Доступны следующие операции:
  - Чтение
  - Запись

5.1.1 Регистры блока CRU для управления eFlash

Некоторые настройки работы eFlash осуществляются с помощью регистров блока CRU.

Таблица 5-2 – Регистры блока CRU для управления eFlash

[Регистр].[Поле]	Описание
`SYSCR0.NVR`	Область памяти, с которой будут производиться операции: `0` – Main `1` – NVR Используется только для непосредственного доступа.
`SYSCR0.FLASHNVRDIS`	Запрет на доступ к NVR: `0` – разрешен доступ к NVR `1` – запрещен доступ к NVR. При этом запросы на операции с NVR будут обрабатываться как запросы на операции с eFlash Main
`FLASHNVRCR.NVR1DIS`	Запрет записи в сектор NVR2: `0` – разрешена запись в сектор NVR2 `1` – запрещена запись в сектор NVR2
`SYSCR0.DAMODDIS`	Запрет записи в Main при непосредственном доступе: `0` – разрешена запись в Main `1` – запрещена запись в Main

5.1.2 Регистры контроллера eFlash

Регистры контроллера eFlash задают режим его работы, а также могут быть использованы для чтения и записи



Перед началом проведения операций с eFlash необходимо убедиться в том, что внутренний сигнал сброса eFlash снят.

Подробное описание регистров контроллера eFlash и их полей приведено в документе «BE-U1000. Reference Manual».

Пример проверки снятия сигнала сброса eFlash:

```
while ((EFLASH->SR & EFLASH_SR_BUSY) != 0);
```

5.1.2.1 Настройка временны́х параметров

Для правильной работы eFlash должны быть настроены временные параметры в соответствии с частотой CCLK, на которой работает контроллер eFlash. Временные параметры настраиваются при помощи регистров `EFLASH_TIME0` – `EFLASH_TIME6`.

Подробное описание битовых полей регистров `EFLASH_TIME*` приведено в документе «BE-U1000. Reference Manual».

Для режимов загрузки («eFlash», «UART», «QSPI», «JTAG Ext», «UART MP», «Multi») источником тактового сигнала CCLK является внешний тактирующий сигнал (25 МГц), а временные параметры определяются значениями регистров по умолчанию, обеспечивающими чтение eFlash на частоте CCLK 25 МГц.

После снятия сигнала NRESET и завершения работы BootROM, для режимов загрузки «USB», «USB CDC», «USB MP», «JTAG INT» источником тактового сигнала CCLK является PLL, настроенный на частоту 100 МГц.

Пример вычисления значения поля `DS_T` регистра `EFLASH_TIME0` при тактовой частоте 100 МГц:

1. Вычисление периода установленной тактовой частоты:

   $$T_{CCLK} = \frac{1}{100\,000\,000} = 10 \times 10^{-9}\ \text{сек} = 10\ \text{нс}$$

2. Вычисление значения, которое необходимо записать в битовое поле `DS_T`:

   $$DS_T = \frac{t_x}{T_{CCLK}} = \frac{15\ \text{нс}}{10\ \text{нс}} = 1{,}5 \approx 2,$$

   где t_x– значение временного интервала для параметра. Полный перечень параметров и их значений приведен в разделе «*Time Intervals Configuration*» документа «BE-U1000. Reference Manual»).

Рекомендуемые значения регистров `EFLASH_TIME0` – `EFLASH_TIME6` для основных частот CCLK (25, 100, 200 МГц), а также значения после начальной загрузки указаны в таблице 5-3.

Таблица 5-3 – Значения полей регистров `EFLASH_TIME0` – `EFLASH_TIME6` в десятичной форме

Регистр	Поле	Значение параметра, мкс	Рекомендуемые значения полей для основных частот			Значения полей после начальной загрузки	
			CCLK = 25 МГц	CCLK = 100 МГц	CCLK = 200 МГц	Режимы «USB», «USB CDC», «USB MP», «JTAG INT»	Режимы «eFlash», «UART», «QSPI», «JTAG Ext», «UART MP», «Multi»
EFLASH_TIME0	AA_T	0,025	1	3	5	2	2
	SCE_T	8 000	200 000	3 200 000	1 600 000	900 000	400 000
	DS_T	0,015	1	2	3	2	1
EFLASH_TIME1	RCV_T	50	1 250	5 000	10 000	5 010	2500
	RHR_T	10	250	1 000	2 000	1010	500
	AS_T	0,002	1	1	1	1	1
EFLASH_TIME2	NVS_T	4	100	400	800	410	200
	PROG_T	8	200	8 000	1 600	900	400
	RW_T	0,1	3	10	20	10	5
EFLASH_TIME3	PGS_T	20	500	2 000	4 000	2010	1000
	CBD_T	0,02	1	2	4	2	1
	PREPROG_T	2,5	63	250	500	300	125
EFLASH_TIME4	TEST_T	0,05	2	5	10	5	3
	SBE_T	4 000	100 000	400 000	800 000	500 000	200 000
	RS_T	0,01	1	1	2	1	1
EFLASH_TIME5	RCR_T	0,2	5	20	40	20	10
	ERASE_T	3200	80 000	320 000	640 000	360 000	160 000
EFLASH_TIME6	PREPGS_T	5	125	500	1 000	510	250

© 2026 АО «БАЙКАЛ ЭЛЕКТРОНИКС»

Пример конфигурации поля `DS_T` регистра `EFLASH_TIME0`:

```
#define EFLASH_TIME0_DS_T_NS (2UL << EFLASH_TIME0_DS_T_Pos)
while (EFLASH->SR & EFLASH_SR_BUSY) != 0; ①
EFLASH->TIME0 &= ~(EFLASH_TIME0_DS_T); ②
EFLASH->TIME0 |= (EFLASH_TIME0_DS_T_NS); ②
EFLASH->CR = EFLASH_CR_OP_CODE_CFG | EFLASH_CR_RUN; ③
while (EFLASH->SR & EFLASH_SR_BUSY) != 0; ①
```

① Ожидание окончания предыдущей операции
② Запись нового значения в регистр `TIME0` контроллера eFlash
③ Выставление кода операции и флага начала операции

Пример конфигурации поля `RCV_T` регистра `EFLASH_TIME1`:

```
#define EFLASH_TIME1_RCV_T_100MHZ (5000UL << EFLASH_TIME1_RCV_T_Pos)
while ((EFLASH->SR & EFLASH_SR_BUSY) != 0); ①
EFLASH->TIME1 &= ~(EFLASH_TIME1_RCV_T); ②
EFLASH->TIME1 |= (EFLASH_TIME1_RCV_T_100MHZ); ②
EFLASH->CR = EFLASH_CR_OP_CODE_CFG | EFLASH_CR_RUN; ③
while ((EFLASH->SR & EFLASH_SR_BUSY) != 0); ①
```

① Ожидание окончания предыдущей операции
② Запись нового значения в регистр `TIME1` контроллера eFlash
③ Выставление кода операции и флага начала операции

Пример конфигурации поля `RW_T` регистра `EFLASH_TIME2`:

```
#define EFLASH_TIME2_RW_T_100MHZ (10UL << EFLASH_TIME2_RW_T_Pos)
while ((EFLASH->SR & EFLASH_SR_BUSY) != 0); ①
EFLASH->TIME2 &= ~(EFLASH_TIME2_RW_T);      ②
EFLASH->TIME2 |= EFLASH_TIME2_RW_T_100MHZ;  ②
EFLASH->CR = EFLASH_CR_OP_CODE_CFG | EFLASH_CR_RUN; ③
while ((EFLASH->SR & EFLASH_SR_BUSY) != 0); ①
```

① Ожидание окончания предыдущей операции
② Запись нового значения в регистр `TIME2` контроллера eFlash
③ Выставление кода операции и флага начала операции

Пример конфигурации поля `PREPROG_T` регистра `EFLASH_TIME3`:

```
#define EFLASH_TIME3_PREPROG_T_100MHZ (250UL << EFLASH_TIME3_PREPROG_T_Pos)
while ((EFLASH->SR & EFLASH_SR_BUSY) != 0);     ①
EFLASH->TIME3 &= ~(EFLASH_TIME3_PREPROG_T);     ②
EFLASH->TIME3 |= (EFLASH_TIME3_PREPROG_T_100MHZ);     ②
EFLASH->CR = EFLASH_CR_OP_CODE_CFG | EFLASH_CR_RUN;   ③
while ((EFLASH->SR & EFLASH_SR_BUSY) != 0);     ①
```

① Ожидание окончания предыдущей операции
② Запись нового значения в регистр `TIME3` контроллера eFlash
③ Выставление кода операции и флага начала операции

Пример конфигурации поля `TEST_T` регистра `EFLASH_TIME4`:

```
#define EFLASH_TIME4_TEST_T_100MHZ (5UL << EFLASH_TIME4_TEST_T_Pos)
while ((EFLASH->SR & EFLASH_SR_BUSY) != 0);     ①
EFLASH->TIME4 &= ~(EFLASH_TIME4_TEST_T);        ②
EFLASH->TIME4 |= (EFLASH_TIME4_TEST_T_100MHZ); ②
EFLASH->CR = EFLASH_CR_OP_CODE_CFG | EFLASH_CR_RUN;  ③
while ((EFLASH->SR & EFLASH_SR_BUSY) != 0);     ①
```

① Ожидание окончания предыдущей операции
② Запись нового значения в регистр `TIME4` контроллера eFlash
③ Выставление кода операции и флага начала операции

Пример конфигурации поля `RCR_T` регистра `EFLASH_TIME5`:

```
#define EFLASH_TIME5_RCR_T_100MHZ (20UL << EFLASH_TIME5_RCR_T_Pos)
while ((EFLASH->SR & EFLASH_SR_BUSY) != 0);    ①
EFLASH->TIME5 &= ~(EFLASH_TIME5_RCR_T);        ②
EFLASH->TIME5 |= (EFLASH_TIME5_RCR_T_100MHZ); ②
EFLASH->CR = EFLASH_CR_OP_CODE_CFG | EFLASH_CR_RUN;  ③
while ((EFLASH->SR & EFLASH_SR_BUSY) != 0);    ①
```

① Ожидание окончания предыдущей операции
② Запись нового значения в регистр `TIME5` контроллера eFlash
③ Выставление кода операции и флага начала операции

Пример конфигурации поля `PREPGS_T` регистра `EFLASH_TIME6`:

```
#define EFLASH_TIME6_PREPGS_T_100MHZ (500UL << EFLASH_TIME6_PREPGS_T_Pos)
while ((EFLASH->SR & EFLASH_SR_BUSY) != 0);
EFLASH->TIME6 &= ~(EFLASH_TIME6_PREPGS_T);
EFLASH->TIME6 |= (EFLASH_TIME6_PREPGS_T_100MHZ);
EFLASH->CR = EFLASH_CR_OP_CODE_CFG | EFLASH_CR_RUN;
while ((EFLASH->SR & EFLASH_SR_BUSY) != 0);
```

① Ожидание окончания предыдущей операции
② Запись нового значения в регистр `TIME6` контроллера eFlash
③ Выставление кода операции и флага начала операции

5.2 Чтение

5.2.1 Регистровый доступ

Алгоритм чтения из eFlash 32-битного значения:

1. В регистр `EFLASH_ADDR` записать адрес чтения.
2. В бит `[4]` регистра `EFLASH_CR` записать флаг желаемого региона чтения (eFlash – `0` или NVR – `1`), код операции чтения и флаг начала операции.
3. Дождаться окончания операции чтения посредством проверки флага занятости в регистре `EFLASH_SR`
4. Считать полученное значение из регистра `EFLASH_DR`
5. Повторить пункты 2-4, если требуется считать несколько подряд идущих адресов памяти eFlash. Значение регистра `EFLASH_ADDR` будет инкрементироваться автоматически.

Пример чтения данных из eFlash при регистровом доступе:

```
typedef enum {
    EFLASH_REGION_MAIN = 0,  // ①
    EFLASH_REGION_NVR  = 1   // ①
} eflash_region_read_e;

uint32_t eflash_read_word(eflash_region_read_e region, uint32_t byte_address) {
    while ((EFLASH->SR & EFLASH_SR_BUSY) != 0);  // ③
    if (region == EFLASH_REGION_MAIN) {          //
        EFLASH->ADDR = (uint16_t)((byte_address & 0x3FFFFUL) >> 2);  // ④
    } else {
        EFLASH->ADDR = (uint16_t)((byte_address & 0x3FFUL) >> 2);    // ④
    }
    EFLASH->CR = region | EFLASH_CR_OP_CODE_READ | EFLASH_CR_RUN;  // ⑤
    while ((EFLASH->SR & EFLASH_SR_BUSY) != 0);  // ③
    return EFLASH->RDATA;  // ⑥
}
uint32_t a = eflash_read_word(EFLASH_REGION_MAIN, 0x400);  // ①②
```

① Задание региона чтения
   `0` – eFlash Main
   `1` – eFlash NVR
② Установка смещения адреса чтения относительно базового адреса
③ Ожидание окончания предыдущей операции
④ Запись адреса чтения в регистр контроллера eFlash в зависимости от выбранного региона
⑤ Выставление флага региона, кода операции чтения и флага начала операции
⑥ Получение считанного значения в переменную `a`

5.2.2 Непосредственный доступ

При непосредственном доступе содержимое памяти eFlash можно получить путем обращения к адресам, указанным в таблице характеристик видов памяти. Дополнительных настроек не требуется.

Пример чтения данных из eFlash при непосредственном доступе:

```
#define EFLASH_BASEADDR 0xA0000000 ①
#define EFLASH_ADDROFFSET 0x1234 ②
uint32_t a = *(uint32_t *)(EFLASH_BASE + EFLASH_ADDROFFSET); ③
```

① Установка базового адреса eFlash
② Установка смещения адреса чтения относительно базового адреса
③ Вычисление адреса для чтения из eFlash и чтение значения из полученного адреса в переменную `a`

5.3 Стирание

Стирание eFlash переводит ячейки памяти в значение `0xFFFFFFFF`.

Т.к. память eFlash имеет иерархическую структуру и разделяется на сектора по 1 КБ и блоки по 8 секторов, то существует несколько вариантов операции стирания:

- Стирание всего eFlash Main или NVR. При этом учитываются настройки, контролирующие возможность проведения операций над отдельными регионами. Например, если поле `FLASHNVRDIS` регистра `SYSCR0` имеет значение `1`, то регион NVR стерт не будет.
- Стирание блока
  - eFlash Main: будет стерто 8 секторов (8 КБ)
  - NVR: будет стерто 2 сектора (2 КБ)
- Стирание сектора (1 КБ)

Для стирания eFlash необходимо:

- В случае стирания блока или сектора в регистр `EFLASH_ADDR` записать адрес стираемого объекта
- В регистр `EFLASH_CR` записать флаг желаемого региона стирания (eFlash Main или NVR), код операции стирания и флаг начала операции.
- Дождаться окончания операции посредством проверки флага занятости в регистре `EFLASH_SR`

Пример стирания основного массива eFlash:

```
uint32_t Region = 0; ①
while ((EFLASH->SR & EFLASH_SR_BUSY) != 0); ②
EFLASH->CR = Region | EFLASH_CR_OP_CODE_CHIP_ERASE | EFLASH_CR_RUN; ③
while ((EFLASH->SR & EFLASH_SR_BUSY) != 0); ②
```

① Выбор основного массива eFlash в качестве объекта стирания
② Ожидание окончания предыдущей операции
③ Выставление флага региона, кода операции стирания всей памяти и флага начала операции

Пример стирания блока в NVR

```
uint32_t Region = EFLASH_CR_NVR; ①
uint32_t BlockAddress = 0x01; ②
while ((EFLASH->SR & EFLASH_SR_BUSY) != 0); ③
EFLASH->ADDR = (uint16_t)((BlockAddress & 0x3FFFFUL) >> 2); ④
EFLASH->CR = Region | EFLASH_CR_OP_CODE_BLOCK_ERASE | EFLASH_CR_RUN; ⑤
while ((EFLASH->SR & EFLASH_SR_BUSY) != 0); ②
```

① Выбор NVR в качестве объекта стирания
② Значение адреса стираемого блока
③ Ожидание окончания предыдущей операции
④ Вычисление значения адреса и запись этого значения в регистр адреса `EFLASH→ADDR`
⑤ Выставление флага региона, кода операции стирания блока и флага начала операции

Пример стирания сектора eFlash:

```
uint32_t Region = 0; ①
uint32_t SectorAddress = 0x01; ②
while ((EFLASH->SR & EFLASH_SR_BUSY) != 0); ③
EFLASH->ADDR = (uint16_t)((SectorAddress & 0x3FFFFUL) >> 2); ④
EFLASH->CR = Region | EFLASH_CR_OP_CODE_SECTOR_ERASE | EFLASH_CR_RUN; ⑤
while ((EFLASH->SR & EFLASH_SR_BUSY) != 0); ②
```

① Выбор eFlash Main в качестве объекта стирания
② Значение адреса сектора стирания
③ Ожидание окончания предыдущей операции
④ Вычисление значения адреса и запись этого значение в регистр адреса `EFLASH→ADDR`
⑤ Выставление флага региона, кода операции стирания сектора и флага начала операции

5.4 Запись

После включения или сброса МК бит `DAMODDIS` регистра `SYSCR0` имеет значение `1`. Это значит, что запись при непосредственном доступе отключена. Для разрешения записи необходимо установить указанный бит в значение `0`.

Память eFlash позволяет записывать в отдельные ячейки памяти значения `0`. Обратное — запись значения `1` в ячейку, хранящую значение `0` — невозможна без проведения операции стирания.



Пример: память по адресу `0xA0000004` содержит значение `0b11100110`. Без предварительного проведения операции стирания в эту ячейку памяти можно корректно записать значения `0b00100010`, `0b11100000`, `0b00000000`. Однако для записи, значения `0b00011001` необходимо стереть регион памяти, к которому принадлежит ячейка памяти. Это можно сделать с помощью стирания всего eFlash Main, отдельного блока или сектора.

5.4.1 Регистровый доступ

Для записи 32-битного значения в память eFlash необходимо:

1. В регистр `EFLASH_ADDR` записать адрес записи 32-битного значения.
2. В регистр `EFLASH_WDATA` записать значение, которое требуется записать по заданному адресу.
3. В регистр `EFLASH_CR` записать флаг региона записи (eFlash Main или NVR), код операции записи и флаг начала операции.
4. Дождаться окончания операции посредством проверки флага занятости в регистре `EFLASH_SR`.
5. Повторить пункты 2-4, если требуется записать несколько подряд идущих адресов памяти eFlash. Значение регистра `EFLASH_ADDR` будет инкрементироваться автоматически.

Пример записи данных в eFlash при регистровом доступе:

```
uint32_t Region = 0; ①
uint32_t Data = 0x42; ②
uint32_t Address = 0x100; ③
EFLASH->ADDR = (uint16_t)((Address & 0x3FFFFUL) >> 2); ④
EFLASH->WDATA = Data; ⑤
EFLASH->CR = Region | EFLASH_CR_OP_CODE_WRITE | EFLASH_CR_RUN; ⑥
while ((EFLASH->SR & EFLASH_SR_BUSY) != 0); ⑦
```

① Выбор eFlash Main в качестве региона записи
② Запись в переменную `Data` значения `0x42`
③ Смещение адреса записи относительно начала выбранного региона
④ Запись значения адреса в регистр `EFLASH_ADDR`
⑤ Запись значения, которое будет записано по выбранному адресу, в регистр `EFLASH_WDATA`
⑥ Выставление флага региона, кода операции записи и флага начала операции
⑦ Ожидание завершения операции записи

5.4.2 Непосредственный доступ

При непосредственном доступе запись в eFlash осуществляется путем обращения к адресам, указанным в таблице характеристик видов памяти. Дополнительных настроек не требуется.

Пример записи данных в eFlash в при непосредственном доступе:

```
#define EFLASH_BASEADDR 0xA0000000 ①
uint32_t Data = 0x42; ②
uint32_t AddressOffset = 0x100; ③
*(uint32_t *)(EFLASH_BASEADDR + AddressOffset) = Data; ④
```

① Базовый адрес eFlash (см. таблицу характеристик видов памяти)
② Переменная, содержащее значение, которое будет записано по выбранному адресу eFlash
③ Смещение адреса записи относительно базового адреса eFlash
④ Запись значения, содержащегося в переменной `Data`, по выбранному адресу в при непосредственном доступе.

5.5 Исполнение

Код, содержащийся в eFlash, может быть запущен на исполнение.

Для этого необходимо:

- Подготовить образ программы для МК
- Записать его любым подходящим способом в eFlash
- Загрузить МК в режиме исполнения кода из eFlash.

Подробнее о подготовке образа, его записи в память и выборе режима загрузки МК см. в документе «BE-U1000. Руководство пользователя».

Возможны гибридные варианты использования eFlash. В ней могут одновременно храниться как часть пользовательского кода, так и данные, требующие энергонезависимого хранения. Например, основная часть программы исполняется из TCM, при этом часть функций вызывается из eFlash.



Адрес в eFlash, из которого будет запущено исполнение кода, должен быть выровнен (кратен 4 байтам).

Пример исполнения кода из eFlash:

```
__attribute__ ((section (".eflash_section")))
uint32_t eflash_function(void) { ①
    return 69;
}
uint32_t (*function)(void) = eflash_function;
uint32_t a = function();②
```

① Функция `eflash_function` расположена в секции `eflash_section`, а значит должна находиться в eFlash Main.

② Переменная `a` получает значение, которое возвращает функция `eflash_function`, исполняемая из eFlash.

Предполагается, что код программы исполняется не из eFlash, и что при компоновке выделена секция `eFlash_section`, которая размещается в eFlash Main.

6 SRAM

Память SRAM предназначена для хранения временных данных, которые используются во время исполнения кода. Время доступа к памяти SRAM выше, чем время доступа к TCM – 4 такта CCLK. SRAM не поддерживает исполнение кода.

Таблица 6-1 – Характеристики SRAM

Объем, КБ	Диапазон адресов	Доступ для ядер	Допустимые операции[3]
32	0x7000 0000 – 0x7000 7FFF	Core 0 Core 1	RW

Пример использования SRAM:

Код исполняется из TCM ядром Core 0 или Core 1.
При этом необходимо организовать обмен информацией между компонентами МК с помощью одного из блоков DMA.
В этом случае SRAM может использоваться в качестве источника или приемника информации. Такая конфигурация позволяет уменьшить конкуренцию компонентов системы (ядер Core 0, Core 1 и блоков DMA) за TCM.

6.1 Чтение

Пример чтения данных из SRAM:

```
#define SRAM_BASEADDR (0x70000000U) ①
uint32_t AddressOffset = 0x400; ②
uint32_t a = *(uint32_t *)(SRAM_BASEADDR + AddressOffset); ③
```

① Установка базового адреса SRAM

② Установка смещения адреса чтения относительно базового адреса

③ Чтение значения, хранящегося в SRAM по указанному смещению и его сохранение в переменную `a`

6.2 Запись

Пример записи данных в SRAM:

```
#define SRAM_BASEADDR (0x70000000U) ①
uint32_t AddressOffset = 0x400; ②
uint32_t Data = 451; ③
*(uint32_t *)(SRAM_BASEADDR + AddressOffset) = Data; ④
```

① Установка базового адреса SRAM
② Установка смещения адреса записи относительно базового адреса
③ Переменная, хранящая значение для записи в SRAM
④ Запись значения, хранящегося в переменной `Data`, в SRAM по указанному смещению.

7 QSPI Flash

К МК через порты QSPI0 и QSPI1 может быть подключена внешняя энергонезависимая память с интерфейсом SPI. Она предназначена для хранения и исполнения кода, а также для пользовательских данных, требующих хранения в энергонезависимой памяти (настройки, журналы событий, медиаданные и т.д.). Порты QSPI0 и QSPI1 могут работать с внешней памятью в режимах Standard SPI, Dual SPI или Quad SPI.

Через порты QSPI0 и QSPI1 можно одновременно подключить две микросхемы памяти объемом не более 16 384 КБ каждая.

Также, для QSPI Flash предусмотрен режим «XIP» в котором QSPI Flash отображается на адресное пространство МК. В режиме «XIP» QSPI Flash доступна только для чтения и исполнения.

Таблица 7-1 – Характеристики QSPI Flash

Тип памяти	Объем, КБ	Диапазон адресов	Доступ для ядер	Допустимые операции[1]
QSPI0	до 16384	0x8000 0000 – 0x80FF FFFF	Core 0 Core 1	RWX[2]
QSPI1	до 16384	0x9000 0000 – 0x90FF FFFF	Core 0 Core 1	RWX[2]

[1] R – чтение, W – запись, X – исполнение
[2] Операция записи (W) доступна через регистры контроллера, если память работает в стандартном (не XIP) режиме.

Алгоритм исполнения кода из QSPI Flash:

- Подготовить образ программы, предназначенного для работы на ядре Core 0 или Core 1 и исполнения из QSPI Flash.
- Записать образ программы в QSPI Flash с помощью внешнего программатора, либо с помощью встроенных средств МК.
- Установить режим загрузки «QSPI» и перезагрузить МК.
- Вспомогательный модуль программы будет скопирован в TCM и запущен. Он настраивает интерфейс QSPI МК и внешней микросхемы памяти, а затем передает управление по адресу записанного на QSPI Flash основного модуля программы. Если

данные действия выполнены корректно, то основной модуль программы начнет исполняться.



Подробнее о подготовке образа программы, его записи в память и выборе режима загрузки МК см. в документе «BE-U1000. Руководство пользователя».

Настройка интерфейса QSPI и порядок его перевода в режим XIP описаны в документе *«BE-U1000. Reference Manual»*. Перечень необходимых настроек зависит от используемой модели SPI-памяти.

Пример вспомогательного модуля см. в SDK: `/Projects/HAL_examples/QSPI/QSPI_XIPLoader`.

Пример запуска кода из QSPI Flash на интерфейсе QSPI1 в режиме XIP (псевдокод):

```
#define __QSPI_APP_ADDR__ (0x10000U) ①
// 1. Configure clock: PLL, CCLK, PCLKx, HCLK, etc. {an_clock}
clock_cfg(); ②
/* 2. Configure {MCU} {QSPI} interface: enable QSPI and GPIO clock,
set AF for corresponding GPIO pins, GPIO drive strength, pull up/down, etc.
*/
mcu_qspi_gpio_clk_en(); ③
mcu_qspi_clk_en();
mcu_qspi_gpio_set_af();
mcu_qspi_gpio_set_driver_strength();
mcu_qspi_gpio_set_pull();
mcu_qspi_init();
mcu_qspi_en();
// 3. Switch external flash memory IC in Quad SPI mode
ext_flash_quad_en(); ④
// 4. Switch {MCU} {QSPI} interface in Quad SPI mode
mcu_qspi_quad_en(); ⑤
// 5. Switch external flash memory IC to XIP mode
ext_flash_xip_en(); ⑥
// 6. Switch {MCU} {QSPI} interface to XIP mode
mcu_qspi_xip_en(); ⑦
// 7. Run user {FW} from QSPI (XIP mode)
void(*qspi_app)(void) = (void(*)(void))(XIP1_BASE + __QSPI_APP_ADDR__); ⑧
qspi_app(); ⑨
```

① Задание значения смещения адреса начала пользовательского кода относительно базового адреса QSPI XIP.
② Настройка тактирования компонентов МК.
③ Настройка блока GPIO и QSPI: включение тактирования блоков, конфигурация выводов GPIO, инициализация регистров QSPI и т.д.
④ Перевод QSPI Flash в режим Quad SPI.
⑤ Перевод периферийного блока QSPI МК, к которому подключена внешняя QSPI Flash, в режим Quad SPI.
⑥ Перевод QSPI Flash в режим XIP.
⑦ Перевод периферийного блока QSPI МК, к которому подключена QSPI Flash, в режим XIP.
⑧ Определение указателя на функцию, код которой размещен во внешней QSPI Flash памяти, с помощью базового адреса QSPI1 и значения смещения адреса (см. пункт 1).
⑨ Вызов функции, находящейся в QSPI Flash.

Некорректное конфигурирование тактирования (например, отключение блока PLL) может привести к ошибке исполнения кода.

8 ROM

ROM – это постоянная неперезаписываемая память МК, предназначенная для хранения служебного кода программы и данных. Разделена на регионы ROM0, ROM1, ROM2, ROM MicroPython.

Таблица 8-1 – Характеристики памяти

Тип памяти	Объем, КБ	Диапазон адресов	Описание
ROM0	64	0x4000 0000 – 0x4000 FFFF	Содержит модули: ▪ Загрузчик BootROM. ▪ Библиотека `math.h` (код математических функций) Регион доступен только ядру Core 0.
ROM1	32	0x4000 8000 – 0x4000 FFFF	Содержит модули: ▪ DSP (функции цифровой обработки сигналов) ▪ Библиотека `math.h` (код математических функций) Регион доступен только ядру Core 1.
ROM2	16	0x4000 0000 – 0x4000 3FFF	Содержит код программно эмулируемого адаптера JTAG, запускаемого на ядре Core 2 и позволяющего проводить отладку кода, исполняемого на ядрах Core 0 и Core 1. Регион доступен только ядру Core2.
ROM MicroPython	192	0xA080 0000 – 0xA082 FFFF	Содержит исполняемый код интерпретатора MicroPython.

Код математических функций, их набор и значения адресов вызова одинаковы для ROM0 и ROM1.

9 Производительность

В этом разделе представлена информация по скорости чтения и копирования данных из / в различные типы и регионы памяти. Скорость может варьироваться в зависимости от места исполнения кода.

Все представленные данные несут ознакомительный характер. Фактические значения зависят от кода и ключей компиляции.

Код, используемый для измерений, находится в SDK.

Таблица 9-1 – Установившаяся скорость копирования при исполнении кода из eFlash

Память-источник	Память-адресат		
	TCMA	TCMB	eFlash
TCMA	360 МБ/с	360 МБ/с	89,4 кБ/с
TCMB	335 МБ/с	337 МБ/с	89,4 кБ/с
eFlash	36,2 МБ/с	36,2 МБ/с	89,2 кБ/с
SRAM	61,0 МБ/с	61,0 МБ/с	89,3 кБ/с
ROM MicroPython	50,4 МБ/с	50,4 МБ/с	89,3 кБ/с
XIP 50 МГц, WC 0	7,81 МБ/с	7,81 МБ/с	88,4 кБ/с
XIP 25 МГц, WC 0	4,92 МБ/с	4,92 МБ/с	87,8 кБ/с
XIP 50 МГц, WC 14	5,05 МБ/с	5,05 МБ/с	87,9 кБ/с
XIP 25 МГц, WC 14	2,92 МБ/с	2,92 МБ/с	86,8 кБ/с

Таблица 9-2 – Установившаяся скорость чтения при исполнении кода из eFlash

Тип / регион памяти	Скорость
TCMA	580 МБ/с
TCMB	560 МБ/с
eFlash	39,3 МБ/с
SRAM	70,3 МБ/с
ROM MicroPython	60,0 МБ/с
XIP50 МГц, WC 0	7,97 МБ/с
XIP25 МГц, WC 0	4,98 МБ/с
XIP 50 МГц, WC 14	5,11 МБ/с
XIP 25 МГц, WC 14	2,93 МБ/с

Таблица 9-3 – Установившаяся скорость записи при исполнении кода из eFlash

Тип / регион памяти	Скорость
TCMA	660 МБ/с
TCMB	660 МБ/с
eFlash	89,4 кБ/с

История изменений

Версия	Дата	Описание
1	10.06.2026	Начальная версия
2	18.06.2026	Исправления ошибок
2.1	29.10.2026	Исправление скорости для eFlash в таблице 9-3.

Контактные данные

Официальный сайт
www.baikalelectronics.ru

Главный офис
www.baikalelectronics.ru/contacts

Электронная почта
info@baikalelectronics.ru

Телефон
+7 495 221-39-47