Использование STM32F3 / STM32G4 CCM SRAM с IAR ™ EWARM, Keil® MDK-ARM и инструментами на основе GNU

Введение

В этом документе представлена SRAM память ядра (CCM), доступная на микроконтроллерах STM32F3 / STM32G4, и описано, что требуется для выполнения части кода приложения из этой области памяти с использованием различных инструментальных средств.

В первом разделе представлен обзор CCM SRAM, а в следующих разделах описаны шаги, необходимые для выполнения части

код приложения из CCM SRAM с использованием следующих инструментов:

- IAR ™ EWARM
- KEIL® MDK-Arm®
- Наборы инструментов на основе GNU

Процедуры, описанные в этом документе, применимы к другим областям SRAM, таким как ОЗУ данных CCM некоторых устройств **STM32F4** или внешняя SRAM.

В таблице ниже перечислены микроконтроллеры STM32, в которые встроено CCM SRAM.

Таблица 1. Применимые продукты

Reference		Products
STM32F3/STM32G4	STM32F3	STM32F303 line, STM32F334 line STM32F328C8, STM32F328K8, STM32F328R8 STM32F358CC, STM32F358RC, STM32F358VC STM32F398RE, STM32F398VE, STM32F398ZE
	STM32G4	STM32G4 Series

Этот документ относится к микроконтроллерам на базе STM32F3 / STM32G4 Arm®.

Примечание.

Arm является зарегистрированным товарным знаком Arm Limited (или ее дочерних компаний) в США и / или в других странах.

1.1 Применение

CCM SRAM тесно связана с ядром Arm® Cortex® для выполнения кода на максимальной системной тактовой частоте без каких-либо потерь состояния ожидания. Это также приводит к значительному сокращению времени выполнения критической задачи по сравнению с выполнением кода из флэш-памяти.

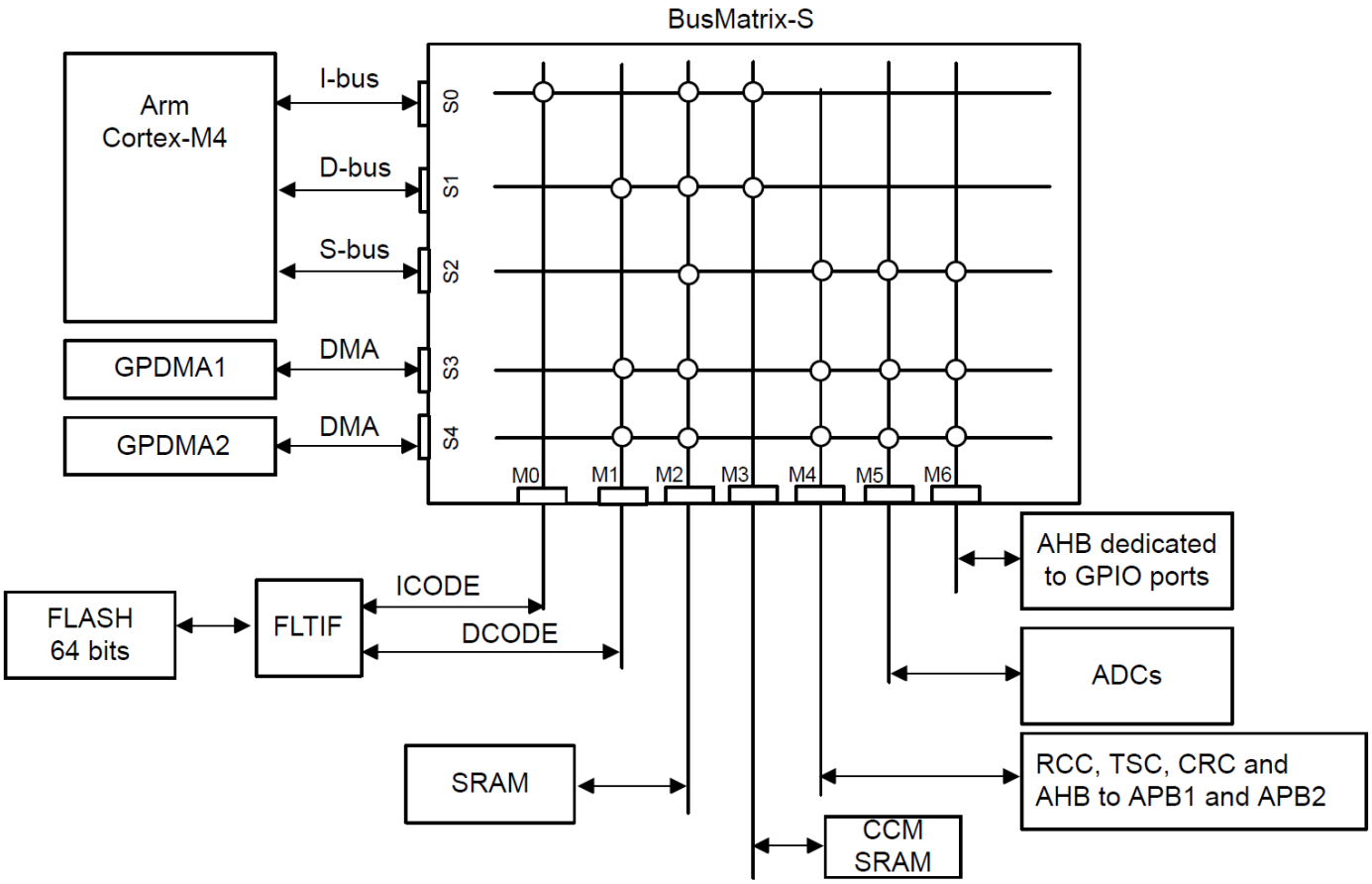
CCM SRAM обычно используется для подпрограмм реального времени и интенсивных вычислений, таких как следующие:

- контуры управления цифровым преобразованием энергии (импульсные источники питания, освещение)
- управление трехфазным двигателем с ориентацией на поле
- задачи DSP (цифровая обработка сигналов) в реальном времени

Когда код находится в SRAM CCM, а данные хранятся в обычном SRAM, ядро Cortex-M4 находится в оптимальной гарвардской конфигурации. Выделенная па-

мять с нулевым временем ожидания подключена к каждой из своих I-bus и D-bus (см. Рисунки ниже) и, таким образом, может работать со скоростью 1,25 DMIPS / МГц с детерминированной производительностью 90 DMIPS в STM32F3 и 213 DMIPS в STM32G4. Это также гарантирует минимальную задержку, если процедуры обслуживания прерывания помещаются в SRAM CCM.

Figure 1. STM32F3 devices system architecture



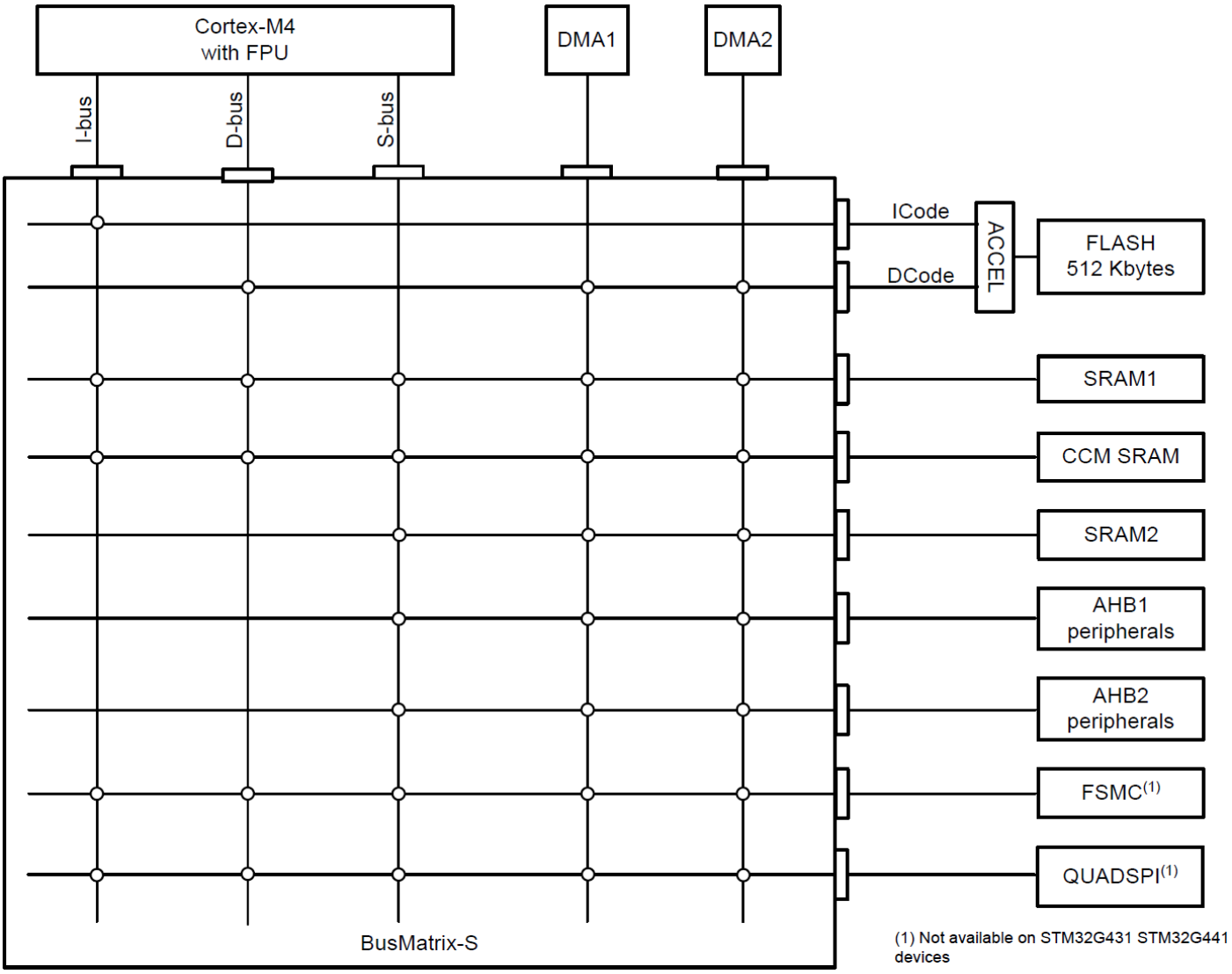
Пример

Сравнительный анализ микроконтроллеров STM32F103 и STM32F303 с использованием библиотеки MC STMicroelectronics V3.4 показывает, что в случае управления одним двигателем с использованием алгоритма трех шунтов общее время выполнения FOC для STM32F303 составляет 16,97 мкс по сравнению с 21,3 мкс для STMF103 (см. примечание ниже), с ядром FOC и контурами сердечника без датчика, работающими из CCM SRAM для STM32F303. Это означает, что STM32F303 на 20,33% быстрее, чем STM32F103, благодаря CCM SRAM.

Примечание: подпрограммы FOC запрограммированы на структурированном языке C, поэтому приведенные выше значения не представляют собой максимально быстрое выполнение как для STM32F103, так и для STM32F303. Кроме того, время выполнения также зависит от используемого компилятора и его версии.

Когда CCM SRAM не используется для кода, она может хранить данные, такие как дополнительная память SRAM. Не рекомендуется размещать в CCM и код, и данные вместе, поскольку ядро Cortex должно извлекать код и данные из одной и той же памяти с риском коллизий. Тогда ядро находится в конфигурации фон Неймана, и его производительность падает с 1,25 DMIPS / МГц до менее 1 DMIPS / МГц.

Figure 2. **STM32G4 devices system architecture**



1.2 Возможности CCM SRAM

В таблице ниже приведены характеристики CCM SRAM различных продуктов STM32. Более подробная информация представлена в следующих разделах.

Table 2. **CCM SRAM main features**

Feature/ products	STM32F303xB/C STM32F358xx	STM32F303x6/8 STM32F334xx STM32F328xx	STM32F303xD/E STM32F398xx	STM32G47xx STM32G84xx	STM32G431x STM32G441x
Size (Kbytes)	8	4	16	32	10
Mapping	0x1000 0000			0x1000 0000 and can be aliased at 0x2001 8000	0x1000 0000 and can be aliased at 0x2000 5800
Parity check	Yes				
Write protection	Yes, with 1-Kbyte page granularity				
Read protection	No			Yes	
Erase	No			Yes	
DMA access	No			• No if mapped at 0x1000 0000 • Yes if mapped at 0x2001 8000	• No if mapped at 0x1000 0000 • Yes if mapped at 0x2000 5800

1.2.1 Отображение SRAM CCM

CCM SRAM отображается по адресу 0x1000 0000.

На устройствах STM32G4 CCM SRAM имеет псевдоним по адресу, следующему за концом SRAM2, предлагая непрерывное адресное пространство с SRAM1 и SRAM2.

1.2.2 Переназначение CCM SRAM

В отличие от обычного SRAM, CCM SRAM не может быть переназначена по адресу 0x0000 0000.

1.2.3 Защита от записи CCM SRAM

CCM SRAM может быть защищена от нежелательных операций записи с размером страницы 1 Кбайт.

Защита от записи включается через регистр защиты SYSCFG CCM SRAM. Это механизм однократной записи: после включения защиты от записи на данной странице CCM SRAM путем программирования соответствующего бита на 1, ее можно очистить только с помощью сброса системы. Дополнительные сведения см. В справочном руководстве по продукту.

1.2.4 Проверка четности CCM SRAM

Реализованная проверка четности отключена по умолчанию и может быть включена пользователем при необходимости с помощью бита опции (бит SRAM_PE). Когда этот бит опции сброшен, проверка четности включена.

1.2.5 Защита от чтения CCM SRAM (только на устройствах STM32G4)

CCM SRAM также может быть защищена от считывания с помощью байта опции RDP. В защищенном состоянии CCM SRAM не может быть прочитана или записана с помощью JTAG или последовательного порта отладки, а также при выборе загрузки в системной флэш-памяти или загрузки в SRAM.

CCM SRAM стирается, когда защита от считывания изменяется с уровня 1 на уровень 0.

1.2.6 Стирание CCM SRAM (только на устройствах STM32G4)

CCM SRAM можно стереть программно, установив бит CCMER в регистре управления конфигурацией и состояния системы CCM SRAM.

CCM SRAM также можно стереть при сбросе системы в зависимости от бита опции CCMSRAM_RST в байтах опции пользователя.

2 Выполнение кода приложения из CCM SRAM с помощью набора инструментов IAR EWARM

2.1 Выполнение простого кода из CCM SRAM (кроме обработчика прерывания)

Простой код может состоять из одной или нескольких функций, на которые не ссылается обработчик прерывания. Если на код ссылается обработчик прерывания, выполните действия, описанные в Разделе 2.2.

EWARM предоставляет возможность разместить одну или несколько функций или весь исходный файл в CCM SRAM. Эта операция требует, чтобы в файле компоновщика (.icf) был определен новый раздел для размещения кода, который будет помещен в CCM SRAM.

Этот раздел копируется в CCM SRAM при запуске. Необходимые шаги перечислены ниже:

1. Определите адресную область для CCM SRAM, указав начальный и конечный адреса.
2. Попросите компоновщик скопировать при запуске раздел с именем .ccmram из флэш-памяти в CCM SRAM.
3. Сообщите компоновщику, что раздел кода .ccmram должен быть помещен в область CCM SRAM.

На рисунке ниже показан пример кода, реализующего эти операции.

Figure 3. EWARM linker update

```
/*###ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\cortex_v1_0.xml" */
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x08000000;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = 0x08000000;
define symbol __ICFEDIT_region_ROM_end__   = 0x0803FFFF;
define symbol __ICFEDIT_region_RAM_start__ = 0x20000000;
define symbol __ICFEDIT_region_RAM_end__   = 0x20009FFF;
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__ = 0x400;
define symbol __ICFEDIT_size_heap__   = 0x200;
/**** End of ICF editor section. ###ICF###*/


define memory mem with size = 4G;
define region ROM_region   = mem:[from __ICFEDIT_region_ROM_start__   to __ICFEDIT_region_ROM_end__];
define region RAM_region   = mem:[from __ICFEDIT_region_RAM_start__   to __ICFEDIT_region_RAM_end__];

1 define region CCMRAM_region   = mem:[from 0x10000000   to 0x10001FFF];

define block CSTACK    with alignment = 8, size = __ICFEDIT_size_cstack__   { };
define block HEAP      with alignment = 8, size =   ICFEDIT_size_heap__     { };

2 initialize by copy { readwrite, section .ccmram };

do not initialize  { section .noinit };

place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec };

place in ROM region   { readonly };

3 place in CCMRAM region   {section .ccmram};

place in RAM_region   { readwrite,
                        block CSTACK, block HEAP };
```

Defines the address zone for CCM SRAM.

'Initialize by copy ' tells the linker to copy this section at startup time.

Places .ccmram section at CCM SRAM defined above.

Примечание. Эта процедура не подходит для обработчиков прерываний.

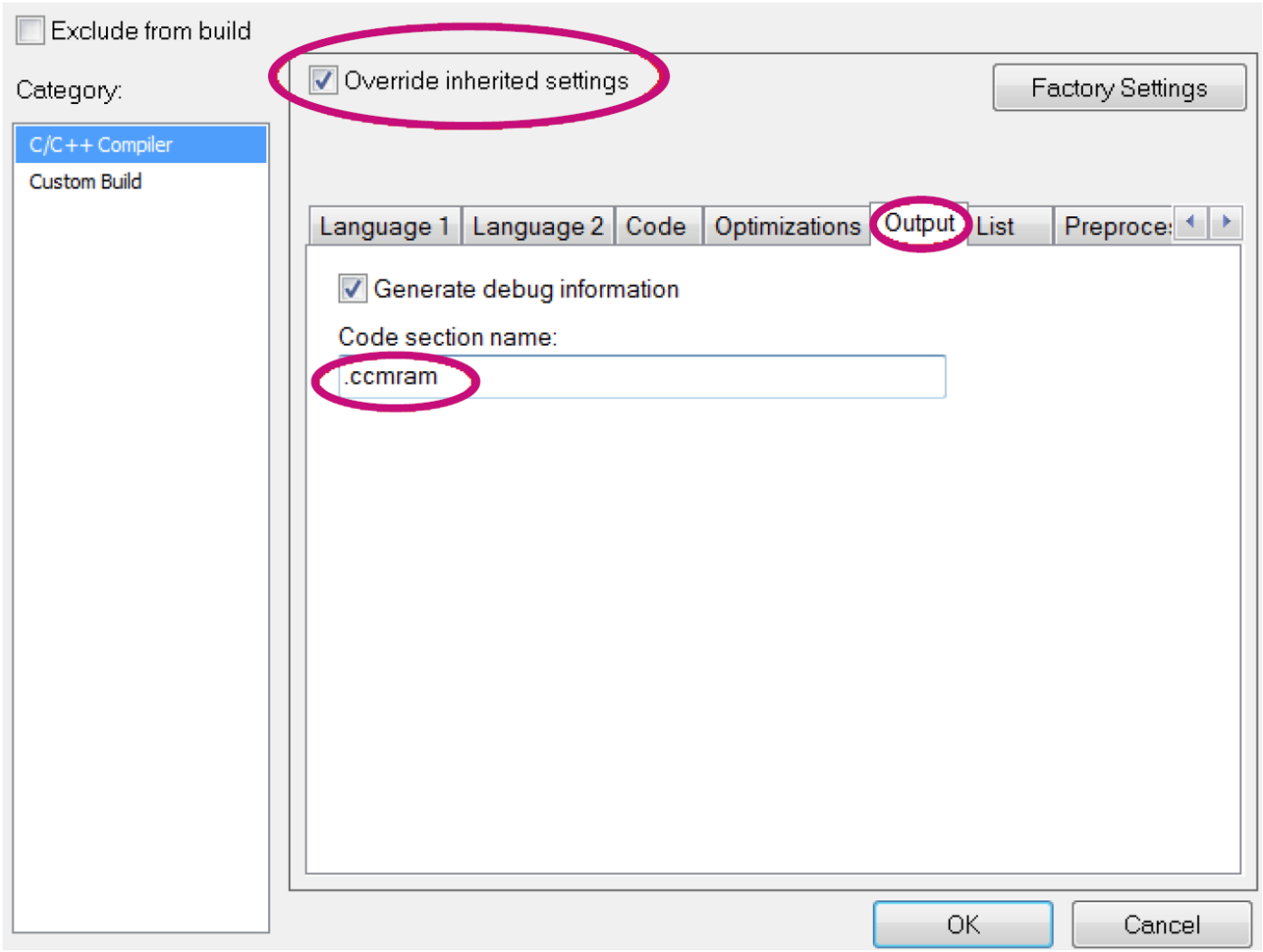
2.1.1 Выполнение исходного файла из CCM SRAM

Выполнение исходного файла из CCM SRAM означает, что все функции, объявленные в этом файле, выполняются из этой области памяти.

Чтобы разместить и выполнить исходный файл из CCM SRAM, используйте окно параметров файла EWARM следующим образом:

1. Добавьте раздел .ccmram (например) в файл компоновщика, как определено в Разделе 2.1.
2. Щелкните правой кнопкой мыши имя файла в окне рабочей области.
3. В открывшемся меню выберите «Параметры».
4. В открывшемся окне установите флажок «Переопределить унаследованные настройки».
5. Выберите вкладку «Вывод» и введите имя раздела, уже определенного в файле компоновщика (.ccmram в данном примере), в поле «Имя раздела кода» (см. Рисунок ниже).

Figure 4. EWARM file placement



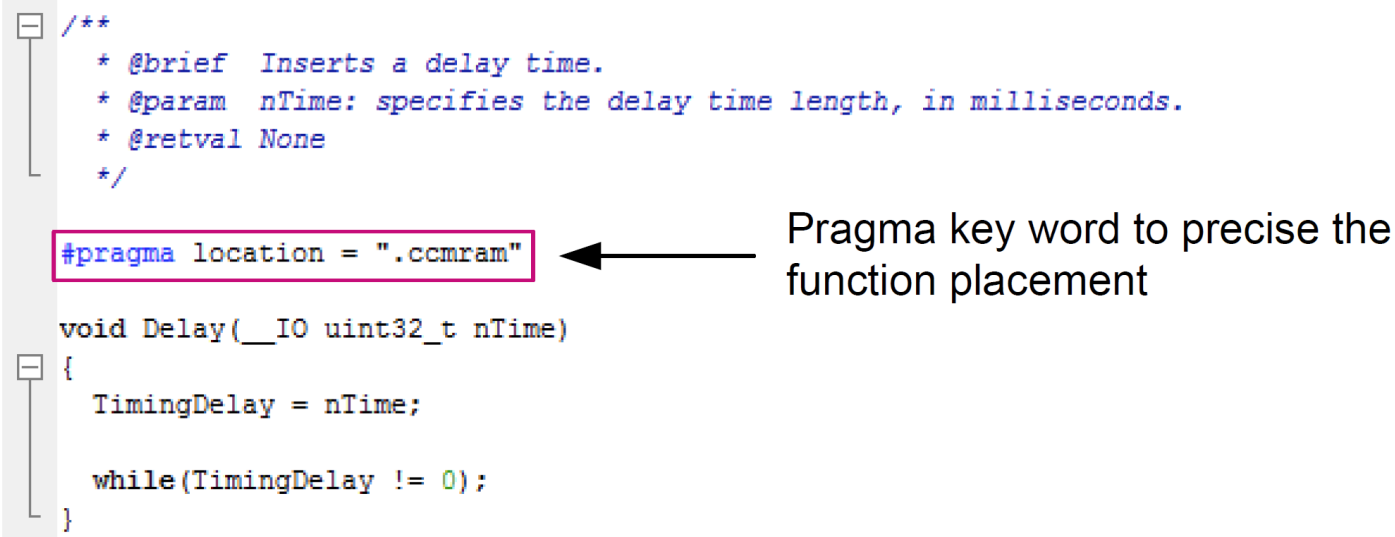
2.1.2 Выполнение одной или нескольких функций из CCM SRAM

Для выполнения функции из CCM SRAM необходимы следующие шаги:

1. Добавьте раздел .ccmram в файл компоновщика, как описано в Разделе 2.1.

2. Используя ключевое слово «расположение прагмы», укажите функцию, которая будет выполняться из CCM SRAM (см. Рисунок ниже).

Figure 5. EWARM function placement



Примечание. Чтобы выполнить более одной функции из CCM SRAM, ключевое слово pragma location должно быть размещено над каждым объявлением функции.

2.2 Выполнение обработчика прерывания из CCM SRAM

Векторная таблица реализована как массив с именем __vector_table и упоминается в коде запуска.

Компоновщик EWARM защищает разделы, на которые есть ссылки из кода запуска, от воздействия директивы «инициализировать копированием». Символ __vector_table не должен использоваться для копирования разделов обработчика прерываний с помощью директивы «инициализировать копированием». Вторая таблица векторов должна быть создана и помещена в CCM SRAM.

Шаги, необходимые для выполнения обработчика прерывания из CCM SRAM, перечислены ниже:

1. Обновите файл компоновщика (.icf).
2. Обновите файл запуска.
3. Поместите обработчик прерывания в CCM SRAM.
4. Переназначьте векторную таблицу в CCM SRAM.

2.2.1 Обновление файла компоновщика (.icf)

Для обновления файла компоновщика .icf необходимо выполнить следующие действия (см. Рисунок 6 на сл. стр.):

1. Определите адрес, по которому находится вторая таблица векторов: 0x1000 0000.
2. Определите адресную область памяти для CCM SRAM, указав начальный и конечный адреса.
3. Попросите компоновщик скопировать при запуске раздел с именем .ccmram и второй раздел таблицы векторов .intvec_CCMRAM из флэш-памяти в CCM SRAM.
4. Сообщите компоновщику, что вторую таблицу векторов необходимо поместить в секцию .intvec_CCMRAM.
5. Укажите, что раздел кода .ccmram необходимо поместить в CCM SRAM.

Figure 6. EWARM linker update for interrupt handler

```
/*###ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\cortex_v1_0.xml" */
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x08000000;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__    = 0x08000000;
define symbol __ICFEDIT_region_ROM_end__      = 0x0803FFFF;
define symbol __ICFEDIT_region_RAM_start__    = 0x20000000;
define symbol __ICFEDIT_region_RAM_end__      = 0x20009FFF;
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__ = 0x1000;
define symbol __ICFEDIT_size_heap__   = 0x0000;
/**** End of ICF editor section. ###ICF###*/

1 define symbol CCMRAM_intvec_start = 0x10000000;

define memory mem with size = 4G;
define region ROM_region      = mem:[from __ICFEDIT_region_ROM_start__   to __ICFEDIT_region_ROM_end__];
define region RAM_region      = mem:[from   ICFEDIT_region_RAM_start     to __ICFEDIT_region_RAM_end__];

2 define region CCMRAM_region   = mem:[from 0x10000000   to 0x10001FFF];

define block CSTACK    with alignment = 8, size = __ICFEDIT_size_cstack__   { };
define block HEAP      with alignment = 8, size =   ICFEDIT_size_heap       { };

3 initialize by copy { readwrite, section .intvec_CCMRAM, section .ccmram, ro object stm32f30_it.o };

do not initialize  { section .noinit };

place at address mem:  ICFEDIT_intvec_start   { readonly section .intvec };

4 place at address mem: CCMRAM_intvec_start { section .intvec_CCMRAM };

5 place in CCMRAM_region   { section .ccmram };

place in ROM_region   { readonly };
place in RAM_region   { readwrite,
                        block CSTACK, block HEAP };
```

2.2.2 Обновление файла запуска

Для обновления файла запуска необходимы следующие шаги: (см. рис. 7)

1. Создайте вторую таблицу векторов для сохранения в CCM SRAM. Файл startup_stm32f30x.s необходимо изменить, удалив все записи, кроме sfe (CSTACK) и Reset_Handler, из исходной таблицы векторов __vector_table.

2. Добавьте вторую таблицу векторов для размещения в CCM SRAM. Он должен содержать все записи. В качестве примера эту таблицу можно назвать __vector_table_CCMRAM. Эта векторная таблица должна быть помещена в раздел .intvec_CCMRAM, определенный в файле компоновщика.

2.2.3 Поместите обработчик прерывания в CCM SRAM

Поместите обработчик прерывания для выполнения в CCM SRAM, как описано в Разделе 2.1.2, или весь файл stm32f_it.c, как описано в Разделе 2.1.1.

Figure 7. **EWARM startup file update for interrupt handler**

```
;; Forward declaration of sections.
SECTION CSTACK:DATA:NOROOT(3)

SECTION .intvec:CODE:NOROOT(2)

EXTERN  __iar_program_start
EXTERN  SystemInit
PUBLIC  __vector_table

DATA
```

1

```
__vector_table
        DCD     sfe(CSTACK)
        DCD     Reset_Handler              ; Reset Handler
```

2

```
        SECTION .intvec_CCMRAM:CODE:ROOT(2)

        PUBLIC  __vector_table_CCMRAM

        DATA
__vector_table_CCMRAM
       DCD      sfe(CSTACK)
        DCD      Reset_Handler              ; Reset Handler
        DCD      NMI_Handler                ; NMI Handler
        DCD      HardFault_Handler          ; Hard Fault Handler
        DCD      MemManage_Handler          ; MPU Fault Handler
        DCD      BusFault_Handler           ; Bus Fault Handler
        DCD      UsageFault_Handler         ; Usage Fault Handler
        DCD      0                          ; Reserved
        DCD      0                          ; Reserved
        DCD      0                          ; Reserved
```

2.2.4 Переназначить векторную таблицу в CCM SRAM

В функции SystemInit переназначьте векторную таблицу в CCM SRAM, изменив регистр VTOR следующим образом:

```
SCB->VTOR = 0x10000000 | VECT_TAB_OFFSET;
```

2.3 Запуск библиотеки (.a) из CCM SRAM

EWARM позволяет запускать библиотеку или библиотечный модуль из CCM SRAM. Необходимые шаги перечислены ниже:

1. Определите адресную область памяти, соответствующую CCM SRAM, указав начальный и конечный адреса.

Figure 8. **CCM SRAM area definition**

```
define memory mem with size = 4G;
define region ROM_region   = mem:[from __ICFEDIT_region_ROM_start__   to __ICFEDIT_region_ROM_end__];
define region RAM_region   = mem:[from __ICFEDIT_region_RAM_start__   to __ICFEDIT_region_RAM_end__];

define region CCMRAM_region   = mem:[from 0x10000000   to 0x10001FFF];

define block CSTACK    with alignment = 8, size = __ICFEDIT_size_cstack__   { };
define block HEAP      with alignment = 8, size = __ICFEDIT_size_heap__     { };
```

Defines the address zone for CCM SRAM.

2. Обновите компоновщик, чтобы при запуске копировать библиотеку или библиотечный модуль в CCM SRAM, используя директиву «инициализировать копированием».

Figure 9. EWARM section initialization

```
initialize by copy { readwrite,ro object iar_cortexM4lf_math.a };

do not initialize  { section .noinit };
```

3. Сообщите компоновщику, что библиотека должна быть помещена в CCM SRAM.

Figure 10. EWARM library placement

```
place in ROM_region    { readonly };

place in CCMRAM_region    {section .text object iar_cortexM4lf_math.a};
```

Чтобы запустить библиотечный модуль из CCM SRAM, выполните шаги 1, 2 и 3, используя имя библиотечного модуля.

Пример на рисунке ниже показывает, как разместить arm_abs_f32.o (модуль библиотеки iar_cortexM4l_math.a) в CCM SRAM.

Figure 11. EWARM library module placement

```
/*###ICF### Section handled by ICF editor, don't touch! ****/
/*-Editor annotation file-*/
/* IcfEditorFile="$TOOLKIT_DIR$\config\ide\IcfEditor\cortex_v1_0.xml" */
/*-Specials-*/
define symbol __ICFEDIT_intvec_start__ = 0x08000000;
/*-Memory Regions-*/
define symbol __ICFEDIT_region_ROM_start__ = 0x08000000;
define symbol __ICFEDIT_region_ROM_end__   = 0x0803FFFF;
define symbol __ICFEDIT_region_RAM_start__ = 0x20000000;
define symbol __ICFEDIT_region_RAM_end__   = 0x20009FFF;
/*-Sizes-*/
define symbol __ICFEDIT_size_cstack__ = 0x400;
define symbol __ICFEDIT_size_heap__   = 0x200;
/**** End of ICF editor section. ###ICF###*/
define memory mem with size = 4G;
define region ROM_region   = mem:[from __ICFEDIT_region_ROM_start__   to __ICFEDIT_region_ROM_end__];
define region RAM_region   = mem:[from __ICFEDIT_region_RAM_start__   to __ICFEDIT_region_RAM_end__];

1  define region CCMRAM_region   = mem:[from 0x10000000   to 0x10001FFF];

define block CSTACK    with alignment = 8, size = __ICFEDIT_size_cstack__   { };
define block HEAP      with alignment = 8, size =   ICFEDIT_size_heap       { };

2  initialize by copy { readwrite, ro object arm_abs_f32.o };

do not initialize  { section .noinit };

place at address mem:__ICFEDIT_intvec_start__ { readonly section .intvec  };
place in ROM_region   { readonly };

3  place in CCMRAM_region   {section .text object arm_abs_f32.o };

place in RAM_region   { readwrite,
                        block CSTACK, block HEAP };
```

3 Выполнение кода приложения из CCM SRAM с помощью набора инструментов MDK - ARM

Возможности MDK-ARM позволяют выполнять простые функции или обработчики прерываний из CCM SRAM. В следующих разделах объясняется, как использовать эти функции для выполнения кода из CCM SRAM.

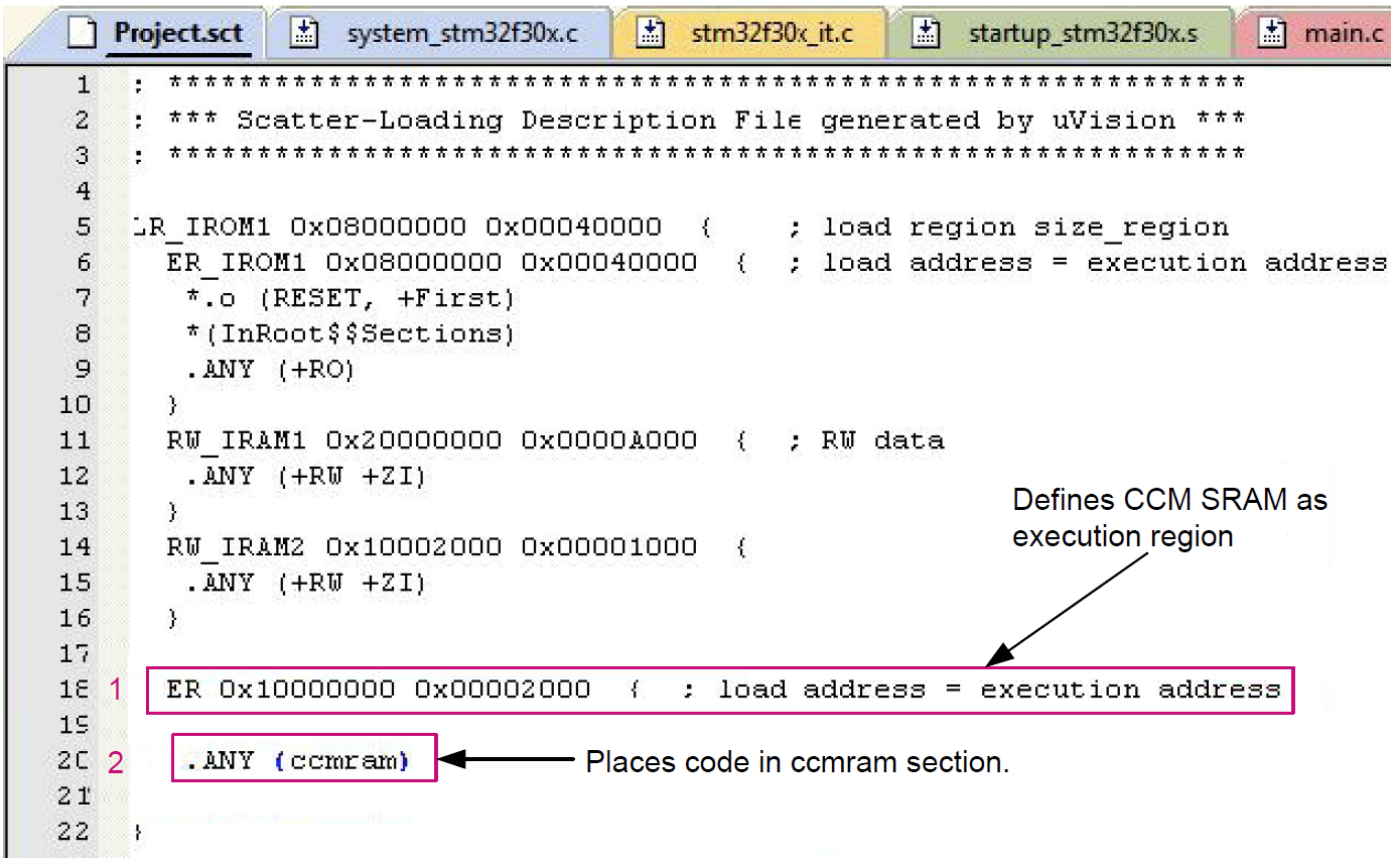
3.1 Выполнение функции или обработчика прерывания из CCM SRAM

Шаги, необходимые для выполнения функции или обработчика прерывания из CCM SRAM, перечислены ниже:

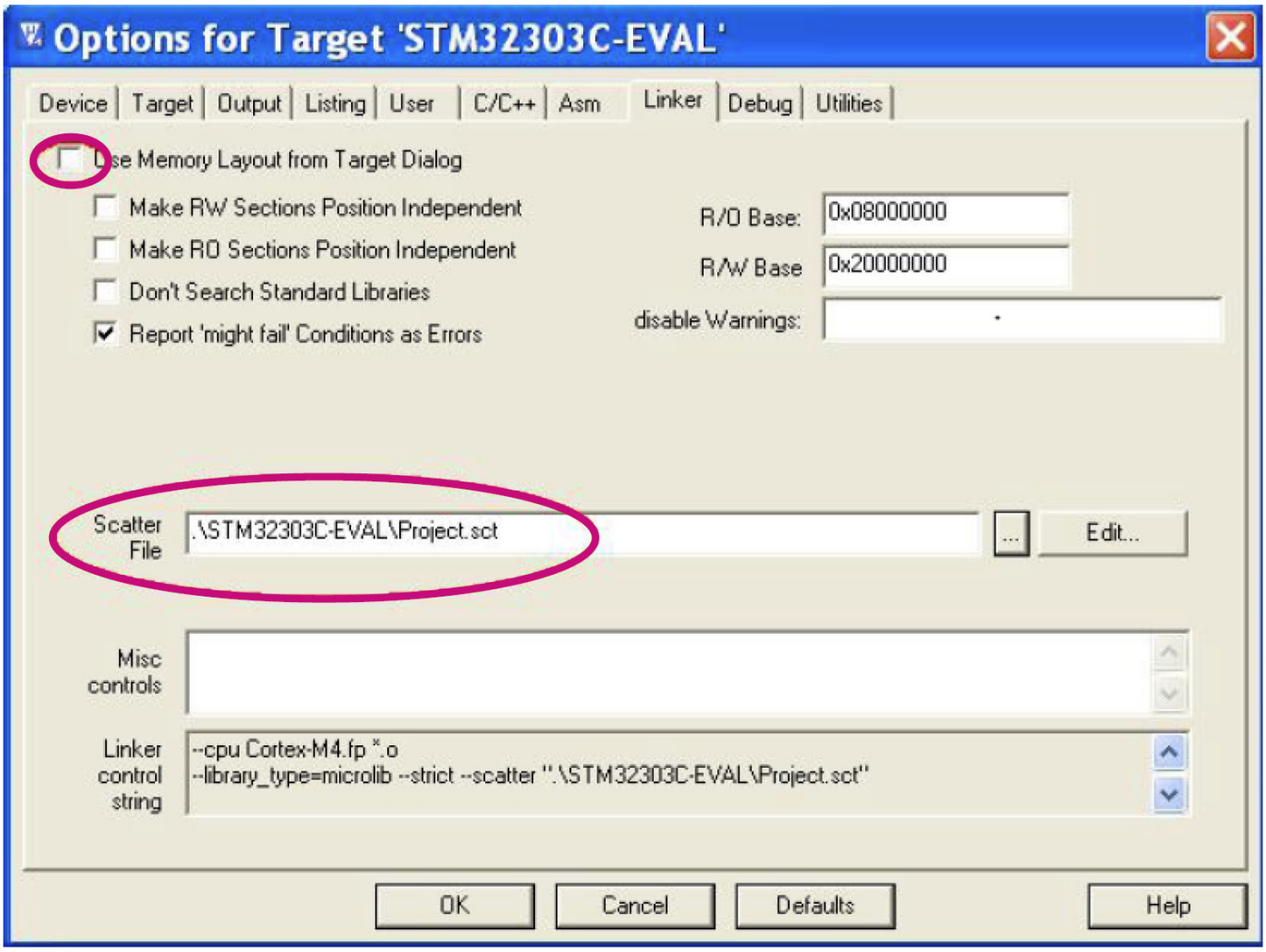
1. Определите новую область (ccmram) в файле разброса, указав начальный и конечный адреса CCM SRAM.

2. Укажите компоновщику, что разделы с атрибутом ccmram должны быть помещены в область CCM SRAM.

Figure 12. MDK-ARM scatter file



3. Для получения информации о параметрах проекта см. MDK-ARM Options menu



4. Поместите часть кода, который будет выполняться из CCM SRAM, в раздел ccmram, определенный выше. Это делается путем добавления ключевого слова атрибута над объявлением функции.

Figure 14. MDK-ARM function placement

```
/**
  * @brief  This function handles SysTick Handler.
  * @param  None
  * @retval None
  */
__attribute__((section ("ccmram"))
void SysTick_Handler(void)
{
  TimingDelay_Decrement();
}
```

Примечание. Чтобы выполнить более одной функции из CCM SRAM, ключевое слово атрибута должно быть размещено над каждым объявлением функции.

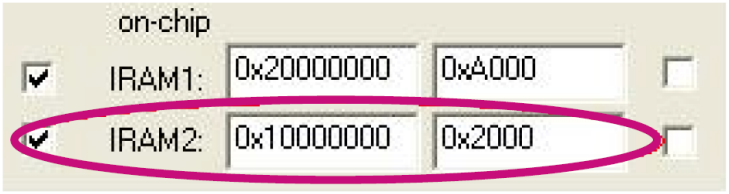
3.2 Выполнение исходного файла из CCM SRAM

Выполнение исходного файла из CCM SRAM означает, что все функции, объявленные в этом файле, выполняются из области CCM SRAM.

Выполните следующие действия, чтобы запустить файл из CCM SRAM:

1. Определите CCM SRAM как область памяти в окне параметров проекта (Project>option>target). (Проект> параметр> цель).

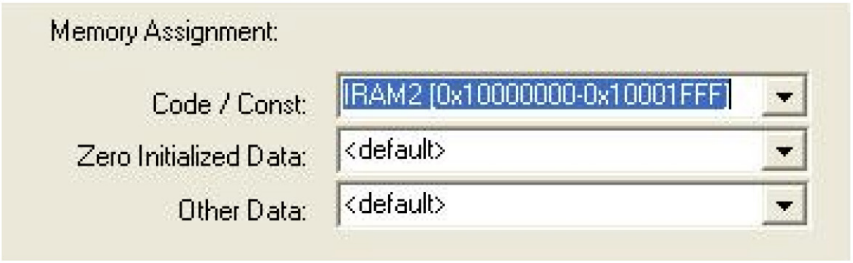
Figure 15. MDK-ARM target memory



2. Щелкните правой кнопкой мыши файл, чтобы поместить его в CCM SRAM, и выберите «Параметры».

3. Выберите область CCM SRAM в меню назначения памяти.

Figure 16. MDK-ARM file placement



3.3 Запуск библиотеки или библиотечного модуля из CCM SRAM

Выполните следующие действия, чтобы запустить библиотеку или библиотечный модуль из CCM SRAM:

1. Определите CCM SRAM как область памяти, как показано на рисунке ниже.

2. Щелкните правой кнопкой мыши библиотеку в рабочей области и выберите «Параметры».

3. Поместите полную библиотеку или модуль из библиотеки в CCM SRAM.

Figure 17. MDK-ARM library placement

4 Выполнение кода приложения из CCM SRAM с использованием инструментальной цепочки на основе GNU

Наборы инструментов на основе GNU позволяют выполнять простые функции или обработчики прерываний из CCM SRAM. В следующих разделах объясняется, как использовать эти функции для выполнения кода из CCM SRAM.

4.1 Выполнение функции или обработчика прерывания из CCM SRAM

Шаги, необходимые для выполнения функции или обработчика прерывания из CCM SRAM, перечислены ниже:

1. Определите новую область (ccmram) в файле компоновщика (.ld), указав начальный адрес и размер области CCM SRAM.

Figure 18. GNU linker update

```
/* Entry Point */
ENTRY(Reset_Handler)

/* Highest address of the user mode stack */
_estack = 0x2000a000;    /* end of 40K RAM on AHB bus*/

/* Generate a link error if heap and stack don't fit into RAM */
_Min_Heap_Size = 0;      /* required amount of heap  */
_Min_Stack_Size = 0x400; /* required amount of stack */

/* Specify the memory areas */
MEMORY
{
  FLASH (rx)      : ORIGIN = 0x08000000, LENGTH = 256K
  RAM (xrw)       : ORIGIN = 0x20000000, LENGTH = 40K
  MEMORY_B1 (rx)  : ORIGIN = 0x60000000, LENGTH = 0K

  CCMRAM (xrw)    : ORIGIN = 0x10000000. LENGTH = 8K
}
```

Defines the address zone for CCM SRAM.

2. Сообщите компоновщику, что разделы кода с атрибутом ccmram должны быть помещены в CCM SRAM.

Figure 19. GNU linker section definition

```
  *(.data)           /* .data sections */
  *(.data*)          /* .data* sections */

  . = ALIGN(4);
  _edata = .;        /* define a global symbol at data end */
} >RAM AT> FLASH

_siccmram = LOADADDR(.ccmram);

/* CCM-RAM section
*
* IMPORTANT NOTE!
* If initialized variables will be placed in this section,
* the startup code needs to be modified to copy the init-values.
*/
.ccmram :
{
  . = ALIGN(4);
  _sccmram = .;       /* create a global symbol at ccmram start */
  *(.ccmram)
  *(.ccmram*)

  . = ALIGN(4);
  _eccmram = .;       /* create a global symbol at ccmram end */
} >CCMRAM AT> FLASH

/* Uninitialized data section */
. = ALIGN(4);
.bss :
{
```

3. Измените файл запуска, чтобы инициализировать данные для размещения в CCM SRAM во время запуска (см. Обновленные строки кода, выделенные жирным шрифтом):

```
.section  .text.Reset_Handler
.weak  Reset_Handler
.type  Reset_Handler, %function
Reset_Handler:
/* Copy the data segment initializers from flash to SRAM and CCMRAM */
  movs  r1, #0
b  LoopCopyDataInit
CopyDataInit:
ldr  r3, =_sidata
ldr  r3, [r3, r1]
str  r3, [r0, r1]
adds  r1, r1, #4
LoopCopyDataInit:
ldr  r0, =_sdata
ldr  r3, =_edata
adds  r2, r0, r1
cmp  r2, r3
bcc  CopyDataInit
movs  r1, #0
b  LoopCopyDataInit1
CopyDataInit1:
ldr  r3, =_siccmram
ldr  r3, [r3, r1]
str  r3, [r0, r1]
adds  r1, r1, #4
LoopCopyDataInit1:
ldr  r0, =_sccmram
ldr  r3, =_eccmram
adds  r2, r0, r1
cmp  r2, r3
bcc  CopyDataInit1
ldr  r2, =_sbss
b  LoopFillZerobss
/* Zero fill the bss segment. */
FillZerobss:
movs  r3, #0
str  r3, [r2], #4
LoopFillZerobss:
ldr  r3, = _ebss
cmp  r2, r3
bcc  FillZerobss
/* Call the clock system intitialization function.*/
bl  SystemInit
/* Call the application's entry point.*/
bl  main
bx  lr
```

4. Поместите часть кода, которая будет выполняться из CCM SRAM, в раздел .ccmram, добавив ключевое слово атрибута в прототип функции.

Figure 20. GNU function placement

```
void NMI_Handler(void);
void HardFault_Handler(void);
void MemManage_Handler(void);
void BusFault_Handler(void);
void UsageFault_Handler(void);
void SVC_Handler(void);
void DebugMon_Handler(void);
void PendSV_Handler(void);

void SysTick_Handler(void) __attribute__((section (".ccmram")));
```

4.2 Выполнение файла из CCM SRAM

Выполнение исходного файла из CCM SRAM означает, что все функции, объявленные в этом файле, выполняются из CCM SRAM.

Чтобы запустить файл из CCM SRAM, выполните следующие действия:

1. Добавьте раздел .ccmram в файл компоновщика, как определено в Разделе 4.1.
2. Поместите файл в CCM SRAM, как показано ниже.

Figure 21. GNU file placement

```
_siccmram = LOADADDR(.ccmram);

  /* CCM-RAM section
  *
  * IMPORTANT NOTE!
  * If initialized variables will be placed in this section,
  * the startup code needs to be modified to copy the init-values.
  */
  .ccmram :
  {
    . = ALIGN(4);
    _sccmram = .;       /* create a global symbol at ccmram start */
    *(.ccmram)
    *(.ccmram*)

      stm32f30x_it.o(*)

    . = ALIGN(4);
    _eccmram = .;       /* create a global symbol at ccmram end */


} >CCMRAM AT> FLASH
```

4.3 Запуск библиотеки из CCM SRAM

Выполните следующие действия, чтобы запустить библиотеку из CCM SRAM:

1. Добавьте раздел .ccmram в файл компоновщика, как определено в Разделе 4.1.
2. Поместите библиотеку в CCM SRAM, как показано ниже.

Figure 22. GNU library placement

```
/* CCM-RAM section
*
* IMPORTANT NOTE!
* If initialized variables will be placed in this section,
* the startup code needs to be modified to copy the init-values.
*/
.ccmram :
{
  . = ALIGN(4);
  _sccmram = .;       /* create a global symbol at ccmram start */
  *(.ccmram)
  *(.ccmram*)

    mylib.a(*)

  . = ALIGN(4);
  _eccmram = .;       /* create a global symbol at ccmram end */


} >CCMRAM AT> FLASH
```

Revision history

Table 3. **Document revision history**

Date	Version	Changes
23-Jul-2013	1	Initial release.
25-Mar-2014	2	Changed STM32F313xC into STM32F358xC. Reworked Section 1: Overview of STM32F303xB/C and STM32F358xC CCM RAM.
2-Sep-2014	3	Added STM32F303x6/x8, STM32F328x8, STM32F334x4/x6/x8 in Table 1: Applicable products. Updated step 2 in Section 2.1: Executing a simple code from CCM RAM (except for interrupt handler), step 3 in Section 2.2.1: Updating the linker file (.icf) and updated Figure 5: EWARM linker update for interrupt handler. Updated Figure 11: MDK-ARM scatter file.
16-Apr-2019	4	Updated: • Title of the document • Introduction • CCM RAM replaced by CCM SRAM in the whole document • Figure 1. STM32F3 devices system architecture Added: • Figure 2. STM32G4 devices system architecture • Table 2. CCM SRAM main features • Section 1.2.5 CCM SRAM read protection (only on STM32G4 devices) • Section 1.2.6 CCM SRAM erase (only on STM32G4 devices) Removed Table 2. CCM RAM organization.