Министерство образования и науки
Российской Федерации

Федеральное государственное бюджетное образовательное
учреждение высшего образования
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
СИСТЕМ УПРАВЛЕНИЯ И РАДИОЭЛЕКТРОНИКИ (ТУСУР)

Кафедра конструирования узлов и деталей радиоэлектронной аппаратуры
(КУДР)

А.А. Бомбизов, А.Г. Лощилов

АССЕМБЛЕР ДЛЯ ARM. СОЗДАНИЕ БАЗОВОГО ПРОЕКТА

Методические указания к выполнению
практических занятий и самостоятельной работы
по дисциплине «Микропроцессорные устройства»

Томск 2017

1 Введение

Основная сущность языка ассемблер – это команда. Программа (упрощенно) – это последовательность команд, работающих одна за другой. Программирование на ассемблере представляет собой доступ к памяти и модификацию значений по определенным адресам, при помощи команд. Язык позволяет группировать последовательности команд в подпрограммы с выполнением условных и безусловных переходов. Но для начала нужно запустить микроконтроллер.

Целью работы является освоение организации запуска микроконтроллера и реализации базового проекта на ассемблере для микроконтроллеров с ядром cortex-m4.

2 Краткая теория

Как же загружается процессор? Типичная ситуация, когда он просто начинает выполнять команды с адреса 0x00000000. В нашем случае процессор несколько более умный, и рассчитывает на специально определенный формат данных в начале памяти, а именно – таблицу векторов прерываний (полная таблица размещена в [1, Table 62. Vector table for STM32F42xxx and STM32F43xxx]):

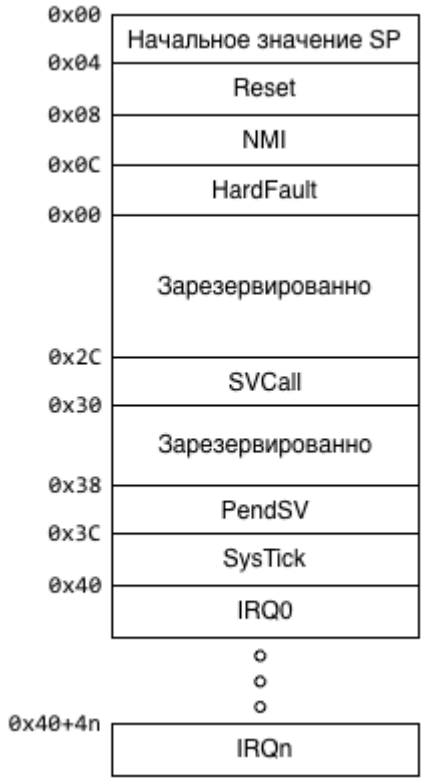

Рисунок 1 – Таблица векторов прерываний

Старт выполнения программы происходит следующим образом: процессор считывает значение по адресу 0x00000000 и записывает его в *SP* (англ. Stack Pointer – регистр, который указывает на вершину стека), после

чего читает значение по адресу 0x00000004 и записывает его в *PC* (Program counter – регистр, который указывает на текущую инструкцию + 4 байта). В программе на ассемблере определение вершины стека и таблицы векторов выполняется следующим образом:

.word адрес вершины стека

.word адрес метки начала программы (обычно Start) + 1

…

и так далее для всех прерываний

Ключевое слово *word* определяет размер ячейки для последующих данных размером четыре байта.

В программе метка записывается в начале строки с двоеточием в конце. Перейдя по метке, процессор начинает последовательно исполнять указанные инструкции.

Таким образом, начинает выполняться программный код пользователя, при этом уже есть стек, указывающий на определенную область памяти. Карта памяти процессора с ядром cortex-m4 выглядит следующим образом:

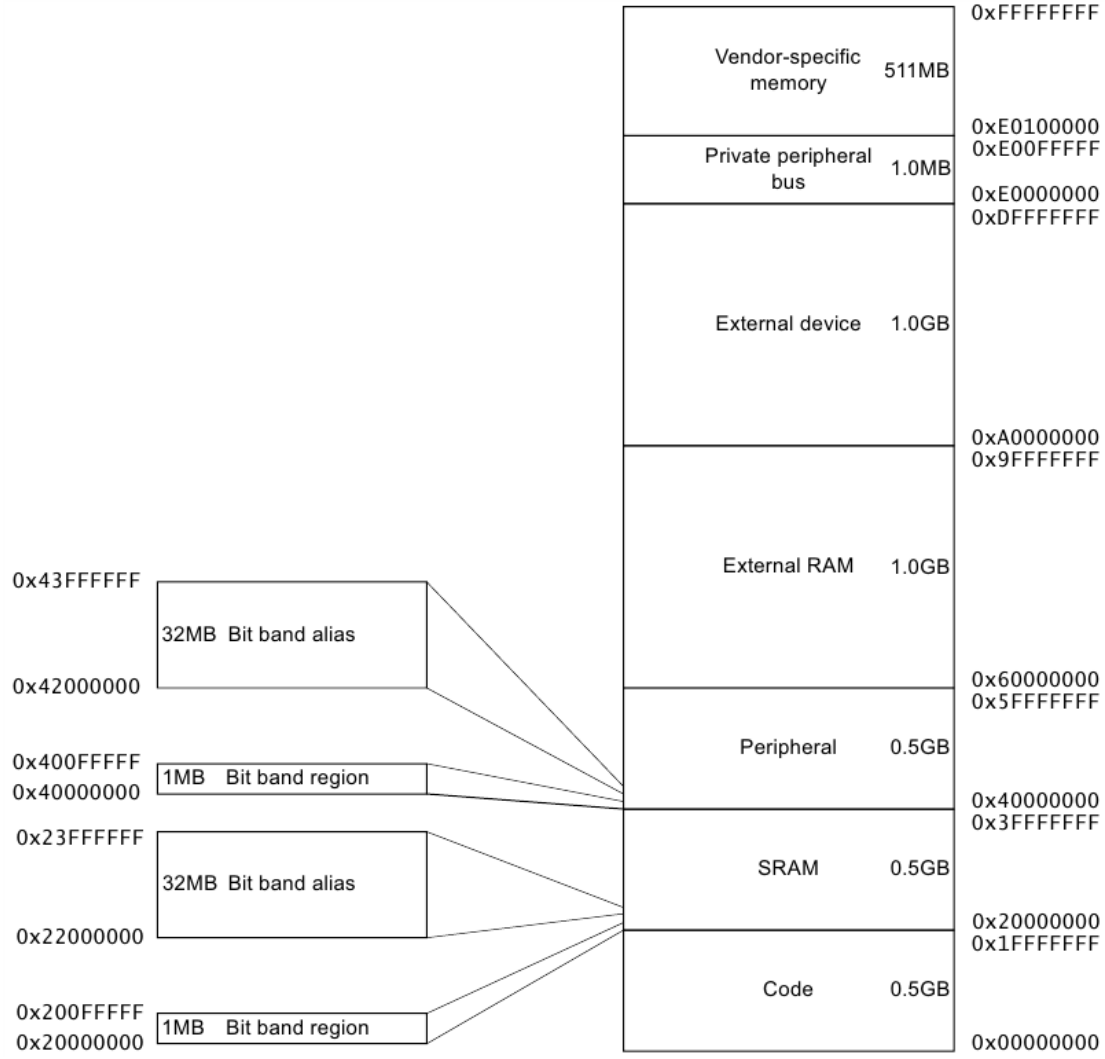

Рисунок 2 – Карта памяти процессора с ядром cortex-m4 [2]

На данном этапе почти вся необходимая информация для написания кода запуска процессора с переходом на управляющую инструкцию предоставлена. Далее из исходного кода создается файл прошивки. Для этого выполняются следующие действия:

– компиляция исходного кода;

– компоновка результата компиляции;

– преобразование скомпонованный файл в файл прошивки нужного формата.

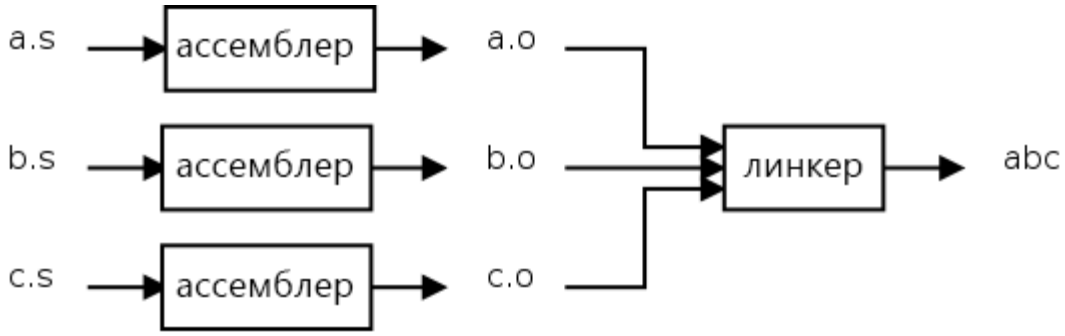

Рисунок 3 – Схема генерации исполняемого файла

Компиляция законченного ассемблерного кода выполняется при помощи утилиты arm-none-eabi-as [3]. Во время трансляции программы, состоящей из нескольких файлов исходного текста (см. рисунок 3), каждый такой файл преобразовывается в объектный. Компоновщик объединяет эти объектные файлы в конечный исполняемый файл.

Во время компоновки линкер (компоновщик) выполняет следующие операции:

а) разрешение символов;

б) перемещение.

В ходе преобразования исходного файла в объектный код транслятор заменяет все ссылки на соответствующие адреса. В многофайловой программе, если в модуле есть какие-либо ссылки на внешние метки, определенные в другом файле, ассемблер помечает их как «нерешённые». Когда эти объектные файлы передаются компоновщику, он определяет значения адресов таких ссылок из других объектных файлов и исправляет код на правильные значения.

Перемещение – процесс изменения адреса, уже заданного ссылке ранее, а также исправления всех ссылок для отражения вновь назначенных адресов [4, 5]. В первую очередь, перемещение осуществляется по следующим двум причинам:

а) слияние секций;

б) размещение секций в исполняемом файле.

Для понимания процесса перемещения важно понимать, что такое секции. В момент выполнения программы код и данные могут обрабатываться по-разному: если, код можно разместить в ПЗУ (постоянное запоминающее устройство ,ROM, read-only memory), то для данных может потребоваться как чтение из памяти, так и запись. Удобнее всего, если код и данные не чередуются, и именно поэтому программы разделены на секции, которые компоновщик должен привести в порядок.

Указания для компоновщика размещаются в специальном скрипт-файле, имеющем стандартизированную архитектуру. Всего для программирования ARM-процессоров с ядром cortex-mX нужно заполнить два раздела скрипт-файла. Первый называется MEMORY. В общем виде без привязки к конкретному процессору может выглядеть следующим образом:

```
MEMORY
{
  FLASH (RX)   : ORIGIN = 0x00000000, LENGTH = 0x8000
  SRAM (RW) : ORIGIN = 0x10000000, LENGTH = 0x2000
}
```

Пробел слева и справа от «:» ставить обязательно.

Конфигурация компоновщика по умолчанию позволяет ему использовать всю доступную память (где-то около 0xFFFFFFFF байт в случае 32-битного ARM). Во-первых, необходимо определить регионы памяти, которые можно использовать: FLASH и SRAM. Буквы в скобках определяют атрибуты: доступ на чтение (R), запись (W), исполнение (X).

Всего в STM32F42xxx четыре региона памяти:

а) FLASH – для хранения кода программы;

б) SRAM – для хранения переменных;

в) CCM – быстрая память ядра**;**

г) BKPSRAM – для хранения данных при отключении питания (только при наличии батареи питания часов реального времени).

Два параметра, *ORIGIN=Значение* и *LENGTH=Значение*, задают начало и длину региона, соответственно. *Значение* – это выражение, т.е., в нем можно выполнять арифметические операции или использовать суффиксы K, M, и т.п. (пример: *LENGTH = 100+50*).

Вторая часть файла – конфигурация секций. В целом, это означает копирование заданных исходных секций в выходные секции.

```
SECTIONS
{
        .text :
        {
```

```
        *(.vectors);    /* Указатели векторов прерываний */
        *(.text)     /* Текст программы */
    } > FLASH
    .data :
    {
        *(.data)
    }>SRAM AT>FLASH     /* Перемещаемые в RAM  данные */
    .bss :
    {
        . = ALIGN(1);
        *(.bss);        /* Переменные в SRAM */
    } > SRAM
}
```

Исходные секции задаются в форме ИМЯ_ФАЙЛА(ИМЯ_СЕКЦИИ), символ * ведет себя стандартным образом, так что запись *(.text) означает: секции .text из всех файлов. У секции есть два адреса: LMA (Load Memory Address) – откуда она загружается, и VMA (Virtual Memory Address) – по какому адресу она доступна в виртуальной памяти. Объясняя проще, LMA – это где она окажется в бинарном файле, а VMA – это куда будут перенаправлены символы, т.е., указатель на символ в коде будет ссылаться на VMA-адрес.

Наиболее важные секции – это код, данные (с заданными начальными значениями) и данные, которые нулевые по умолчанию. Таким образом, мы копируем код (.text) во flash-память, данные (.data) – во flash-память, но из расчета, что они будут доступны в оперативной памяти, и переменные (.bss) – в оперативную память. В представленном выше коде видно, что перенаправление секций в соответствующую память выполняется при помощи символа «>».

Программа для выполнения компоновки и формирования файла прошивки называется arm-none-eabi-ld [3].

Затем файл прошивки должен быть преобразован в формат, пригодный для программатора, при помощи программы arm-none-eabi-objcopy.

Загрузка подготовленного файла прошивки выполняется при помощи программы STM32 ST-LINK Utility [6], предназначенной для работы с программатором типа ST-LINK. Внешний вид программы изображен на рисунке 4.

Работа с программой осуществляется следующим образом:

1) Запустить программу;

2) Подключить отладочную плату STM32F429I-DISC1;

3) В программе выбрать в меню Target пункт Connect;

4) Загрузить файл прошивки, выбрав меню File пункт Open file… В появившемся диалоге выберете файл прошивки с расширением *hex* [7];

5) Выполнить программирование устройства путём выбора в меню Target пункт Program & Verify…

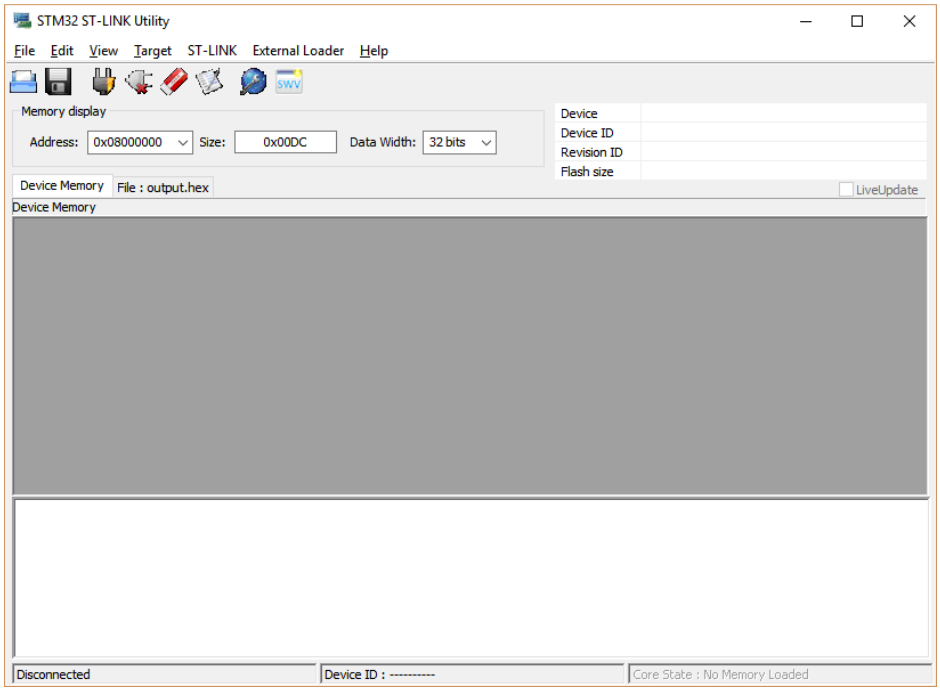

Рисунок 4 – Внешний вид ST-LINK Utility

3 Порядок выполнения работы

В ходе данной работы необходимо создать и откомпилировать базовый проект на языке ассемблер для микроконтроллера STM32F42xx. И выполнить его программирование. Для этого необходимо выполнить следующие действия:

3.1 Изучите предложенный в п. 2 теоретический материал.

3.2 Скопируйте папку template_Start.base в свой рабочий каталог.

3.3 Создайте внутри папки template_Start.base пустой файл main.s.

3.4 Откройте файл в текстовом редакторе. Для удобства лучше использовать Notepad++.

3.5 Определите рабочий набор инструкций, используемых в проекте, как thumb путём добавление в файл main.s директивы *.thumb* (с точкой впереди).

3.6 Определите используемый тип процессорного ядра cortex-m4 путём добавления в файл main.s директивы *.cpu cortex-m4*.

3.7 Определите начало секции векторов прерываний строчкой *.section .vectors*.

3.8 Установите вершину стека, которая определяется суммой начального адреса оперативной памяти (SRAM) и её размером. Начальный

адрес описан в [1, п. 2.3.1]. Размер описан в [1, п. 2.1] (в разделе для используемого в работе процессора).

3.9 Добавьте в программу адрес начала программы.

3.10 Определите секцию *.text*.

3.11 Установите метку начала программы.

3.12 Запустите консоль windows: Пуск->выполнить->cmd.

3.13 Перейдите в каталог template_Start.base путем выполнения команды *cd "путь_к_каталогу"*. Если требуется поменять диск, то необходимо набрать букву диска, двоеточие и нажать на «Enter» («D:»).

3.14 Выполните компиляцию проекта путем вызова команды *bin\arm-none-eabi-as.exe main.s -o main.o* , где инструкция *-o main.o* указывает на имя выходного файла.

Далее выполняются действия по компоновке.

3.15 Создайте в каталоге проекта файл stm32f42x_map.ld.

3.16 Откройте файл в Notepad++ и определите в нём секцию MEMORY. Начальный адрес и размер FLASH-памяти взять в [1, п. 3.3]. Начальный адрес SRAM-памяти описан в [1, п. 2.3.1]. Размер описан в [1, п. 2.1] (в разделе для используемого в работе процессора).

3.17 Определите секцию *SECTIONS* (см. пункт «Краткая теория»).

3.18 Перейдите в консоль и выполните команду *bin\arm-none-eabi-ld.exe -T stm32f42x_map.ld -o main.elf main.o* . В результате выполнения команды, если не было ошибок в файле скрипта, в рабочем каталоге должен появиться исполняемый файл *main.elf*.

3.19 Для преобразования исполняемого файла в пригодный для программирования файл выполните команду *bin\arm-none-eabi-objcopy.exe -O ihex main.elf main.hex* .

3.20 Запрограммируйте микроконтроллер при помощи ST-LINK Utility как описано в разделе теоретическая часть.

3.21 Оформите отчет, содержащий титульный лист, введение, ход выполнения работы, ответы на контрольные вопросы и выводы.

3.22 Защитите отчет у преподавателя.

4 Контрольные вопросы

4.1 Из каких этапов состоит сборка программы?

4.2 Почему нужно компоновать программу?

4.3 Какие виды памяти установлены на контроллере STM32F42xxx?

4.4 Что значит ORIGIN?

4.5 Откуда микроконтроллер узнает вершину стека?

4.6 Как рассчитать вершину стека?

4.7 Сколько оперативной памяти в микроконтроллере STM32F42xxx?

4.8 Назначение flash-памяти?

4.9 Сколько flash-памяти в микроконтроллере STM32F42xxx?

4.10 С какого адреса в виртуальном пространстве используемого микроконтроллера начинается flash-память?

4.11 Каким образом в программе описывается таблица прерываний?

4.12 Какие наиболее важные секции в исполняемом файле?

4.13 Для чего требуется компоновка?

Список литературы

1. RM0090. Reference manual. STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced ARM®-based 32-bit MCUs.– URL: www.st.com/resource/en/reference_manual/DM00031020.pdf (дата обращения: 10.01.2017);

2. Cortex™-M4 Devices: Generic User Guide.– URL: http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf (дата обращения: 10.01.2017);

3. GNU ARM Embedded Toolchain.– URL: https://launchpad.net/gcc-arm-embedded (дата обращения: 10.01.2017);

4. Тонкости компиляции и компоновщик.– URL: https://habrahabr.ru/post/191058 (дата обращения: 10.01.2017);

5. Linker Script Guide.– URL: www.emprog.com/support/documentation/thunderbench-Linker-Script-guide.pdf (дата обращения: 10.01.2017);

6. STM32 ST-LINK utility.– URL: http://www.st.com/en/embedded-software/stsw-link004.html (дата обращения: 10.01.2017);

7. Intel HEX.– URL: https://ru.wikipedia.org/wiki/Intel_HEX (дата обращения: 10.01.2017).