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. ЛИНЕЙНОЕ ПРОГРАММИРОВАНИЕ. ПОРТЫ ВВОДА-ВЫВОДА

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

Томск 2017

1 Введение

В предыдущей работе были разобраны инструменты для компиляции и компоновки (линковки) проекта на языке ассемблера, а так же способа программирования микроконтроллера посредством программатора.

Целью настоящей работы является освоение механизмов управления микропроцессором на примере изменения состояния порта ввода-вывода (General Purpose Input/Output – GPIO) микроконтроллера с ядром cortex-m4.

2 Краткая теория

В данной работе используется микроконтроллер STM32F429, основанный на ядре ARM Cortex-m4. Это 32-битное высокопроизводительное процессорное ядро с трёхстадийным конвейером Гарвардской архитектуры. Трёхстадийный конвейер в упрощенном виде подразумевает следующее: одна команда процессора сейчас выбирается из кода программы, предыдущая уже декодируется, а пред-предыдущая уже исполняется.

Ассемблерный код – это текстовый файл, в котором одна строка текста, как правило, описывает одну команду процессора. Компилятор ассемблера преобразует этот текст в бинарный файл, в котором команды закодированы специальным образом. Процессор будет читать эти команды из памяти и исполнять их по очереди. Иногда он может переходить к выполнению команд по другому адресу, выполняя условный или безусловный переход. Для работы с данными в процессорном ядре есть набор регистров ядра, включающий в себя регистры общего назначения. Условно говоря – это временные переменные программы. Весь состав регистров ядра изображен на рисунке 1. Далее приведена их расшифровка:

R0–R12 – 32-битные регистры общего назначения, предназначенные для операций с данными;

R13 (SP) – указатель текущей позиции в стеке. Альтернативное название в программе SP, что есть сокращение от Stack pointer;

R14 (LR) – регистр возврата из процедуры;

R15 (PC) – программный счетчик. Содержит текущий адрес программы.

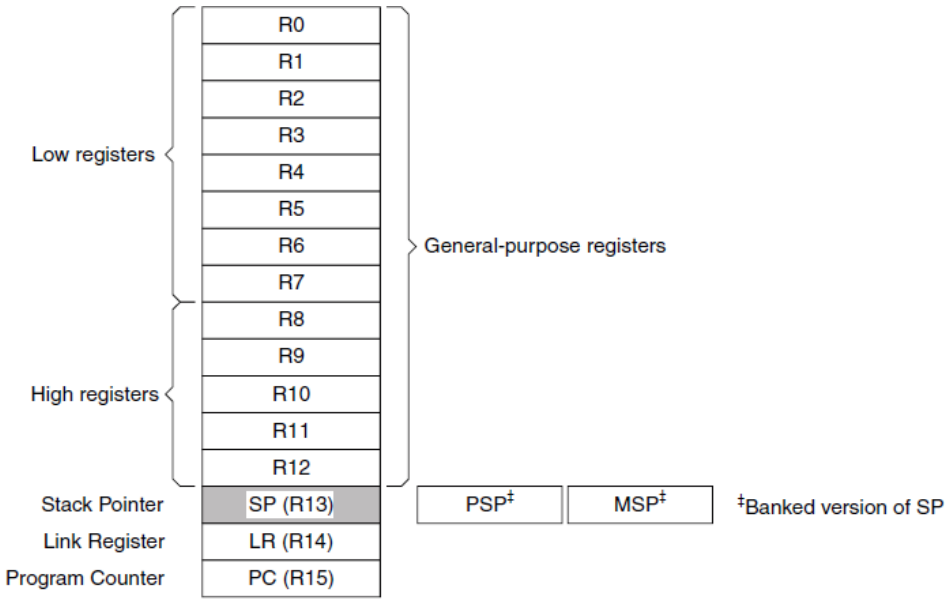

Рисунок 1 – Регистры ядра [1, п. 2.1.3]

Процессор может манипулировать значениями в этих регистрах с помощью арифметических, логических или команд сдвига. Так же он может загружать в регистры значения из памяти или выгружать в память.

На языке ассемблер можно определять константы в виде

.equ CONSTANT_NAME, VALUE

где CONSTANT_NAME – имя константы; VALUE – значение константы.

Загрузка значения в регистр из другого регистра производится при помощи команды [2]:

LDR Rd, Rn ,

где *Rd* – регистр назначения, *Rn* – регистр источник. (*t*, *n* – номера регистров)

Загрузка константы в регистр производится следующим образом

LDR Rd, =5 ,

где 5 – это константа. Префикс «=» ставить обязательно.

Допускается выполнять математические операции над константами:

LDR Rd, =((1<<3)|(1<<2))&~(1<<3) ,

в результате в регистр *Rd* будет загружено значение 4. На языке Си «<<» – сдвиг влево, «|» – побитовое ИЛИ, «&» – побитовое И, «~» – инверсия.

Загрузка ранее определенной константы:

LDR Rd, =CONSTANT_NAME .

Загрузка значения из ячейки памяти по определенному адресу производится следующим образом:

LDR Rd, [Rn] ,

где адрес ячейки памяти, содержащийся в регистре *Rn*, записывается в квадратных скобках.

Операция выгрузки значения в ячейку памяти по указанному адресу:

STR Rd, [Rn] ,

где *Rd* – значение, которое требуется сохранить; *Rn* – адрес ячейки, в которую нужно сохранить.

Выполнение базовых арифметических и логических операции процессором может выполняться при помощи следующих функций [2]:

Сложение регистров:

ADD {Rd,} Rn, Operand2 ,

где *Rd* – регистр, в который помещается результат; *Rn* – регистр, в который происходит прибавление; *Operand2* – что прибавлять. В приведенных командах фигурными скобками обозначены необязательные параметры. Если *Rd* отсутствует, то результат вычисления будет помещен в *Rn*.

Разность регистров выполняется при помощи команды SUB. Параметры аналогичны, как и в команде *ADD*.

Побитовое умножение (операция И):

AND {Rd,} Rn, Operand2 ,

где *Rd* – регистр, в который помещается результат; *Rn* – первый множитель; *Operand2* – второй множитель.

ORR – побитовое сложение (операция ИЛИ).

EOR – исключающее ИЛИ.

BIC – сброс бита по маске. Аналогично операции AND NOT…

LSL – сдвиг влево.

LSR – сдвиг вправо.

Это минимальный, но далеко не полный набор команд для совершения операций над регистрами. Полный набор можно посмотреть в [2].

Вся периферия микроконтроллера на ядре cortex-m4 доступна в виде наборов регистров, начинающихся с определенного адреса (далее базовые адреса – Base address) [1, п. 2.3, Table 1. STM32F4xx register boundary addresses]. Согласно представленной таблице, за каждым блоком микроконтроллера закреплен определенный диапазон адресов в виртуальном адресном пространстве. В общем виде периферийные регистры можно разделить на два типа: для хранения данных и конфигурации. Так как каждый конфигурационный регистр является 32-битной ячейкой памяти, то его можно представить в виде линейки, состоящей из 32 переключателей. Поэтому настройка каждого периферийного блока представляет собой изменение состояния (в единицу или ноль) определенной комбинации переключателей. Другой особенностью является то, что все регистры, относящиеся к определенному блоку и описанные в [1], представлены в виде смещения адреса (Address offset) относительно базового.

Активная периферия «на кристалле» STM32 тактируется. Тактирование может быть включено/выключено отдельно для каждого устройства микроконтроллера – это сделано и для экономии энергии и для устранения конфликтов различных устройств при использовании одних и тех же ресурсов. Поэтому, чтобы активировать какой-либо блок микроконтроллера, нужно обратиться к документации и узнать к какой шине (BUS) относится этот блок. Затем включить тактирование блока на этой шине. Для включения тактирования существует несколько пар регистров [1, п. 7.3.25, Table 34. RCC register map and reset values]:

RCC_<название шины тактирования>RSTR – этот регистр отвечает за отключение тактирования;

RCC_<название шины тактирования>ENR – этот регистр включает тактирование.

Из всего вышесказанного можно описать алгоритм активизации блока микроконтроллера:

2.1 Определить название шины тактирования выбранного устройства [1, п. 2.3, Table 1. STM32F4xx register boundary addresses, столбец Bus].

2.2 Рассчитать адрес для регистра RCC_<название шины тактирования>ENR. Адрес рассчитывается путём сложения базового адреса [1, п. 2.3, Table 1. STM32F4xx register boundary addresses, столбец Boundary address для устройства Reset and clock control (RCC)] и смещения Address offset [1, п. 6.3.5–6.3.19] целевого регистра.

2.3 Включение тактирования путём переключения с использованием логической операции (операция побитового «ИЛИ») в единицу бита, соответствующего выбранному устройству.

2.4 Расчет адресов требуемых регистров для выбранного устройства.

2.5 Работа через регистры с выбранным устройством.

3 Порядок выполнения работы

В ходе данной работы необходимо настроить порт ввода/вывода таким образом, чтобы зажечь размещенный на плате и подключенный к микроконтроллеру светодиод. Для этого потребуется выполнить следующие действия:

3.1 Изучите предложенный в п. 2 теоретический материал.

3.2 Для начала работы скопируйте папку template_GPIO_OUT.base в свой рабочий каталог.

3.3 Откройте в блокноте (или в программе Notepad++) файл main.s, содержащийся в каталоге template_GPIO_OUT.base. Как видно, это базовый проект, созданный на прошлом занятии.

3.4 Определите по [3] или по внешнему виду отладочной платы порт ввода/вывода и номера бит пользовательских светодиодов.

Определение констант порта ввода вывода:

3.5 В файле main.s после определения секции .text опишите константу GPIO_LED, содержащую базовый адрес порта ввода/вывода светодиода.

3.6 Опишите константы с номерами бит для красного (LED_RED) и зеленого (LED_GREEN) светодиодов, соответственно (см. [3]).

3.7 Определите название регистра, отвечающего за переключение режима (на вход, на выход…) порта ввода/вывода [1, п. 8.3.3 I/O port control registers].

3.8 Рассчитайте адрес регистра режима порта ввода/вывода путём соответствующего смещения относительного базового [1, п. 8.4.11 GPIO register map].

3.9 Опишите константу GPIO_LED_DIR, содержащую адрес регистра режима порта ввода/вывода.

3.10 Определите название регистра данных, отвечающего за выход порта ввода/вывода [1, п. 8.3.4 I/O port data registers].

3.11 Опишите константу GPIO_LED_OUT и рассчитайте адрес регистра выходных данных порта ввода/вывода путём соответствующего смещения относительного базового адреса [1, п. 8.4.11 GPIO register map].

3.12 Определите название шины для тактирования порта ввода/вывода светодиодов.

3.13 Опишите константу RCC_LED с базовым адресом источника тактирования.

3.14 Опишите константу RCC_ BUS_LED_EN с адресом регистра для включения тактирования порта ввода/вывода светодиодов.

3.15 Опишите константу RCC_LED_EN с номером бита для включения тактирования порта ввода/вывода светодиодов.

Включение тактирования порта ввода/вывода светодиодов:

3.16 В тексте программы (после метки Start:) загрузите адрес регистра RCC_ BUS_LED_EN в регистр общего назначения R0.

3.17 Загрузите в регистр R1 содержимое ячейки, адрес которой указан в регистре R0.

3.18 Установите в регистре R1 бит по номеру RCC_LED_EN в единицу.

3.19 Выгрузите содержимое регистра R1 в ячейку памяти по адресу, содержащемуся в регистре R0. После этого тактирование выбранного порта ввода/вывода будет запущено. Удерживать в регистре R0 адрес регистра RCC_ BUS_LED_EN больше необязательно.

Настройка режима порта ввода/вывода светодиодов:

3.20 В тексте программы загрузите адрес регистра GPIO_LED_DIR в регистр общего назначения R0.

3.21 Загрузите в регистр R1 содержимое ячейки, адрес которой указан в регистре R0.

3.22 Установите в единицу в регистре R1 биты, отвечающие за режим «выхода» для зеленого и красного светодиодов [1, п. 8.4 GPIO registers]. Используйте ранее определенные константы LED_RED и LED_GREEN.

3.23 Выгрузите содержимое регистра R1 в ячейку памяти по адресу, содержащемуся в регистре R0. После этого минимальная настройка порта «на выход» будет выполнена.

Включение/выключение светодиодов:

3.24 В тексте программы загрузите адрес регистра GPIO_LED_OUT в регистр общего назначения R0.

3.25 Загрузите в регистр R1 содержимое ячейки, адрес которой указан в регистре R0.

3.26 Установите в единицу в регистре R1 биты, отвечающие за состояние логической «1» для зеленого и красного светодиодов [1, п. 8.4 GPIO registers].

3.27 Выгрузите содержимое регистра R1 в ячейку памяти по адресу, содержащемуся в регистре R0. После этого минимальная настройка режима порта «на выход» будет выполнена.

3.28 Выполните компиляцию и сборку проекта. Для этого запустите файл make_project.bat.

3.29 Запрограммируйте микроконтроллер при помощи ST-LINK Utility.

3.30 В результате работы должны засветиться зеленый и красный светодиоды.

3.31 Далее включите только красный и только зеленый светодиоды.

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

3.33 Защитите отчет у преподавателя.

4 Контрольные вопросы

4.1 Для чего в микропроцессоре используются регистры общего назначения. Сколько их?

4.2 Опишите назначение регистров SP, LR, PC.

4.3 Что такое конвейер выполнения команд?

4.4 Какую последовательность операций необходимо выполнить, чтобы изменить состояние регистра блока периферии микроконтроллера?

4.5 Как установить определенный бит в регистре в единицу, при этом не изменив состояние других бит?

4.6 Как сбросить определенный бит регистре в ноль, при этом не изменив состояние других бит?

4.7 Какие нужно выполнить операции, чтобы активировать порт ввода/вывода?

4.8 Каким образом рассчитывается адрес требуемого регистра?

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. PM0214. Programming manual. STM32F3, STM32F4 and STM32L4 Series. Cortex®-M4 programming manual.– URL: www.st.com/resource/en/programming_manual/DM00046982.pdf (дата обращения: 10.01.2017).

3. MB1075. STM32F429I-DISCO schematics.– URL: http://www.st.com/resource/en/schematic_pack/stm32f429i-disco_sch.zip.

4. 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).

5. GNU ARM Embedded Toolchain.– URL: https://launchpad.net/gcc-arm-embedded (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).

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

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