Лабораторная работа #1

Принципы программирования микроконтроллера STM32F407VGT6 в среде Keil µVision

СОДЕРЖАНИЕ

1 Принципы программирования микроконтроллера STM32F407VGT6 в среде Keil µVision

Цель: изучить основные функциональные элементы интегрированной среды разработки keil µVision на примере исследования элементарных функций ввода/вывода.

Оборудование и программное обеспечение: плата STM32F4 Discovery, интегрированная среда разработки Keil uVision, библиотеки StdPeriph и CMSIS.

1.1 Теоретические сведения

При проведении работы используется отладочная плата STM32F4Discovery, внешний вид и структура которой представлены на рис.1.

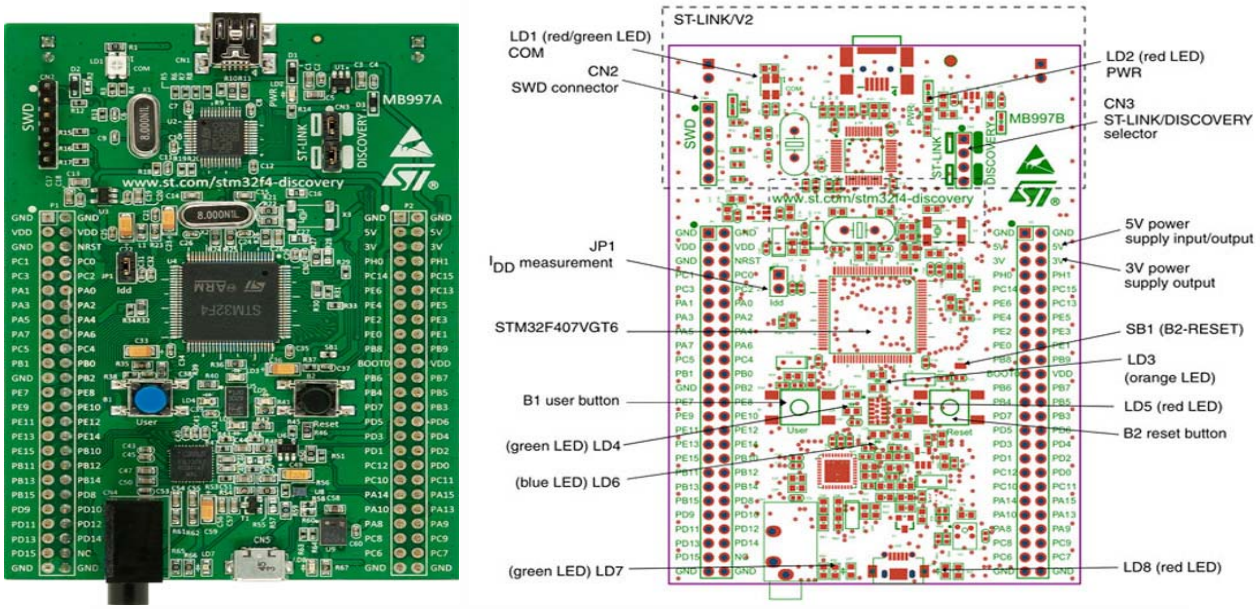


Рисунок 1 - внешний вид (слева) и структура (справа) платы STM32F4Discovery

В составе платы используется 32-битный ARM микроконтроллер STM32F407VGT6 семейства Cortex-M4. Микроконтроллер выполнен с возможностью питания от 1.8 до 3.6 Вольт; максимальной частотой работы до 168 МГц; с внутренней памятью программ (Flash) до 1 Мбайт и RAM 196 Кбайт; 4 Кбайт backup SRAM; имеется 82 контакта для программирования ввода и вывода(I/O) или 82xGPIO; поддерживает инструкции DSP с возможностью отладки как по SWD так и по JTAG. Кроме того на «борту»

находятся 16 и 32-битные таймеры, full-speed-USB 2.0, MAC уровень10/000 Ethernet, IEEE 1588v2, MII/RMII, 3xSPI, 3xI2G, 4xUSART, 2xUART(10,5 Мбит/с), ISO 7816, LIN, IrDA, 2xCAN, SDIO(SD) DCMI(до 54 Мбайт/с), RTC, 3x12 бит, АЦП (52 МГц); 2x12битных ЦАП, DMA(16 каналов + FIFO), LCD(8080/6800), FSMC(Compact Flash, SRAM, PSRAM, NOR, NAND); отладчик ST-Link выведен разъемом mini-USB, предназначенный для отладки и прошивки программы МК; все 100 выводов МК выведены по бокам платы, среди них есть выводы для внешнего питания 5 и 3,3 В. На плате присутствует кнопка сброса, четыре светодиода и одна пользовательская кнопка. LIS302DL – MEMS-датчик движения, 3х-осевой цифровой акселерометр; MP4DT02 – цифровой MEMS-микрофон; CS43L22 – аудио-ЦАП со встроенным драйвером динамиков класса D. Для USB OTG выведен разъем micro-USB. Если плату с заводской прошивкой подключить через этот разъем к компьютеру, то она будет вести себя как джойстик класса USB HID.

Так же при проведении работ используется интегрированная среда разработки Keil µVision - набор утилит для выполнения полного комплекса мероприятий по написанию программного обеспечения для микроконтроллеров.

Keil µVision позволяет работать с проектами любой степени сложности, начиная с введения и правки исходных текстов и заканчивая внутрисхемной отладкой кода и программированием ПЗУ микроконтроллера. К основным программным средствам Keil µVision относят:
1. Базу данных микроконтроллеров, содержащую подробную информацию обо всех поддерживаемых устройствах. В ней хранятся конфигурационные данные и ссылки на источники информации с дополнительными техническими описаниями. Поэтому При добавлении нового устройства в проект все его уникальные опции интегрируются автоматически.
2. Менеджер проекта, объединяющий отдельные текстовые программные модули и файлы в группы, обрабатываемые по единым правилам, что позволяет легко ориентироваться среди множества файлов.

3. Встроенный редактор, облегчающий работу с исходным текстом. При этом редактирование остается доступным и во время отладки программы, что позволяет на «горячую» исправлять участки кода.
4. Средства автоматической компиляции, ассемблирования и компоновки проекта, которые предназначены для создания исполняемого (загрузочного) модуля программы. Функция глобальной оптимизации проекта позволяет достичь наилучшего использования регистров микроконтроллера путем неоднократной компиляции исходного кода. Компиляторы uVision работают с текстами, написанными на Си или ассемблере.
5. Отладчик-симулятор, отлаживающий работу скомпилированной программы на виртуальной модели микропроцессора; в нём довольно достоверно моделируется работа ядра контроллера и его периферийного оборудования: портов ввода-вывода, таймеров, контроллеров прерываний.

1.2 Начало работы

Для начала работы в Keil µVision, необходимо во вкладке Project, создать новый проект; выбрать папку расположения всех будущих файлов проекта.

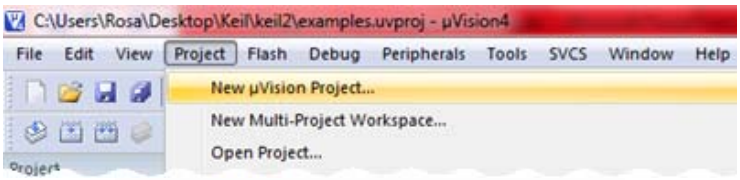


Рисунок 2 - Создание нового проекта

В появившемся окне выбрать программируемое устройство – в нашем случае это stm32f407VG(STMicroelectronics).

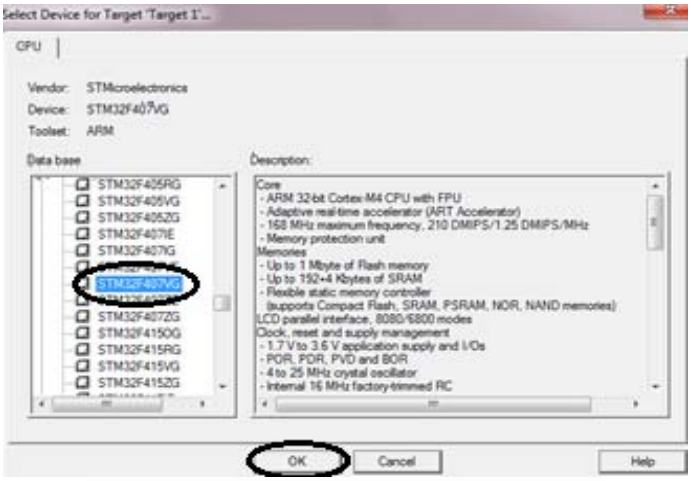


Рисунок 3 - Выбор микроконтроллера

После выбора устройства необходимо согласиться с установкой Startup file, что добавляет в проект первый файл **startup_stm32f40xx.s (**набор файлов для каждой линейки семейства STM32, обеспечивающих инициализацию стека и таблицу векторов прерываний), и его можно будет найти в папке проекта **Source Group 1.** Для удобства рекомендуется переименовать эту группу в **Startup,** что облегчит понимание того, что здесь расположено.

Для дальнейшей возможности использования простейших функций при программировании необходимо подключить к проекту библиотеки CMSIS StdPeriph. Их можно скачать с сайта st.com, пройдя по следующему маршруту Products→STM32 32-bit ARM Cortex MCUs→ STM32F4 Series→ STM32F 407/417→STM32F407VG(Port Number) → design resources. В пункте **Related Tools and Software** находим нужный пакет файлов **STSW-STM3206** (STM32F4 DSP and standart peripherals library). Из архива **stm32f4 dsp stdperiph lib.zip** нам необходима только папка **Libraries** и файл **stm32f4xx_conf.h.** В корень папки с нашим проектом добавляем папку **Libraries**. При этом в самой папке **Libraries** располагается два каталога. В каталоге **CMSIS** необходимо оставить только папку **Device**, а всё остальное в этом каталоге – удаляем.

С файлов **stm32f4xx.h** и **system_stm32f4xx.h** находящихся по пути **%Каталог проекта%Libraries\CMSIS\Device\ST\STM32F4xx\Include** снимаем атрибут «только чтение».

Файл **stm32f4xx_conf.h** размещаем в каталоге проекта **%Libraries\STM32F4xx_StdPeriph_Driver.** Сам файл берётся из скаченного с st.com архива в папке с любым примером по пути **Project\ STM32F4xx_StdPeriph_Examples\любой.**

Для простоты понимания в будущем сложных, многомерных программ удобно разделить инициализацию библиотек и встраиваемых кодов. Поэтому, в папке **User** нашего проекта создадим два каталога **inc** и **src.** В **src** разместим пустые файлы **main.c** и **init.c.** В одном будет находиться основной код программы, а во втором – функции инициализации. В **inc** разместим пустые файлы **main.h** и **init.h.** В одном будут находиться определения и основные библиотеки, а в другом – библиотеки инициализации.

При создании файлов, как «ни странно» они сами в проекте keil не прописываются. Поэтому их нужно инициализировать и подключить к нашему проекту. Для этого:

1. Создадим в проекте папки(группы) **CMSIS**, **StdPeriph** и **User** (правой кнопкой мыши нажимаем на **target 1** и выбираем **add group**);
2. Добавим файлы соответствующие группы как на рис.4

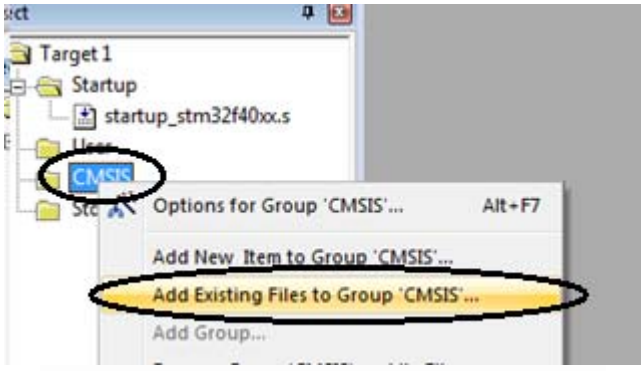


Рисунок 4 – Добавление файлов в группу

а) **system_stm32f4xx.c** в группу **CMSIS** из каталога %CMSIS\Device\ST\STM32F4xx\Source\Templates. **system_stm32f4xx.c** – файл начальной инициализации тактовой частоты микроконтроллера.

б) из каталога %STM32F4xx_StdPeriph_Driver\src все файлы в группу **StdPeriph**, кроме **stm32f4xx_fmc**, так как у данного МК контроллера внешней памяти нет.

В эту же группу добавляем файл **stm32f4xx_conf.h**

в) в группу **User** добавляем файлы **init.c, main.c.**

Для корректной работы Keil необходимо сконфигурировать параметры компилятора, отладчика и среды разработки именно под наш тип микроконтроллера. Через Target Options (рисунок 5)

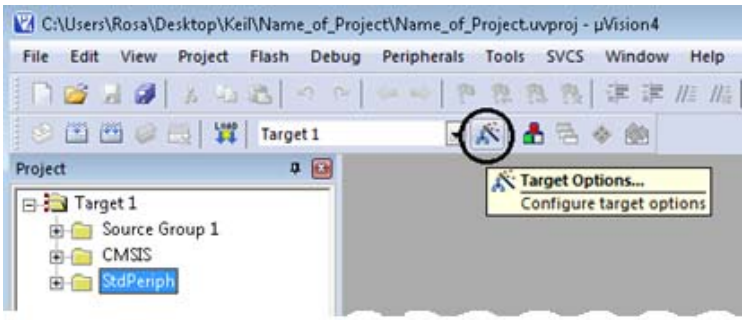


Рисунок 5 – Открытие Target Options

переходим во вкладку C/C++ и указываем пути ко всем файлам проекта (рисунок 6)

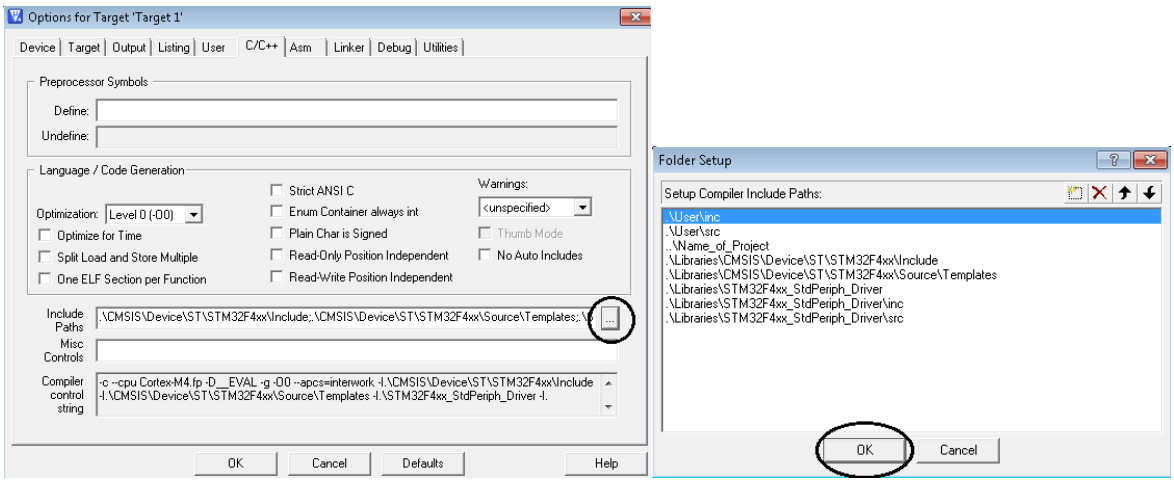


Рисунок 6 - Указание путей для компилятора

Во вкладке Debug из списка отладчиков выбираем ST-Link Debugger и проверяем расположение выбранных пунктов согласно рисунку 7

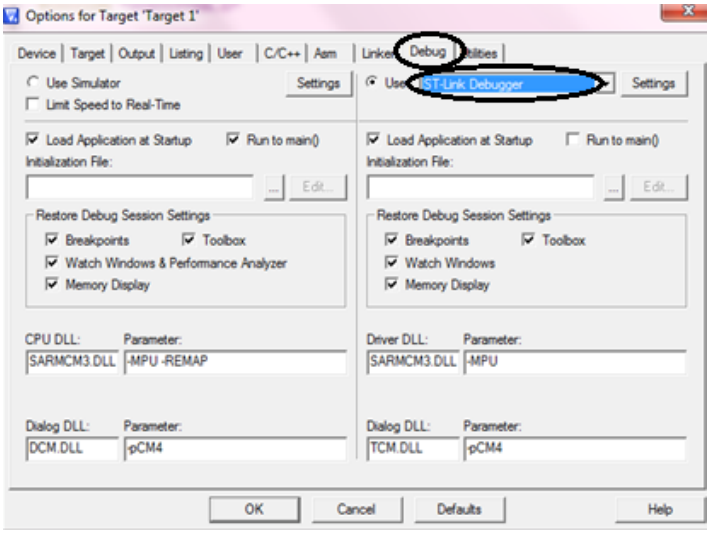


Рисунок 7 - Указание параметров отладчику

В настройках отладчика выбираем порт SW, так как предполагается использование ST-Link исключительно в режиме SW.

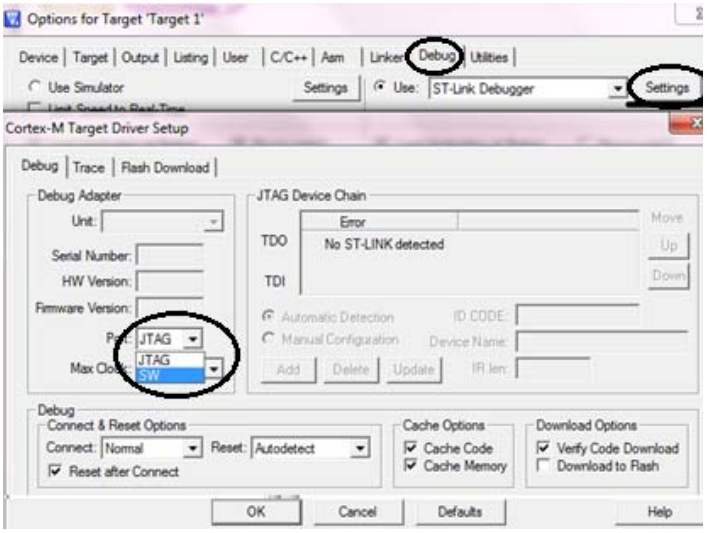


Рисунок 8 – Выбор портов отладчика

Здесь же выбираем алгоритм прошивания программы в Flash память микроконтроллера.

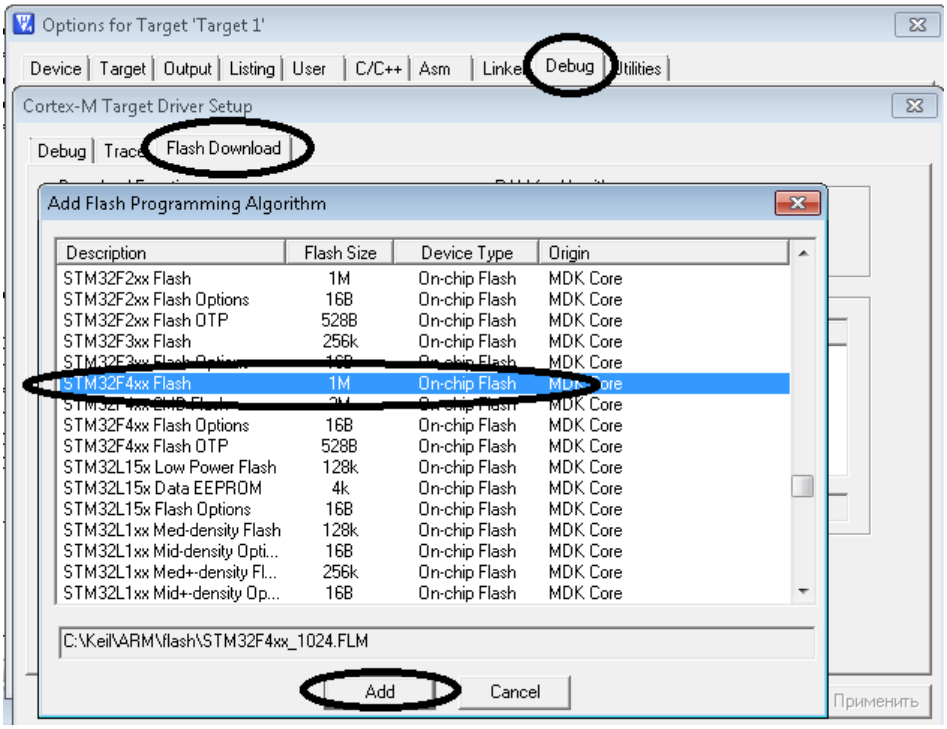


Рисунок 9 – Выбор алгоритма прошивания

Во вкладке **Utilites** выбираем **ST-Link Debugger**.

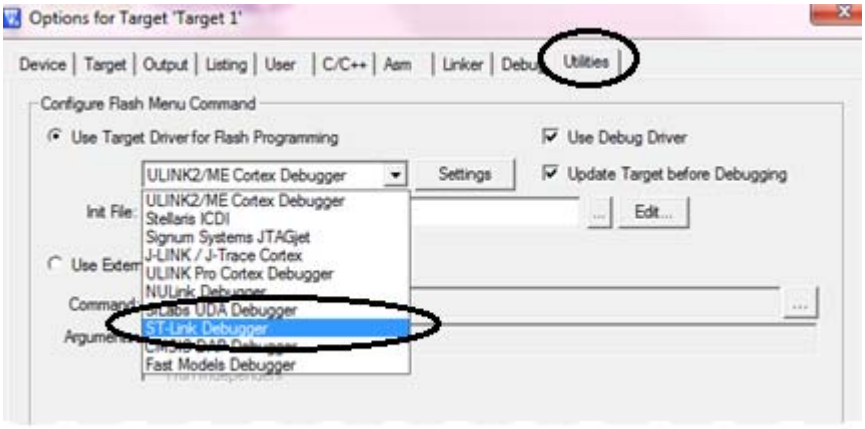


Рисунок 10 – Выбор отладчика

Библиотеки и подключаемые файлы (заголовочные файлы) инициализируются в тексте программы с помощью *директивы препроцессора* `#include`. При этом имя файла может быть указано двумя различными по функциональности способами: `#include <some_file.h>` `#include "my_file.h"`.

Если имя файла заключено в угловые скобки, это означает, что нужна стандартная библиотека и компилятор ищет её в самом Keil. Двойные кавычки

означают, что заголовочный файл – внешний файл пользователя, и его поиск начинается с каталога, в котором находится исходный текст программы.

Минимальный текст программы, которая инициализирует контроллер и выполняет код, записанный в цикле `while`(т.е. ничего не выполняет в нашем случае) в файле **main.c** имеет следующий вид:

```
#include "main.h"
//Declare your global variables here
int main(void)
{
   //Declare your local variables here
    while(1)
    {
        //Place your code here
     }
  }
```

При этом в **init.c** должна быть «подрублена» библиотека:

```
#include "init.h"
```

В дальнейшем дополнительные файлы и библиотеки будут подключаться в файлах **main.h** и **init.h**. Сейчас же они пустые. Отметим, что не очень внимательные инженеры при программировании могут инициализировать заголовочные файлы в нескольких местах программы, что увеличивает время выполнения исходного кода. Для защиты от этого рекомендуется использовать *условные директивы препроцессора*.

Поэтому в **main.h** (файл, не отображающийся в дереве папок слева, но его можно открыть по правой кнопке мыши, клацнув на его название в месте инициализации.) необходимо задать следующую структуру:

```
#ifndef MAIN_H
#define MAIN_H
  //библиотеки, подключаемые файлы, макроподстановки
#endif
```

Условная директива `#ifndef` проверяет, не было ли значение `MAIN_H` определено ранее. Препроцессор обрабатывает следующие строки вплоть до директивы `#endif`. В противном случае он пропускает строки от `#ifndef` до `#endif`.

Директива `#define MAIN_H` определяет константу препроцессора `MAIN_H`. Поместив эту директиву непосредственно после директивы `#ifndef`, мы можем гарантировать, что содержательная часть заголовочного файла **main.h** будет включена в исходный текст только один раз. Аналогичная структура прописывается в файле **init.h**.

В **main.h** включаем библиотеку: `#include "init.h"`

В **init.h** включаем библиотеку `#include "stm32f4xx.h"`.

Файл **stm32f4xx.h** содержит:

1. Раздел Конфигураций, что позволяет:
   а) Выбрать устройство, используемое в целевом приложении
   б) Указать, использовать или нет драйвер для периферий в коде приложения
   в) Указать частоту резонатора
2. Структуры данных и отображение адреса для всех периферийных устройств
3. Декларацию периферийных регистров
4. Экспортируемые константы
   а) Определение битов периферийных регистров
   б) Экспортируемые макросы

В библиотеке **stm32f4xx.h** необходимо раскомментировать строку макроопределения для нужного микроконтроллера – STM32F407VG, который относится к группе STM32F40_41xxx. (Рисунок 11)

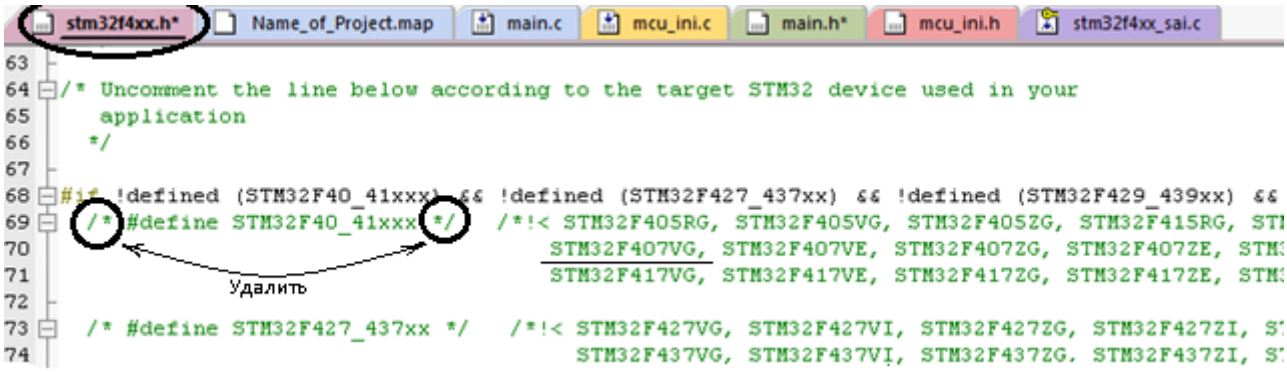


Рисунок 11 – Редактирование файла stm32f4xx.h

По умолчанию код приложения основывается на прямом доступе регистрам периферии. Чтобы использовать драйверы периферийных устройств, нужно в этом же файле снять комментирование со строки `#define USE_STDPERIPH_DRIVER`

Здесь же можно задать значение тактовой частоты, которая по умолчанию равна 25 МГц. Установим её значение 8МГц. (Чтобы упростить работу с файлами в Keil можно пользоваться поиском)

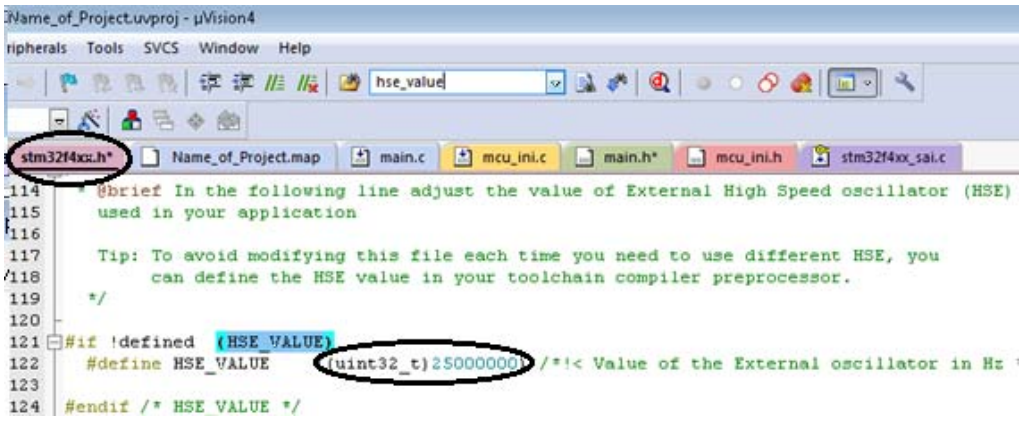


Рисунок 12 - Изменение значения тактовой частоты

При нажатии F7 (Build) происходит сбор проекта и проверка на ошибки. Если при этом не выдаются какие-либо предупреждения, значит, подготовка проекта по вышеуказанному алгоритму прошла успешно.

Теперь проект окончательно сформирован и можно приступать к написанию программы.

1.3 Примеры простых программ

Написать программу можно используя одну из библиотек: CMSIS или StdPeriph, а так же их «симбиоз».

CMSIS это стандартная библиотека для всех Cortex контроллеров, несмотря на то, что периферия у всех ARM Cortex контроллеров разная, даже в пределах одной линейки. Это возможно из-за того, что доступ к регистрам периферии из Си стандартизирован и представлен в виде слоя аппаратной абстракции Cortex Microcontroller Software Interface Standart (CMSIS).

StdPeriph (Standard Peripherals Library) - библиотека, созданная компанией STMicroelectronics на языке Си для своих микроконтроллеров семейства STM32F4xx и содержит структуры, функции и макросы для облегчения работы с периферией микроконтроллера.

Рассмотрим два варианта написания одной и той же простейшей программы как с использованием CMSIS так и Std Periph.

1.3.1 Проект управления диодом через библиотеку CMSIS

Схема подключения четырёх светодиодов на плате STM32F4Discovery представлена на рисунке 13.

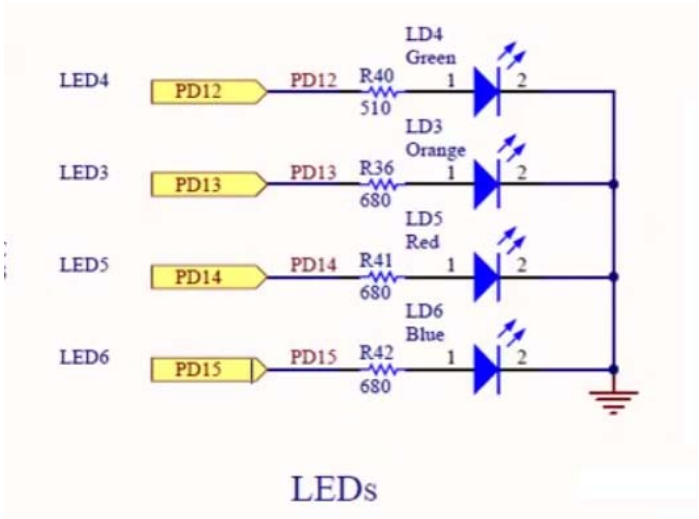


Рисунок 13 - Схема подключения светодиодов на плате STM32F4 Discovery

Как видно, диоды находятся на 12, 13, 14, 15 выводах порта D.

GPIO (General Purpose Input/Output) - самый простой и примитивный способ организации работы с внешними устройствами. Порты могут работать в двух режимах: вход (прием сигнала) и выход (передача сигнала). Работают они только с логическими уровнями 0 (Low) или 1 (Hight). Например, если подключить к порту в режиме выхода светодиод (как указано на рисунке 13), то при подаче сигнала высокого уровня светодиод будет светиться, а при подаче низкого – потухнет. Если включить вывод в режим входа и подключить к нему кнопку,соединённую с землёй, то с помощью микроконтроллера можно отслеживать ее состояние: нажатое или отжатое.

Ядро Cortex подключено к Flash памяти по отдельной шине инструкций. Шина данных и системная шина Cortex подключены к матрице высокоскоростных шин AHB, с которой так же связан блок прямого доступа к памяти (ПДП). Подключение встроенных устройств ввода/вывода (Input/Output или периферии) распределены между двумя шинами (мостами) APB1 и APB2, соединёнными с матрицей шин AHB, причем к каждому мосту подключен свой набор периферии. Дело в том, что у этих мостов различаются максимальные рабочие частоты, и менее быстрая периферия подключена к мосту APB1. В качестве шинных мастеров могут выступать как ЦПУ Cortex, так и блок ПДП. Благодаря свойственной матрице шин параллелизму, необходимость в арбитраже возникает только в случае попыток одновременного доступа обеих мастеров к статическому ОЗУ, шины APB1 и APB2. Тем не менее, шинный арбитр гарантированно предоставляет 2/3 времени доступа для блока ПДП и 1/3 для ЦПУ Cortex.

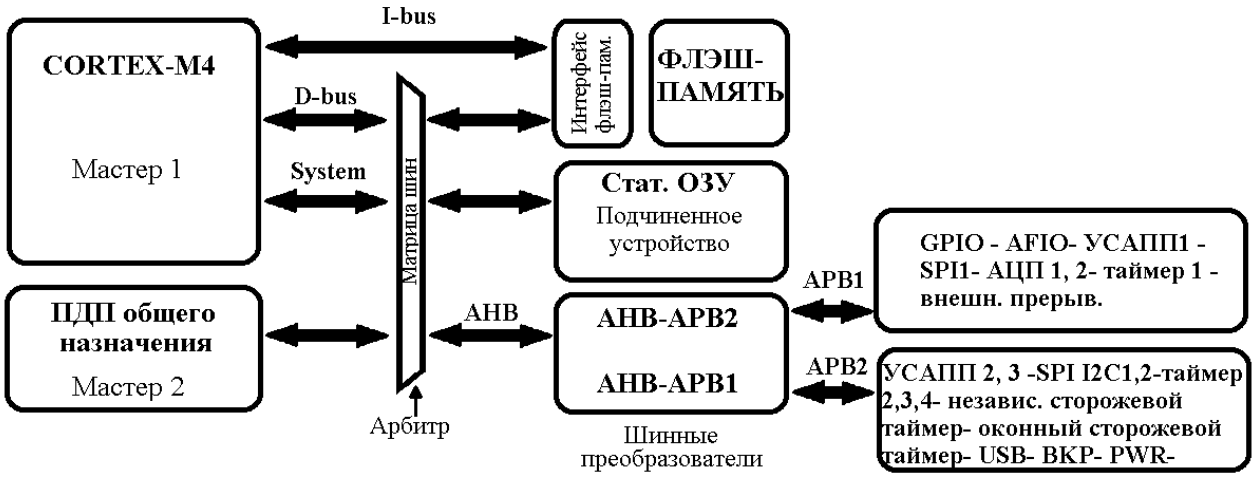
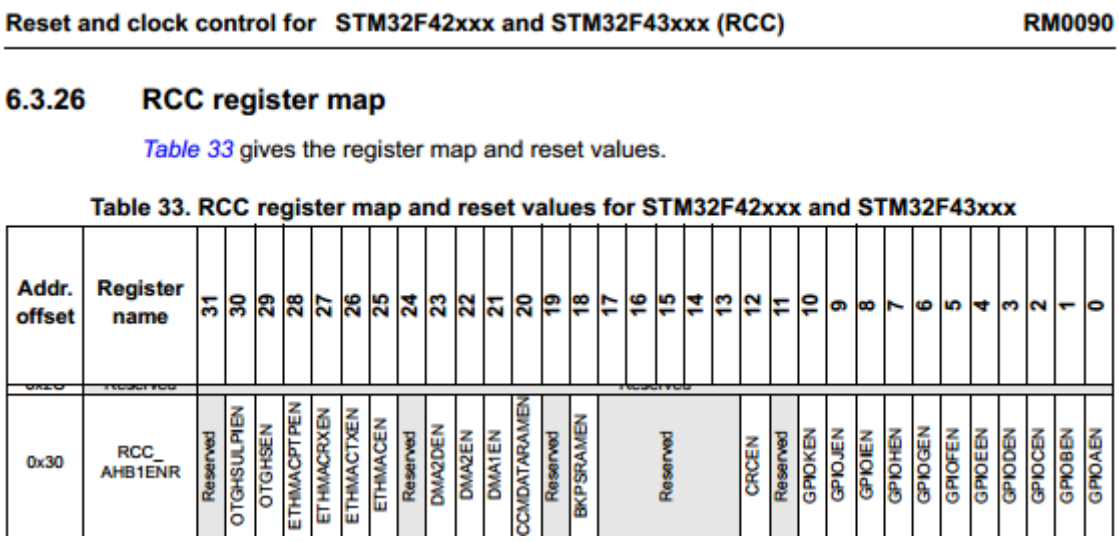


Рисунок 14 –Схема подключения к флеш-памяти

Для снижения электропотребления микроконтроллера практически все периферийные устройства блоки после включения микроконтроллера – отключены. Включение периферийного устройства производится передачей тактового сигнала на его вход.

Для того чтобы включить тактирование (микропроцессору будет указано, что ему нужно выделить некоторое количество времени в своей работе для

периодического контроля состояния своего периферийного порта, в данном случае PD), нужно обратиться к двум datasheets: Reference Manual и Product Specification (Показывает, какая ножка за что отвечает), которые расположены на st.com. В Reference Manual, находим раздел Reset and clock control for STM32F42xx and STM32F43xx (RCC). RCC – означает группу управления сбросом и синхронизацией, в ней находятся все регистры управления настройками генераторов, ФАПЧ и шин.



Reset and clock control for STM32F42xxx and STM32F43xxx (RCC) RM0090

6.3.26 RCC register map

Table 33 gives the register map and reset values.

Table 33. RCC register map and reset values for STM32F42xxx and STM32F43xxx

Addr. offset	Register name	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0x30	RCC_ AHB1ENR	Reserved	OTGHSULPIEN	OTGHSEN	ETHMACPTPEN	ETHMACRXEN	ETHMACTXEN	ETHMACEN	Reserved	DMA2DEN	DMA2EN	DMA1EN	CCMDATARAMEN	Reserved	BKPSRAMEN	Reserved						CRCEN	Reserved	GPIOKEN	GPIOJEN	GPIOIEN	GPIOHEN	GPIOGEN	GPIOFEN	GPIOEEN	GPIODEN	GPIOCEN	GPIOBEN	GPIOAEN

Рисунок 15 Таблица регистров RCC

Находим **GPIODEN**, где **GPIO** означает способ связи с внешней средой, а имя порта – **D**, при этом **EN** – сокращение от англ. Enable(активный). Как видно, за него отвечает регистр **RCC AHB1ENR**.

Для включения тактирования порту D в файле **init.c** (В этом файле будут располагаться все инициализации) прописываем следующее:

```
void LEDs_ini(void) //функция для инициализации диодов
{
  RCC->AHB1ENR|=RCC_AHB1ENR_GPIODEN;
}
```

Эта строка устанавливает 3-ий бит регистра **RCC AHB1ENR** в 1 и таким образом активируется тактирование порта D.

Периферийный блок не может начать работу сразу после включения тактового сигнала, необходимо подождать несколько тактов, пока он запустится. Взаимодействие ядра микроконтроллера с периферийным блоком

осуществляется с помощью спецрегистров (есть ещё взаимодействие через механизм прерываний в DMA, но об этом в следующих лабораторных работах). Спецрегистры обычно разделены на битовые поля. Один (или несколько) бит управляют определённым параметром периферийного блока, обычно независимо.

Теперь нужно настроить выводы D12,D13,D14,D15 как выходы. Согласно последней главе раздела General-purpose I/Os (GPIO) Reference Manual:

1. GPIOx_Moder - спецрегистр, который определяется для каждого порта микроконтроллера и отвечает за режим работы. За инициализацию каждого вывода регистра отвечает 2 бита, при этом их значение соответствует следующим режимам:

   00: Вход

   01: Выход

   10:Альтернативная функция

   11:Аналоговый режим

8.4.1 GPIO port mode register (GPIOx_MODER) (x = A..I/J/K)

Address offset: 0x00

Reset values:

- 0xA800 0000 for port A
- 0x0000 0280 for port B
- 0x0000 0000 for other ports

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
MODER15[1:0]		MODER14[1:0]		MODER13[1:0]		MODER12[1:0]		MODER11[1:0]		MODER10[1:0]		MODER9[1:0]		MODER8[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
MODER7[1:0]		MODER6[1:0]		MODER5[1:0]		MODER4[1:0]		MODER3[1:0]		MODER2[1:0]		MODER1[1:0]		MODER0[1:0]	
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw
01	01	01	01	00	00	00	00	00	00	00	00	00	00	00	00

Рисунок 16 GPIOx_Moder

Ножкам микроконтроллера к которым подключены диоды необходимо настроить как выходы, поэтому на месте 12,13,14 и 15 регистра выводов

записываем 01 и получаем: 01010101 00000000 00000000 00000000, что эквивалентно 0x5500 0000.

В файле **init.c**, в функции для инициализации диодов, после строки, включающей тактирование порта D, активируем ножки D12, D13, D14, D15 как выходы:

```
GPIOD->MODER=0x55000000;
```

2. GPIOx_OTYPER отвечает за подтягивание нижки к земле. Это означает, что если подтягивание включено, то на выходе в качестве низкого сигнала будет сигнал с земли, в противном случае – ножка будет подвешена в свободное состояние: 0 – подтянуть, а 1-без подтягивания.

8.4.2 GPIO port output type register (GPIOx_OTYPER) (x = A..I/J/K)

Address offset: 0x04

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OT15	OT14	OT13	OT12	OT11	OT10	OT9	OT8	OT7	OT6	OT5	OT4	OT3	OT2	OT1	OT0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Bits 31:16 Reserved, must be kept at reset value.

Рисунок 17 GPIOx_OTYPER

Таким образом, добавляем запись в функцию инициализации диодов:

```
GPIOD->OTYPER=0;
```

3. GPIOx_OSPEED отвечает за то, как часто микропроцессор будет проверять состояние регистра I/O т.е. фактически за скорость срабатывания

Значения:

00 – Низкая (2 МГц)

01 – Средняя(25 МГц)

10 – Высокая(50 МГц)

10 – Наиболее высокая(100 МГц)

Поскольку скорость нас сильно не интересует:

```
GPIOD->OSPEEDR=0;
```

4. GPIOx_ODR определяет, высокое или низкое напряжение будет на выходе.

Значения:

1 – Высокое

0 – Низкое

Светодиоды светятся при высоком напряжении, следовательно на ODR 15,14,13,12 нужно установить логическую единицу.

8.4.6 GPIO port output data register (GPIOx_ODR) (x = A..I/J/K)

Address offset: 0x14

Reset value: 0x0000 0000

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Reserved															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ODR15	ODR14	ODR13	ODR12	ODR11	ODR10	ODR9	ODR8	ODR7	ODR6	ODR5	ODR4	ODR3	ODR2	ODR1	ODR0
rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw	rw

Рисунок 17 GPIOx_ODR.

В файле **main.c**, внутри цикла `while` записываем: `GPIOD->ODR=0xF000;` В результате выполнения этой команды будут светиться все четыре диода. Чтобы светился только один светодиод, мы устанавливаем логическую единицу только на одном выводе. Например, чтобы светился только красный диод(PD14): `GPIOD->ODR=0x4000;`

Перед запуском проекта убедитесь, что вы вызвали функцию инициализации `LEDs_ini()` в файле **main.c** (перед циклом while) и определили её в файле **init.h**. : `void LEDs_ini(void);`

Чтобы диод мигал, используем внешнюю функцию задержки (в **main.c**):

```
void Delay(uint32_t nCount)
{
  while(nCount--){}
}
```

Для удобства и наглядности можно использовать директиву #define, которая располагается в файле **main.h:**

```
#define RED_ON()    GPIOD->ODR=0x4000
#define RED_OFF()   GPIOD->ODR=0
```

Все `RED_ON()` будут автоматически заменяться макроподстоновкой на `GPIOD->ODR=0x4000`, а `RED_OFF()` на `GPIOD->ODR=0`.

Таким образом, файл **main.c** будет выглядеть следующим образом:

```
#include "main.h"
void Delay(volatile uint32_t nCount)
{
  while(nCount--) {}
}
int main(void)
{
  LEDs_ini();
  while(1)
  {
    RED_ON();
    Delay(8000000);
    RED_OFF();
    Delay(8000000);
  }
  }
```

Для прошивки программы в микроконтроллер необходимо подключить устройство к компьютеру через mini-Usb и установить драйвер.

Драйвер можно скачать с сайта st.com, пройдя по следующему маршруту : Products→STM32 32-bit ARM Cortex MCUs→ STM32F4 Series→ STM32F 407/417→STM32F407VG(Port Number) →design resources. В пункте **Related Tools and Software** находим нужный пакет файлов **ST-LINK/V2** (ST-LINK/V2 in-circuit debugger/programmer for STM8 and STM32) и скачиваем подходящий драйвер.

Теперь если нажать ctrl + F5 проект соберется и загрузится в flash память микроконтроллера. Можно наблюдать, как мигает красный светодиод.

1.3.2 Проект управления диодом через библиотеку StdPeriph

Эта библиотека представляется компанией ST Microelectronics для использования со своими микроконтроллерами. Данная библиотека, как следует из ее описания самим производителем, предназначена лишь в качестве руководства к действию, но может облегчить жизнь программисту и сократить время разработки проекта.

По сути это просто набор функций, позволяющих намного легче и проще выполнять какие-то основные моменты. В основном, упор сделан на функции инициализации периферии, ее включения/выключения, опроса состояния и т.д.

Код становится проще (хотя на данном простейшем примере это вопрос спорный), но как минимум не требуется знать, что и в какие регистры надо прописать. Так как используются функции с вполне говорящими названиями вместо битовых операций, то код становится понятнее. Написать такой код начинающему инженеру гораздо проще и быстрее чем «грызть» даташит, чтобы узнать какие регистры и как нужно проинициализировать, также не надо тасовать «вагон» PDF'ок и помнить, что куда ссылается. Большая часть необходимой информации теперь находится непосредственно в проекте.

Разумеется, нужно проделать те же процедуры, что и в случае с использованием библиотеки CMSIS, а именно:

1. Включить тактирование порта. За тактирование в данной библиотеке отвечает файл: stm32f4xx.rcc.c→ **stm32f4xx.rcc.h.** Нужно найти `#define`, отвечающий за GPIOD

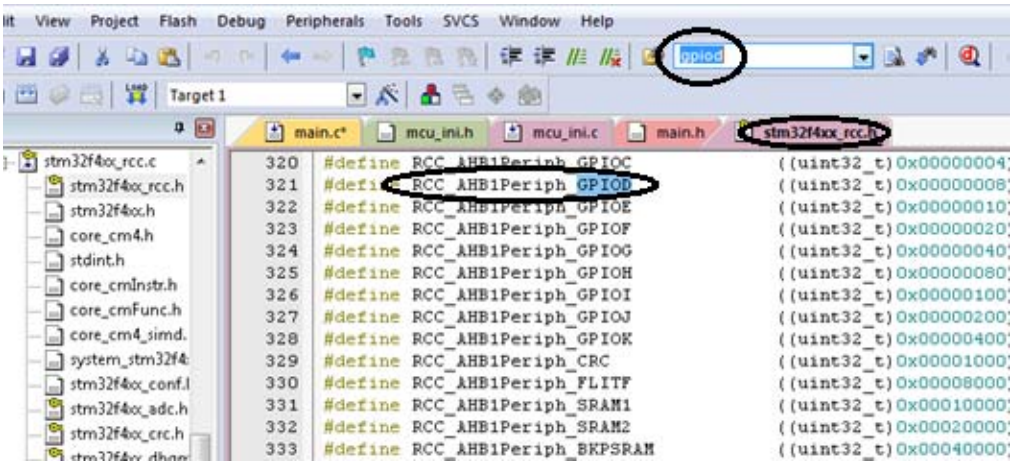


Рисунок 19 –Поиск функции, отвечающей за тактирование порта D

Функция, отвечающая за тактирование порта D, будет начинаться с `RCC_AHB1Periph`. Найдем по этому ключу в этом же файле следующую функцию:`RCC_AHB1PeriphClockCmd(uint32_tRCC_AHB1Periph,FunctionalState NewState);` и скопируем данную функцию в **init.c** На первом месте в скобках пишем макроопределение `#define`, отвечающего за GPIO, найденный ранее, на втором месте – состояние: в данном случае – ENABLE. В итоге получим:

```
void LEDs_ini()//функция для инициализации диодов
{
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
```

```
}
```

Пропишем эту функцию в **init.h .** `( void LEDs_ini(void);)`

2. Настроить работу порта D. В Библиотеке StdPeriph используются готовые структуры, которые необходимо заполнить нужными значениями. Соответствующую структуру для порта D можно найти в файле:

stm32f4xx_gpio.c→ **stm32f4xx_gpio.h**

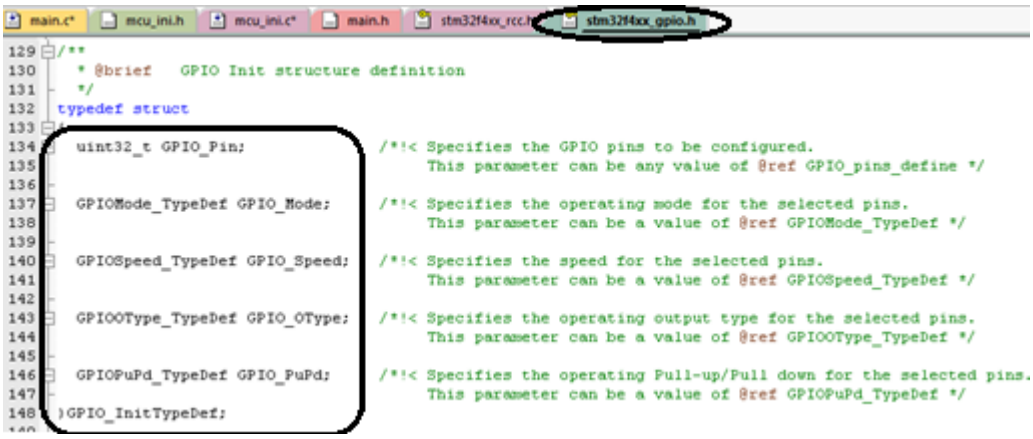


Рисунок 20 - GPIO_InitTypeDef

Как видно, название структуры - **GPIO_InitTypeDef** (используется как тип данных). В файле **init.c** создадим переменную `Init_LEDs` с таким типом: `GPIO_InitTypeDef  Init_LEDs`, причем, так как переменные определяются до функций в которых они используются, объявлять эту переменную нужно до включения тактирования.

Далее, в конце функции инициализации диодов описываем структуру `Init_LEDs`**.** Указываем выводы, на которых расположены диоды: `Init_LEDs.GPIO_Pin=GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPIO_Pin_15`

устанавливаем режим работы – на выход:

`Init_LEDs.GPIO_Mode=GPIO_Mode_OUT;`

устанавливаем скорость работы=2МГц :

`Init_LEDs.GPIO_Speed=GPIO_Speed_2MHz;`

подтягиваем вывод к земле:

`Init_LEDs.GPIO_OType=GPIO_OType_PP;`

`Init_LEDs.GPIO_PuPd=GPIO_PuPd_NOPULL` (используется при режиме работы на вход, поэтому на данный момент значение неважно).

После этого нужно инициализировать эту структуру. Функция инициализации `GPIO_Init(GPIOD, &Init_LEDs);` находится в **stm32f4xx_gpio.h** в самом конце. На первом месте пишется название порта, на втором – указатель на структуру.

2. Теперь нам нужно менять состояние светодиодов. В файле **stm32f4xx_gpio.h** описаны все необходимые для этого функции.
   `void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);` - устанавливает 1 на указанной ножке;
   `void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);`- сбрасывает значение;
   `void GPIO_WriteBit(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin, BitAction BitVal);` - записывает определённое значение на выводе;
   `void GPIO_Write(GPIO_TypeDef* GPIOx, uint16_t PortVal);` - записывает определённое значение на весь порт;
   `void GPIO_ToggleBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);`- инвертирует состояние бита/

Для изменения состояния светодиода удобнее всего использовать функцию GPIO_ToggleBits( ). Поэтому в файле **main.h** прописываем соответствующее макроопределение:

```
#define CHANGE_RED()  GPIO_ToggleBits(GPIOD, GPIO_Pin_14)
```

Пример программы в **main.c** будет выглядеть следующим образом:

```
#include "main.h"
void Delay(volatile uint32_t nCount)
{
  while(nCount--) {}
}
int main(void)
{
  LEDs_ini();
  while(1)
  {
    CHANGE_RED();
    Delay(8000000);
  }
}
```

1.4 Задания и методические указания к проведению работы

1. Для закрепления теоретических навыков, в среде Keil µVision создать проекты, повторяющие примеры, описанные ранее и проверить их работу на плате STM32F4Discovery.
2. На основе ознакомления с примерами проектов и описанием файлов библиотеки CMSIS создать в интегрированной среде разработки Keil µVision проект, в котором организован код для поочерёдного, циклического включения светодиодов RGBY – по кругу. Проверить работу программы, загрузив её в память микроконтроллера.
3. Повторить задание по предыдущему пункту, но вместо библиотеки CMSIS использовать библиотеки StdPeriph.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Белов А.В. Микроконтроллеры AVR в радиолюбительской практике А.В.Белов. – М.: Наука и Техника, 2007.– 339 с.

2. Лабораторный практикум по изучению микроконтроллеров архитектуры ARM Cortex-M4 В. И. Бугаев, М. П. Мусиенко, Я. М. Крайнык Москва-Николаев – 2013, 52с.

3. http://www.diller-technologies.de/stm32_wide.html

4. http://mikrocontroller.bplaced.net/

5. http://www.youtube.com/user/Sappise

6. http://www.ebay.com/itm/Open407V-D-B-STM32F407VGT6-STM32-Cortex-M4-ARM-Development-Board-16-Module-Kits-/261050694929

7. Aveal. Программирование STM32F4. Создание нового проекта в Keil. март 20, 2013 // (Рус.). – URL: http://microtechnics.ru/programmirovanie-stm32f4-sozdanie-novogo-proekta-v-keil/ [15 апреля 2013].

Приложение А

Файл “main.c”

```
#include "main.h"
void Delay(volatile uint32_t nCount)
{
  while(nCount--) {}
}
int main(void)
{
  LEDs_ini();
  while(1)
  {
    RED_ON();
    Delay(8000000);
    RED_OFF();
    Delay(8000000);
  }
}
```

Файл “main.h”

```
#ifndef MAIN_H
#define MAIN_H
#include "init.h."
#define RED_ON()   GPIOD->ODR=0x4000
#define RED_OFF()  GPIOD->ODR=0
#endif
```

Файл “init.c”

```
#include "init.h"
void LEDs_ini(void) //функция для инициализации диодов
{
  RCC->AHB1ENR|=RCC_AHB1ENR_GPIODEN;

  GPIOD->MODER=0x55000000;
  GPIOD->OTYPER=0;
  GPIOD->OSPEEDR=0;
  GPIOD->ODR=0x4000;
}
```

Файл “init.h”

```
#ifndef INIT_H
#define INIT_H
#include "stm32f4xx.h"
void LEDs_ini(void);

#endif
```

Приложение Б

Файл “main.c”

```
#include "main.h"
void Delay(volatile uint32_t nCount)
{
  while(nCount--) {}
}
int main(void)
{
  LEDs_ini();
  while(1)
  {
    RED_ON();
    Delay(8000000);
    RED_OFF();
    Delay(8000000);
  }
}
```

Файл “main.h”

```
#ifndef MAIN_H
#define MAIN_H
#include "init.h."
#define RED_ON()            GPIO_SetBits(GPIOD, GPIO_Pin_14)
#define RED_OFF()           GPIO_ResetBits(GPIOD, GPIO_Pin_14)
#endif
```

Файл “init.c”

```
#include "init.h"
void LEDs_ini(void)
{
  GPIO_InitTypeDef GPIO_Init_LED;
  RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

  GPIO_Init_LED.GPIO_Pin=GPIO_Pin_12|GPIO_Pin_13|GPIO_Pin_14|GPI
  O_Pin_15;
  GPIO_Init_LED.GPIO_Mode=GPIO_Mode_OUT;
  GPIO_Init_LED.GPIO_Speed=GPIO_Speed_2MHz;
  GPIO_Init_LED.GPIO_OType=GPIO_OType_PP;
  GPIO_Init_LED.GPIO_PuPd=GPIO_PuPd_NOPULL;
  GPIO_Init(GPIOD, &GPIO_Init_LED);

}
```

Файл “init.h”

```
#ifndef INIT_H
#define INIT_H
#include "stm32f4xx.h"
void LEDs_ini(void);

#endif
```