**Лабораторная работа №1**

**Создание базового проекта для микроконтроллера STM32F407. Порты ввода вывода**

**Теоретические сведения**

Микроконтроллеры семейства STM32 позволяют производить тонкую настройку почти каждого вывода микросхемы. Сам микроконтроллер в корпусе LQFP100 показан на рисунке 1.

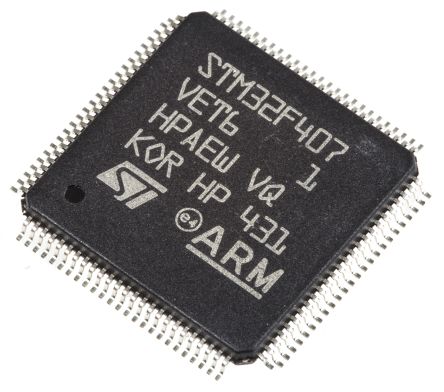


Рисунок 1 – Микроконтроллер STM32F407VET6

Схематично микросхема STM32F407VET6 показана на рисунке 10. Каждый вывод подписан определённым названием. Есть несколько выводов для подключения питания, внешнего резонатора, земли и др. Оставшиеся выводы являются портами ввода вывода микроконтроллера. Порт, в данном случае, означает объединение 16-ти выводов. Портами в микроконтроллерах STM32 называются буквами латинского алфавита (А, B, C…). На рисунке 2 такие выводы обозначаются буквой P, затем буквой порта и в конце номером вывода (от 0 до 15).

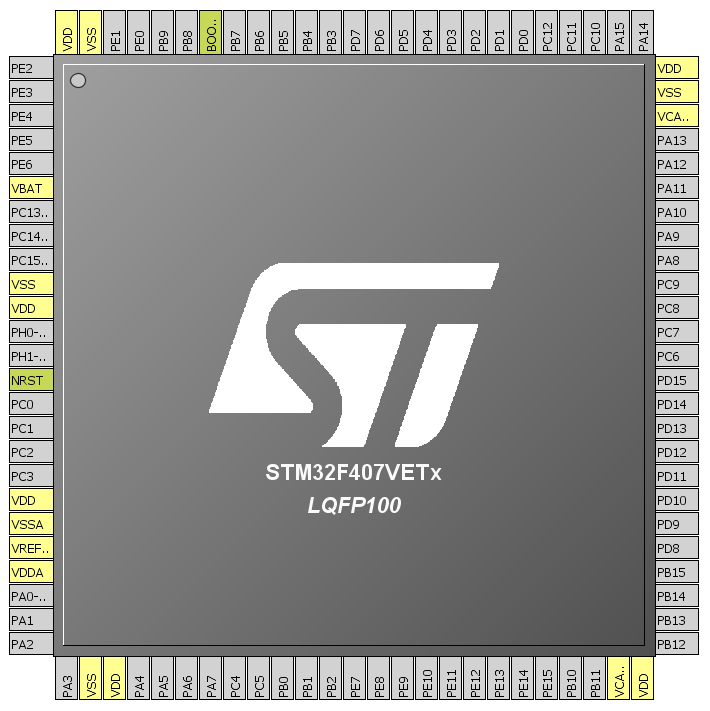


Рисунок 10 – Схематичное изображение микроконтроллера STM32F407VET6

Для предоставления максимальной гибкости работы каждый вывод общего назначения имеет в микроконтроллере структуру конфигурации (Рисунок 3), позволяющей настраивать режим работы вывода. У самого выхода находятся два защитных диода, защищающие микроконтроллер при подаче на вывод микросхемы напряжение ниже земли (например –3,3В) или выше напряжения питания микроконтроллера (например +6В).

Существуют следующие режимы работы вывода общего назначения:

а) Высокоимпедансный вход. На выходе установлена пара комплементарных полевых транзисторов (один p-типа, другой n-типа). Полевой транзистор в закрытом состоянии имеет почти бесконечное сопротивление между стоком и истоком. В этом режиме оба транзистора в закрытом состоянии, поэтому вывод не подключён ни к земле, ни к питанию – поэтому ведёт себя как не подключённый к схеме.

б) Вход с подтяжкой к питанию. Между входом и напряжением питания включается подтягивающий резистор (порядка 1 кОм), что позволяет находится в высоком состоянии, когда к входу не приложено внешнее напряжение. Это позволяет избежать спонтанных появлений 0 на входе.

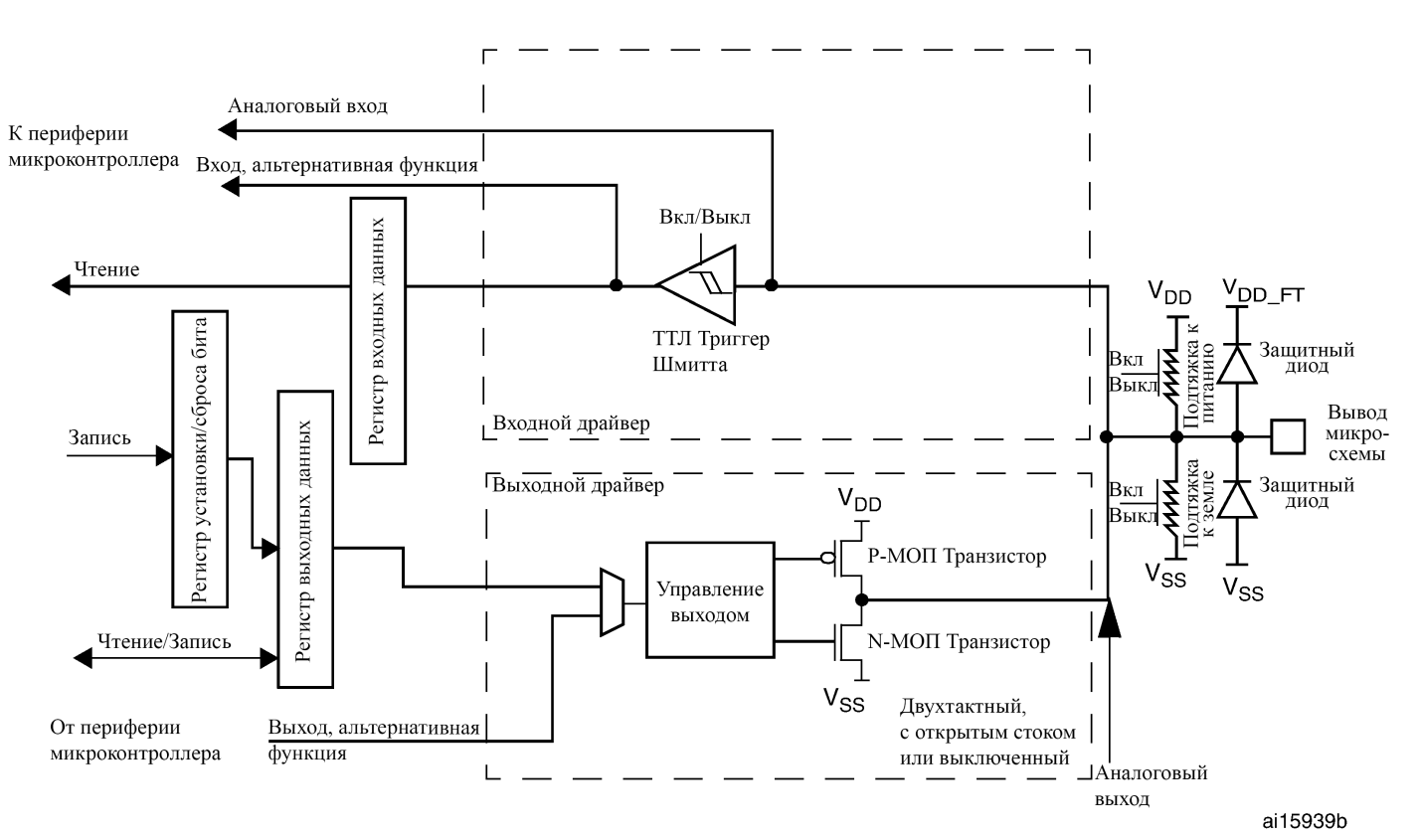


Рисунок 3 – Структурная схема интерфейса ввода/вывода общего назначения

в) Вход с подтяжкой к земле. Между входом и землей включается подтягивающий резистор (обычно 40 кОм), что позволяет находиться в низком состоянии, когда не приложено внешнее напряжение.

г) Аналоговый вход/выход. В этом случае вывод подключается к АЦП/ЦАП.

д) Выход с открытым стоком. Также возможно включать подтяжку к питанию/земле. В данном случае происходит управление только транзистора n-типа, когда транзистор p-типа закрыт. Это позволяет подключать вывод микросхемам с другим напряжением питания (2,5 В; 5 В и др.).

е) Двухтактный выход. Также возможно включать подтяжку к питанию/земле. Оба транзистора управляются, т.е. когда подаём на вывод 1-цу, то транзистор n-типа закрывается, а транзистор p-типа открывается, тем самым подавая на выход микросхемы напряжение питания (чуть меньшее). Когда подаём на вывод 0, то транзистор p-типа открывается, а транзистор n-типа закрывается, тем самым подавая на выход микросхемы напряжение земля (чуть большее).

ж) Альтернативная функция. Переключает вывод в режим альтернативной функции. У каждого вывода своя альтернативная функция (ЦАП, UART, SPI, и.т.д.), значение которой нужно смотреть в документации к микроконтроллеру.

**1. Установка необходимого программного обеспечения**

Для программирования микроконтроллеров семейства STM32 необходимо, чтобы на персональном компьютере были установлены следующие программы:

– CooСox CoIDE 1.7.8.Это интегрированная среда разработки программного обеспечения для микроконтроллеров архитектуры ARM. Отличается скоростью и простотой установки, освоения и настройки решений. Скачать установочный файл можно с официального сайта <http://www.coocox.org/download/Tools/CoIDE-1.7.8.exe> , либо взять у преподавателя. Для установки запустите установочный файл CoIDE-1.7.8.exe. Далее следуйте стандартной установке. На рабочем столе появятся два ярлыка: CoCentral и CoIDE.

– GNU ARM Embedded Toolchain. Это компилятор с языка программирования Си и С++ для микроконтроллеров семейства Cortex-R/Cortex-M (У микроконтроллера STM32F407 ядро Cortex-M4). С его помощью из исходного кода создаётся сам исполнительный файл для микроконтроллера. Скачать GNU ARM Embedded Toolchain можно по ссылке <https://developer.arm.com/open-source/gnu-toolchain/gnu-rm/downloads> , либо взять у преподавателя. Установка стандартна и не должна вызвать вопросов.

– ST-LINK USB драйвера для Windows. Они нужны для работы с платой STM32F4DISCOVERY (Рисунок 4), чтобы можно было её прошивать и отлаживать. Скачать драйвера можно с официального сайта производителя <http://www.st.com> , либо взять у преподавателя.



Рисунок 4 – Исследовательская плата STM32F4DISCOVERY

**2. Запуск CooCox IDE 1.7.8 и создание пустого проекта**

Запустите CoIDE с рабочего стола. В центральной панели открытой среды нажмите Create a New Project (Рисунок 5). В появившемся окне введите имя проекта BlinkingLED, затем нажимайте кнопку «Next >» (Рисунок 6).

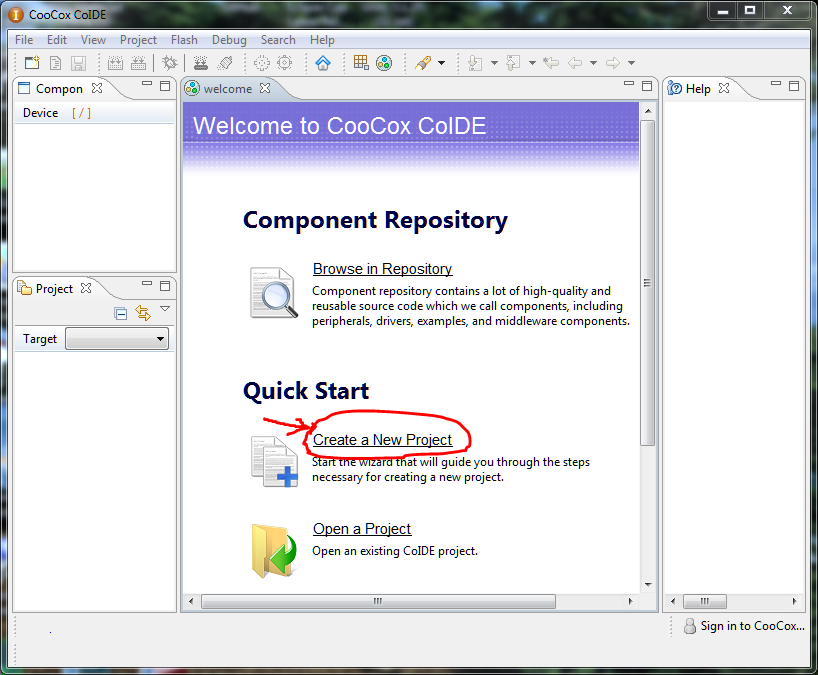


Рисунок 5 – Создание нового проекта

Далее необходимо выбрать какой проект создать: Board (для конкретной отладочной платы), либо Chip (просто под микроконтроллер (МК)). В данном случае необходимо выбрать Board, т.к. программировать будем отладочную плату STM32F4DISCOVERY. Нажимаем кнопку «Next >».

В появившемся окне вводим в строку поиска STM32F4DISCOVERY. В спске ниже выбираем STM32F4DISCOVERY (Рисунок 7). Нажимаем ктопку «Finish». Происходит генерация проекта.

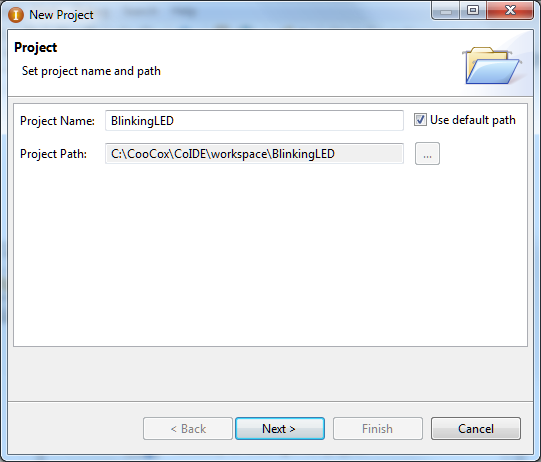


Рисунок 6 – Выбор имени проекта

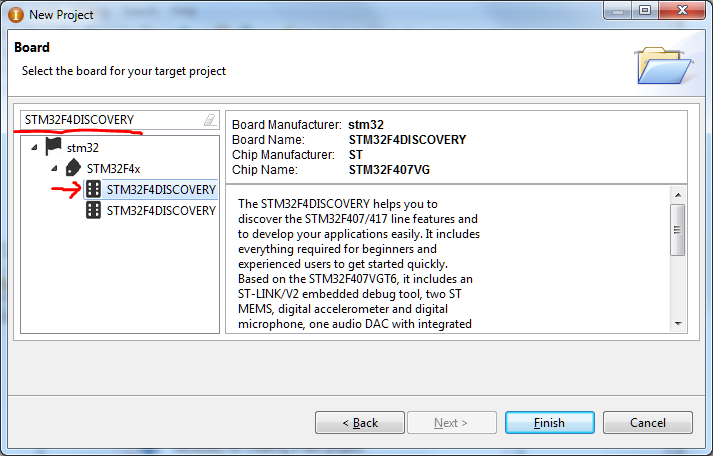


Рисунок 7 – Выбор отладочной платы

После загрузки должен появиться проект BlinkingLED с одним файлом main.c. В центральной панели будет высвечена вкладка Repository, позволяющая подключить стандартные компоненты для работы с микроконтроллером (Рисунок 8). В нижней вкладке выберите Peripherals, и добавьте следующие компоненты:

* M4 CMSIS Core. Этот компонент предоставляет стандартизированный интерфейс для работы с Cortex-M4;
* CMSIS BOOT. Производит начальную настройку микроконтроллера при запуске, и вызывает функцию main();

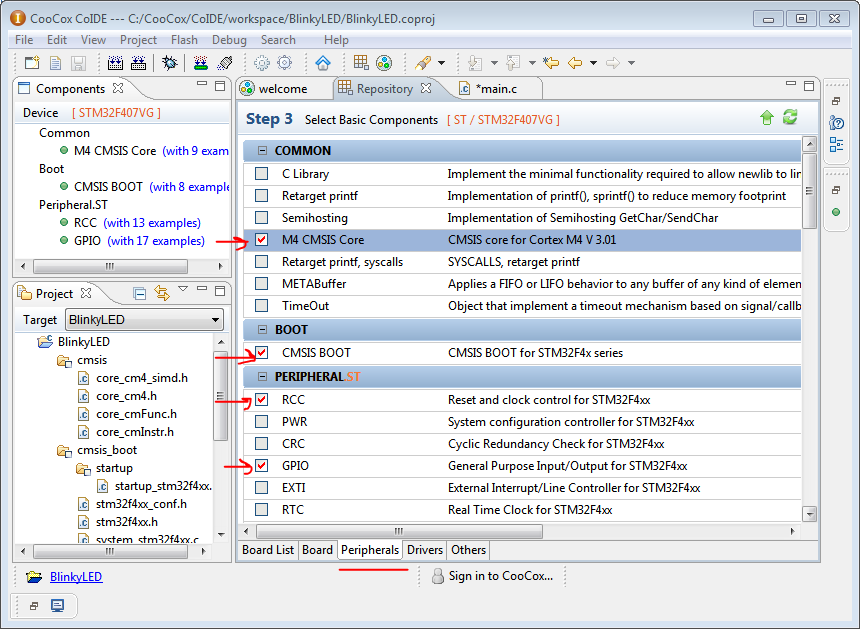


Рисунок 8 – Подключение периферии в проект

* RCC. Отвечает за сброс, настройку частоты тактирования микроконтроллера, а также за включение и настройку тактирования различных частей микроконтроллера;
* GPIO. Предоставляет функции и структуры для настройки и управления портами ввода вывода;

Библиотеки содержащие в названии CMSIS – стандартизированные для всех Cortex-M библиотеки абстракции, которые позволяют при написании кода думать на более высоком уровне (архитектуры в целом и имеющейся периферии).

Слева, в панели Components появились включённые компоненты. Если щелкнуть на какой либо компонент, при подключенном интернете, появятся примеры работы с данным компонентом. В правой нижней панели Project в проекте BlinkingLED откройте файл main.c (Рисунок 9). Это главный файл, в котором пока находится только пустая функция main, с которой начинается работа программы (точка входа в программу).

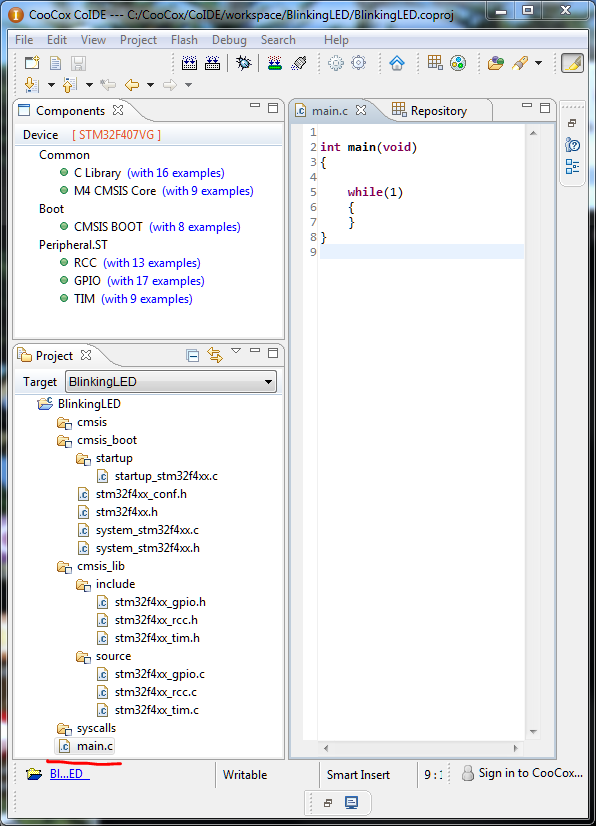


Рисунок 9 – Открытие файла исходного кода main.c

Можно попробовать собрать данный проект кнопкой  (Build) на верхней панели. При первой попытке выдастся предупреждение, что компилятор не настроен, и необходимо указать его месторасположение (Рисунок 10).

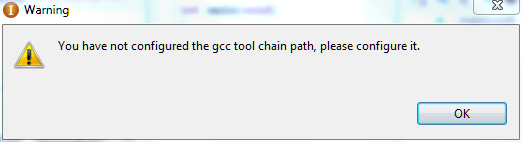


Рисунок 10 – Ошибка конфигурации компилятора

Нажимаем OK и в следующем окне нажимаем Browse для указания месторасположения файла компилятора (Рисунок 11). Если вы устанавливали GNU Tools for ARM Embedded Processor стандартно, то месторасположение будет приблизительно таким:

C:\Program Files\GNU Tools ARM Embedded\5.4 2016q2\bin\

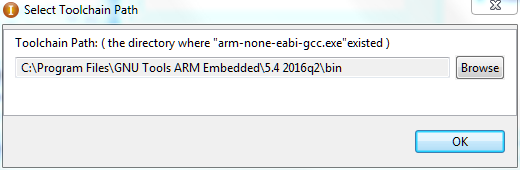


Рисунок 11 – Настройка расположения компилятора

После этого произойдёт построение проекта.

3. Написание программы

* 1. Подключение необходимых библиотек

Для того чтобы подключить библиотеки для работы в файл main.c, необходимо в самом начале файла необходимо написать:

**#include** "stm32f4xx.h"

**#include** "stm32f4xx\_gpio.h"

**#include** "stm32f4xx\_rcc.h"

stm32f4xx.h – это файл библиотеки CMSIS, позволяющий работать с семейством микроконтроллеров stm32f4xx.

stm32f4xx\_gpio.h – подключение компонента GPIO.

stm32f4xx\_rcc.h – подключение компонента RCC.

* 1. Настройка интерфейса ввода/вывода общего назначения

Для работы необходимо включить тактирование интересующего порта. Для этого необходимо в функции main написать строку:

RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOD, *ENABLE*);

где RCC\_AHB1PeriphClockCmd – это функция, принимающая два параметра:

* название блока, в котором необходимо включить тактирование.
* ENABLE или DISABLE, что означает включение или выключение тактирования.

В данном случае тактирование включается в блоке RCC\_AHB1Periph\_GPIOD, что означает порт ввода вывода D. Настраиваем именно порт D, потому что на плате stm32f4discovery именно к порту D подключены четыре пользовательских светодиода. Посмотреть схему платы можно в документации к плате stm32f4discovery (stm32f4Discovery\_UserManual.pdf) (Рисунок 12).

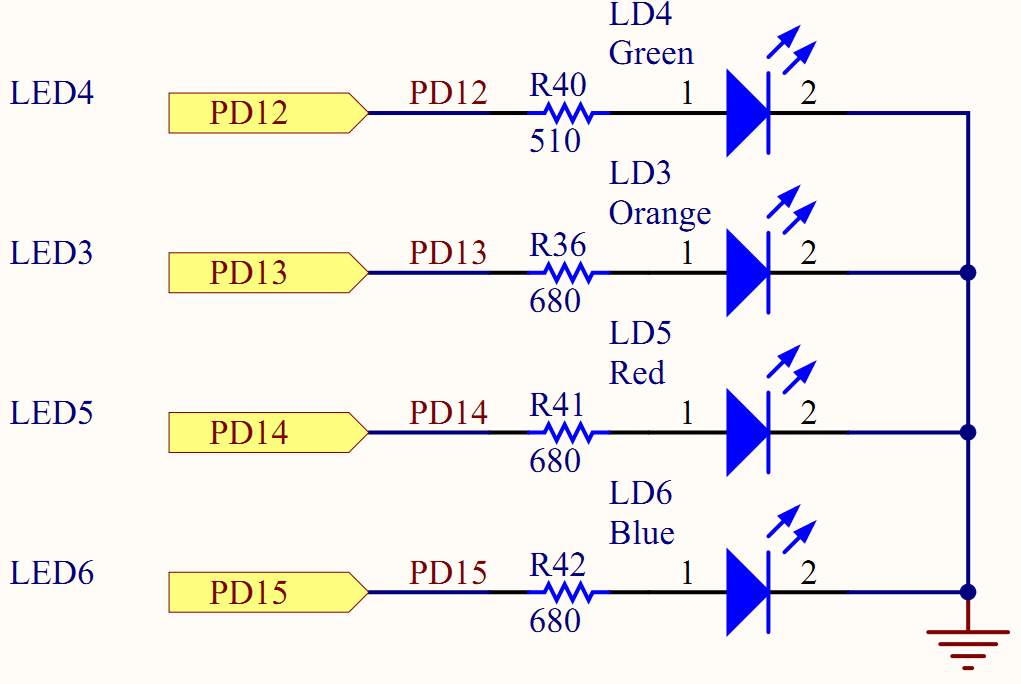


Рисунок 12 – Схема принципиальная подключения светодиодов на отладочной плате STM32F4DISCOVERY

Для настройки портов ввода вывода используется стандартная структура из библиотеки GPIO, c названием GPIO\_InitTypeDef. Для этого создаем переменную этой структуры:

GPIO\_InitTypeDef GPIO\_InitStruct;

Далее структуру нужно инициализировать значениями по умолчанию:

GPIO\_StructInit(&GPIO\_InitStruct);

Это необходимо для предотвращения инициализации структуры ошибочными значениями. В языке C при инициализации новой структуры под неё выделяется память, но никто не гарантирует, что эта память будет очищена. Так получается, что в этом участке памяти остались предыдущие значения совершенно других данных. Функция GPIO\_StructInit позволяет заполнить структуру стандартными значениями. Символ «&» перед названием структуры указывает на то, что мы передаем указатель на структуру.

В структуре нужно указать настройки порта, для этого пишем:

GPIO\_InitStruct.GPIO\_Pin = GPIO\_Pin\_13 | GPIO\_Pin\_14;

GPIO\_InitStruct.GPIO\_Mode = *GPIO\_Mode\_OUT*;

GPIO\_InitStruct.GPIO\_Speed = *GPIO\_Speed\_2MHz*;

GPIO\_InitStruct.GPIO\_OType = *GPIO\_OType\_PP*;

GPIO\_InitStruct.GPIO\_PuPd = *GPIO\_PuPd\_NOPULL*;

В параметре GPIO\_Pin указываем, какие выводы порта настраиваются (в данном примере выводы 13 и 14, что соответствует светодиодам LED3 и LED5 на схеме).

В параметре GPIO\_Mode указывается направление настраиваемых выводов. Возможные значения:

* GPIO\_Mode\_IN Вход (МК считывает данные)
* GPIO\_Mode\_OUT Выход (МК устанавливает данные)
* GPIO\_Mode\_AF Альтернативная функция (Интерфейсы и др.)
* GPIO\_Mode\_AN Аналоговый (ЦАП, АЦП)

В данном случае направление выводов GPIO\_Mode\_OUT, т.к. необходимо подавать сигналы на светодиоды.

В параметре GPIO\_Speed указываем скорость тактирования настраиваемых выводов. От скорости тактирования зависит энергопотребление (чем выше частота тактирование, тем выше энергопотребление) и максимальная частота считывания (либо записи) информации с вывода (на вывод). Возможные значения:

* GPIO\_Speed\_2MHz низкая скорость 2MHz;
* GPIO\_Speed\_25MHz средняя скорость 25MHz;
* GPIO\_Speed\_50MHz повышенная скорость 50MHz;
* GPIO\_Speed\_100MHz высокая скорость 100MHz.

В данном случае выбираем GPIO\_Speed\_2MHz, так как “мигать” светодиодом будем приблизительно с частотой 1-2Гц, так что 2 МГц более чем достаточно.

В параметре GPIO\_OType указываем тип вывода.

* GPIO\_OType\_PP Push-pull (двухтактный);
* GPIO\_OType\_OD Open drain (открытый сток).

В параметре GPIOx\_PUPDR задаем подтяжку.

* GPIO\_PuPd\_NOPULL Нет подтяжки;
* GPIO\_PuPd\_UP Подтяжка к питанию;
* GPIO\_PuPd\_DOWN Подтяжка к земле.

Последнее что необходимо сделать – произвести инициализацию строкой:

GPIO\_Init(GPIOD, &GPIO\_InitStruct);

Где GPIOD – имя порта;

GPIO\_InitStruct – название нашей структуры.

**3.3 Мигание светодиодом**

Для мигания используется две функции:

GPIO\_SetBits(GPIOD, GPIO\_Pin\_14); //Подать лог. единицу

GPIO\_ResetBits(GPIOD, GPIO\_Pin\_14); // Подать лог. нуль

Обе функции принимают два параметра:

* Название порта;
* Номера вывода (можно указывать несколько через знак “или” – “|”);

Если в бесконечном цикле попеременно вызывать эти две функции (подача логических 0 и 1 на вывод 14 порта D), и запустит программу, то светодиод будет постоянно тускло гореть, но не переключаться. Дело в том, то была указана частота записи битов в регистр выхода 2 МГц, значит с такой же частотой происходит переключение светодиода. Человеческий глаз способен различить переключение состояний не более сотни герц. Следовательно, необходимо производить переключение медленнее. Для этого выше функции main() создадим функцию задержки, самую простую, которая будет просто нагружать микроконтроллер работой (пускай и не полезной) необходимое время:

void Delay(volatile uint32\_t tick)

{

for(uint32\_t i = 0; i < tick; i++);

}

Теперь между переключениями можно вызывать функцию Delay с необходимой задержкой. Укажем в бесконечном цикле следующие строки:

**while**(1)

{

GPIO\_SetBits(GPIOD, GPIO\_Pin\_14);

Delay(500000);

GPIO\_ResetBits(GPIOD, GPIO\_Pin\_14);

Delay(500000);

}

Это значит, что микроконтроллер будет включает светодиод, зам 500000 раз считать переменную i (задержка), затем выключает светодиод, опять задержка, а затем начинает заново. Визуально происходит мигание светодиода.

**Индивидуальные задания:**

1. Мигать попеременно красным и зелёным светодиодами. Время горения светодиода равна времени простоя.
2. Мигать синим светодиодом: попеременно быстро и медленно. Время горения светодиода равна времени простоя.
3. Мигать красным светодиодом медленно, зелёным светодиодом в два раза быстрее. Время горения светодиода равна времени простоя.
4. Зажигать и гасить по очереди все четыре светодиода.
5. Зелёный и красный светодиод горит попеременно с желтым светодиодом. Желтый горит быстро, зелёный и красный медленно.
6. Два любых светодиода горят постоянно, оставшиеся два горят попеременно.
7. Мигать синим и зеленым светодиодом. Время горения светодиода в два раза больше чем время простоя. Красный светодиод горит постоянно.