Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет информационных технологий и управления

Кафедра теоретических основ электротехники

ВСТРОЕННЫЕ МИКРОПРОЦЕССОРНЫЕ СИСТЕМЫ. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве пособия для специальностей 1-36 04 01 «Программно-управляемые электронно-оптические системы», 1-36 04 02 «Промышленная электроника», 1-39 02 02 «Проектирование и производство программно-управляемых электронных средств», 1-39 02 03 «Медицинская электроника»

Минск БГУИР 2019

УДК 004.31-022.53(076.5)
ББК 32.971.32-04я73
В85

Авторы:

М. В. Давыдов, С. И. Городко, Н. С. Давыдова, П. В. Камлач, О. П. Высоцкий

Рецензенты:

кафедра электротехники и систем электропитания учреждения образования
«Военная академия Республики Беларусь»
(протокол №6 от 19.02.2018);

доцент кафедры информатики Белорусского государственного университета
кандидат физико-математических наук,
доцент С. А. Лысенко

Встроенные микропроцессорные системы. Лабораторный практикум : пособие / М. В. Давыдов [и др.]. – Минск : БГУИР, 2019. – 68 с.
ISBN 978-985-543-485-7.

Рассматриваются основы разработки программного обеспечения для микроконтроллеров на языке программирования С. Представлены четыре лабораторные работы по изучению внутренней структуры микропроцессоров и особенностей программирования на языке С, регистрации аналоговых сигналов посредством аналого-цифрового преобразователя. Приведена работа микроконтроллера с универсальным асинхронным приёмопередатчиком USART.

Предназначено для закрепления и углубления теоретических знаний, приобретения практических навыков работы в области микропроцессорного управления и создания программ для микропроцессорной техники.

УДК 004.31-022.53(076.5)
ББК 32.971.32-04я73

ISBN 978-985-543-485-7

Содержание

Лабораторная работа №1
Создание базового проекта для микроконтроллера STM32F407. Порты ввода/вывода

Цель работы: сформировать общее представление о разработке программного обеспечения для микроконтроллеров серии STM32; создать проект и научиться управлять портами ввода/вывода микроконтроллера, а также прерываниями микроконтроллера.

1.1 Теоретические сведения

Микроконтроллеры семейства STM32 позволяют производить настройку почти каждого вывода микросхемы. Микроконтроллер в корпусе LQFP100 показан на рисунке 1.1.

Рисунок 1.1 – Микроконтроллер STM32F407VET6

Схематично микросхема STM32F407VET6 показана на рисунке 1.2. Каждый вывод подписан определённым названием. Есть несколько выводов для подключения питания, внешнего резонатора, земли и др. Оставшиеся выводы являются портами ввода/вывода микроконтроллера. В данном случае порт означает объединение 16 выводов. Порты в микроконтроллерах STM32 называются буквами латинского алфавита (A, B, C…). На рисунке 1.2 такие выводы обозначаются буквой P, затем – буквой порта и в конце – номером вывода (от 0 до 15).

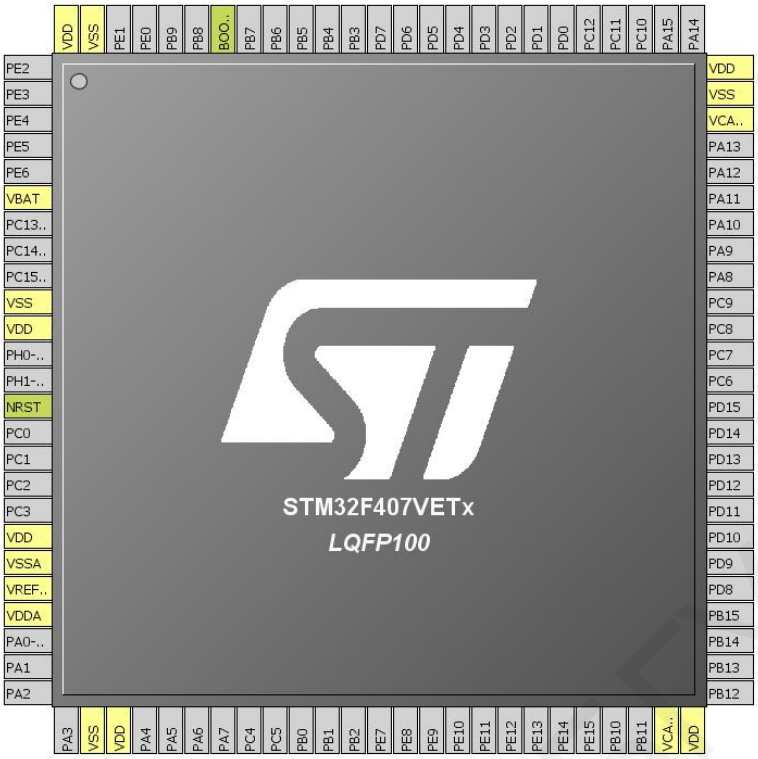

Рисунок 1.2 – Схематичное изображение
микроконтроллера STM32F407VET6

Для предоставления максимальной гибкости работы каждый вывод общего назначения может быть индивидуально настроен. У самого выхода находятся два защитных диода, защищающие микроконтроллер при подаче на вывод микросхемы напряжения ниже земли (например, –3,3 В) или выше напряжения питания микроконтроллера (например, +6 В) (рисунок 1.3).

Существуют следующие режимы работы вывода общего назначения:

1. 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 находятся в закрытом состоянии, поэтому вывод не подключён ни к земле, ни к питанию и, следовательно, ведёт себя как неподключённый к схеме. Данный режим используется для приема данных, когда логическая единица (3,3 В) или логический нуль (0 В) формируются внешними схемами.

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

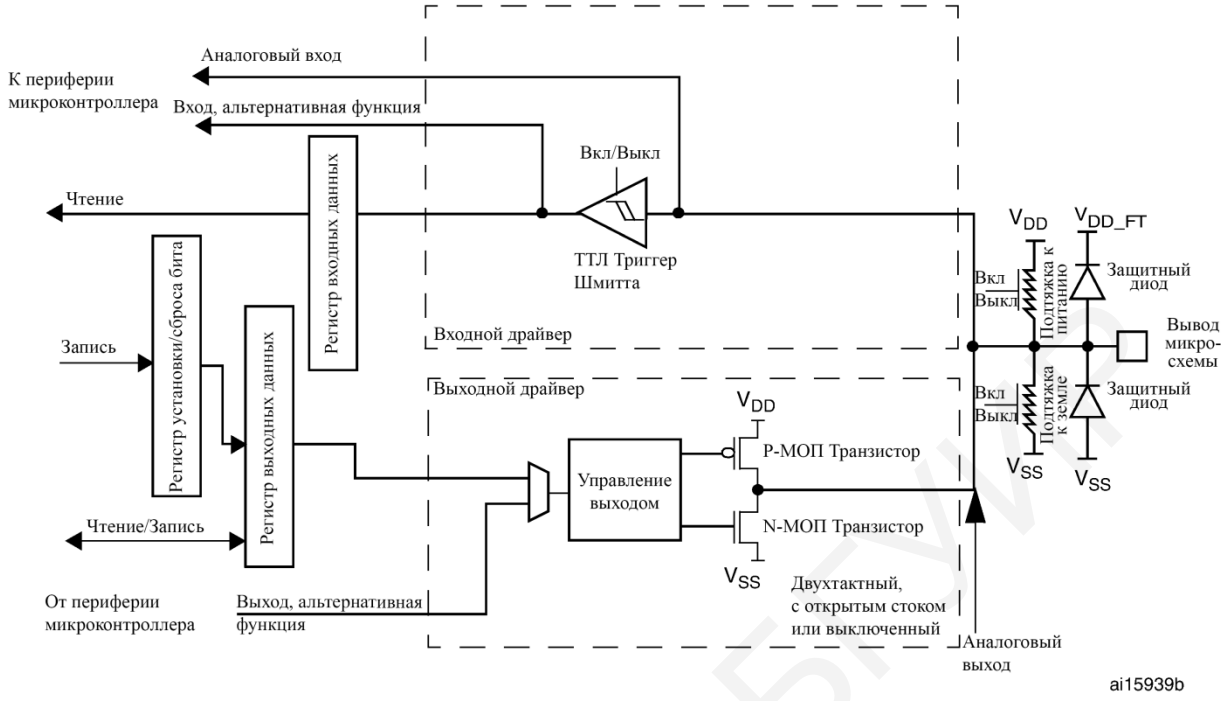

Рисунок 1.3 – Структурная схема интерфейса ввода/вывода общего назначения

3. 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.

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

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

6. Двухтактный выход. Также возможно включать подтяжку к питанию/земле. Происходит управление обоими транзисторами, т. е. когда подаём на вывод единицу, то транзистор n-типа закрывается, а транзистор p-типа открывается, тем самым подавая на выход микросхемы напряжение питания (чуть меньшее). Когда подаём на вывод нуль, то транзистор p-типа открывается, а транзистор n-типа закрывается, тем самым подавая на выход микросхемы напряжение земли (чуть большее).

7. Альтернативная функция. Вывод переключается в режим альтернативной функции. У каждого вывода свой набор альтернативных функций (ЦАП, UART, SPI и т. д.), значение которых нужно смотреть в документации к микроконтроллеру.

1.2 Порядок выполнения работы

1.2.1 Установка необходимого программного обеспечения

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

– CooCox CoIDE 1.7.8. Это интегрированная среда разработки программного обеспечения для микроконтроллеров архитектуры ARM. Отличается скоростью и простотой установки, освоения и настройки решений. Установочный файл можно взять у преподавателя либо скачать с официального сайта http://www.coocox.org/download/Tools/CoIDE-1.7.8.exe. 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 C и C++ 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 получить у преподавателя. Установка стандартна и не должна вызвать вопросов;

– ST-LINK USB драйвера для Windows. Они нужны для работы с платой STM32F4DISCOVERY (рисунок 1.4), чтобы можно было её прошивать и отлаживать. Скачать драйвера можно с официального сайта производителя http://www.st.com либо получить у преподавателя.

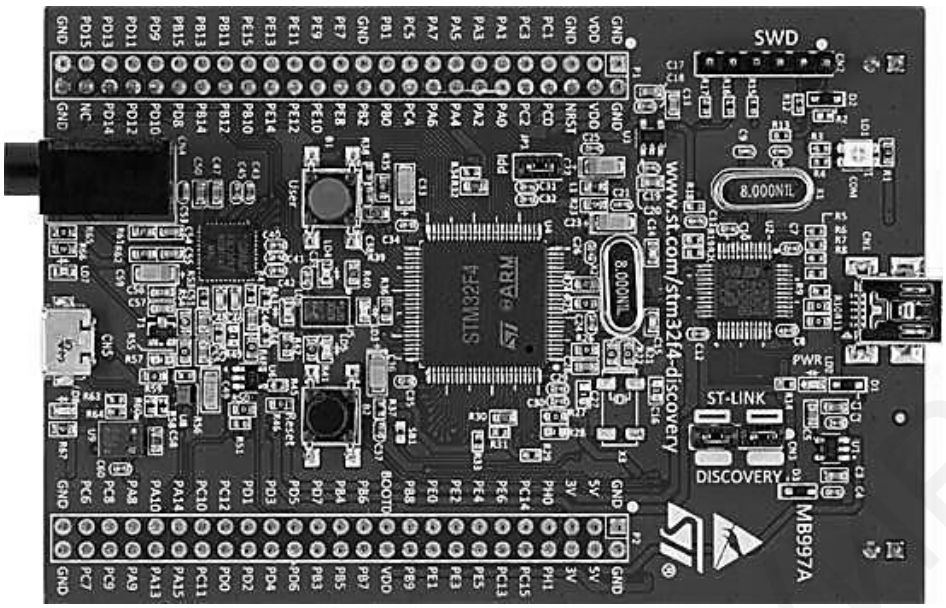

Рисунок 1.4 – Исследовательская плата STM32F4DISCOVERY

1.2.2 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 1.5). В появившемся окне введите имя проекта (например, BlinkingLED), затем нажимайте кнопку Next > (рисунок 1.6).

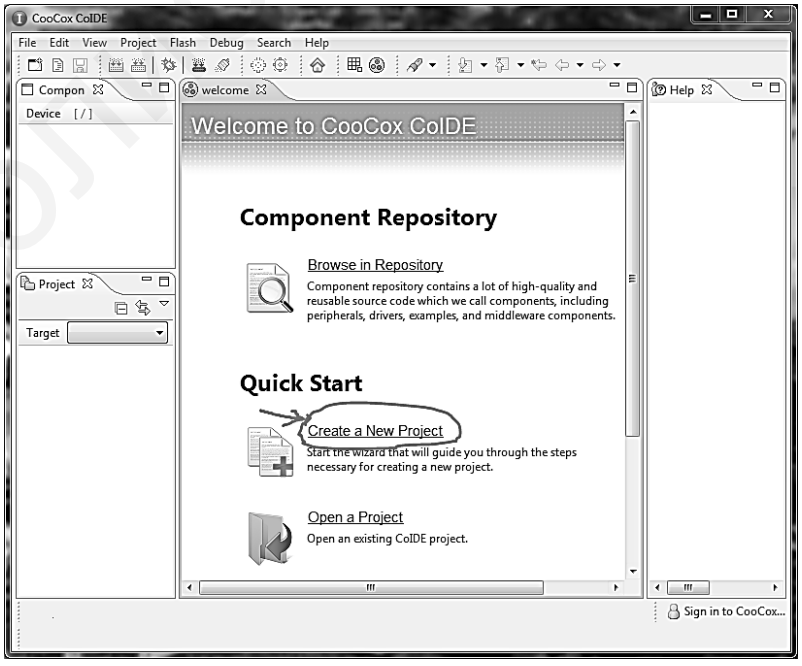

Рисунок 1.5 – Создание нового проекта

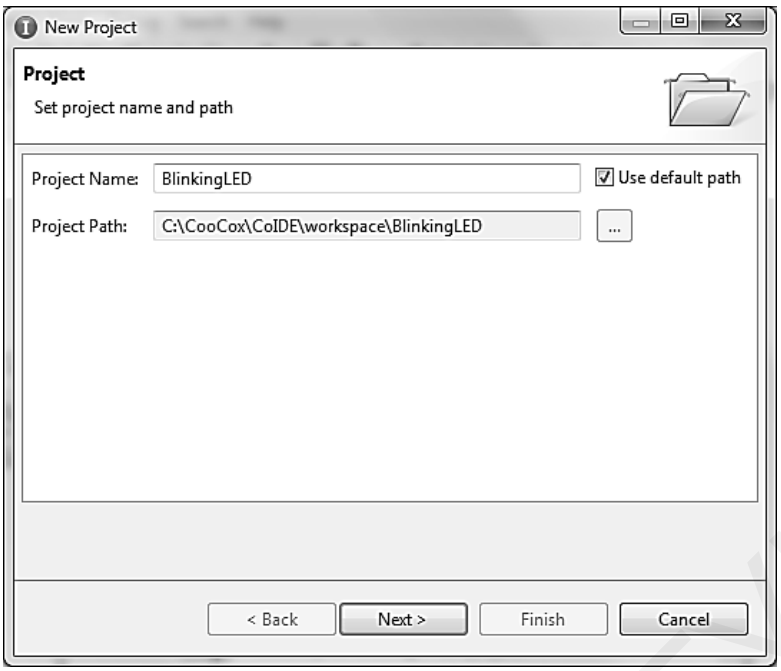

Рисунок 1.6 – Выбор имени проекта

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 списке ниже выбираем STM32F4DISCOVERY (рисунок 1.7). Нажимаем кнопку Finish. Происходит генерация проекта.

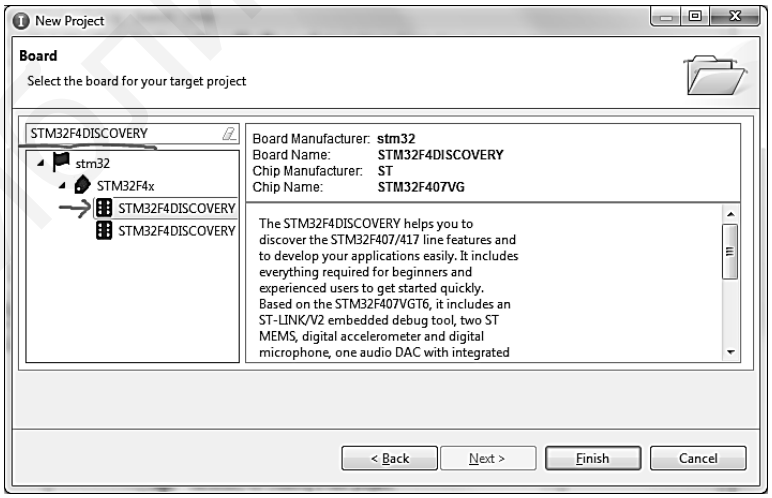

Рисунок 1.7 – Выбор отладочной платы

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 1.8). 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()`;

– 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.

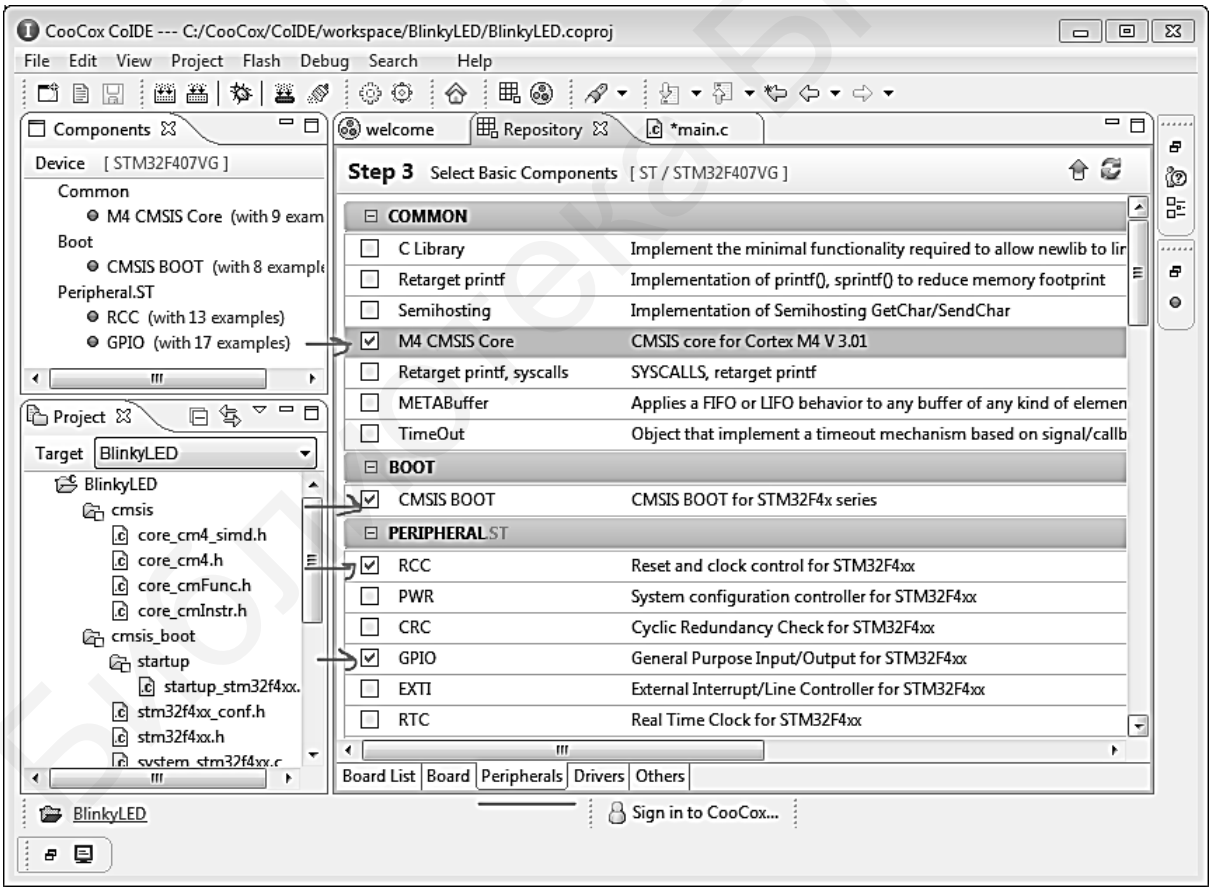

Рисунок 1.8 – Подключение периферии в проект

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 на панели Components появились включённые компоненты. Если щёлкнуть на каком-либо компоненте при подключенном интернете, появятся примеры работы с данным компонентом. В правой нижней панели Project в проекте BlinkingLED откройте файл main.c (рисунок 1.9). Это главный файл, в котором пока находится только пустая функция main, с которой начинается работа программы (точка входа в программу).

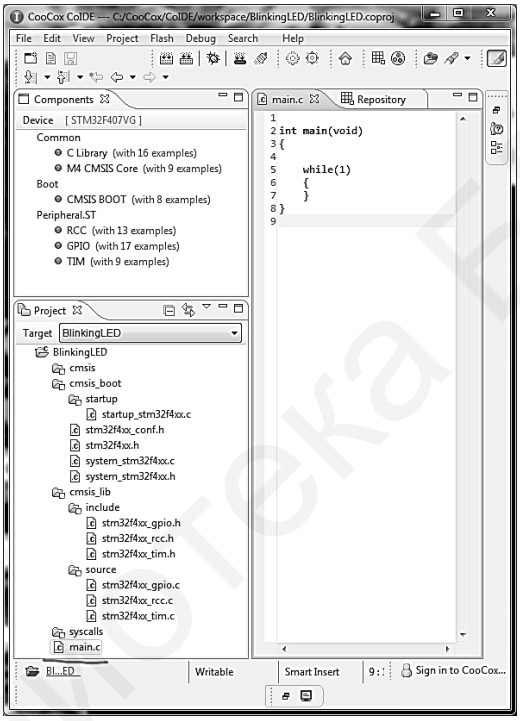

Рисунок 1.9 – Открытие файла исходного кода main.c

Можно попробовать собрать данный проект кнопкой  (Build) на верхней панели. При первой попытке выдаётся предупреждение, что компилятор не настроен и необходимо указать его месторасположение (рисунок 1.10).

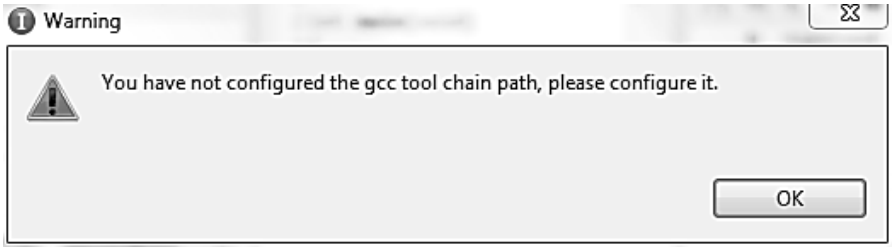

Рисунок 1.10 – Ошибка конфигурации компилятора

Нажимаем OK и в следующем окне – Browse (для указания месторасположения файла компилятора) (рисунок 1.11). 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\
```

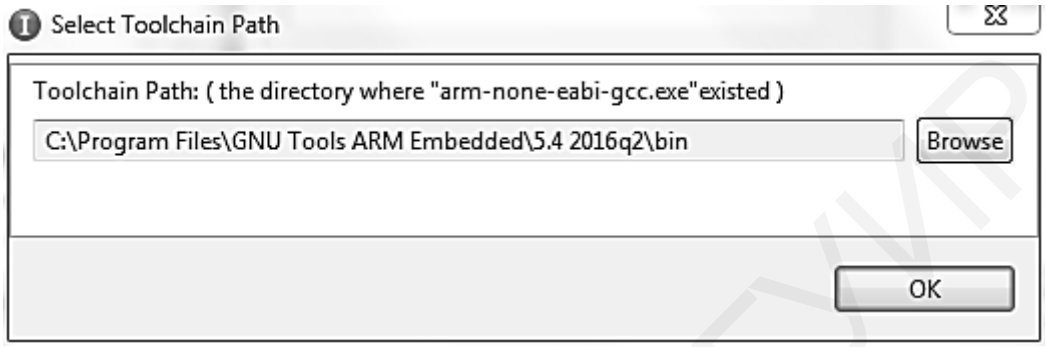

Рисунок 1.11 – Настройка расположения компилятора

После этого произойдёт построение проекта.

1.2.3 Написание программы

1.2.3.1 Подключение необходимых библиотек

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

```
#include "stm32f4xx.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
```

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

`stm32f4xx_gpio.h` – подключение компонента GPIO;

`stm32f4xx_rcc.h` – подключение компонента RCC.

1.2.3.2 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 1.12).

Для настройки портов ввода/вывода используется стандартная структура из библиотеки GPIO с названием `GPIO_InitTypeDef`. Для этого создаём переменную этой структуры

```
GPIO_InitTypeDef GPIO_InitStruct;
```

Далее структуру нужно инициализировать значениями по умолчанию:

```
GPIO_StructInit(&GPIO_InitStruct);
```

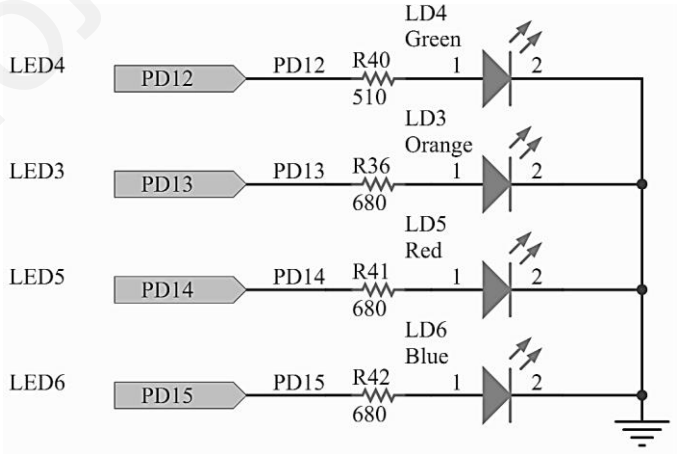

Рисунок 1.12 – Схема принципиальная подключения светодиодов на отладочной плате STM32F4DISCOVERY

Это необходимо для предотвращения инициализации структуры ошибочными значениями. В языке С 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` – выход (МК устанавливает данные);

– `GPIO_Mode_OUT` – вход (МК считывает 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 тактирования, тем выше энергопотребление) и максимальная частота считывания (либо записи) информации с вывода (на вывод). Возможные значения:

– `GPIO_Speed_2MHz` – низкая скорость (2 МГц);

– `GPIO_Speed_25MHz` – средняя скорость (25 МГц);

– `GPIO_Speed_50MHz` – повышенная скорость (50 МГц);

– `GPIO_Speed_100MHz` – высокая скорость (100 МГц).

В данном случае выбираем `GPIO_Speed_2MHz`, т. к. «мигать» светодиодом будем приблизительно с частотой 1–2 Гц, так что 2 МГц – более чем достаточно.

В параметре GPIO_OType указываем тип вывода:

– GPIO_OType_PP, где Push-pull – двухтактный;

– GPIO_OType_OD, где 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.

1.2.3.3 Мигание светодиодом

Для мигания используются две функции:

```
GPIO_SetBits(GPIOD, GPIO_Pin_14); //подать логическую единицу;
GPIO_ResetBits(GPIOD, GPIO_Pin_14); // подать логический 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 нуля и единицы на вывод 14 порта D) и запустить программу, то светодиод будет постоянно тускло гореть, но не переключаться. Дело в том, что 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 неполезной) 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 включает светодиод, затем 500 тыс. раз инкрементирует переменную `i` (таким образом формируется задержка), потом выключает светодиод, и далее снова производится задержка. После выполнения данных действий цикл начинается заново. Визуально наблюдаем мигание светодиода.

1.2.3.4 Обработка внешних прерываний

1.2.3.4.1 Настройка библиотек

В проекте нужны следующие компоненты (как включаются компоненты, смотрите в лабораторной работе №2):

– M4 CMSIS Core;
– CMSIS BOOT;
– RCC;
– GPIO;
– EXTI (нужен для настройки внешних прерываний);
– MISC (нужен для настройки и обработки прерываний);
– SYSCFG.

В файле `main.c` подключаем заголовочные файлы:

```
#include "stm32f4xx.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_syscfg.h"
#include "stm32f4xx_exti.h"
#include "misc.h"
```

1.2.3.4.2 Инициализация портов ввода/вывода для работы со светодиодами и кнопкой

Для работы кнопки необходимо произвести инициализацию GPIO: инициализировать кнопку, расположенную на выводе PA0 (рисунок 1.13).

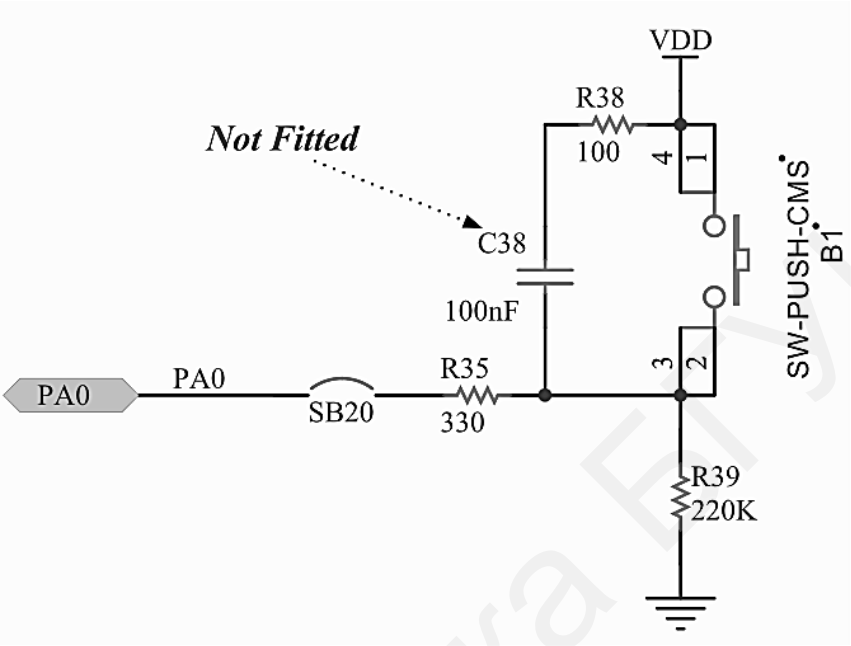

Рисунок 1.13 – Схема кнопки на отладочной плате
(всю схему смотрите в документации к плате
stm32f4Discovery_UserManual.pdf)

1.2.3.4.3 Настройка внешних прерываний EXTI при нажатии кнопки на PA0

Прерывание (от англ. interrupt) – сигнал, сообщающий процессору о наступлении какого-либо события. При этом выполнение текущей последовательности команд приостанавливается и управление передаётся обработчику прерывания, который реагирует на событие и обслуживает его, после чего возвращает управление в прерванный код.

Прерывание – это событие, как правило, связанное с каким-либо блоком периферии микроконтроллера STM32. Событий, которые могут породить прерывание, – множество. Например, если идёт речь о таком блоке периферии, как USART, то там могут быть такие события: «передача завершена», «приём завершен», «возникла ошибка чётности» и т. д. Использование прерываний позволит нашей программе мгновенно реагировать на подобные

события. Сам термин «прерывание» говорит о том, что что-то должно прерваться, и в данном случае прервется выполнение основного кода вашей программы и управление будет передано некоторому другому отрезку кода, который называется *обработчиком прерывания*. Таких обработчиков достаточно много, т. к. периферийных устройств в STM32 предостаточно. Стоит отметить важный момент: в случае возникновения двух разных прерываний от одного блока периферии возникает одно и то же прерывание. Например, если произойдет прерывание по приёму байта через USART и прерывание по завершении передачи через тот же USART, то в обоих случаях будет вызван один и тот же обработчик. Для того чтобы определить, какое из возможных прерываний произошло, нужно смотреть на флаги состояния, и, само собой, очищать их перед выходом из прерывания. Когда обработчик прерывания отработает, управление будет передано той самой строчке кода, во время выполнения которой наступило прерывание. То есть основная программа продолжит работать дальше как ни в чем не бывало.

Важная информация о прерываниях:

– все прерывания независимо включаются/выключаются;

– прерывания имеют приоритет;

– прерывания могут быть вызваны программно;

– если для прерывания нет обработчика, а оно возникло, то будет вызван обработчик по умолчанию.

Так как прерывание у нас внешнее (с кнопки), то перед включением прерываний необходимо настроить источник внешних прерываний. Для этого нужно включить тактирование модуля SYSCFG, которым и будет производиться настройка:

```
RCC_AHB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
```

Далее указываем источник внешних прерываний. У STM32 за внешние прерывания отвечает EXTI-контроллер. Его основные возможности:

– до 20 линий прерываний (в реальности несколько меньше, зависит от контроллера);

– независимая работа со всеми линиями – каждой линии присвоен собственный статусный бит в специальном регистре;

– улавливает импульсы, длительность которых ниже (меньше) периода частоты APB2.

Сама структура связи EXTI показана на рисунке 1.14.

То есть имеется шестнадцать EXTI-линий, к которым мы можем подключать выводы порта. Причём группируются они по номерам выводов. Значит, мы не можем на разные прерывания навесить, например, PA0 и PB0.

Для выбора порта и номера вывода, изменение сигнала на которых будет вызывать прерывания, запишем

```
SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0);
```

где `EXTI_PortSourceGPIOA` – порт, к которому подключена кнопка;
`EXTI_PinSource0` – номер вывода, к которому подключена кнопка.

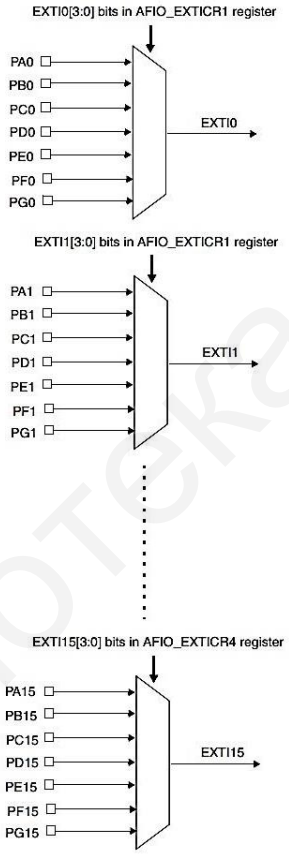

Рисунок 1.14 – Структура связи внешних прерываний EXTI

Дальше инициализируем структуру:

```
EXTI_InitTypeDef EXTI_InitStruct;
EXTI_InitStruct.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStruct.EXTI_Trigger = EXTI_Trigger_Rising;
EXTI_InitStruct.EXTI_Line = EXTI_Line0;
EXTI_InitStruct.EXTI_LineCmd = ENABLE;
EXTI_Init(&EXTI_InitStruct);
```

Разберём некоторые параметры:

– `EXTI_Mode` – режим работы; может принимать следующие значения:

• `EXTI_Mode_Interrupt`. Прерывания – это когда происходит переход на обработчик прерываний;

• `EXTI_Mode_Event` – события, когда обработчик не вызывается, а просто поднимается флажок. Может разбудить процессор или включить какую-нибудь периферию, например АЦП;

– `EXTI_Trigger` – триггер реакции. Выбираем на какой фронт реагировать. Варианты:

`EXTI_Trigger_Rising` – фронт подъема (при нажатии кнопки);
`EXTI_Trigger_Falling` – фронт спада (при отпускании кнопки);
`EXTI_Trigger_Rising_Falling` – фронт подъема и спада (и то и другое).

– `EXTI_Line` – указываем номер линии внешних прерываний. Может принимать значения от `EXTI_Line0` до `EXTI_Line15`, если это прерывания с `GPIO`, и от `EXTI_Line16` до `EXTI_Line22`, если они с другой переферии (например, USB WakeUp).

1.2.3.4.4 Настройка приоритета и источника прерываний с помощью NVIC, запуск прерываний

Для управления прерываниями существует специальный модуль NVIC.

NVIC – это контроллер вложенных векторизированных прерываний STM32 (Nested vectored interrupt controller). В зависимости от модели контроллера он может осуществлять обслуживание до 68 источников прерываний IRQ от перефирийных устройств и выполняет следующие функции:

– разрешение и запрет прерывания;

– назначение и изменение приоритета прерывания от 0 (максимальный приоритет) до 15 (минимальный);

– автоматическое сохранение данных при выполнении одиночного или вложенных прерываний.

Реализуем вышеперечисленные функции:

```
NVIC_InitTypeDef NVIC_InitStruct;
NVIC_InitStruct.NVIC_IRQChannel = EXTI0_IRQn;
NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x01;
NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x01;
```

```
NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStruct);
```

Разберём некоторые параметры:

– NVIC_IRQChannel – указывается источник прерываний. Всего их более 90, посмотреть их можно в документации или в заголовочном файле stm32f4xx.h;

– IRQChannelPreemptionPriority – приоритет прерывания (чем меньше значение данного параметра, тем выше приоритет прерывания). Данный параметр позволяет определить очередность выполнения обработчиков прерываний;

– IRQChannelSubPriority – подприоритет прерывания, позволяющий определить очередность выполнения обработчиков прерываний в пределах одинакового приоритета.

1.2.3.4.5 Обработка прерываний

Для обработки прерываний существуют специальные функции, объявления которых находятся в файле startap_stm32f4xx.c. Нас интересует обработчик с именем EXTI0_IRQHandler. Если создать определение функции с данным именем, то, как только происходит прерывание, программа заходит в данную функцию. На одни и те же обработчики прерываний могут приходить до шестнадцати (например, USART) разных прерываний. Следовательно, при входе в обработчик прерываний необходимо проверить, какое именно произошло прерывание; после обработки прерывания необходимо сбросить флаг прерывания. Если флаг прерывания не будет сброшен, то после выхода из функции обработки прерывания произойдет новый вызов функции обработки прерывания и т. д. Таким образом, выполнение программы будет нарушено:

```
void EXTI0_IRQHandler(void){
  if(EXTI_GetITStatus(EXTI_Line0) != RESET){
    Delay(10000);
    if(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0) == 1){
      GPIO_ToggleBits(GPIOD, GPIO_Pin_13);
    }
    EXTI_ClearITPendingBit(EXTI_Line0);
  }
}
```

В данном случае строкой if (EXTI_GetITStatus(EXTI_Line0) != RESET) мы проверяем, внешнее ли это прерывание линии 0. Если да, то делаем задержку функцией Delay:

```
void Delay(u32 ticks){
  while(ticks != 0) ticks--;
}
```

Это необходимо, чтобы предотвратить множественное срабатывание от дребезга контактов кнопки (проверьте выполнение программы без защиты от дребезга и с ней). В данном случае используется самый простой способ защиты – выжидание некоторого фиксированного времени, пока переходный процесс дребезга не закончится. Если по истечении этого времени кнопка нажата, то производится изменение режима мигания диода в соответствии с таблицей индивидуальных заданий.

Вызывая функцию `EXTI_ClearITPendingBit(EXTI_Line0)`, мы сбрасываем флаг прерывания от внешнего источника линии 0. Без этого не произойдёт дальнейших прерываний. Это позволяет предотвратить возникновения прерывания `EXTI_Line0` во время обработки предыдущего прерывания.

Проверьте работу прерываний в режиме отладки программы, выбрав пункт Debug → Debug. При этом запустится режим отладки, и можно в прерывании поставить точку останова.

1.3 Индивидуальные задания

Индивидуальные задания к лабораторной работе №1 представлены в таблице 1.1.

Таблица 1.1 – Индивидуальные задания к лабораторной работе №1

Вариант	Задания
1	2
1	Мигать синим светодиодом с переменной: F1 = 0,5 Гц, F2 = 2 Гц. Скважность равна 2. При нажатии кнопки происходит смена частоты. Повторное нажатие кнопки возвращает в первоначальное состояние
2	Мигать попеременно красным и зелёным светодиодами. F = 3 Гц. Скважность равна 2. При нажатии кнопки скважность равна 4. Повторное нажатие кнопки возвращает в первоначальное состояние

Продолжение таблицы 1.1

1	2
3	Мигать красным светодиодом с F = 0,25 Гц, зелёным светодиодом – с F = 1 Гц. Скважность равна 4. При нажатии кнопки частота увеличивается в 2 раза. Повторное нажатие кнопки возвращает в первоначальное состояние
4	Зажигать и гасить по очереди все четыре светодиода, F = 1 Гц. При нажатии кнопки порядок мигания светодиодов меняется на противоположный. Повторное нажатие кнопки возвращает в первоначальное состояние
5	Зелёный и красный светодиоды горят попеременно с желтым светодиодом. F = 1 Гц. Скважность равна 5. При нажатии кнопки частота F = 5 Гц. Повторное нажатие кнопки возвращает в первоначальное состояние
6	Два любых светодиода горят постоянно, оставшиеся два горят попеременно. F = 3 Гц. Скважность равна 2. При нажатии кнопки мигающие светодиоды начинают постоянно гореть, постоянно горевшие светодиоды начинают мигать. Повторное нажатие кнопки возвращает в первоначальное состояние
7	Мигать синим и зеленым светодиодом. F = 1 Гц. Скважность равна 4. Красный светодиод горит постоянно. При нажатии кнопки мигающие светодиоды начинают постоянно гореть, постоянно горевшие светодиоды начинают мерцать. Повторное нажатие кнопки возвращает в первоначальное состояние
8	Мигать попеременно красным и зелёным светодиодами. F = 5 Гц. Скважность равна 3. При нажатии кнопки F = 2 Гц, скважность равна 2. Повторное нажатие кнопки возвращает в первоначальное состояние
9	Мигать синим светодиодом с F = 5 Гц, зелёным светодиодом с F = 1 Гц. Скважность равна 4. При нажатии кнопки начинают мигать красный и желтый светодиоды (параметры те же). Повторное нажатие кнопки возвращает в первоначальное состояние
10	Зажигать и гасить по очереди все четыре светодиода. F1 = 1 Гц, F2 = 2 Гц, F3 = 3 Гц, F4 = 4 Гц. При нажатии кнопки F1 = 4 Гц, F2 = 3 Гц, F3 = 2 Гц, F4 = 1 Гц. Повторное нажатие кнопки возвращает в первоначальное состояние

Продолжение таблицы 1.1

1	2
11	Мигать синим светодиодом с переменной частотой: F1 = 0,5 Гц, F2 = 2 Гц, F3 = 3 Гц, F4 = 4 Гц. Скважность равна 2. При нажатии кнопки происходит циклическая смена частоты
12	Мигать красным светодиодом с переменной частотой: F1 = 5 Гц, F2 = 4 Гц, F3 = 3 Гц, F4 = 2 Гц. Скважность равна 4. При нажатии кнопки происходит циклическая смена частоты
13	Мигать синим светодиодом с переменной частотой: F1 = 0,5 Гц, F2 = 1 Гц, F3 = 2 Гц, F4 = 4 Гц. Скважность равна 0,5. При нажатии кнопки происходит циклическая смена частоты
14	Первоначально светодиоды не горят. При нажатии кнопки зажигать по очереди все четыре светодиода с переменной частотой от 40 до 1 Гц с шагом 0,5 Гц
15	Первоначально светодиоды горят. При нажатии кнопки гасить по очереди все четыре светодиода с переменной частотой от 20 до 1 Гц с шагом 0,25 Гц

1.4 Контрольные вопросы

1. Каковы области применения микроконтроллеров серии STM32?
2. В каких режимах могут работать порты ввода/вывода микроконтроллеров серии STM32?
3. Каковы особенности работы в режиме GPIO_Mode_IN?
4. Каковы особенности работы в режиме GPIO_Mode_OUT?
5. Каковы особенности работы в режиме GPIO_Mode_AF?
6. Каковы особенности работы в режиме GPIO_Mode_AN?
7. В чем отличие типов выходного сигнала GPIO_OType_PP и GPIO_OType_OD?
8. Что такое прерывание и как это используется в проектировании программного обеспечения микроконтроллеров серии STM32?
9. Каков алгоритм настройки прерывания?

Лабораторная работа №2
Таймеры. Настройка тактирования микроконтроллера STM32F407

Цель работы: сформировать общее представление о работе таймеров микроконтроллера; изучить особенности формирования тактирующих сигналов микроконтроллера.

2.1 Теоретические сведения

Для любого микроконтроллера таймер является одним из важнейших узлов, который позволяет точно отсчитывать интервалы времени, считать импульсы, поступающие на входы, генерировать внутренние прерывания, формировать сигналы с широтно-импульсной модуляцией (ШИМ) и поддерживать процессы прямого доступа к памяти (ПДП).

Микроконтроллер STM32 [1] имеет в своём составе несколько типов таймеров, отличающихся друг от друга по функциональному назначению. Первый тип таймеров является самым простым и представляет собой базовые таймеры (Basic Timers). К данному типу принадлежат таймеры TIM6 и TIM7. Эти таймеры просто настраиваются и управляются при помощи минимума регистров. Они способны отсчитывать интервалы времени и генерировать прерывания при достижении таймером заданного значения.

Второй тип представляет собой таймеры общего назначения (General-Purpose Timers). К нему относятся таймеры с TIM2 по TIM5 и таймеры с TIM9 по TIM14. Они могут генерировать ШИМ, считать импульсы, поступающие на определённые выводы микроконтроллера, обрабатывать сигналы от энкодера и т. п.

Третий тип определяет таймеры с развитым управлением (Advanced-Control Timer). К этому типу относятся таймеры TIM1 и TIM8, которые способны выполнять все перечисленные выше операции. Кроме того, на основе данных таймеров можно построить устройство, способное управлять трёхфазным электроприводом. В таблице 2.1 представлены типы таймеров микроконтроллера STM32F407, а также их основные технические характеристики.

Таблица 2.1 – Таймеры в микроконтроллере STM32F407VG

Тип таймера	Таймер	Разрешение счётчика	Направление счёта	Значение предделителя	Генерация запроса DMA	Каналы захвата/сравнения	Комплементарные выходы	Max (МГц)	Max (МГц)
Advanced-Control	TIM1, TIM8	16-bit	Up, Down, Up/Down	Любое число от 1 до 65 536	Да	4	Да	84	168
General-Purpose	TIM2, TIM5	32-bit	Up, Down, Up/Down	Любое число от 1 до 65 536	Да	4	Нет	42	84
	TIM3, TIM4	16-bit	Up, Down, Up/Down	Любое число от 1 до 65 536	Да	4	Нет	42	84
	TIM9	16-bit	Up	Любое число от 1 до 65 536	Нет	2	Нет	84	168
	TIM10, TIM11	16-bit	Up	Любое число от 1 до 65 536	Нет	1	Нет	84	168
	TIM12	16-bit	Up	Любое число от 1 до 65 536	Нет	2	Нет	42	84
	TIM13, TIM14	16-bit	Up	Любое число от 1 до 65 536	Нет	1	Нет	42	84
Basic	TIM6, TIM7	16-bit	Up	Любое число от 1 до 65 536	Да	0	Нет	42	84

2.1.1 Базовый модуль таймера

Базовая часть программируемого таймера включает в себя 16-разрядный счётчик с независимым регистром автоматической перезагрузки (рисунок 2.1). Счётчик может работать в следующих режимах: сложение, вычитание, сложение/вычитание. Тактовый сигнал на вход счётчика подаётся через делитель с настраиваемым коэффициентом деления.

Базовый модуль включает в себя:

– счётный регистр (TIMx_CNT);

– регистр делителя (TIMx_PSC);

– регистр автоматической перезагрузки (TIMx_ARR).

Счётный регистр, регистр автоматической перезагрузки и регистр делителя могут быть записаны или прочитаны программно. Это справедливо даже во время работы счётчика (рисунок 2.1).

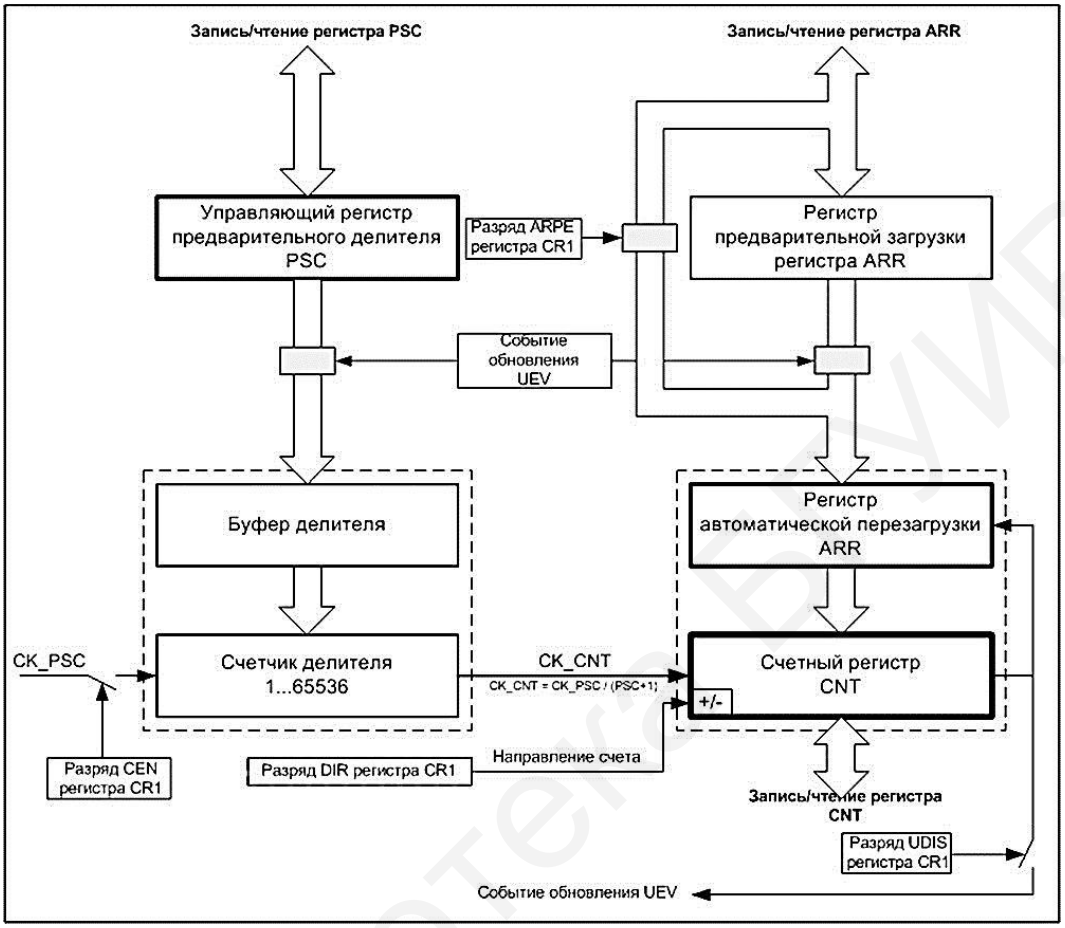

Рисунок 2.1 – Структура базового таймера

Регистр автоматической перезагрузки является предварительно загружаемым. Запись и чтение этого регистра доступны посредством регистра предварительной загрузки. Содержимое регистра предварительной загрузки передается в TIMx_ARR постоянно или после каждого события обновления (UEV) в зависимости от состояния бита ARPE регистра TIMx_CR1. Событие обновления формируется, когда происходит переполнение счётчика (при работе на вычитание или сложение) и если бит UDIS равен нулю. Событие обновления также может генерироваться программно. Генерирование события обновления подробно описано для каждой конфигурации таймера [1].

2.1.2 Тактирование счётного регистра

Таймер может тактироваться от разных источников. На выходе схемы выбора источника тактирования формируется сигнал CK_PSC, который и является тактовым сигналом таймера, но он подается на счётный регистр не напрямую, а через предварительный делитель с переменным коэффициентом деления (1…65 536). Сигнал, подаваемый непосредственно на вход счётного регистра, называется CK_CNT (см. рисунок 2.1).

Коэффициент деления предварительного делителя задается управляющим регистром PSC.

Управление подачей тактовых импульсов на вход предварительного делителя (а также включение/выключение таймера) осуществляется с помощью разряда CEN регистра TIMx_CR1.

2.1.3 Переполнение счётного регистра и событие обновления UEV

Перед началом счёта в регистр TIMx_CNT загружается его начальное значение. При счёте вверх (up) это значение равно нулю, а при счёте вниз (down) – содержимому регистра TIMx_ARR. От каждого тактового импульса содержимое TIMx_CNT увеличивается на единицу (при счёте вниз уменьшается на единицу), пока не достигнет своего максимального значения, которое определяется содержимым регистра TIMx_ARR (при счёте вниз – пока не достигнет своего минимального значения, т. е. нуля).

С приходом следующего тактового импульса произойдет сброс счётного регистра в нуль (при счёте вниз в него автоматически будет записано значение регистра TIMx_ARR). Этот переход и называется переполнением счётного регистра, в результате которого может формироваться событие обновления счётного регистра (UEV).

В микроконтроллере имеется возможность запретить формирование сигнала UEV. Делается это с помощью разряда UDIS регистра TIMx_CR1: если он равен нулю, то генерирование события обновления разрешено, единице – запрещено.

Помимо формирования сигнала обновления счётным регистром, его можно формировать программно (имитация переполнения счётного регистра). Для этого предназначен разряд UG регистра EGR. После записи в него единицы произойдет перезагрузка счётного регистра и формирование сигнала обновления (если он разрешён разрядом UDIS).

Событие обновления счётного регистра TIMx_CNT используется:

– для генерирования запроса прерывания от таймера;

– генерирования запроса DMA от таймера;

– записи нового значения регистров TIMx_ARR и TIMx_PSC;

– управления другим таймером.

Приведенная диаграмма иллюстрирует формирование события обновления по переполнению счётного регистра, который работает в режиме счёта вверх (рисунок 2.2). В регистре TIMx_ARR содержится 0x36.

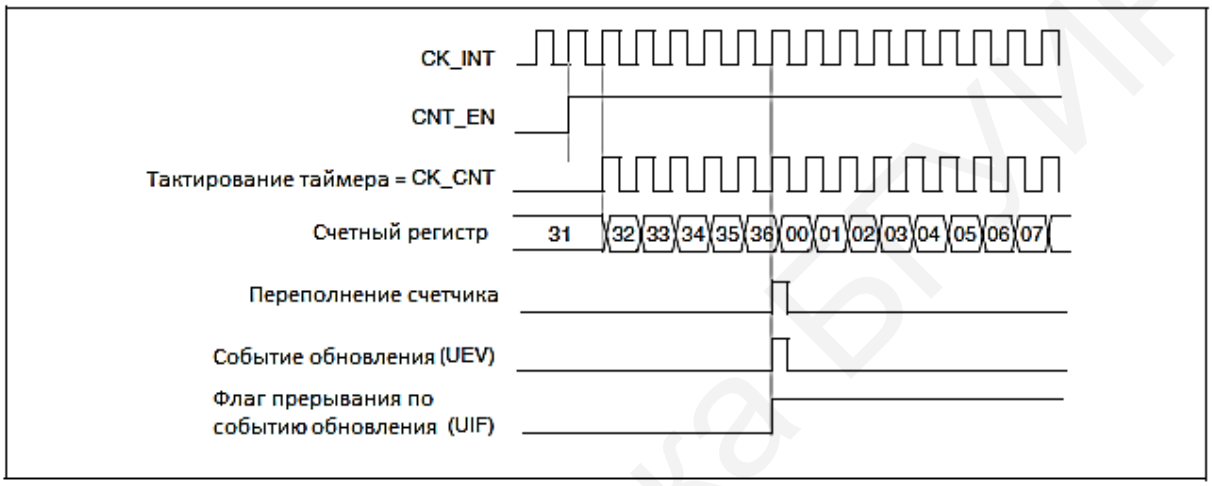

Рисунок 2.2 – Формирование события обновления
по переполнению счётного регистра

2.1.4 Таймеры общего назначения

Таймеры общего назначения имеют до четырёх независимых каналов, которые могут использоваться:

– для захвата сигнала;

– сравнения;

– генерации ШИМ;

– генерации одиночного импульса.

Таймеры микроконтроллера STM32F407 16-битные (т. е. могут считать до 65 535), могут работать с инкрементальными энкодерами и датчиками Холла, несколько таймеров можно синхронизировать между собой. Есть прерывания на разные события, а именно:

– переполнение;

– захват сигнала;

– сравнение;

– событие-триггер.

При наступлении любого из этих событий таймеры могут генерировать запрос к DMA (DMA – прямой доступ к памяти). Далее подробнее рассмотрим каждый из режимов работы таймеров.

Режим захвата сигнала. Это особый режим работы таймера. Суть режима захвата в следующем: по специальному сигналу (обычно с вывода контроллера) значение счётного регистра переписывается в специальный регистр, который называется регистром захвата (рисунок 2.3).

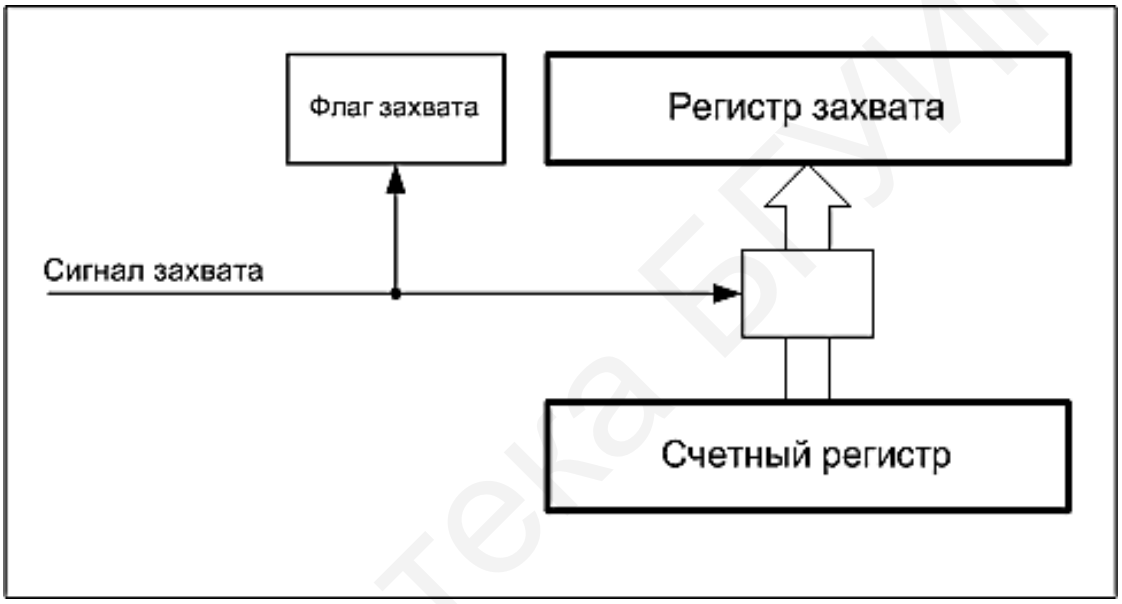

Рисунок 2.3 – Упрощенная схема работы таймера в режиме захвата

Помимо фиксации значения счётного регистра, происходит установка флага захвата, что может приводить к генерированию прерывания или запроса DMA.

Этот режим может быть полезным, например, при измерении периода сигнала: по фронту измеряемого сигнала значение счётного регистра сохраняется в регистре захвата, которое после прерывания можно переписать в память; следующий фронт также приведёт к сохранению значения счётного регистра; по разнице этих значений можно вычислить длительность сигнала.

Микроконтроллеры STM32 имеют дополнительные возможности для режима захвата:

– каждый таймер общего назначения оснащён не одним, а четырьмя каналами захвата. Соответственно имеется четыре входа контроллера

(CH1, CH2, CH3 и CH4), на которые можно подавать измеряемые сигналы, и четыре регистра захвата;

– сигнал захвата каждого канала можно обработать с помощью входного фильтра;

– имеется возможность выбрать активный уровень сигнала (фронт или спад) для каждого канала;

– сигнал захвата можно пропустить через делитель с изменяемым коэффициентом деления (1, 2, 4, 8);

– источником сигнала захвата может быть другой таймер;

– для каждого канала таймера предусмотрено по два флага. Второй флаг устанавливается, если произошел захват при установленном первом флаге.

Помимо этого, имеется возможность канал 1 подключить к выводу CH2, а канал 2 – к выводу CH1, и канал 3 – к выводу CH4, а канал 4 – к выводу CH3 (рисунок 2.4).

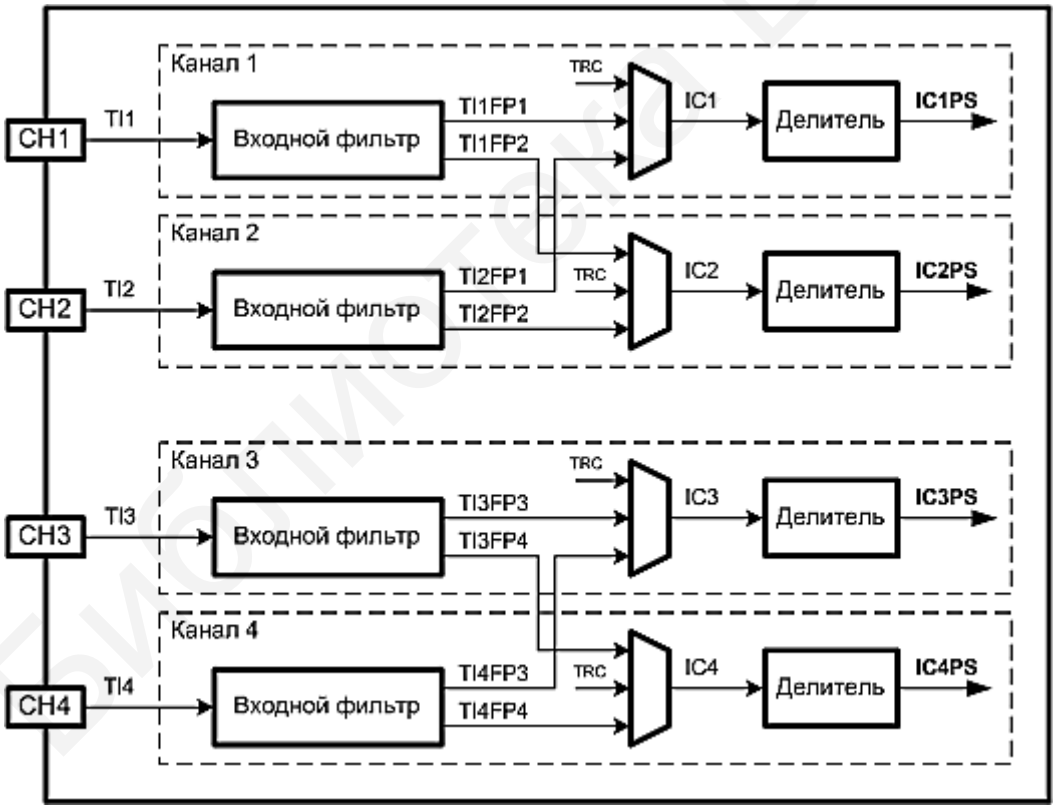

Рисунок 2.4 – Возможности подключения модулей захвата-сравнения

Итак, каналы 1 и 2 могут получать сигнал с вывода CH1 или CH2, а каналы 3 и 4 – с вывода CH3 или CH4. Это позволяет использовать их в паре для измерения параметров ШИМ-сигнала.

IC1PS, IC2PS, IC3PS и IC4PS – это сигналы захвата соответствующих каналов. Именно по этим сигналам происходит перезапись значения счётного регистра в регистр захвата соответствующего канала.

Регистры захвата называются CCR1, CCR2, CCR3 и CCR4 (точнее, это регистры захвата/сравнения). Если канал настроен в режим захвата, то соответствующий ему регистр CCRx является регистром захвата, если настроен в режим сравнения – регистром сравнения.

Режим сравнения. Режим сравнения является востребованным режимом работы таймеров. Благодаря ему возможно формирование микроконтроллером выходных импульсов с минимальным участием ЦПУ, формирование ШИМ и др.

Ниже приведена упрощённая схема, которая показывает принцип режима сравнения (рисунок 2.5).

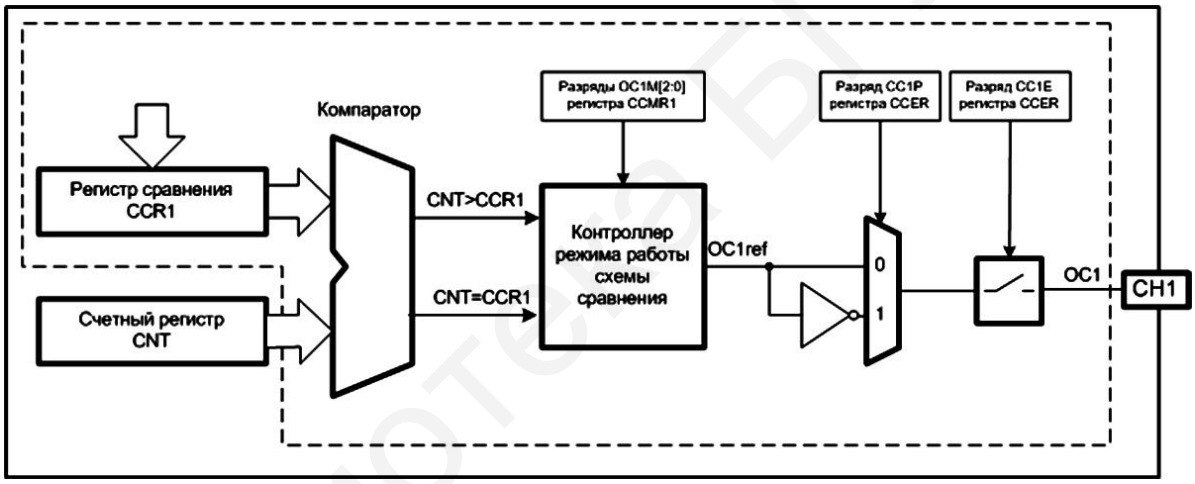

Рисунок 2.5 – Упрощённая схема работы таймера в режиме сравнения

В регистр сравнения заносится значение, которое постоянно сравнивается со значением счётного регистра. Когда значение счётного регистра станет равным значению регистра сравнения (или больше его), будет сформирован специальный сигнал, который может использоваться внутри контроллера (генерирование прерывания, запроса DMA) или управлять определённым внешним выводом контроллера (устанавливать его в единицу, сбрасывать, инвертировать).

Сигнал с компаратора, который сравнивает значения счётного регистра и регистра сравнения, поступает на контроллер режима работы схемы сравнения. Это устройство в зависимости от разрядов OC1M регистра CCMR1 формирует сигнал совпадения OC1ref, который в дальнейшем используется для формирования сигнала, поступающего на выход таймера CH1. С помо-

щью разряда CC1P регистра CCER можно выполнить инверсию сигнала (если записать в этот разряд нуль), с помощью разряда CC1E регистра CCER можно отключить выход CH1 от схемы сравнения.

Режим широтно-импульсной модуляции. Этот режим заслуживает отдельного рассмотрения, т. к. весьма часто используется. ШИМ (pulse-width modulation, PWM) – это изменение скважности импульсов при его постоянной длительности. Благодаря ШИМ легко организовать, например, регулирование оборотов двигателя постоянного тока, при этом КПД такой схемы будет весьма высоким. Временные диаграммы, поясняющие принцип широтно-импульсной модуляции, изображены на рисунке 2.6.

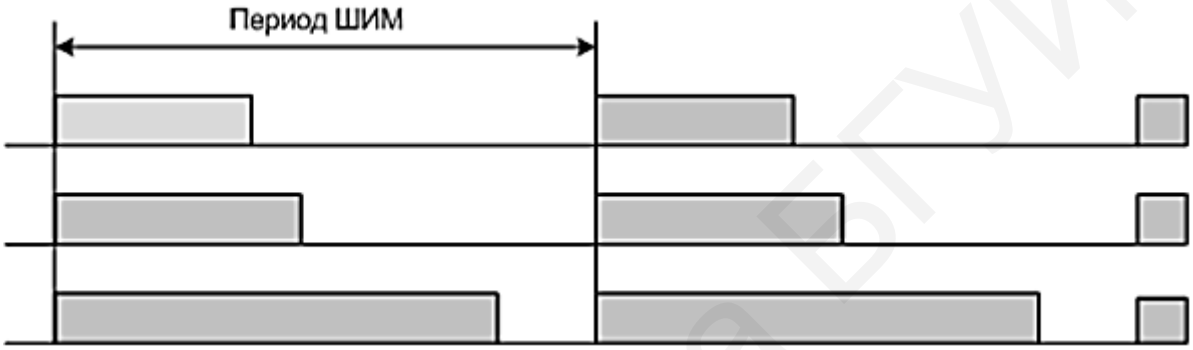

Рисунок 2.6 – Принцип широтно-импульсной модуляции

Работа ШИМ посредством таймеров. В данном режиме используются три регистра: счётный регистр (TIMx_CNT), регистр предварительной автозагрузки (TIMx_ARR) и регистр сравнения (TIMx_CCRx). В регистр TIMx_ARR необходимо загрузить значение, до которого будет считать счётный регистр. Это значение определяет период ШИМ. После этого, изменяя содержимое регистра сравнения, можно изменять скважность импульсов. Если значение счётного регистра равно нулю, сигнал OCxREF будет установлен равным единице (данный сигнал используется для управления внешним выводом микроконтроллера). Если значение счётного регистра больше либо равно значению регистра TIMx_CCRx, сигнал OCxREF будет установлен равным нулю (рисунок 2.7).

Рисунок 2.7 – Работа ШИМ посредством таймера. Прямой режим

Режимы работы ШИМ. Предусмотрено два режима: №1 (прямой ШИМ) и №2 (инвертированный ШИМ). Рисунок 2.7 показывает работу таймера в прямом режиме. Инвертированный ШИМ представлен на рисунке 2.8.

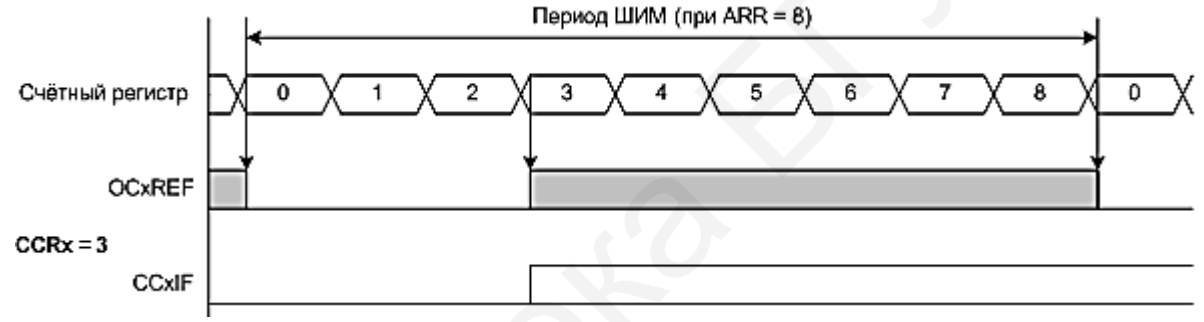

Рисунок 2.8 – Работа ШИМ посредством таймера. Инвертированный режим

Также можно задать способ выравнивания ШИМ-сигнала: выравнивание бывает по краям или по центру (задается разрядами CMS регистра CR1). Приведенные выше рисунки 2.7 и 2.8 – это ШИМ с выравниванием по краям (CMS = 00). Если значение CMS отличается от 00, то будет включен режим выравнивания по центру. Временная диаграмма этого режима представлена на рисунке 2.9.

Рисунок 2.9 – Работа ШИМ посредством таймера.
Режим выравнивания по центру

При CMS > 00 счётный регистр начинает работать *по-особому*: сразу работает на сложение, дойдя до максимального значения (определяемого ARR), начинает работать на вычитание, затем все повторяется. При этом сигнал OCxREF меняет свое значение на противоположное, когда происходит совпадение счётного регистра и регистра сравнения. Событие совпадения (а значит флаг совпадения, прерывание, запрос DMA) генерируется в зависимости от разрядов CMS:

– CMS = 01 – при совпадении во время счёта вверх;
– CMS = 10 – при совпадении во время счёта вниз;
– CMS = 01 – при совпадении во время счёта вверх и вниз (т. е. два раза).

2.2 Порядок выполнения работы

2.2.1 Настройка библиотек

Создайте пустой проект. В нём должны быть включены следующие компоненты (как подключаются компоненты библиотек, смотрите в лабораторной работе №1):

– M4 CMSIS Core;
– CMSIS BOOT;
– RCC;
– GPIO;
– MISC (нужен для настройки и обработки прерываний);
– TIM (для настройки и включения таймеров).

Далее в проект необходимо добавить сгенерированный файл. Для этого перетащите файл `system_stm32f4xx.c` в область Project в папку `cmsis_boot` (рисунок 2.10). Данный файл настроит тактирование всех систем микроконтроллера, способ его создания рассмотрен в пункте 1.2.3 лабораторной работы №1.

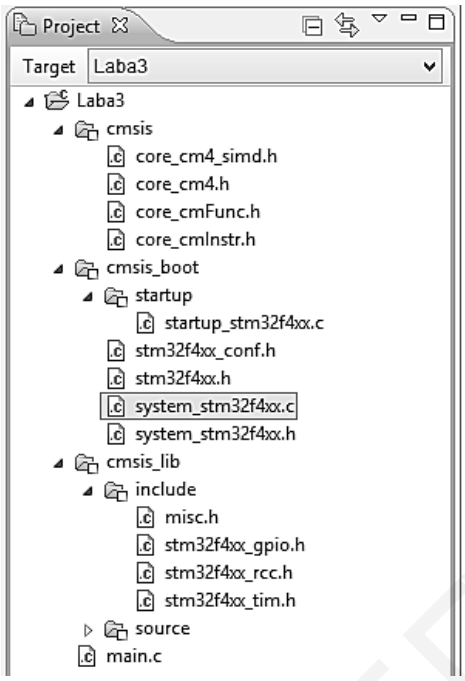

Рисунок 2.10 – Расположение файла в дереве проекта

Необходимо согласиться с заменой (рисунок 2.11).

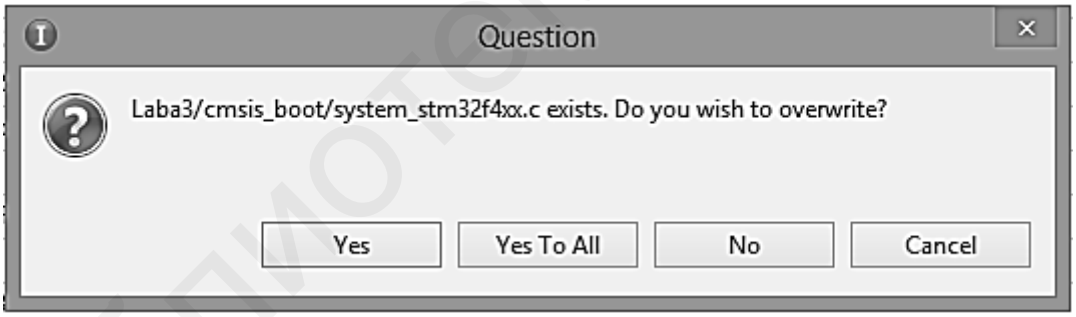

Рисунок 2.11 – Диалоговое окно замены файла

В файле `main.c` подключаем заголовочные файлы:

```
#include "stm32f4xx.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
#include "system_stm32f4xx.h"
#include "misc.h"
```

Теперь для настройки тактирования необходимо вызвать функцию SystemInit(); при инициализации микроконтроллера

```
int main(void)
{
    SystemInit();

    while(1)
    {
    }
}
```

Данная функция прописана в подключенном файле system_stm32f4xx.c.

Также необходимо написать функцию инициализации светодиода на выводе PD12, аналогично тому, как это делалось в лабораторной работе №1.

2.2.2 Инициализация таймера

Для инициализации таймера необходимо включить тактирование этого таймера:

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);
```

Как вы, наверное, заметили, название функции немного отличается от того, какое мы использовали при инициализации GPIO. Отличается наименованием шины: вместо AHB (Advanced High-performance Bus или Расширенная высокопроизводительная шина) используется APB (Advanced Peripheral Bus, или Расширенная шина периферии).

Следующим шагом будет объявление структуры инициализации:

```
TIM_TimeBaseInitTypeDef TIM_InitStruct;
```

Далее необходимо проинициализировать параметры объявленной структуры значениями по умолчанию:

```
TIM_TimeBaseStructInit(&TIM_InitStruct);
```

Подобные функции существуют почти для всех структур инициализации микроконтроллера. Нужны они для того, чтобы предотвратить ввод некорректных данных в значения параметров структуры. При объявлении структуры на языке С происходит передача адреса ячейки памяти, начиная с

которой находится экземпляр структуры. Проблема в том, что при этом не происходит очистка ячеек памяти, в которых расположен экземпляр структуры. В итоге, если не присвоить каждому параметру структуры конкретное значение, структура может иметь недопустимые значения параметров, что вызывает ошибку, которую очень сложно выявить.

Структура для инициализации таймера имеет следующие поля:

– TIM_Period = N – период для записи в регистр перезагрузки ARR, N = 0…65565;

– TIM_Prescaler = N – значение делителя частоты, N = 0...6556;

– TIM_ClockDivision = 0 – дополнительный делитель системной частоты;

– TIM_CounterMode = TIM_CounterMode_Up – режим работы таймера, возможные режимы можно уточнить в документации [2];

– TIM_RepetitionCounter = N – счетчик переполнений для генерации прерывания от таймера. Прерывание генерируется не каждый раз при переполнении, а через заданное количество переполнений (N+1).

Для настройки модулей захвата/сравнения используется управляющая структура, определяемая типом TIM_OCInitTypeDef:

– TIM_OCMode = TIM_OCMode_PWM1 – задает режим работы модуля захвата/сравнения;

– TIM_OutputState = TIM_OutputState_Enable – включает или выключает вывод сигнала на внешние выводы;

– TIM_Pulse = CCR1_Val – определяет значение, загружаемое в регистр захвата/сравнения;

– TIM_OCPolarity = TIM_OCPolarity_High – определяет активный уровень на выходе модуля захвата/сравнения.

Функция TIM_OC1Init служит для настройки модуля захвата/сравнения в соответствии со значениями, указанными в управляющей структуре:

```
TIM_OC1Init(TIM2,&TIM_OCInitStructure);
```

2.2.3 Настройка тактирования

Для настройки тактирования микроконтроллера необходимо сгенерировать С-файл конфигурации при помощи специального Excel-файла – «STM32F4xx_Clock_Configuration_V1.1.0.xls». Этот файл специально разработан производителями микроконтроллера для удобной и интерактивной настройки системы тактирования. Для начала работы необходимо запустить этот файл. Вид открывшегося окна показан на рисунке 2.12.

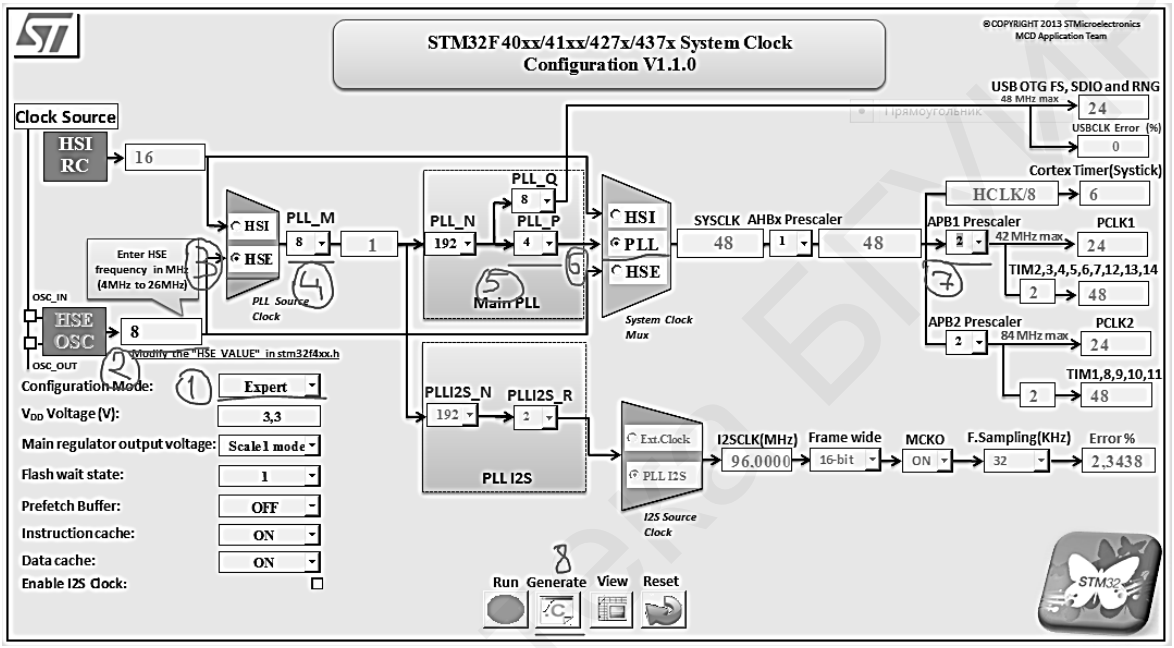

Рисунок 2.12 – Настройка тактирования микроконтроллера STM32F407VG

Разберём каждый пункт настройки (в соответствии с цифрами на рисунке 2.12):

1 – выбор режима настройки Expert;

2 – выбор частоты тактирования внешнего осциллятора. На плате STM32F4DISCOVERY запаян осциллятор на 8 МГц;

3 – выбор тактирования: HSI (внутренний осциллятор) либо HSE (внешний осциллятор). Выбираем HSE;

4 – выбор делителя частоты (в данном случае делим на 8);

5 – выбор делителя для частоты 192 МГц после PLL (делим на 4);

6 – выбор источника тактирования SYSCLK как PLL;

7 – выставление величины предделителя частоты тактирования шины APB2 равной 2. Таким образом, таймеры 2, 3, 4, 5, 6, 7, 12, 13, 14 тактируются частотой 48 МГц;

8 – нажатие кнопки `Generate`. Откроется окно `File successfully generated` и в папке рядом с Excel-файлом должен появиться файл `system_stm32f4xx.c`;

9 – содержание данного файла необходимо поместить в соответствующий файл проекта, после этого частоты тактирования должны соответствовать заданным.

2.2.4 Настройка приоритета и источника прерываний с помощью модуля NVIC, запуск прерываний

Для управления прерываниями существует специальный модуль NVIC – контроллер вложенных векторизированных прерываний STM32 (Nested vectored interrupt controller). Для настройки прерывания от таймера необходимо задать значение поля структуры `NVIC_InitStruct.NVIC_IRQChannel=TIMx_IRQn.`

Например, установка прерывания от таймера 2 с приоритетом и субприоритетом равна единице:

```
NVIC_InitTypeDef NVIC_InitStruct;
NVIC_InitStruct.NVIC_IRQChannel = TIM2_IRQn;
NVIC_InitStruct.NVIC_IRQChannelPreemptionPriority = 0x01;
NVIC_InitStruct.NVIC_IRQChannelSubPriority = 0x01;
NVIC_InitStruct.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStruct);
```

2.2.4.1 Обработка прерываний

Для обработки прерываний существуют специальные функции, объявления которых находятся в файле `startap_stm32f4xx.c`. Для таймера 2 обработчик прерывания имеет имя `TIM2_IRQHandler`. Если создать определение функции с данным именем, то, как только происходит прерывание, программа заходит в данную функцию. На одни и те же обработчики прерываний могут приходить до шестнадцати (например, USART) разных прерываний. Следовательно, при входе в обработчик прерываний необходимо проверить, какое именно произошло прерывание. После обработки прерывания необходимо сбросить флаг прерывания, для того чтобы прерывание снова произошло.

Приведём пример функции обработчика прерывания для таймера 2:

```
void TIM2_IRQHandler(void){
    if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET){
         TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
         //здесь необходимо поместить свой код
    }
}
```

Проверьте работу прерываний в режиме отладки программы, выбрав пункт Debug → Debug. При этом запустится режим отладки и можно в прерывании поставить точку останова.

2.3 Индивидуальные задания

Индивидуальные задания к лабораторной работе №2 представлены в таблице 2.2.

Таблица 2.2 – Индивидуальные задания для лабораторной работы №2

Вариант	Задания
1	2
1	Мигать синим светодиодом с переменной: F1 = 0,5 Гц, F2 = 2 Гц. Скважность равна 2. При нажатии кнопки происходит смена частоты. Повторное нажатие кнопки возвращает в первоначальное состояние. Реализовать на основе прерывания таймера
2	Мигать попеременно красным и зелёным светодиодами. F = 3 Гц. Скважность равна 2. При нажатии кнопки скважность равна 4. Повторное нажатие на кнопки возвращает в первоначальное состояние. Реализуйте на основе прерывания таймера
3	Мигать красным светодиодом с F = 0,25 Гц, зелёным светодиодом – с F = 1 Гц. Скважность равна 4. При нажатии кнопки частота увеличивается в 2 раза. Повторное нажатие кнопки возвращает в первоначальное состояние. Реализовать на основе прерывания таймера

Продолжение таблицы 2.2

1	2
4	Зажигать и гасить по очереди все четыре светодиода, F = 1 Гц. При нажатии кнопки порядок мигания светодиодов меняется на противоположный. Повторное нажатие кнопки возвращает в первоначальное состояние. Реализовать на основе прерывания таймера
5	Зелёный и красный светодиоды горят попеременно с жёлтым светодиодом. F = 1 Гц. Скважность равна 5. При нажатии кнопки F = 5 Гц. Повторное нажатие кнопки возвращает в первоначальное состояние. Реализовать на основе прерывания таймера
6	Два любых светодиода горят постоянно, оставшиеся два горят попеременно. F = 3 Гц. Скважность равна 2. При нажатии кнопки мигающие светодиоды начинают постоянно гореть, постоянно горевшие светодиоды начинают мигать. Повторное нажатие кнопки возвращает в первоначальное состояние. Реализовать на основе прерывания таймера
7	Мигать синим и зелёным светодиодами. F = 1 Гц. Скважность равна 4. Красный светодиод горит постоянно. При нажатии кнопки мигающие светодиоды начинают постоянно гореть, постоянно горевшие светодиоды начинают мерцать. Повторное нажатие кнопки возвращает в первоначальное состояние. Реализовать на основе прерывания таймера
8	Мигать попеременно красным и зелёным светодиодами. F = 5 Гц. Скважность равна 3. При нажатии кнопки F = 2 Гц, скважность равна 2. Повторное нажатие кнопки возвращает в первоначальное состояние. Реализовать на основе прерывания таймера
9	Мигать синим светодиодом с F = 5 Гц, зелёным светодиодом с F = 1 Гц. Скважность равна 4. При нажатии кнопки начинают мигать красный и жёлтый светодиоды (параметры те же). Повторное нажатие кнопки возвращает в первоначальное состояние. Реализовать на основе прерывания таймера

Продолжение таблицы 2.2

1	2
10	Зажигать и гасить по очереди все четыре светодиода, F = 1 Гц. При нажатии кнопки F = 5 Гц. Повторное нажатие кнопки возвращает в первоначальное состояние. Реализовать на основе прерывания таймера
Задания повышенной сложности	
11	Зажигать и гасить по очереди все четыре светодиода, F1 = 1 Гц, F2 = 2 Гц, F3 = 3Гц, F4 = 4 Гц. При нажатии кнопки порядок мигания светодиодов меняется на противоположный. Повторное нажатие кнопки возвращает в первоначальное состояние. Реализовать на основе прерывания таймера
12	Зажигать и гасить по очереди все четыре светодиода, F1 = 1 Гц, F2 = 2 Гц, F3 = 3 Гц, F4 = 4 Гц. При нажатии кнопки F1 = 4Гц, F2 = 3 Гц, F3 = 2 Гц, F4 = 1 Гц. Повторное нажатие кнопки возвращает в первоначальное состояние. Реализовать на основе прерывания таймера
13	Мигать синим светодиодом с переменной частотой: F1 = 0,5 Гц, F2 = 2 Гц, F3 = 3 Гц, F4 = 4 Гц. Скважность равна 2. При нажатии кнопки происходит циклическая смена частоты. Реализовать на основе прерывания таймера
14	Мигать красным светодиодом с переменной частотой: F1 = 5 Гц, F2 = 4 Гц, F3 = 3 Гц, F4 = 2Гц. Скважность равна 4. При нажатии кнопки происходит циклическая смена частоты. Реализовать на основе прерывания таймера
15	Мигать синим светодиодом с переменной частотой: F1 = 0,5 Гц, F2 = 1 Гц, F3 = 2 Гц, F4 = 4 Гц. Скважность равна 0,5. При нажатии кнопки происходит циклическая смена частоты. Реализовать на основе прерывания таймера
16	Первоначально светодиоды не горят. При нажатии кнопки зажигать по очереди все четыре светодиода с переменной частотой от 40 до 1 Гц с шагом 0,5 Гц. Реализовать на основе прерывания таймера
17	Первоначально светодиоды горят. При нажатии кнопки гасить по очереди все четыре светодиода с переменной частотой от 20 до 1 Гц с шагом 0,25 Гц. Реализовать на основе прерывания таймера

2.4 Контрольные вопросы

1. Каковы области применения таймеров микроконтроллеров серии STM32?

2. В каких режимах могут работать таймеры микроконтроллеров серии STM32?

3. Каковы особенности работы в режиме TIM_CounterMode_Up?

4. Каковы особенности работы в режиме TIM_CounterMode_Down?

5. Каковы особенности работы в режиме TIM_CounterMode CenterAligned1?

6. Что такое прерывание и как это используется при работе таймеров микроконтроллеров серии STM32?

7. В каких режимах могут работать модули захвата/сравнения?

8. В каких ситуациях применяют режим сравнения?

9. В каких ситуациях применяют режим захвата сигнала?

10. Для чего используется режим широтно-импульсной модуляции?

Лабораторная работа №3
Работа с аналого-цифровым преобразователем микроконтроллера STM32F407

Цель работы: сформировать общее представление о работе аналого-цифрового преобразователя (АЦП) микроконтроллера STM32F407; изучить особенности регистрации цифровых сигналов.

3.1 Теоретические сведения

Микроконтроллер STM32F4xx имеет три 12-разрядных АЦП. Каждое АЦП может быть подключено к любому из 24 аналоговых входов. Более того, каждое из АЦП может сканировать эти входы, снимая с них данные в заданном пользователем порядке.

По окончании преобразования АЦП может выдать прерывание. В общем случае АЦП может выдать одно из трёх прерываний: об окончании преобразования обычного (регулярного) канала, об окончании преобразования по инжектированному каналу и событие по Watchdog.

Каналы АЦП микроконтроллеров STM32 делятся на две группы: регулярные каналы (regular) и инжектированные (injected). Количество регулярных каналов для одного АЦП – 18, среди них 16 внешних и 2 внутренних (опорное напряжение и температурный датчик). Количество инжектированных каналов – 4. Регулярные каналы подразумевают сохранение результатов преобразования через DMA-контроллер в памяти микроконтроллера, а инжектированные каналы имеют собственные регистры для хранения результата. Существует возможность настраивать работу каналов АЦП в произвольном порядке, несколько раз преобразовывать подряд одни и те же каналы, использовать внешние и программные события для старта преобразования (рисунок 3.1).

Для параллельного снятия данных сразу по нескольким каналам предусмотрена возможность одновременного запуска нескольких АЦП. Данный режим получил название Dual Mode и Triple Mode.

АЦП имеет хорошую разрешающую способность – 12 бит – и высокую скорость преобразования – 2,4 млн преобразований в секунду (MIPS) в одиночном режиме и 7,2 MIPS – в режиме Triple Mode. Гибкая система настроек встроенного аналогового мультиплексора позволяет задавать любые последовательности преобразования аналоговых каналов (за исключением одновременного преобразования одного канала на нескольких АЦП). Настройки

АЦП позволяют производить однократные и циклические измерения. Для проведения преобразования на максимальных скоростях необходимо соблюдать диапазон напряжения питания от 2,4 до 3,6 В. При снижении напряжения до 1,8(1,7) В скорость преобразования снижается до 1,2 MIPS.

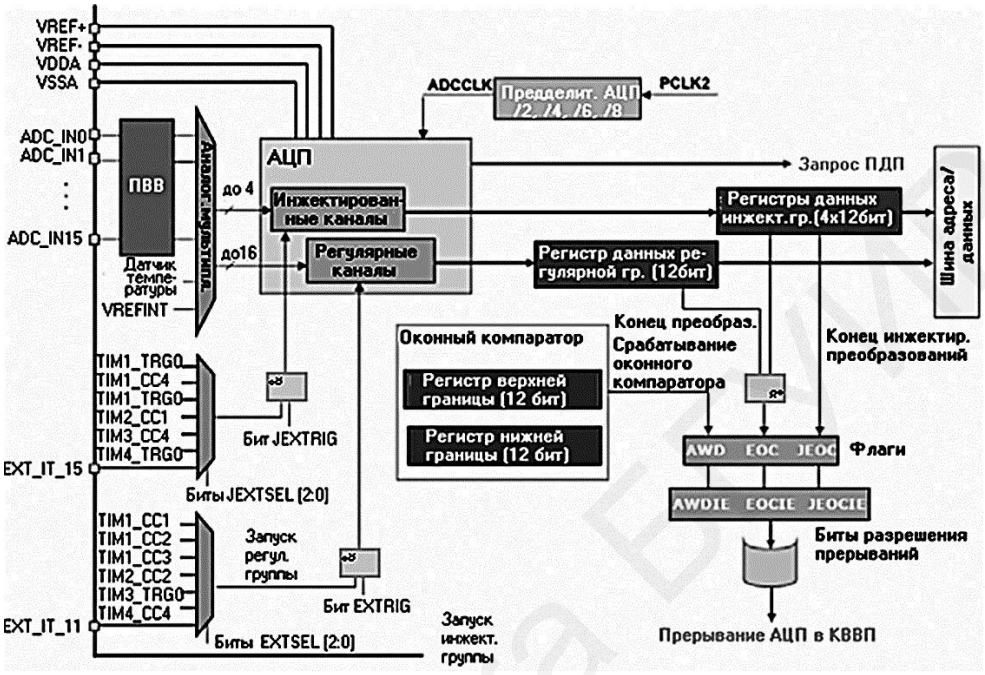

Рисунок 3.1 – Структура аналого-цифрового преобразователя микроконтроллера STM32F407

Для контроля внутренней температуры микроконтроллера встроен температурный датчик. На его выходе формируется напряжение в зависимости от окружающей температуры. Выход датчика через мультиплексор подключается к АЦП. Используя температурный датчик, можно измерять температуру от –40 до 125 °С с точностью ±1,5 °С. Типовая схема подключения АЦП приведена на рисунке 3.2.

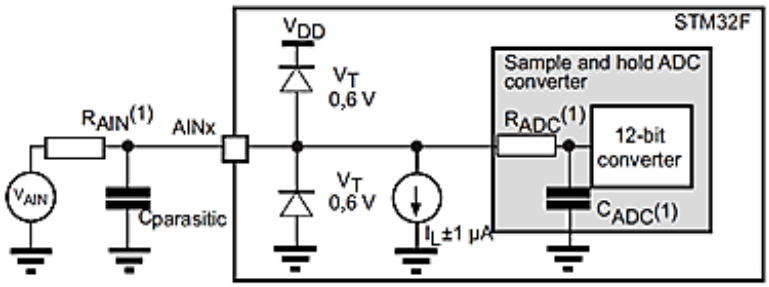

Рисунок 3.2 – Типовая схема подключения АЦП

3.2 Порядок выполнения работы

3.2.1 Настройка библиотек

В проекте нужны следующие компоненты (как подключаются компоненты библиотек, смотрите в лабораторной работе №1):

– M4 CMSIS Core;

– CMSIS BOOT;

– RCC;

– GPIO;

– ADC (модуль отвечает за АЦП);

– TIM (нужен для настройки таймера);

– MISC (нужен для настройки и обработки прерываний).

В файле main.c подключаем заголовочные файлы:

```
#include "stm32f4xx.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_adc.h"
#include "stm32f4xx_tim.h"
#include "misc.h"
```

3.2.2 Инициализация аппаратных средств

Объявляем функцию инициализации аппаратных средств микроконтроллера:

```
void Board_Init(void
{
    GPIO_LEDS_Init();
    ADC1_Init();
    Timer3_Init();
    Timer4_Init();
}
```

3.2.3 Настройка портов ввода/вывода для работы со светодиодами и для АЦП

Для работы светодиодов необходимо произвести инициализацию GPIO:

– инициализировать порты ввода/вывода для светодиодов (смотрите лабораторную работу №1);

– инициализация портов ввода/вывода для АЦП.

Включаем тактирование портов ввода/вывода для АЦП:

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
```

3.2.4 Инициализация АЦП

Определяем функцию для инициализации АЦП:

```
GPIO_ADC1_Init();
```

Включаем тактирование АЦП:

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
```

3.2.4.1 Структура общей настройки АЦП

Определим и инициализируем структуру для общей настройки АЦП:

```
ADC_CommonInitTypeDef ADC_CommonInitStruct;
ADC_CommonStructInit(&ADC_CommonInitStruct);
```

Зададим значение полей структуры:

```
ADC_CommonInitStruct.ADC_Mode = ADC_Mode_Independent;
ADC_CommonInitStruct.ADC_Prescaler = ADC_Prescaler_Div2;
ADC_CommonInitStruct.ADC_DMAAccessMode=
ADC_DMAAccessMode_Disabled;
ADC_CommonInitStruct.ADC_TwoSamplingDelay=
ADC_TwoSamplingDelay_5Cycles;
```

Инициализируем АЦП периферийных устройств в соответствии с заданными параметрами в ADC_CommonInitStruct:

```
ADC_CommonInit(&ADC_CommonInitStruct);
```

Разберём некоторые параметры:

– ADC_Mode – настраивает АЦП для работы в автономном режиме или нескольких. Этот параметр может принимать значения:

```
ADC_Mode_Independent //АЦП работает независимо от других (необхо-
димо использовать в лабораторной работе)
ADC_DualMode_RegSimult_InjecSimult; – работа в режиме Dual
ADC_DualMode_RegSimult_AlterTrig;
ADC_DualMode_InjecSimult;
ADC_DualMode_RegSimult;
ADC_DualMode_Interl;
ADC_DualMode_AlterTrig;
ADC_TripleMode_RegSimult_InjecSimult; – работа в режиме Triple
ADC_TripleMode_RegSimult_AlterTrig;
ADC_TripleMode_InjecSimult;
ADC_TripleMode_RegSimult;
ADC_TripleMode_Interl;
ADC_TripleMode_AlterTrig;
```

– ADC_Prescaler – указывает предделитель, используемый для настройки частоты тактового сигнала на АЦП. Частота является общим параметром для всех АЦП. Параметр может принимать значения:

```
ADC_Prescaler_Div2; //(Предделитель устанавливаем на 2)
ADC_Prescaler_Div4; //(Предделитель устанавливаем на 4)
ADC_Prescaler_Div6; //(Предделитель устанавливаем на 6)
ADC_Prescaler_Div8; //(Предделитель устанавливаем на 8)
```

– ADC_TwoSamplingDelay – настраивает задержку между двумя сэмплами (фазами) выборки. Этот параметр может принимать значения от 5 до 20 тактов:

```
ADC_TwoSamplingDelay_5Cycles;
ADC_TwoSamplingDelay_6Cycles;
ADC_TwoSamplingDelay_7Cycles;
ADC_TwoSamplingDelay_8Cycles;
```

– ADC_DMAAccessMode – настраивает режим прямого доступа к памяти для режима нескольких АЦП. Может принимать значения:

```
ADC_DMAAccessMode_Disabled;
ADC_DMAAccessMode_1;
ADC_DMAAccessMode_2;
ADC_DMAAccessMode_3;
```

3.2.4.2 Основная структура настройки параметров АЦП

Определим и инициализируем структуру для настройки параметров АЦП:

```
ADC_InitTypeDef       ADC_InitStruct;
ADC_StructInit(&ADC_InitStruct);
```

Зададим значение полей структуры:

```
ADC_InitStruct.ADC_Resolution = ADC_Resolution_12b;
ADC_InitStruct.ADC_ScanConvMode = DISABLE;
ADC_InitStruct.ADC_ContinuousConvMode = DISABLE;
ADC_InitStruct.ADC_ExternalTrigConvEdge =
ADC_ExternalTrigConvEdge_None;
ADC_InitStruct.ADC_ExternalTrigConv =
ADC_ExternalTrigConv_T1_CC1;
ADC_InitStruct.ADC_DataAlign = ADC_DataAlign_Right;
ADC_InitStruct.ADC_NbrOfConversion = 1;
```

Выполним инициализацию АЦП:

```
ADC_Init(ADC1, &ADC_InitStruct);

ADC_EOCOnEachRegularChannelCmd(ADC1, ENABLE);
ADC_Cmd(ADC1, ENABLE);
```

Рассмотрим некоторые функции:

– ADC_EOCOnEachRegularChannelCmd(ADC_TypeDef*ADCx, FunctionalState NewState) – включает или выключает End–of–Conversion (EOC – конечное преобразование) на каждом очередном преобразовании канала:

• ADCx – где x может быть 1, 2 или 3 для выбора АЦП периферийного устройства;

• NewState – этот параметр может быть ENABLE (включить) или DISABLE (отключить).

Рассмотрим значения полей структуры для настройки параметров АЦП:

– ADC_Resolution – настраивает двойной режим разрешения АЦП (выбираем число значащих разрядов преобразования). Может принимать значения:

```
ADC_Resolution_12b; // на выходе преобразования 12-битные значе-
ния
ADC_Resolution_10b; // на выходе преобразования 10-битные значе-
ния
ADC_Resolution_8b; // на выходе преобразования 8-битные значения
ADC_Resolution_6b; // на выходе преобразования 6-битные значения
```

– ADC_ScanConvMode – указывает, выполняется ли преобразование в Scan (многоканальности) или Single (один канал). Этот параметр может быть установлен для ENABLE (включения) или DISABLE (отключения). То есть этот параметр определяет, будет ли АЦП сканировать несколько каналов. Если этот режим включен, то АЦП будет последовательно оцифровывать данные с заданных каналов в заданной последовательности;

– ADC_ContinuousConvMode – указывает, выполняется ли преобразование в непрерывном или одиночном режиме. Этот параметр может быть установлен в ENABLE (включен) или DISABLE (отключен). В данном режиме сразу по окончании предыдущего преобразования запускается следующее преобразование. Так можно добиться максимальной скорости работы АЦП. В данной лабораторной работе этого не требуется;

– ADC_ExternalTrigConvEdge – настраивает запуск преобразования по какому-либо событию, например переполнению таймера. Этот параметр может быть значением:

```
ADC_ExternalTrigConvEdge_None; (без внешнего триггера)
ADC_ExternalTrigConvEdge_Rising; (по переднему фронту сигнала)
ADC_ExternalTrigConvEdge_Falling; (по заднему фронту сигнала)
ADC_ExternalTrigConvEdge_RisingFalling; (по переднему и заднему
фронту сигнала)
```

– ADC_DataAlign – указывает, в какую сторону выравниваются данные (регистр 16-битный, а значащих данных всего 12 или меньше).

Пример

С АЦП пришло значение 12 бит.

Если выравнивание вправо, то `0b0000xxxxxxxxxxxx`, где x – значения битов, полученные в процессе аналого-цифрового преобразования. Если выравнивание влево, то `0bxxxxxxxxxxxx0000`).

Этот параметр может иметь значение:

– ADC_DataAlign_Right; (выравнивание вправо);

– ADC_DataAlign_Left; (выравнивание влево);

– ADC_ExternalTrigConv – устанавливает, какие именно события запустят преобразования.

Этот параметр может принимать значения:

```
ADC_ExternalTrigConv_T1_CC1;  запуск преобразования от модуля захвата/сравнения 1 таймера 1
ADC_ExternalTrigConv_T1_CC2; запуск преобразования от модуля захвата/сравнения 2 таймера 1
ADC_ExternalTrigConv_T1_CC3;
ADC_ExternalTrigConv_T2_CC2;
ADC_ExternalTrigConv_T2_CC3;
ADC_ExternalTrigConv_T2_CC4;
ADC_ExternalTrigConv_T2_TRGO;
ADC_ExternalTrigConv_T3_CC1;
ADC_ExternalTrigConv_T3_TRGO;
ADC_ExternalTrigConv_T4_CC4;
ADC_ExternalTrigConv_T5_CC1;
ADC_ExternalTrigConv_T5_CC2;
ADC_ExternalTrigConv_T5_CC3;
ADC_ExternalTrigConv_T8_CC1;
ADC_ExternalTrigConv_T8_TRGO;
ADC_ExternalTrigConv_Ext_IT11;
```

– ADC_NbrOfConversion – указывает число переходов АЦП, которое будет проводиться с использованием секвенсеров (устройство для записи в реальном времени) для регулярной группы каналов. Этот параметр должен находиться в диапазоне от 1 до 16 – это число каналов, которые будет сканировать МК. Сюда записывается требуемое значение, а ниже, если это число больше единицы и ADC_ ScanConvMode = ENABLE, описывается, какие каналы и в какой последовательности будут сканироваться.

3.2.5 Настройка таймеров, приоритета и источника прерываний с помощью NVIC, запуск прерываний

Для управления прерываниями производится инициализация NVIC (подробнее смотрите в лабораторной работе №1).

Следует инициализировать таймеры (смотрите лабораторную работу №2).

3.2.6 Настройка конкретного канала АЦП

В нашем случае это всего один канал, поэтому настройка будет выглядеть следующим образом:

```
ADC_RegularChannelConfig(ADC1,channel,1,ADC_SampleTime_480Cycles);
```

В данном случае настраивается первый канал АЦП1 с длительностью преобразования, равной 480 циклов тактирующего сигнала.

Рассмотрим основные функции для работы с АЦП:

1. ADC_RegularChannelConfig(ADC_TypeDef*ADCx,ADC_Channel, Rank, ADC_SampleTime) настраивает для выбранного регулярного канала АЦП его порядковый номер в цепочке преобразований и его время преобразования.

Его параметры:

– ADCx – номер АЦП, где x может быть 1, 2 или 3;

– ADC_Channel – задаёт канал АЦП (от Channel1 до Channel18) ;

– Rank – показывает, в каком порядке этот канал будет оцифровываться. Этот параметр должен быть в пределах от 1 до 16. В нашем случае канал один, поэтому и rank равен единице;

– ADC_SampleTime – задаёт, за какое время будет произведена оцифровка аналогового сигнала. Чем больше время преобразования, тем точнее получаемое значение. Этот параметр может принимать одно из следующих значений:

```
ADC_SampleTime_3Cycles;
ADC_SampleTime_15Cycles;
ADC_SampleTime_28Cycles;
ADC_SampleTime_56Cycles;
ADC_SampleTime_84Cycles;
ADC_SampleTime_112Cycles;
ADC_SampleTime_144Cycles;
ADC_SampleTime_480Cycles;
```

2. ADC_SoftwareStartConv(ADC_TypeDef * ADCx) включает выбранное программное обеспечение ADC, запускает преобразование каналов.

3. ADC_GetFlagStatus(ADC_TypeDef* ADCx,ADC_FLAG) проверяет, установлен ли указанный флаг АЦП.

Параметры:

– ADCx – номер АЦП, где x может быть 1, 2 или 3;

– ADC_FLAG – задает флаг для проверки. Этот параметр может принимать одно из следующих значений:

- ADC_FLAG_AWD – флаг детектора диапазона;
- ADC_FLAG_EOC – флаг окончания преобразования;
- ADC_FLAG_JEOC – флаг окончания преобразования инжектированной группы;
- ADC_FLAG_JSTRT – флаг начала преобразования инжектированной группы;
- ADC_FLAG_STRT – флаг начала преобразования регулярной группы;
- ADC_FLAG_OVR – флаг устанавливается, если выполнено преобразование и в регистр АЦП записаны данные, но при этом не были прочитаны данные предыдущего преобразования.

4. ADC_GetConversionValue(ADC_TypeDef*ADCx) возвращает последний результат преобразования данных для регулярного канала ADCx.

Пример выполнения одиночного преобразования

```
ADC_SoftwareStartConv(ADC1);
while(ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
return ADC_GetConversionValue(ADC1);
```

3.3 Индивидуальные задания

На основе заданий лабораторной работы №2 выполнить анализ значения напряжения на заданном канале и переключить мигание светодиода, используя данные из таблицы 3.1.

Таблица 3.1 – Индивидуальные задания к лабораторной работе №3

Вариант	Пороговое значение АЦП	Номер отслеживаемых каналов	Группа
1	3610	11, 3, 8	Регулярные
2	5439	1, 13, 5	Регулярные
3	3529	13, 8, 6	Регулярные
4	5822	15, 2, 15	Регулярные
5	2937	6, 13, 12	Регулярные
6	2126	11, 15, 10	Регулярные
7	3638	7, 15, 16	Регулярные
8	2189	14, 9, 8	Регулярные
9	5591	9, 2, 12	Регулярные
10	4858	16, 10, 11	Регулярные
11	5654	13, 4, 5	Регулярные
12	2942	7, 6, 5	Регулярные
13	5274	13, 9, 6	Регулярные
14	4571	1, 10, 2	Регулярные
15	5313	12, 7, 2	Регулярные
16	2454	3, 14, 13	Регулярные
17	3276	8, 12, 6	Регулярные
18	2754	9, 16, 5	Регулярные

3.4 Контрольные вопросы

1. Что такое аналого-цифровое преобразование?
2. На что влияет эффект квантования уровня сигнала?
3. Что такое дискретизация сигнала?
4. Как вычислить необходимую частоту дискретизации сигнала?
5. В чем отличие регулярных и инжектированных каналов АЦП?
6. Какие параметры задаются при настройке АЦП?

Лабораторная работа №4
Интерфейс передачи данных USART и протоколы RS-232, RS-485, RS-422

Цель работы: сформировать общее представление о работе универсального синхронно-асинхронного приёмопередатчика (модуль USART) микроконтроллера STM32F407; изучить особенности передачи цифровых сигналов.

4.1 Теоретические сведения

Сокращения типа RS-232, RS-485, RS-422…RS (Recommended Standard) – это рекомендованный стандарт . Все RS-протоколы можно разделить на полудуплексные (half-duplex) и дуплексные (full-duplex). Есть еще такой вид протоколов как симплексные (simplex), но, в виду ряда причин, в компьютерной технике они не применяются.

Симплексные протоколы позволяют передавать данные только в одну сторону, т. е. только с передатчика на приёмник, но не обратно. *Хороший пример симплексного протокола – FM-радио или телевидение, если только не принимать во внимание возможность позвонить на радиостанцию. Применяется в тех случаях, когда надо просто передать информацию какому-либо устройству без необходимости подтверждения и обратной связи.*

Полудуплексные протоколы снимают главное ограничение симплексных протоколов – односторонняя связь. Они позволяют двум устройствам обмениваться информацией, причём оба устройства могут быть и приемниками и передатчиками, но это не может осуществляться одновременно (рисунок 4.1). То есть каждое устройство может либо передавать, либо принимать данные (*классический протокол RS-485 – полудуплексный*).

Дуплексные протоколы. Применение дуплексного протокола позволяет производить и приём, и передачу информации одновременно, т. е. оба устройства могут быть и приёмником, и передатчиком одновременно (рисунок 4.2). *Например, USART и RS-232 – дуплексные протоколы.*

Наиболее простыми и одновременно часто используемыми в индустрии являются два протокола – RS-232 и RS-485. Важное их отличие заключается в том, что протокол RS-232 использует небалансный (unbalanced) сигнал, в то время как RS-422/RS-485 используют балансный (balanced) сигнал.

Небалансный сигнал передается по несбалансированной линии, представляющей собой сигнальную землю и одиночный сигнальный провод, уровень напряжения на котором используется, чтобы передать или получить двоичные единицы или нуль (рисунок 4.3).

Балансный сигнал, напротив, передаётся по сбалансированной линии, которая представлена сигнальной землей и парой проводов, разница напряжений между которыми используется для передачи/приёма бинарной информации (все вместе составляет экранированную витую пару). Сбалансированный сигнал передается быстрее и дальше, чем несбалансированный (рисунок 4.4).

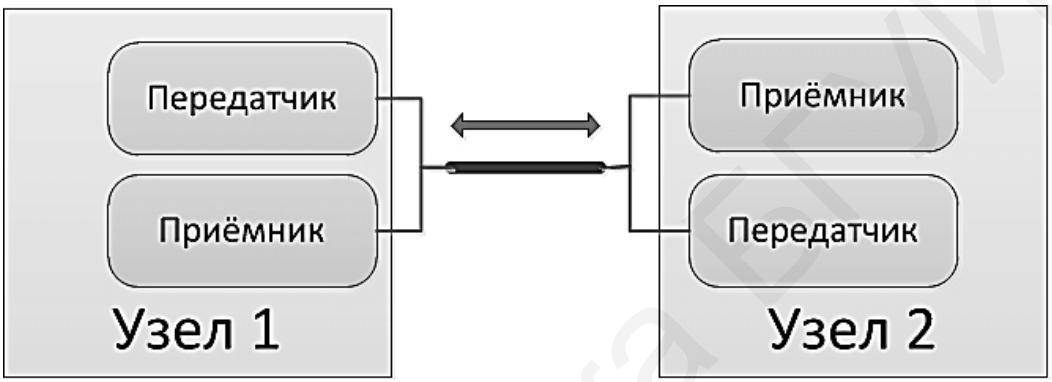

Рисунок 4.1 – Схематичное изображение работы полудуплексного протокола

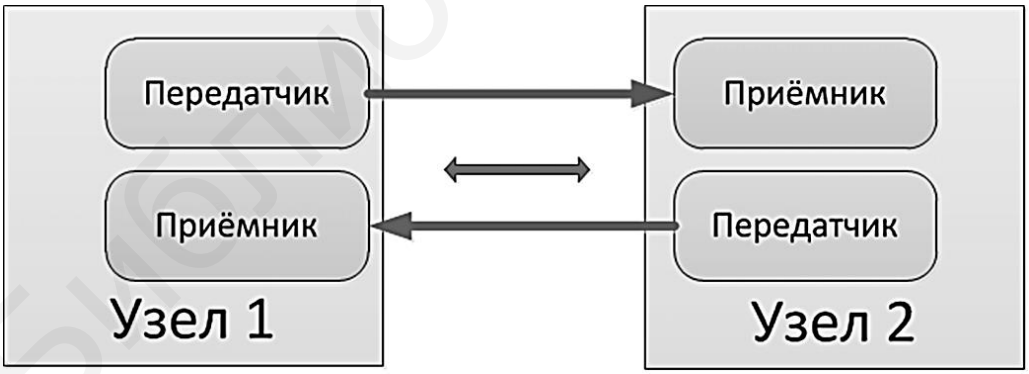

Рисунок 4.2 – Схематичное изображение работы дуплексного протокола

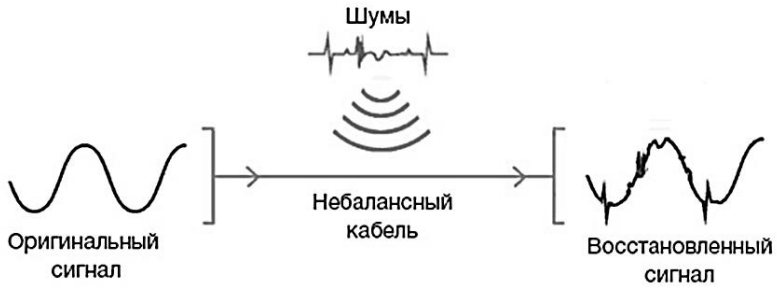

Рисунок 4.3 – Пример небалансного сигнала, который передается по несбалансированной линии

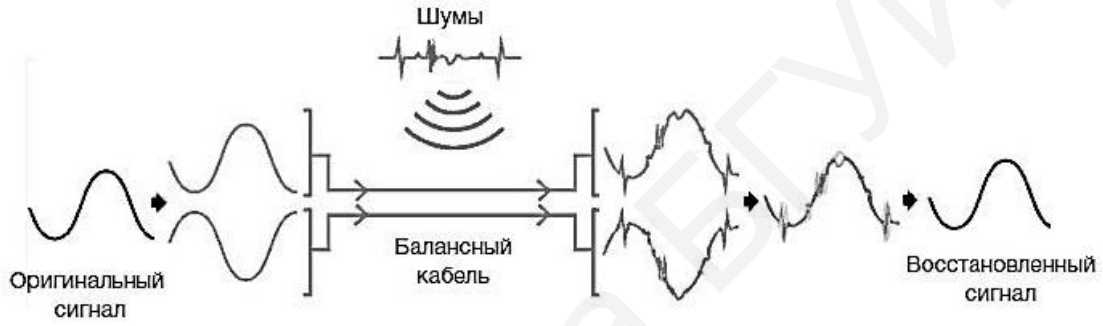

Рисунок 4.4 – Пример балансного сигнала, который передается по сбалансированной линии

Различают также параллельные и последовательные интерфейсы: *в параллельном интерфейсе* данные передаются по нескольким линиям одновременно, *в последовательном интерфейсе* – одна информационная линия, передача осуществляется побитно, т. е. биты информации передаются последовательно один за одним (рисунок 4.5).

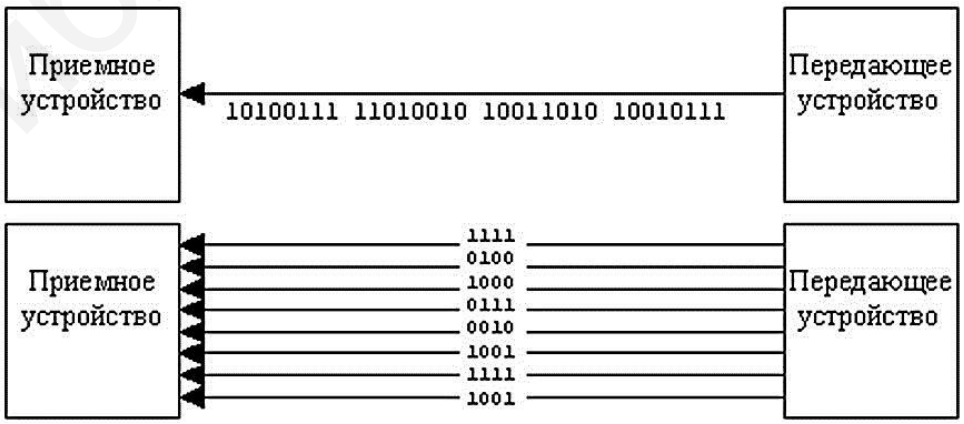

Рисунок 4.5 – Последовательные и параллельные интерфейсы передачи данных

USART является последовательным интерфейсом передачи данных. Формат передаваемых USART данных показан на рисунке 4.6. Получив стартовый бит, приёмник выбирает из линии биты данных через определённые интервалы времени. Очень важно, чтобы тактовые частоты приёмника и передатчика были одинаковыми, допустимое расхождение – не более 10 %. Скорость передачи по USART или RS-232C может выбираться из ряда: 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19 200, 38 400, 57 600, 115 200 бит/с. Все сигналы RS-232C передаются специально выбранными уровнями, обеспечивающими высокую помехоустойчивость связи. Отметим, что данные передаются в инверсном коде (логической единице соответствует низкий уровень, логическому нулю – высокий уровень).

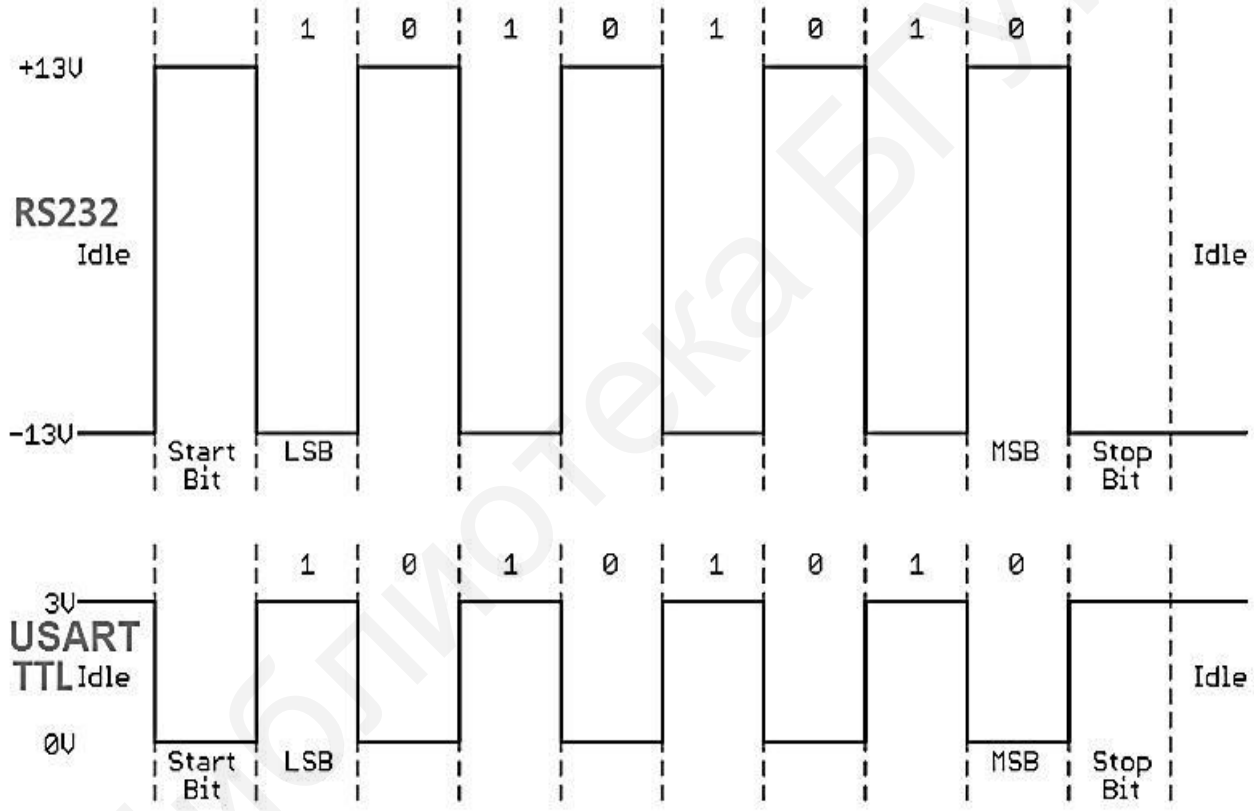

Рисунок 4.6 – Формат передаваемых данных

4.1.1 USART в микроконтроллерах STM32

USART в микроконтроллерах STM32 предоставляет гибкие средства для полнодуплексного обмена данными с внешними устройствами в последовательном формате с возможностью поддержки сигналов CTS/RTS; под-

держивает полудуплексный обмен по однопроводной линии; может работать в широком диапазоне скоростей передачи.

В мультибуферном режиме DMA достигается высокая скорость передачи данных, максимальное значение составляет 3 Мбит/с. Также для режима DMA характерно: поддержание однонаправленной передачи в синхронном режиме; мультипроцессорная связь; LIN (local interconnection network) – сеть для локальной связи; smartcard-протокол; инфракрасный протокол в соответствии со спецификацией IrDA (infrared data association) SIR ENDEC.

Основные возможности:

– асинхронная полнодуплексная связь;

– асинхронная однопроводная полудуплексная связь;

– настраиваемый метод оверсэмплинга (супердискретизации) даёт возможность выбора между скоростью передачи и допустимым отклонением скорости;

– передатчик и приёмник используют общую программируемую скорость передачи, которая может настраиваться в широких пределах; максимальное значение достигает 3 Мбит/с при 8-кратном оверсэмплинге;

– программируемая длина слова (8 или 9 бит);

– настраиваемое количество стоп-битов (один или два);

– в LIN-режиме поддерживается отправка и обнаружение приёмником Break-посылки (генерируется 13-битная и детектируется 10/11-битная);

– имеется выход тактового сигнала для синхронной передачи;

– IrDA SIR-кодек для инфракрасной связи (поддерживается длительность бита 3/16 в нормальном режиме);

– интерфейс Smartcard поддерживает асинхронный протокол смарт-карт, как определено в стандарте ISO 7816-3; используется 0,5 либо 1,5 стоп-битов в операциях со смарт-картой;

– конфигурируемая мультибуферная связь с использованием DMA (direct memory access);

– флаги, устанавливаемые при обнаружении событий во время обмена данными (приёмный буфер заполнен; буфер для передачи пуст; передача завершена);

– контроль чётности (можно настроить передатчик на формирование бита чётности и приёмник на контроль бита чётности);

– четыре флага, устанавливаемые при обнаружении ошибок (ошибка переполнения; обнаружен шум в принимаемом сигнале; ошибка фрейма; ошибка чётности);

– десять источников прерывания USART, связанных с флагами регистра состояния SR (изменение состояния CTS; обнаружение посылки LIN Break; регистр данных передатчика пуст; передача завершена; регистр данных приёмника заполнен; обнаружение события «линия свободна» (Idle line); ошибка переполнения; ошибка фрейма; обнаружение шума; ошибка чётности);

– мультипроцессорная связь (переход в тихий режим, если не произошло сопоставление адреса);

– пробуждение из тихого режима при обнаружении свободной линии (Idle line) или при обнаружении адресной метки;

– два режима пробуждения приёмника – по адресному биту (9-й, старший бит) или при обнаружении, что линия свободна.

4.2 Порядок выполнения работы

4.2.1 Настройка библиотек

Для работы с USART необходимо при создании проекта во вкладке Repository-Peripherals выбрать следующие компоненты:

– M4 CMSIS Core;

– CMSIS BOOT;

– RCC (отвечает за тактирование);

– GPIO;

– USART.

4.2.2 Передача данных по USART

4.2.2.1 Инициализация порта ввода/вывода для работы передатчика

Инициализация структуры, содержащая настройки порта ввода/вывода:

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

GPIO_InitTypeDef PORT_SETUP;
PORT_SETUP.GPIO_Mode = GPIO_Mode_AF;
PORT_SETUP.GPIO_OType = GPIO_OType_PP;
PORT_SETUP.GPIO_Pin = GPIO_Pin_2;
PORT_SETUP.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_Init(GPIOA, &PORT_SETUP);
```

```
GPIO_PinAFConfig(GPIOA,GPIO_PinSource2,GPIO_AF_USART2);
```

Таким образом, порт А вывод 2 настраивается как выход передатчика Tx.

4.2.2.2 Инициализация USART

Для инициализации USART используется структура типа USART_InitTypeDef. Данная структура имеет следующие поля:

1. USART_BaudRate – поле принимает численное значение и содержит скорость обмена данными по USART. Скорость выбирается из ряда: 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19 200, 38 400, 57 600, 115 200 бит/с.

2. USART_HardwareFlowControl – поле, дающее возможность подключения дополнительных выводов, которые принимают значение единицы при осуществлении передачи или приёма информации. Как правило, данные дополнительные выводы не используются, и значение по умолчанию для данного поля – USART_HardwareFlowControl_None.

3. USART_Mode – виды режимов USART:

– USART_Mode_Rx – осуществляется только приём данных;

– USART_Mode_Tx – осуществляется только передача данных;

– USART_Mode_Rx|USART_Mode_Tx – осуществляется и приём, и передача данных.

4. USART_Parity – позволяет автоматически контролировать целостность данных методом контроля битовой чётности. Если контроль целостности данных не производится, устанавливается значение USART_Parity_No.

5. USART_StopBits – поле настраивает длительность бита Стоп при пересылке отдельного байта информации. Данное поле может принимать следующие значения:

– USART_StopBits_1 – длительность бита Стоп равна длительности информационного бита;

– USART_StopBits_0_5 – длительность бита Стоп равна половине длительности информационного бита;

– USART_StopBits_2 – длительность бита Стоп равна двойной длительности информационного бита;

– USART_StopBits_1_5 – длительность бита Стоп равна полуторной длительности информационного бита.

6. USART_WordLength – поле настраивает количество битов в передаваемом байте. Возможны следующие значения:

– USART_WordLength_8b – байт содержит 8 бит;

– USART_WordLength_9b – байт содержит 9 бит.

Приведём пример инициализации USART2 с помощью управляющей структуры, содержащей настройки USART:

```
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);
//Тактирование модуля USART происходит через шину APB1

    USART_InitTypeDef USART_setup;
    USART_setup.USART_BaudRate = 9600;
    USART_setup.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
    USART_setup.USART_Mode =  USART_Mode_Tx;
    USART_setup.USART_Parity = USART_Parity_No;
    USART_setup.USART_StopBits = USART_StopBits_1;
    USART_setup.USART_WordLength = USART_WordLength_8b;
    USART_Init(USART2, &USART_setup);
    USART_Cmd(USART2, ENABLE); //включение USART2
```

4.2.2.3 Примеры функций для передачи данных по USART

Функция для пересылки байта данных, использующая стандартную функцию USART_SendData:

```
int putcharx(uint8_t ch)
{
    while (USART_GetFlagStatus(USART2, USART_FLAG_TXE) == RE-
SET);
    USART_SendData(USART2, (uint8_t)ch);
    return ch;
}
```

Функция для передачи байта данных, использующая управляющие регистры микроконтроллера:

```
void send_to_USART(uint8_t data)
{
    while(!(USART2->SR & USART_SR_TC));
    USART2->DR=data;
}
```

Функция send_str для передачи строки данных. В качестве входных переменных функция получает указатель на передаваемую строку

```
void send_str(char * string)
{
    uint8_t i=0;
    while(string[i])
    {
        send_to_USART(string[i]);
        i++;
    }
}
```

4.2.3 Приём данных и прерывания по USART

Можно настроить микроконтроллер так, чтобы при приёме данных происходило прерывание. Для этого нужно объявить переменную, в которую будут заноситься новые данные, и счётчик принятых байтов:

```
uint8_t receivedData[16];// Счётчик принятых байтов
uint8_t receivedDataCounter = 0;
```

Инициализация USART для приёма данных может выглядеть следующим образом:

```
// Тактирование модулей USART и GPIO
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

//Настройка GPIO
GPIO_InitTypeDef gpio;
GPIO_StructInit(&gpio);

gpio.GPIO_Mode = GPIO_Mode_AF;
gpio.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_10;
gpio.GPIO_Speed = GPIO_Speed_50MHz;
gpio.GPIO_OType = GPIO_OType_PP;
gpio.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_Init(GPIOA, & gpio);

// Настройка портов GPIO на альтернативную функцию
GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_USART1);
GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_USART1);

// Настраиваем модуль USART
```

```
USART_InitTypeDef usart;
USART_StructInit(&usart);
usart.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
usart.USART_BaudRate = 9600;
USART_Init(USART1, &usart);

// настройка прерываний от USART
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);

// Включаем прерывания и запускаем USART
NVIC_EnableIRQ(USART1_IRQn);
SART_Cmd(USART1, ENABLE);

// включаем прерывание по приему данных
USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
```

Приведем пример программы обработчика прерываний (данная подпрограмма принимает 16 байт данных, после этого приём прекращается):

```
void USART1_IRQHandler()
{
// Убеждаемся, что прерывание вызвано новыми данными в регистре
данных
if (USART_GetITStatus(USART1, USART_IT_RXNE) != RESET)
{
// принимаем данные
receivedData[receivedDataCounter] = USART_ReceiveData(USART1);
// Увеличиваем значение счетчика
receivedDataCounter ++;
// Приняли 16 байт – выключаем
if  (receivedDataCounter == 16)
{
USART_ITConfig(USART1, USART_IT_RXNE, DISABLE);
}
// Очищаем флаг прерывания
USART_ClearITPendingBit(USART1, USART_IT_RXNE);
}
}
```

4.2.4 Приём и передача данных на персональный компьютер

Для приёма и передачи данных на персональном компьютере может использоваться программа Tera Term – это свободная, распространяемая по лицензии BSD, служебная программа для работы через реальные и виртуальные COM-порты (RS232, USB) в интерактивном режиме или в режиме командной строки с возможностью управления сессией при помощи встроенного макроязыка.

Программа Tera Term позволяет автоматизировать подключение к устройствам по протоколам Telnet, SSH1, SSH2 через COM-порты. Может использоваться в следующих целях:

– для сохранения логов;

– программирования устройств;

– перезагрузки устройств.

Характеристика программы Tera Term:

– поддерживает протоколы: IPv4, IPv6, Telnet, SSH1, SSH2, Kermit, XMODEM, ZMODEM, B-PLUS, Quick-VAN, SCP;

– эмулирует терминалы: DEC VT100, VT101, VT102, VT282, VT320, VT382, VT420, VT520, VT525, Tektronix TEK4010, Altair 8800;

– поддерживает раскладки клавиатуры: английскую, русскую, японскую;

– имеет многоязыковую поддержку, в том числе переведена на русский язык;

– поддерживает кодировки: ASCII, UTF-8, KOI8-R;

– поддерживает скриптовый язык: Tera Term Language;

– поддерживает соединения: TCP/IP, COM-port;

– поддерживает создание логов сессий;

– работает с терминалами в интерактивном режиме.

4.3 Индивидуальные задания

На основе заданий лабораторной работы №3 выполнить передачу данных на персональный компьютер о состоянии светодиодов. Например, при включении одного светодиода передать на ПК строку VD1 on, при выключении – строку VD1 off. При использовании нескольких светодиодов им необходимо присвоить имена (основываясь на порядковом номере или на цвете светодиода) и также передавать их состояния в момент переключения.

Усложнённые знания содержат возможность включения или выключения мерцания светодиода командой от персонального компьютера. Формат команды необходимо разработать самостоятельно.

4.4 Контрольные вопросы

1. Каковы особенности симплексных интерфейсов передачи данных?
2. Каковы особенности полудуплексных интерфейсов передачи данных?
3. Каковы особенности дуплексных интерфейсов передачи данных?
4. Чем отличаются параллельный и последовательный интерфейсы?
5. Есть ли отличия в формате передаваемых данных при использовании протоколов UART и RS232?
6. Какие основные возможности блока USART в микроконтроллере STM32?
7. Как задать скорость работы блока USART?
8. Сколько битов используется при передаче одного байта?

Список использованных источников

1. Reference manual. STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced ARM. ®. -based 32-bit MCUs [Электронный ресурс]. – Режим доступа : http://www.st.com/resource/en/reference_manual/dm00031020.pdf.

2. STM32 TIMER general-purpose. Описание базового молуля [Электронный ресурс]. – Режим доступа : http://mycontroller.ru/old_site/stm32-timer-general-purpose-opisanie-bazovogo-molulya/default.htm.