Оглавление

[Лабораторная работа №1. Создание базового проекта для микроконтроллера STM32F407. Порты ввода вывода 4](#_Toc506996978)

[Лабораторная работа №2. Таймеры. Настройка тактирования микроконтроллера STM32F407 25](#_Toc506996980)

[Лабораторная работа №3. Работа с аналого-цифровым преобразователем микроконтроллера STM32F407 44](#_Toc506996982)

[Лабораторная работа №4. Интерфейс передачи данных USART и протоколы RS-232, RS-485, RS-422 54](#_Toc506996984)

# Лабораторная работа №1

# Создание базового проекта для микроконтроллера STM32F407. Порты ввода вывода

1. **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. **Теоретические сведения**

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

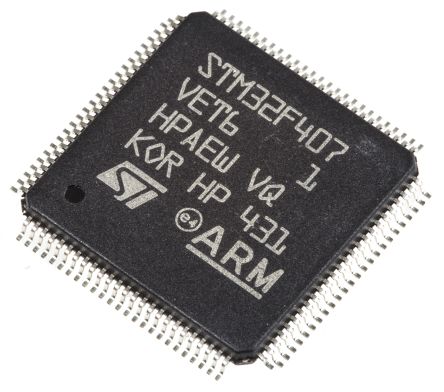


Рисунок 1.1 – Микроконтроллер STM32F407VET6

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

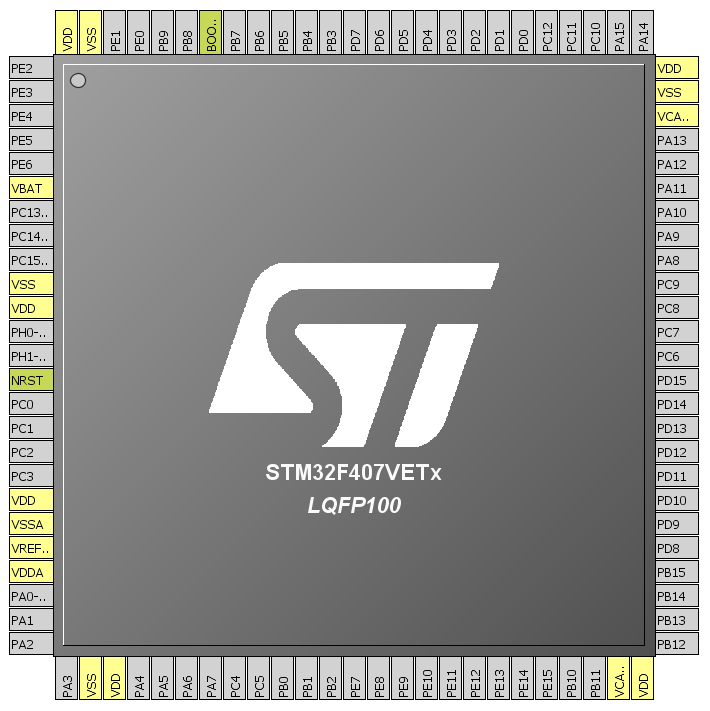


Рисунок 1.2 – Схематичное изображение микроконтроллера STM32F407VET6

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 напряжение ниже земли (например –3,3В) или выше напряжения питания микроконтроллера (например +6В) (Рисунок 3).

Существуют следующие режимы работы вывода общего назначения:

а) Высокоимпедансный вход. На выходе установлена пара комплементарных полевых транзисторов (один p-типа, другой n-типа). Полевой транзистор в закрытом состоянии имеет почти бесконечное сопротивление между стоком и истоком. В этом режиме оба транзистора в закрытом состоянии, поэтому вывод не подключён ни к земле, ни к питанию – поэтому ведёт себя как не подключённый к схеме. Данный режим используется для приема данных, когда логическая 1 (3,3 В) или логический 0 (0 В) формируется внешними схемами.

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

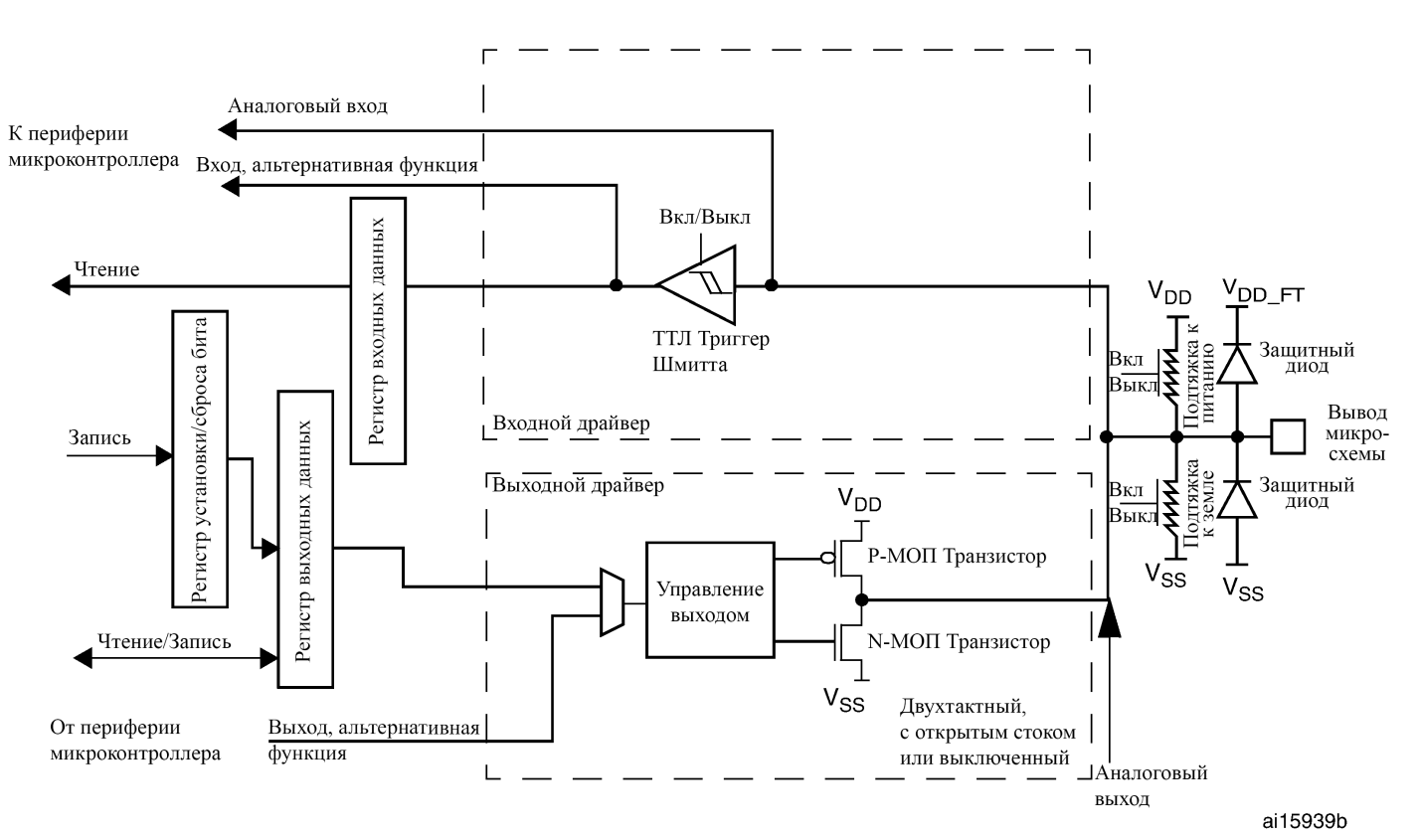


Рисунок 3 – Структурная схема интерфейса ввода/вывода общего назначения

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

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

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

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

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

1. **Порядок выполнения работы.**

**3.1. Установка необходимого программного обеспечения**

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

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

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

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

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

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



Рисунок 4 – Исследовательская плата STM32F4DISCOVERY

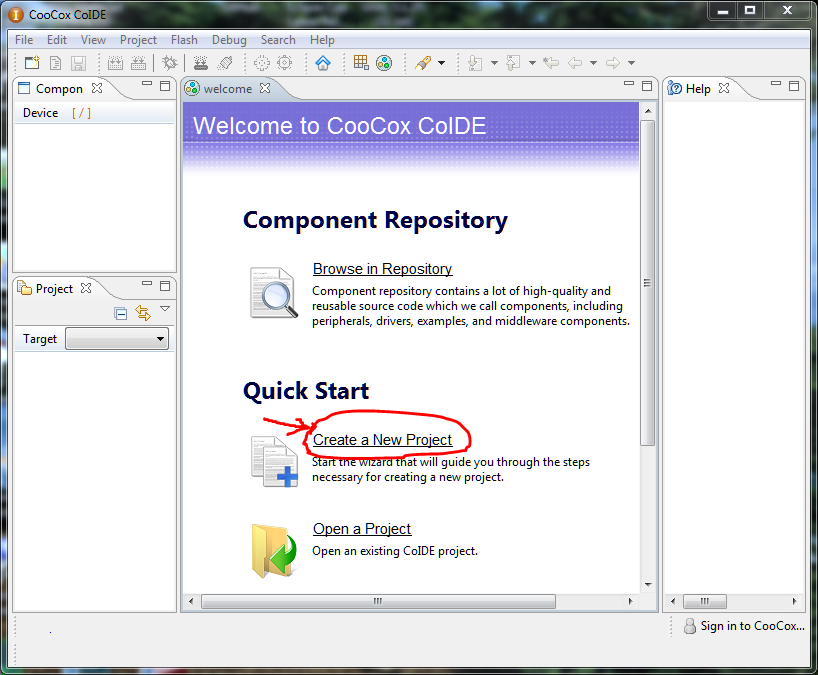


Рисунок 5 – Создание нового проекта

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

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

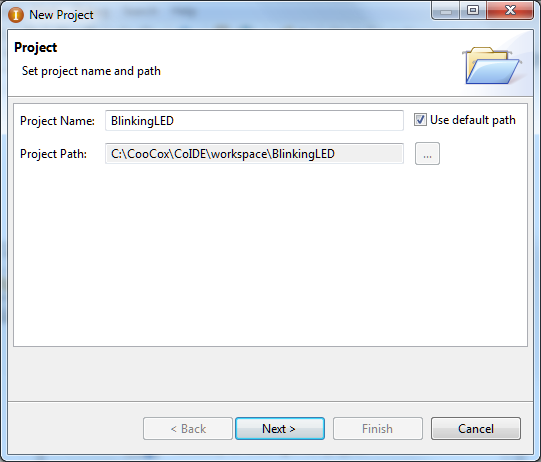


Рисунок 6 – Выбор имени проекта

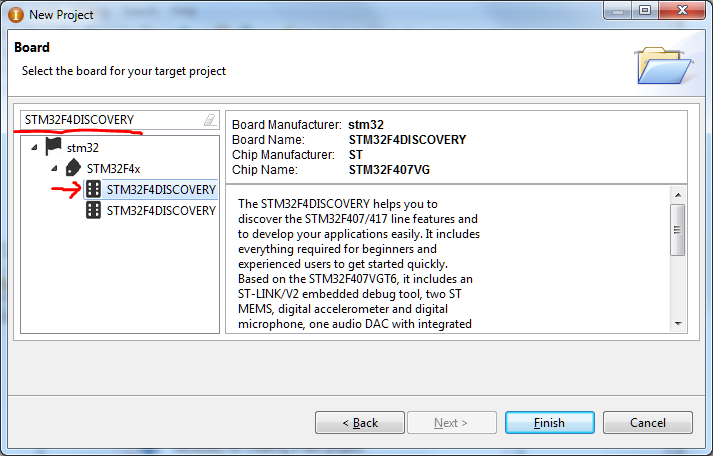


Рисунок 7 – Выбор отладочной платы

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

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

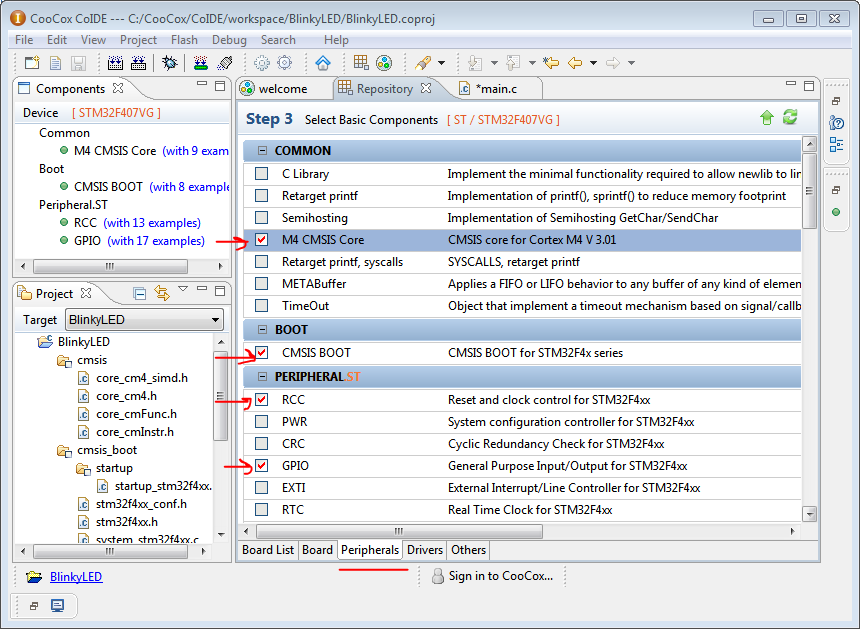


Рисунок 8 – Подключение периферии в проект

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

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

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

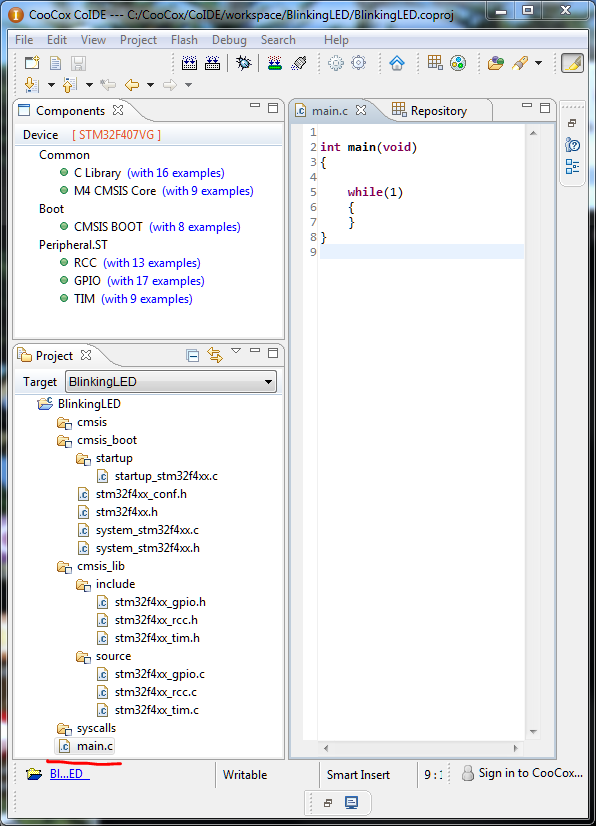


Рисунок 9 – Открытие файла исходного кода main.c

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

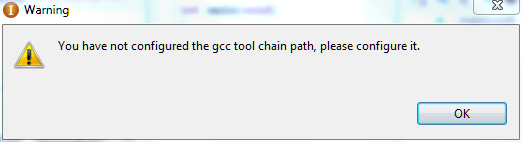


Рисунок 10 – Ошибка конфигурации компилятора

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

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

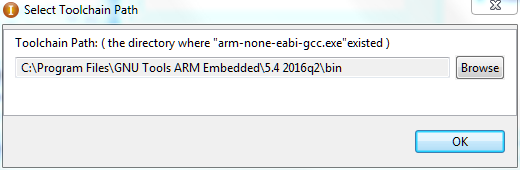


Рисунок 11 – Настройка расположения компилятора

После этого произойдёт построение проекта.

3.3. Написание программы

* 1. Подключение необходимых библиотек

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

**#include** "stm32f4xx.h"

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

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

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

stm32f4xx\_gpio.h – подключение компонента GPIO.

stm32f4xx\_rcc.h – подключение компонента RCC.

* 1. Настройка интерфейса ввода/вывода общего назначения

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

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

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

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

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

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

GPIO\_InitTypeDef GPIO\_InitStruct;

Далее структуру нужно инициализировать значениями по умолчанию:

GPIO\_StructInit(&GPIO\_InitStruct);

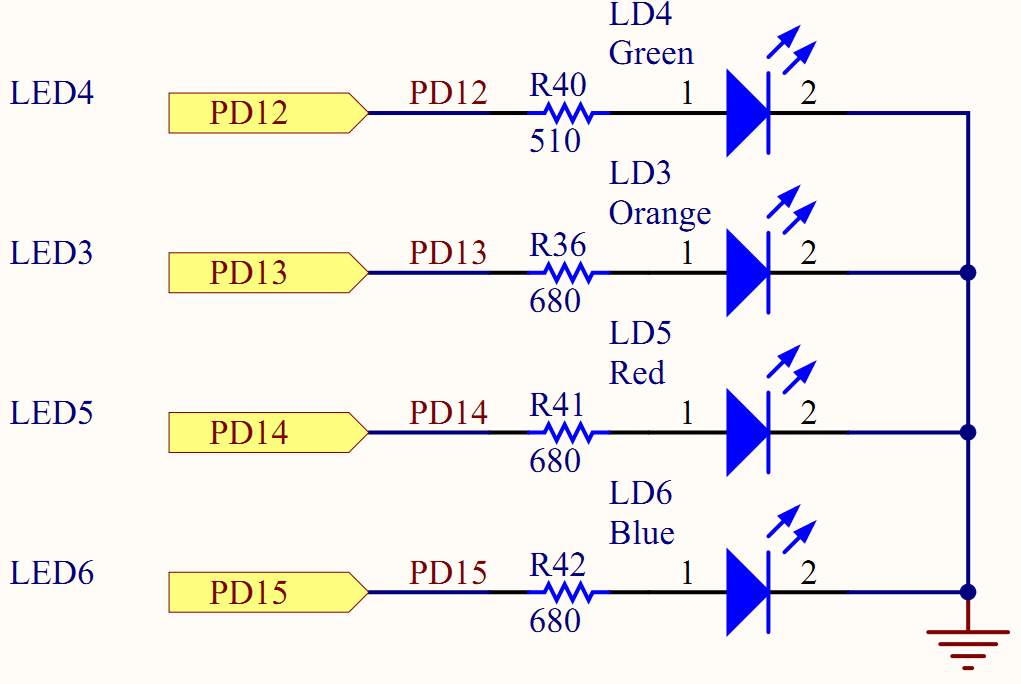


Рисунок 12 – Схема принципиальная подключения светодиодов на отладочной плате STM32F4DISCOVERY

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

В структуре нужно указать настройки порта, для этого пишем:

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

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

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

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

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

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

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

* GPIO\_Mode\_IN 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 тактирование, тем выше энергопотребление) и максимальная частота считывания (либо записи) информации с вывода (на вывод). Возможные значения:

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

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

В параметре GPIO\_OType указываем тип вывода.

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

В параметре GPIOx\_PUPDR задаем подтяжку.

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

Последнее что необходимо сделать – произвести инициализацию строкой:

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

Где GPIOD – имя порта;

GPIO\_InitStruct – название нашей структуры.

**3.3.3 Мигание светодиодом**

Для мигания используется две функции:

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

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

Обе функции принимают два параметра:

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

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

void Delay(volatile uint32\_t tick)

{

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

}

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

**while**(1)

{

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

Delay(500000);

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

Delay(500000);

}

Это значит, что микроконтроллер включает светодиод, затем 500000 раз инкрементирует переменную i (таким образом формируется задержка), затем выключает светодиод, далее снова производится задержка. После выполнения данных действий цикл начинается заново. Визуально происходит мигание светодиода.

**3.3.4Обработка внешних прерываний**

**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"

**Инициализация портов ввода вывода для работы со светодиодами и с кнопкой**

Для работы кнопки необходимо произвести инициализацию **GPIO**:

* инициализировать кнопку, расположенную на выводе **PA0** (Рисунок 1);

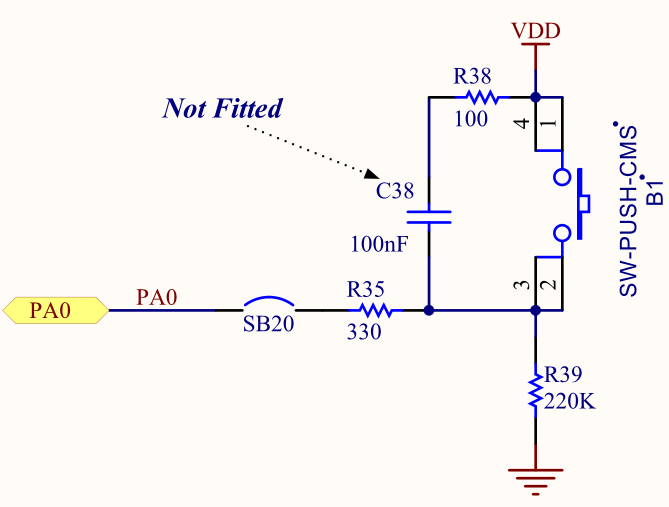


Рисунок 1 – Схема кнопки на отладочной плате (Всю схему смотрите в документации к плате *stm32f4Discovery\_UserManual.pdf*)

**Настройка внешних прерываний EXTI при нажатии кнопки на PA0**

**Прерывание (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 может быть множество. Например, если речь о таком блоке периферии как [UART](http://easystm32.ru/interfaces/14-uart-in-stm32-part-0), 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 нашем случае прервется выполнение основного кода вашей программы и управление будет передано некоторому другому куску кода который называется обработчиком прерывания. Таких обработчиков достаточно много, ибо периферийных устройств в STM32 предостаточно. Стоит отметить важный момент: В случае возникновения двух разных прерываний от одного блока периферии возникает одно и тоже прерывание. Например если произойдет прерывание по приёму байта через UART и прерывание по завершению передачи через тот же UART, то в обоих случаях будет вызван один и тот же обработчик. Для того чтоб 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.

Что нужно знать о прерываниях:

* они все независимо включаются/выключаются;
* имеют приоритет;
* могут быть вызваны программно;
* если для прерывания нет обработчика, а оно возникло, то будет вызван обработчик по умолчанию;

Т.к. 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 спец регистре;
* улавливает импульсы длительность которых ниже меньше периода частоты APB2.

Сама структура связи **EXTI** показана на рисунке 2.

Т.е. имеется16 EXTI линий к которым мы можем подключать выводы порта. Причем группируются они по номерам выводов. Т.е. 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.

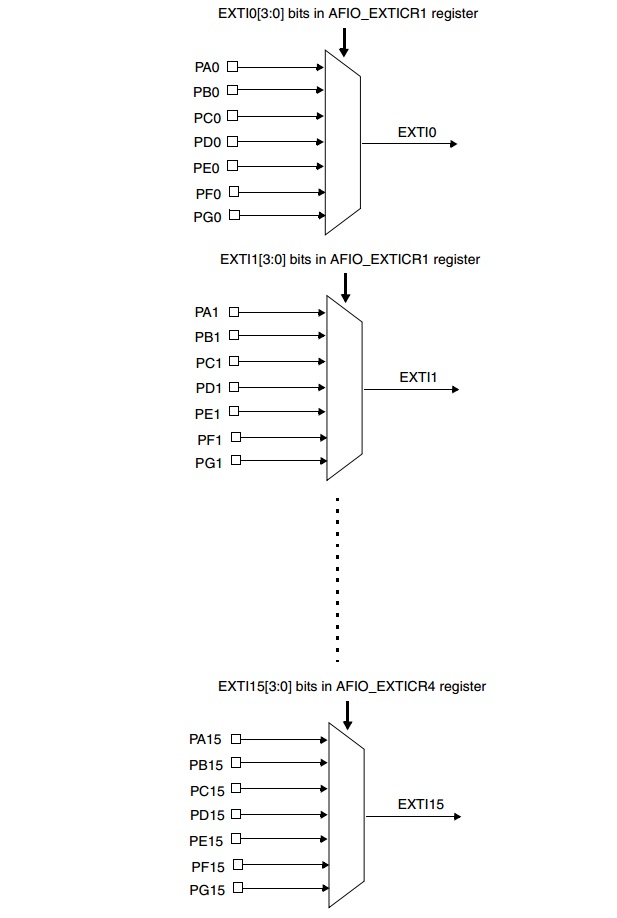


Рисунок 2 – Структура связи внешних прерываний **EXTI**

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, просто поднимается флажок. Может разбудить процессор или включить какую ни будь периферию, АЦП, например.
* EXTI\_Trigger. Триггер реакции. Выбираем на какой фронт реагировать. Может быть:
  + EXTI\_Trigger\_Rising – фронт подъема (при нажатии на кнопку);
  + 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 с другой переферии (Например USB WakeUp).

**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;

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 различных, Всех их можно посмотреть в документации, либо 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. Позволяет определить очередность выполнения обработчиков прерываний в пределах одинакового приоритета.

**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 16-ти (например, UART) разных прерываний. Следовательно, при входе в обработчик прерываний необходимо проверить – какое именно произошло прерывание, после обработки прерывания необходимо сбросить влаг прерывания. Если флаг прерывания не будет сброшен, то после выхода из функции обработки прерываня 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 переходной 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. **Индивидуальные задания:**

|  |  |
| --- | --- |
| **вариант** | **Задание** |
| **1** | Мигать синим светодиодом с переменной: F1 = 0.5Гц, F2 = 2Гц. Скважность = 2. По нажатию на кнопку происходит смена частоты. Повторное нажатие на кнопку возвращает первоначальное состояние. |
| **2** | Мигать попеременно красным и зелёным светодиодами. F = 3Гц. Скважность = 2. По нажатию на кнопку скважность = 4. Повторное нажатие на кнопку возвращает первоначальное состояние. |
| **3** | Мигать красным светодиодом с F=0.25Гц, зелёным светодиодом с F=1Гц. Скважность = 4. По нажатию на кнопку частота увеличивается в 2 раза. Повторное нажатие на кнопку возвращает первоначальное состояние. |
| **4** | Зажигать и гасить по очереди все четыре светодиода, F=1Гц. По нажатию на кнопку порядок мигания светодиодов меняется на противоположный. Повторное нажатие на кнопку возвращает первоначальное состояние. |
| **5** | Зелёный и красный светодиод горит попеременно с желтым светодиодом. F = 1Гц. Скважность = 5. По нажатию на кнопку частота F = 5 Гц. Повторное нажатие на кнопку возвращает первоначальное состояние. |
| **6** | Два любых светодиода горят постоянно, оставшиеся два горят попеременно. F = 3Гц. Скважность = 2. По нажатию на кнопку мигающие светодиоды начинают постоянно гореть, постоянно горевшие светодиоды начинают мигать. Повторное нажатие на кнопку возвращает первоначальное состояние. |
| **7** | Мигать синим и зеленым светодиодом. F = 1Гц. Скважность = 4. Красный светодиод горит постоянно. По нажатию на кнопку мигающие светодиоды начинают постоянно гореть, постоянно горевшие светодиоды начинают мерцать. Повторное нажатие на кнопку возвращает первоначальное состояние. |
| **8** | Мигать попеременно красным и зелёным светодиодами. F = 5Гц. Скважность = 3. По нажатию на кнопку F = 2 Гц, скважность = 2. Повторное нажатие на кнопку возвращает первоначальное состояние. |
| **9** | Мигать синим светодиодом с F=5Гц, зелёным светодиодом с F=1Гц. Скважность = 4. По нажатию на кнопку начинают мигать красный и желтый светодиоды (параметры теже). Повторное нажатие на кнопку возвращает первоначальное состояние. |
| **10** | Зажигать и гасить по очереди все четыре светодиода, F=1Гц. По нажатию на кнопку F=5Гц. Повторное нажатие на кнопку возвращает первоначальное состояние. |
| **11** | Зажигать и гасить по очереди все четыре светодиода, F1=1Гц, F2=2Гц, F3=3Гц, F4=4Гц. По нажатию на кнопку порядок мигания светодиодов меняется на противоположный. Повторное нажатие на кнопку возвращает первоначальное состояние. |
| **12** | Зажигать и гасить по очереди все четыре светодиода, F1=1Гц, F2=2Гц, F3=3Гц, F4=4Гц. По нажатию на кнопку F1=4Гц, F2=3Гц, F3=2Гц, F4=1Гц. Повторное нажатие на кнопку возвращает первоначальное состояние. |
| **13** | Мигать синим светодиодом с переменной частотой: F1 = 0.5Гц, F2 = 2Гц, F3 = 3Гц, F4 = 4Гц. Скважность = 2. По нажатию на кнопку происходит циклическая смена частоты. |
| **14** | Мигать красным светодиодом с переменной частотой: F1 = 5Гц, F2 = 4Гц, F3 = 3Гц, F4 = 2Гц. Скважность = 4. По нажатию на кнопку происходит циклическая смена частоты. |
| **15** | Мигать синим светодиодом с переменной частотой: F1 = 0.5Гц, F2 = 1Гц, F3 = 2Гц, F4 = 4Гц. Скважность = 0,5. По нажатию на кнопку происходит циклическая смена частоты. |
| **16** | Первоначально светодиоды не горят. По нажатию кнопки зажигать по очереди все четыре светодиода с переменной частотой от 40 до 1 Гц с шагом 0,5 Гц. |
| **17** | Первоначально светодиоды горят. По нажатию кнопки гасить по очереди все четыре светодиода с переменной частотой от 20 до 1 Гц с шагом 0,25 Гц. |

1. **Контрольные вопросы**
2. Каковы области применения микроконтроллеров серии STM32?
3. В каких режимах могут работать порты ввода вывода микроконтроллеров серии STM32?
4. Каковы особенности работы в режиме GPIO\_Mode\_IN?
5. Каковы особенности работы в режиме GPIO\_Mode\_OUT?
6. Каковы особенности работы в режиме GPIO\_Mode\_AF?
7. Каковы особенности работы в режиме GPIO\_Mode\_AN?
8. В чем отличие типов выходного сигнала GPIO\_OType\_PP и GPIO\_OType\_OD?
9. Что такое прерывание и как это используется при проектировании программного обеспечения микроконтроллеров серии STM32?
10. Какой алгоритм настройки прерывания?

# Лабораторная работа №2

# Таймеры. Настройка тактирования микроконтроллера STM32F407

1. **Цель работы:**

Сформировать общее представление о работе таймеров микроконтроллера. Изучить особенности формирования тактирующих сигналов микроконтроллера.

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 относится таймер TIM1 и TIM8, который способен выполнять все перечисленные выше операции. Кроме того, на основе данного таймера можно построить устройство, способное управлять трёхфазным электроприводом. В таблице 1 представлены типы таймеров микроконтроллера STM32F407, а также их основные технические характеристики.

Таблица 1 – Таймеры в микроконтроллере STM32F407VG

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Тип таймера | Таймер | Разрешение счётчика | Направление счёта | Значение пред-делителя | 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 65536 | Да | 4 | Да | 84 | 168 |
| General-Purpose | TIM2, TIM5 | 32-bit | Up, Down,  Up/Down | Любое число от 1 до 65536 | Да | 4 | Нет | 42 | 84 |
| TIM3, TIM4 | 16-bit | Up, Down,  Up/Down | Любое число от 1 до 65536 | Да | 4 | Нет | 42 | 84 |
| TIM9 | 16-bit | Up | Любое число от 1 до 65536 | Нет | 2 | Нет | 84 | 168 |
| TIM10, TIM11 | 16-bit | Up | Любое число от 1 до 65536 | Нет | 1 | Нет | 84 | 168 |
| TIM12 | 16-bit | Up | Любое число от 1 до 65536 | Нет | 2 | Нет | 42 | 84 |
| TIM13, TIM14 | 16-bit | Up | Любое число от 1 до 65536 | Нет | 1 | Нет | 42 | 84 |
| Basic | TIM6, TIM7 | 16-bit | Up | Любое число от 1 до 65536 | Да | 0 | Нет | 42 | 84 |

**Базовый модуль таймера**

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

Базовый модуль включает в себя:

* счетный регистр (TIMx\_CNT);
* регистр делителя (TIMx\_PSC);
* регистр автоматической перезагрузки (TIMx\_ARR);

Счетный регистр, регистр автоматической перезагрузки и регистр делителя может быть записан или прочитан программно. Это справедливо даже во время работы счетчика (рисунок 2.1).

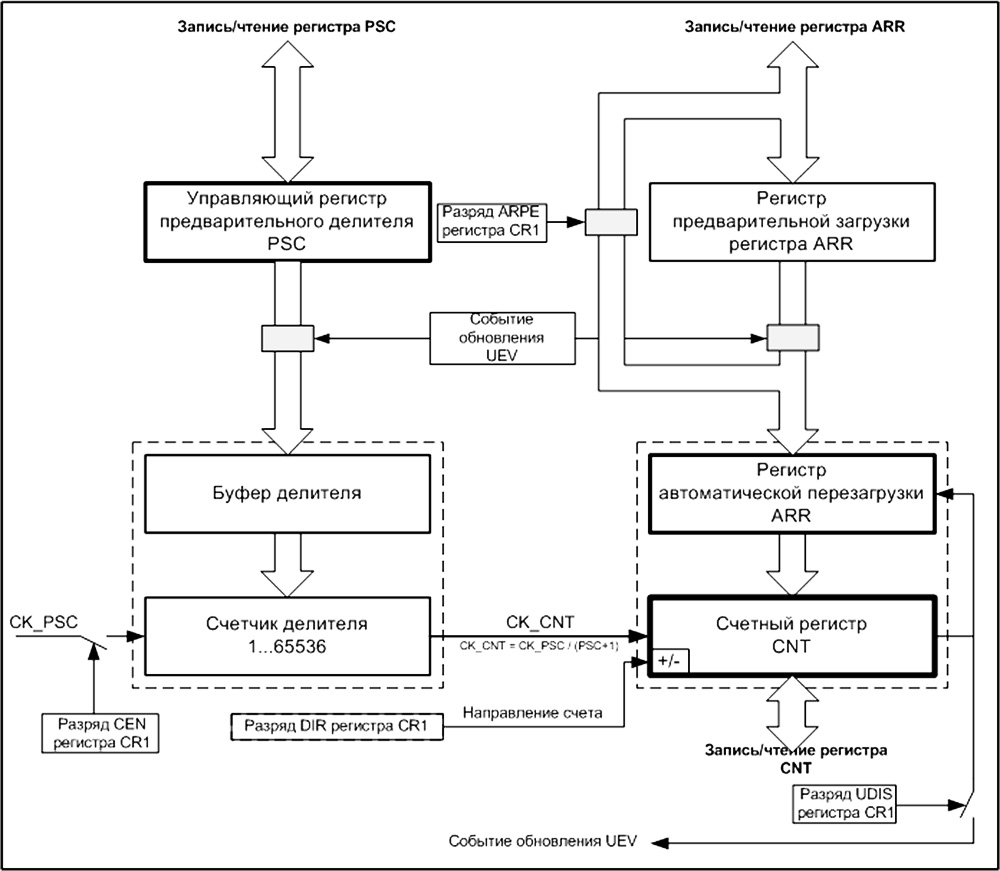


Рисунок 2.1 – Структура базового таймера

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 счетчика (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].

**Тактирование счетного 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 счетный регистр не напрямую, а через предварительный делитель с переменным коэффициентом деления (1…65536). Сигнал, подаваемый непосредственно на вход счетного регистра, называется CK\_CNT (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.

**Переполнение счетного регистра  и событие обновления UEV**

Перед началом счета в регистр TIMx\_CNT загружается его начальное значение. При счете «вверх» (up) это значение равно нулю, а при счете «вниз» (down) – содержимому регистра TIMx\_ARR. От каждого тактового импульса содержимое TIMx\_CNT увеличивается на 1 (при счете вниз уменьшается на 1), пока не достигнет своего максимального значения, которое определяется содержимым регистра TIMx\_ARR (при счете вниз – пока не достигнет своего минимального значения, т.е. нуля).

С приходом следующего тактового импульса произойдет сброс счетного регистра в ноль (при счете вниз в него автоматически будет записано значение регистра TIMx\_ARR). Этот переход и называется переполнением счетного регистра, в результате которого может формироваться событие обновления счетного регистра (UEV).

В микроконтроллере имеется возможность запретить формирование сигнала UEV. Делается это с помощью разряда UDIS регистра TIMx\_CR1: если он равен нулю – генерирование события обновления разрешено, единице – запрещено.

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

Событие обновления счетного регистра TIMx\_CNT используется для:

генерирования запроса прерывания от таймера;

генерирования запроса DMA от таймера;

запись нового значения регистров TIMx\_ARR и TIMx\_PSC;

управления другим таймером

Приведенная диаграмма иллюстрирует формирования события обновления по переполнению счетного регистра, который работает в режиме счета вверх (рисунок 2.2). В регистре TIMx\_ARR содержится 0х36:

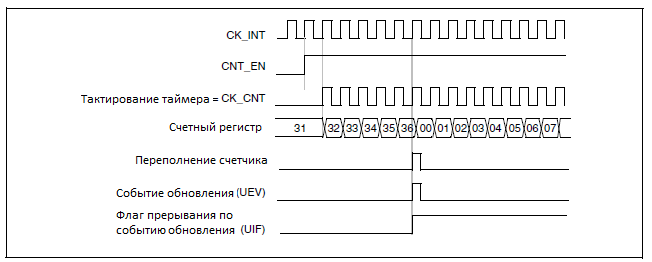
[](http://mycontroller.ru/old_site/wp-content/uploads/image105.png)

Рисунок 2.2 – Формирование события обновления по переполнению счетного регистра

**Таймеры общего назначения**

Таймеры общего назначения имеют до 4 независимых каналов, которые могут использоваться для:

* + Захвата сигнала
  + Сравнения
  + Генерации ШИМ
  + Генерации одиночного импульса

Таймеры микроконтроллера STM32F407 16 битные (то есть могут считать до 65535), 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 счетного регистра переписывается в специальный регистр, который называется регистром захвата (рисунок 2.3).

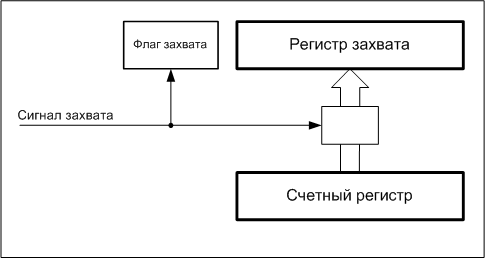


Рисунок 2.3 – Упрощенная схема работы таймера в режиме захвата

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

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

Микроконтроллеры STM32 имеют дополнительные возможности для режима захвата:

* каждый таймер общего назначения оснащен 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).

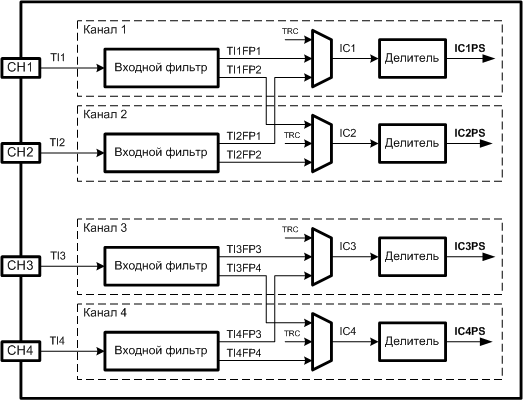


Рисунок 2.4 – Возможности подключения модулей захвата-сравнения

Итак, каналы 1 и 2 могут получать сигнал с вывода CH1 или CH2, а каналы 3 и 4 могут получать сигнал 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 счетного 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 CCRх 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 ШИМа и др.

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

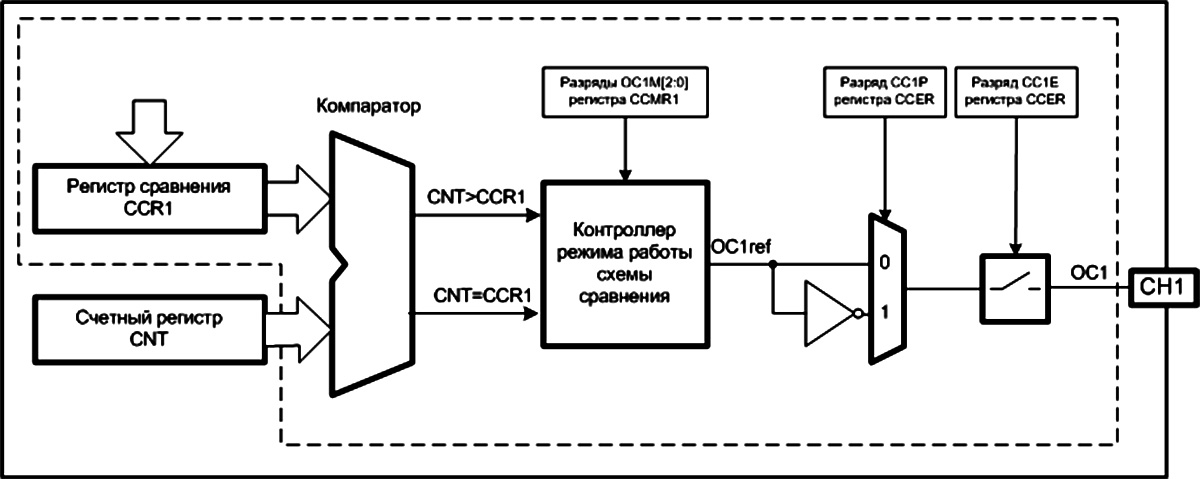


Рисунок 2.5 – Упрощенная схема работы таймера в режиме сравнения

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

Сигнал с компаратора, который сравнивает значения счетного 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. C помощью разряда CC1P регистра CCER можно выполнить инверсию сигнала (если записать в этот разряд 0), с помощью разряда CC1E регистра CCER можно отключить выход CH1 от схемы сравнения.

**Режим широтно-импульсной модуляции (ШИМ)**

Этот режим заслуживает отдельного рассмотрения, т.к. он весьма часто используется. ЩИМ (он же pulse-width modulation, PWM) – это изменение скважности импульсов при его постоянной длительности. Благодаря ШИМу 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.

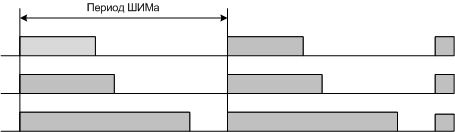
[](http://mycontroller.ru/old_site/wp-content/uploads/image166.png)

Рисунок 2.6 – Принцип широтно-импульсной модуляции

**Работа ШИМ посредством таймеров.** В данном режиме используются 3 регистра: Счетный регистр (TIMx\_CNT), регистр авто предварительной загрузки (TIMx\_ARR) и регистр сравнения (TIMx\_CCRx). В регистр TIMx\_ARR необходимо загрузить значение, до которого будет считать счетный регистр. Это значение определяет  период ШИМа. После этого, изменяя содержимое регистра сравнения, можно изменять скважность импульсов. Если значение счетного регистра равно нулю, сигнал OCxREF будет установлен равным единице (данный сигнал используется для управления внешним выводом микроконтроллера). Если значение счетного регистра больше, либо равно значению регистра TIMx\_CCRx, сигнал OCxREF будет установлен равным нулю (рисунок 2.7).

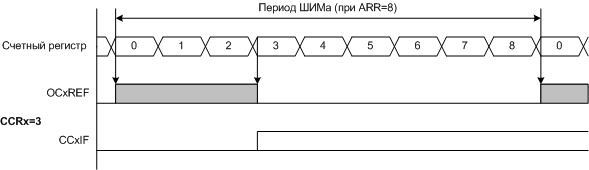
[](http://mycontroller.ru/old_site/wp-content/uploads/image168.png)

Рисунок 2.7 – Работа ШИМ посредством таймера. Прямой режим.

Режимы работы ШИМа.

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:

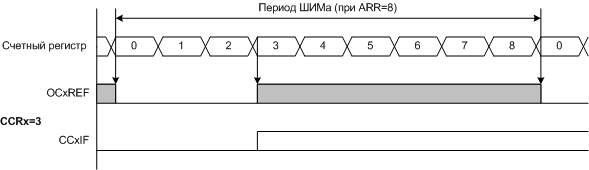
[](http://mycontroller.ru/old_site/wp-content/uploads/image169.png)

Рисунок 2.8 – Работа ШИМ посредством таймера. Инвертированный режим.

Также можно задать способ выравнивания ШИМ-сигнала: выравнивание 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 отлично от “00”, будет включен режим выравнивания по центру.  Временная диаграмма этого режима представлена на рисунке 2.9:

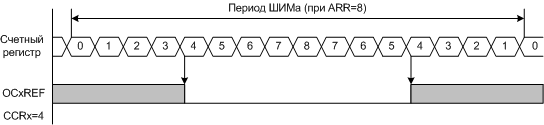
[](http://mycontroller.ru/old_site/wp-content/uploads/image171.png)

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

При CMS>00 счетный 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 счетного регистра и регистра сравнения. Событие совпадение (а значит флаг совпадения, прерывание, запрос DMA) генерируется в зависимости от разрядов CMS:

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

1. **Порядок выполнения работы**
2. **Настройка библиотек**

Создайте пустой проект. В нем 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 подпункте 3.3.

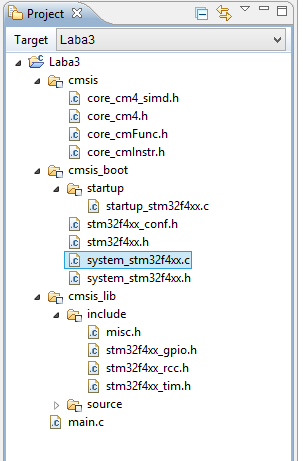


Рисунок 2.10 – Расположение файла в дереве проекта.

Необходимо согласиться с заменой (Рисунок 2.11)

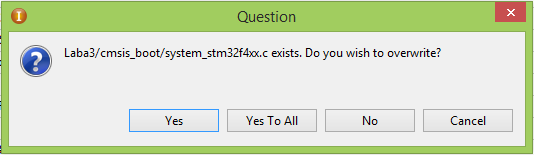


Рисунок 2.11 – Диалоговое окно замены файла

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.

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

1. **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, которую 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 некорректных данных в значения параметров структуры. При объявлении структуры на языке C 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..65565;;

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 (N+1) переполнений.

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);

1. **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.

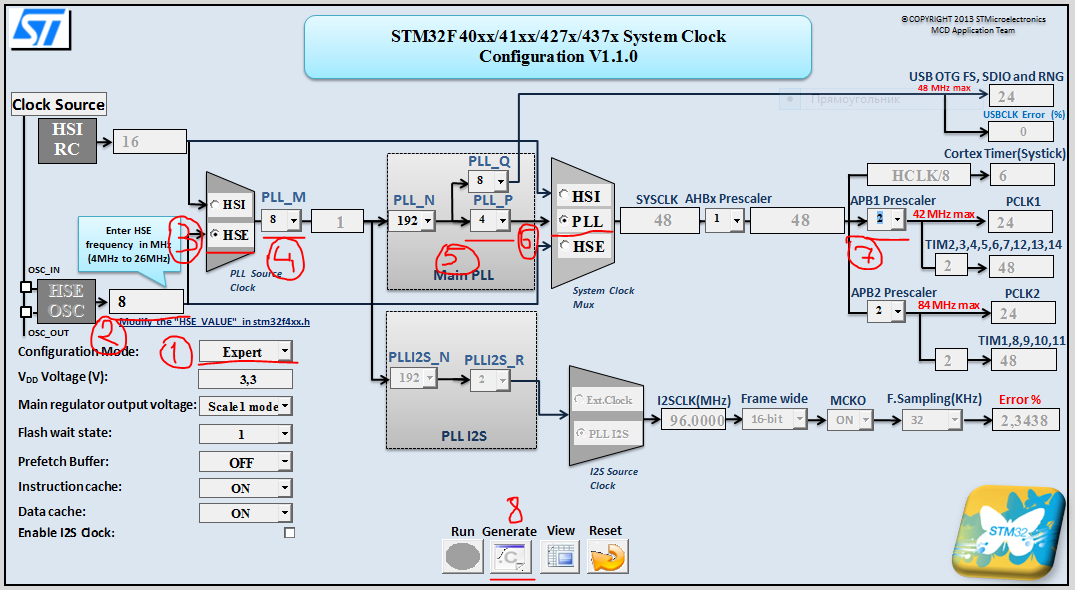


Рисунок 2.12 – Настройка тактирования микроконтроллера STM32F407VG

Разберём каждый пункт настройки (в соответствии с цифрами на рисунке 1):

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. Выбираем делитель частоты, в данном случае делим на 8.
5. Выбираем на сколько делить частоту 192 МГц после PLL. Делим на 4.
6. Выбираем источник тактирования SYSCLK как PLL.
7. Выставляем величину предделителя частоты тактирования шины APB2 равным 2-м. Таким образом таймеры 2,3,4,5,6,7,12,13,14 тактируются частотой 48 МГц.
8. Нажимаем кнопку Generate. Должно выскочить окошко «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.
10. **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 = 1:

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);

**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 16-ти (например, UART) разных прерываний. Следовательно, при входе в обработчик прерываний необходимо проверить – какое именно произошло прерывание, после обработки прерывания необходимо сбросить влаг 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 необоходимо 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.

1. **Индивидуальные задания:**

|  |  |
| --- | --- |
| **вариант** | **Задание** |
| **1** | Мигать синим светодиодом с переменной: F1 = 0.5Гц, F2 = 2Гц. Скважность = 2. По нажатию на кнопку происходит смена частоты. Повторное нажатие на кнопку возвращает первоначальное состояние. Реализовать на основе прерывания таймера. |
| **2** | Мигать попеременно красным и зелёным светодиодами. F = 3Гц. Скважность = 2. По нажатию на кнопку скважность = 4. Повторное нажатие на кнопку возвращает первоначальное состояние. Реализовать на основе прерывания таймера. |
| **3** | Мигать красным светодиодом с F=0.25Гц, зелёным светодиодом с F=1Гц. Скважность = 4. По нажатию на кнопку частота увеличивается в 2 раза. Повторное нажатие на кнопку возвращает первоначальное состояние. Реализовать на основе прерывания таймера. |
| **4** | Зажигать и гасить по очереди все четыре светодиода, F=1Гц. По нажатию на кнопку порядок мигания светодиодов меняется на противоположный. Повторное нажатие на кнопку возвращает первоначальное состояние. Реализовать на основе прерывания таймера. |
| **5** | Зелёный и красный светодиод горит попеременно с желтым светодиодом. F = 1Гц. Скважность = 5. По нажатию на кнопку частота F = 5 Гц. Повторное нажатие на кнопку возвращает первоначальное состояние. Реализовать на основе прерывания таймера. |
| **6** | Два любых светодиода горят постоянно, оставшиеся два горят попеременно. F = 3Гц. Скважность = 2. По нажатию на кнопку мигающие светодиоды начинают постоянно гореть, постоянно горевшие светодиоды начинают мигать. Повторное нажатие на кнопку возвращает первоначальное состояние. Реализовать на основе прерывания таймера. |
| **7** | Мигать синим и зеленым светодиодом. F = 1Гц. Скважность = 4. Красный светодиод горит постоянно. По нажатию на кнопку мигающие светодиоды начинают постоянно гореть, постоянно горевшие светодиоды начинают мерцать. Повторное нажатие на кнопку возвращает первоначальное состояние. Реализовать на основе прерывания таймера. |
| **8** | Мигать попеременно красным и зелёным светодиодами. F = 5Гц. Скважность = 3. По нажатию на кнопку F = 2 Гц, скважность = 2. Повторное нажатие на кнопку возвращает первоначальное состояние. Реализовать на основе прерывания таймера. |
| **9** | Мигать синим светодиодом с F=5Гц, зелёным светодиодом с F=1Гц. Скважность = 4. По нажатию на кнопку начинают мигать красный и желтый светодиоды (параметры теже). Повторное нажатие на кнопку возвращает первоначальное состояние. Реализовать на основе прерывания таймера. |
| **10** | Зажигать и гасить по очереди все четыре светодиода, F=1Гц. По нажатию на кнопку F=5Гц. Повторное нажатие на кнопку возвращает первоначальное состояние. Реализовать на основе прерывания таймера. |
| Задания повышенной сложности | |
| **11** | Зажигать и гасить по очереди все четыре светодиода, F1=1Гц, F2=2Гц, F3=3Гц, F4=4Гц. По нажатию на кнопку порядок мигания светодиодов меняется на противоположный. Повторное нажатие на кнопку возвращает первоначальное состояние. Реализовать на основе прерывания таймера. |
| **12** | Зажигать и гасить по очереди все четыре светодиода, F1=1Гц, F2=2Гц, F3=3Гц, F4=4Гц. По нажатию на кнопку F1=4Гц, F2=3Гц, F3=2Гц, F4=1Гц. Повторное нажатие на кнопку возвращает первоначальное состояние. Реализовать на основе прерывания таймера. |
| **13** | Мигать синим светодиодом с переменной частотой: F1 = 0.5Гц, F2 = 2Гц, F3 = 3Гц, F4 = 4Гц. Скважность = 2. По нажатию на кнопку происходит циклическая смена частоты. Реализовать на основе прерывания таймера. |
| **14** | Мигать красным светодиодом с переменной частотой: F1 = 5Гц, F2 = 4Гц, F3 = 3Гц, F4 = 2Гц. Скважность = 4. По нажатию на кнопку происходит циклическая смена частоты. Реализовать на основе прерывания таймера. |
| **15** | Мигать синим светодиодом с переменной частотой: F1 = 0.5Гц, F2 = 1Гц, F3 = 2Гц, F4 = 4Гц. Скважность = 0,5. По нажатию на кнопку происходит циклическая смена частоты. Реализовать на основе прерывания таймера. |
| **16** | Первоначально светодиоды не горят. По нажатию 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. По нажатию 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. |

1. **Контрольные вопросы**
2. Каковы области применения таймеров микроконтроллеров серии STM32?
3. В каких режимах могут работать таймеры микроконтроллеров серии STM32?
4. Каковы особенности работы в режиме TIM\_CounterMode\_Up?
5. Каковы особенности работы в режиме TIM\_CounterMode\_Down?
6. Каковы особенности работы в режиме TIM\_CounterMode\_CenterAligned1?
7. Что такое прерывание и как это используется при работе таймеров микроконтроллеров серии STM32?
8. В каких режимах могут работать модули захвата/сравнения?
9. В каких ситуациях применяют режим сравнения?
10. В каких ситуациях применяют Режим захвата сигнала?
11. Для чего используется режим широтно-импульсной модуляции (ШИМ)?
12. **Литература**
13. Reference manual. STM32F405/415, STM32F407/417, STM32F427/437 and. STM32F429/439 advanced ARM. ®. -based 32-bit MCUs URL: http://www.st.com/resource/en/reference\_manual/dm00031020.pdf (дата обращения: 09.01.2018).
14. STM32 TIMER general-purpose Описание базового молуля . URL: http://mycontroller.ru/old\_site/stm32-timer-general-purpose-opisanie-bazovogo-molulya/default.htm (дата обращения: 09.01.2018).

# Лабораторная работа №3

# Работа с аналого-цифровым преобразователем микроконтроллера STM32F407

1. **Цель работы:**

Сформировать общее представление о работе аналого-цифрового преобразователя (АЦП) микроконтроллера STM32F407. Изучить особенности регистрации цифровых сигналов.

1. **Теоретические сведения**

Микроконтроллер STM32F4хх имеет 3 12-ти 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 два 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 миллионов 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 2,4…3,6 В. При снижении напряжения до 1,8 (1,7) В, скорость преобразования снижается до 1,2 MIPS.

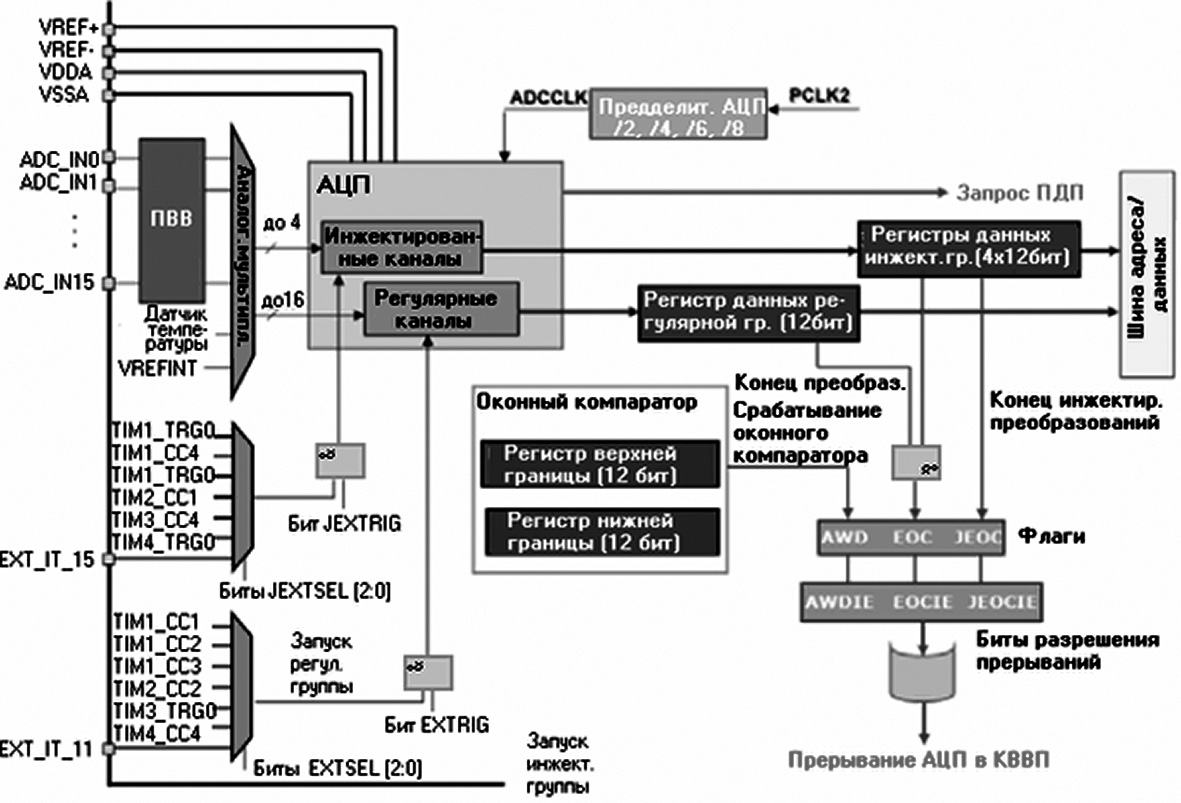


Рисунок 3.1 – структура аналого-цифрового преобразователя (АЦП) микроконтроллера STM32F407

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°C с точностью ±1,5°C.Типовая схема подключения АЦП приведена на рисунке 3.2.

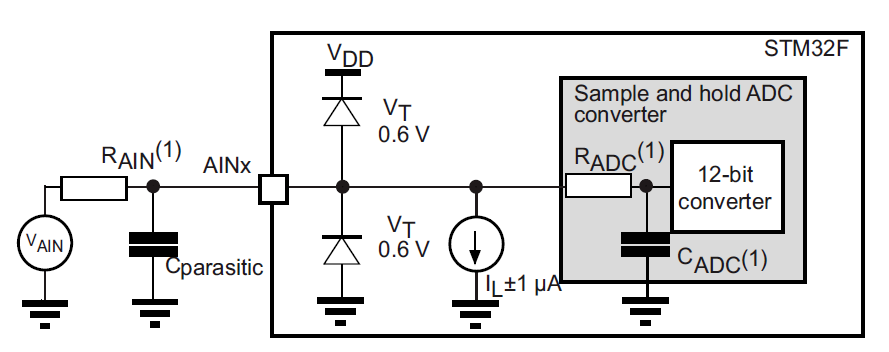


Рисунок 3.2 – Типовая схема подключения АЦП

1. **Порядок выполнения работы.**
   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"

* 1. **Инициализация аппаратных средств**

Объявляем функцию инициализации аппаратных средств микроконтроллера:

void Board\_Init(void

{

GPIO\_LEDS\_Init();

ADC1\_Init();

Timer3\_Init();

Timer4\_Init();

}

* 1. **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);

* 1. **Инициализация АЦП**

Определяем функцию для инициализации АЦП:

GPIO\_ADC1\_Init();

Включаем тактирование АЦП:

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_ADC1, ENABLE);

**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);

Разберем 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 (необходимо 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 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 2 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

**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 х 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 значения

ADC\_Resolution\_10b // на выходе преобразования 10 битные значения

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-ти битный, а значащих данных всего 12 или меньше.

Пример:

С АЦП пришло значение 12 бит.

Если выравнивание вправо: 0b0000xxxxxxxxxxxx, x – значения бит полученные в процессе аналого-цифрового преобразования.

Если выравнивание влево: 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, который 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 1 и ADC\_ ScanConvMode=ENABLE, описывается какие каналы и в какой последовательности они будут сканироваться
  1. ***Настройка таймеров, приоритета и источника прерываний с помощью NVIC, запуск прерываний***
* Для управления прерываниями производится инициализация NVIC (подробнее смотрите в лабораторной работе №1)
* Инициализировать таймеры (смотреть лабораторную работу №2)
  1. ***Настраиваем конкретный канал АЦП***

В нашем случае это всего один канал, потому 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.

Параметры:

* ADCx: номер АЦП, где х может быть 1, 2 или 3.
* ADC\_Channel: задает канал АЦП (с Channel1- 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, потому и rank=1.
* 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

1. ADC\_SoftwareStartConv(ADC\_TypeDef \* ADCx ) – включает выбранное программное обеспечение ADC, запускает преобразование каналов
2. ADC\_GetFlagStatus(ADC\_TypeDef\* ADCx,ADC\_FLAG) – проверяет, установлен ли указанный флаг АЦП.

Параметры:

* ADCx,: номер АЦП, где х может быть 1, 2 или 3.
* ADC\_FLAG ,: задает флаг для проверки. Этот параметр может пригимать 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.

1. 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);

1. **Индивидуальные задания:**

На основе задания по лабораторной работе №2 выполнить анализ значения напряжения на заданном канале и переключить мигание светодиода.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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 |

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

# Лабораторная работа №4

# Интерфейс передачи данных USART и протоколы RS-232, RS-485, RS-422

1. **Цель работы:**

Сформировать общее представление о работе универсального синхронно-асинхронного приемопередатчика (модуль USART) микроконтроллера STM32F407. Изучить особенности передачи цифровых сигналов.

1. **Теоретические сведения**

Сокращения типа RS-232, RS-485, RS-422.. RS - это Recommended Standard (рекомендованный стандарт). Все RS-протоколы можно приблизительно 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 не применяются.

* Симплексные протоколы позволяют передавать данные только в одну сторону, т.е. только с передатчика на приемник, 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, причем оба устройства могут быть и приемниками и передатчиками, но это не может осуществляться одновременно (рисунок 4.1). Т.е. каждое устройство может либо передавать, либо принимать *(кстати, классический /рекомендованный/ RS-485 именно полудуплексный).*
* Дуплексные протоколы. Применение дуплексного протокола позволяет и прием и передачу информации одновременно, т.е. оба устройства могут быть и приемником 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 отличие: 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 1 или 0 (рисунок 4.3).

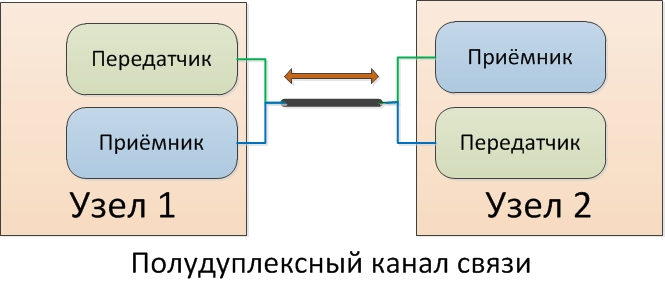


Рисунок 4.1 – Схематичное изображение работы полудуплексного протокола

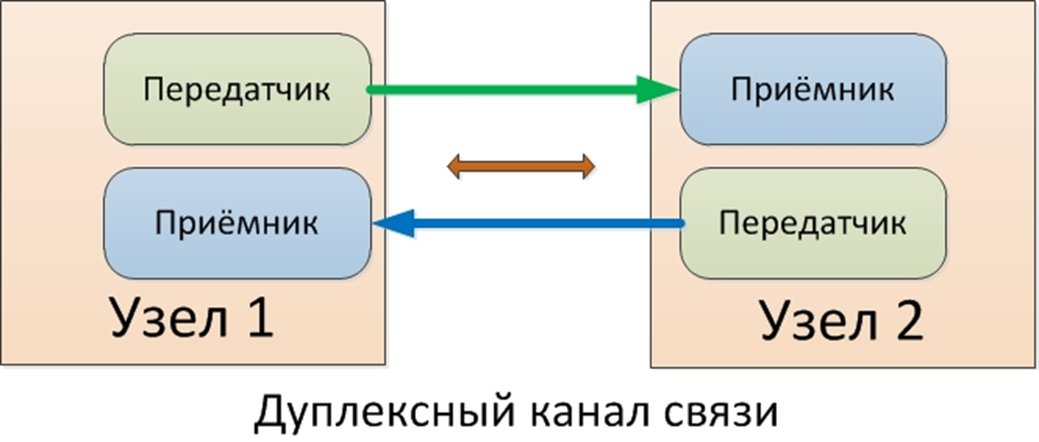
**

Рисунок 4.2 – Схематичное изображение работы дуплексного протокола

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

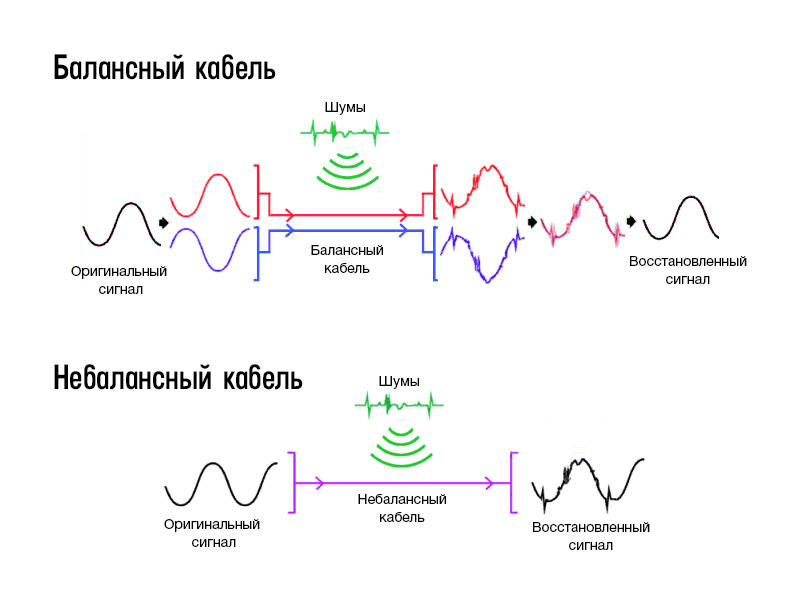


Рисунок 4.3 – Пример небалансного сигнала, который передается по несбалансированной линии

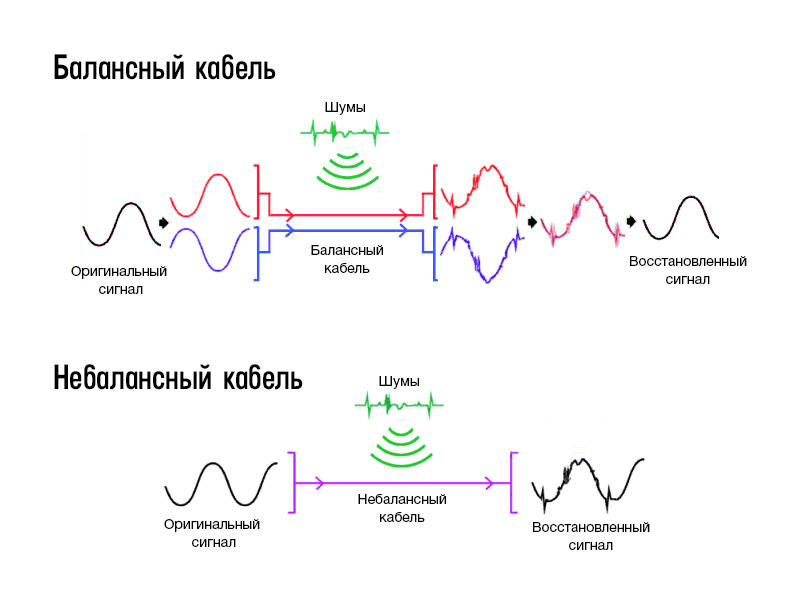


Рисунок 4.4 – Пример балансного сигнала, который передается по сбалансированной линии

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).

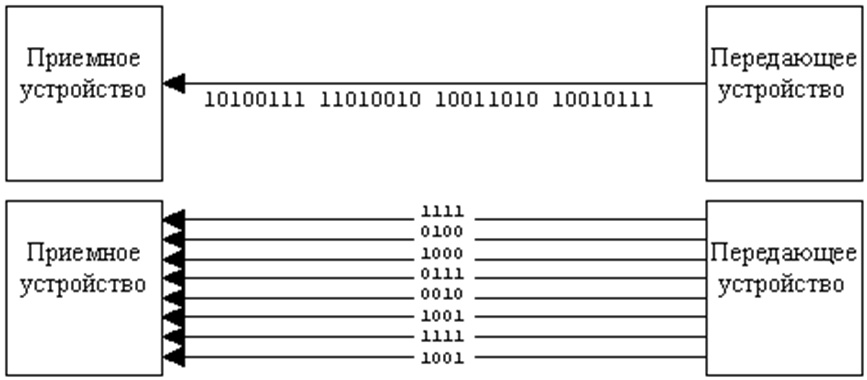


Рисунок 4.5 – последовательные и параллельные интерфейсы передачи данных

Формат передаваемых данных показан на рисунке 4.6. Получив стартовый бит, приемник выбирает из линии биты данных через определённые интервалы времени. Очень важно, чтобы тактовые частоты приемника и передатчика были одинаковыми, допустимое расхождение - не более 10%. Скорость передачи по USART иди RS-232C может выбираться из ряда: 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 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).

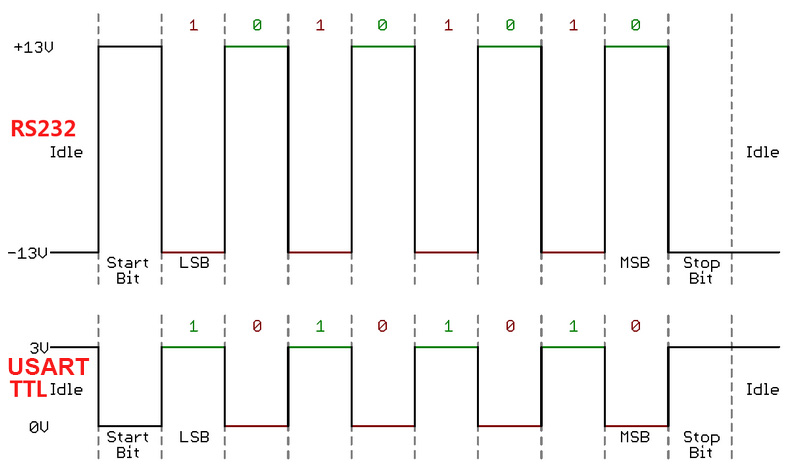


Рисунок 4.6 – Формат передаваемых данных

**USART в STM32 (STM32F100xx)**

USART в микроконтроллерах STM32 предоставляет гибкие средства для полнодуплексного обмена данными с внешними устройствами в последовательном формате с возможностью поддержки сигналов CTS/RTS; поддерживает 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 поддерживается однонаправленная передача 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 (1 или 2).
* 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, 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).
* 10 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.

1. **Порядок выполнения работы.**
   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
  1. **Передача данных по USART**

**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 Тх.

**Инициализация USART**

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

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, 19200, 38400, 57600, 115200 бит/с.

USART\_HardwareFlowControl – поле дающее возможность подключения дополнительных выводов, которые принимают значение 1 при осуществлении передачи или приема информации. Как правило данные дополнительные выводы не используются и значение по умолчанию для данного поля USART\_HardwareFlowControl\_None.

* 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.

USART\_Parity позволяет автоматически контролировать целостность данных методом контроля битовой чётности. Если контроль целостности данных не производится, устанавливается значение USART\_Parity\_No.

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.

USART\_WordLength – поле настраивает количество бит в передаваемом байте. Возможны следующие значения:

* USART\_WordLength\_8b – байт содержит 8 бит.
* USART\_WordLength\_9b – байт содержит 9 бит.

Приведем пример инициализации USART2 с помощью управляющей структуры, содержащей настройки USART.

RCC\_APB1PeriphClockCmd(RCC\_APB1Periph\_USART2, ENABLE);

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); запись структуры

USART\_Cmd(USART2, ENABLE); //включение USART2

**Примеры функций для передачи данных по USART**

Функция для пересылки одного байта данных, использующая стандартную функцию USART\_SendData:

int putcharx(uint8\_t ch)

{

while (USART\_GetFlagStatus(USART2, USART\_FLAG\_TXE) == RESET);

USART\_SendData(USART2, (uint8\_t)ch);

return ch;

}

Функция для передачи байта 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++;

 }

}

* 1. **Прием данных по USART**

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

uint8\_t receivedData[16];

*// Счетчик принятых байт*

uint8\_t receivedDataCounter = 0;

Инициализация для приёма данных может выглядеть следующим образом:

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_USART1, ENABLE);

RCC\_AHB1PeriphClockCmd(RCC\_AHB1Periph\_GPIOA, ENABLE);

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);

USART\_Cmd(USART1, ENABLE);

// включаем прерывание по приему данных

USART\_ITConfig(USART1, USART\_IT\_RXNE, ENABLE);

Пример программы обработчика прерываний. Данная подпрограмма принимает 16 байт данных, после этого прием прекращаетсмя:

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);

}

}

* 1. **Прием и передача данных на персональный компьютер**

Для приема и передачи данных на персональном компьютере может использоваться программа Tera Term — это свободная, распространяемая по [лицензии BSD](https://ru.wikipedia.org/wiki/%D0%9B%D0%B8%D1%86%D0%B5%D0%BD%D0%B7%D0%B8%D1%8F_BSD), служебная программа для работы через реальные и виртуальные COM-порты ([RS232](https://ru.wikipedia.org/wiki/RS232), [USB](https://ru.wikipedia.org/wiki/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;
* для программирования устройств;
* для перезагрузки устройств.

Программа Tera Term

1. поддерживает протоколы: IPv4, IPv6, Telnet, SSH1, SSH2, Kermit, XMODEM, ZMODEM, B-PLUS, Quick-VAN, SCP;
2. эмулирует терминалы: DEC VT100, VT101, VT102, VT282, VT320, VT382, VT420, VT520, VT525, Tektronix TEK4010, Altair 8800;
3. поддерживает раскладки клавиатуры: английскую, русскую, японскую;
4. имеет многоязыковую поддержку, в том числе переведена на русский язык;
5. поддерживает кодировки: ASCII, UTF-8, KOI8-R;
6. поддерживает скриптовый язык: Tera Term Language;
7. поддерживает соединения: TCP/IP, COM-port;
8. поддерживает создание логов сессий;
9. работа с терминалами в интерактивном режиме.
10. **Индивидуальные задания:**

На основе задания по лабораторной работе №3 выполнить передачу данных на персональных компьютер о состоянии светодиодов. Например при включении 1 светодиода передать на ПК строку “VD1 on”, при выключении передать 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.

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

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