**Лабораторная работа №2**

**Прерывания. Обработка внешних прерываний**

1. **Настройка библиотек**

В проекте нужны следующие компоненты (как включаются компоненты смотрите в лабораторной работе 1):

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

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

* инициализировать светодиоды (смотрите лабораторную работу №1);
* инициализировать кнопку, расположенную на выводе **PA0** (Рисунок 1);

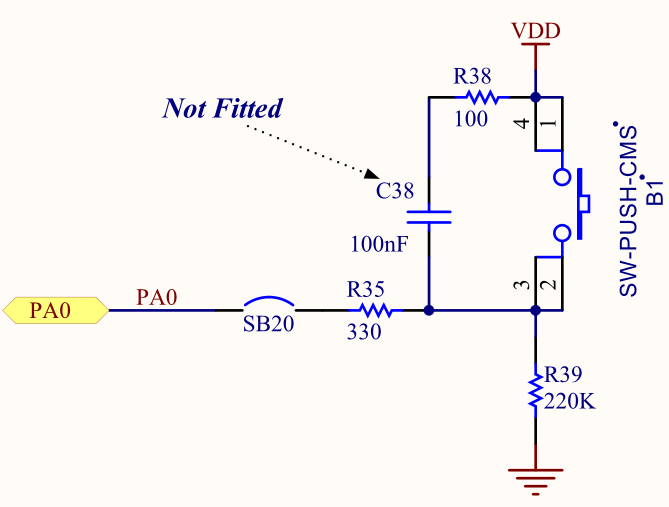


Рисунок 1 – Схема кнопки на отладочной плате (Всю схему смотрите в документации к плате stm32f4Discovery\_UserManual.pdf)

1. **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.

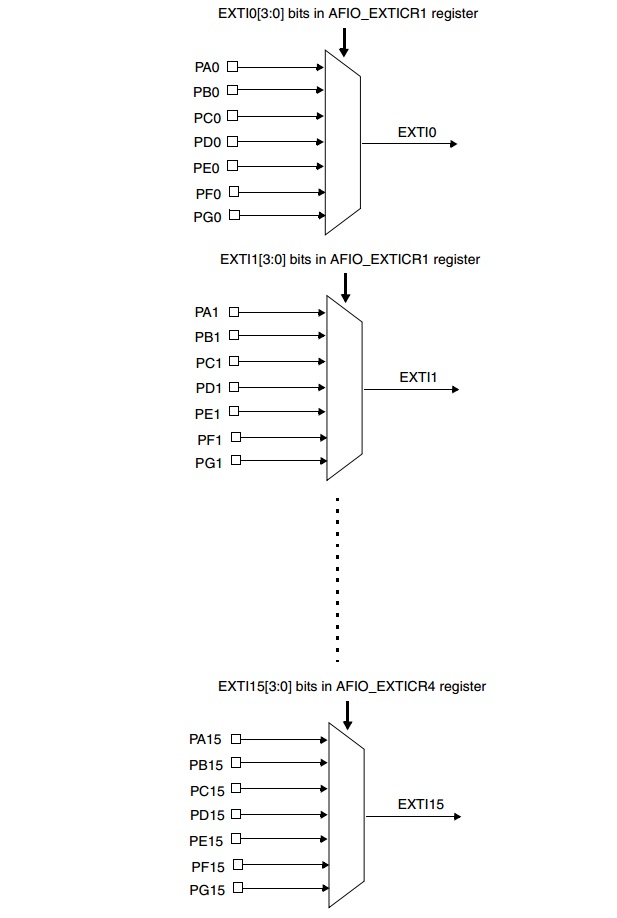


Рисунок 2 – Структура связи внешних прерываний **EXTI**

Т.е. у нас есть 16 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. Либо одно, либо другое. Придется выбирать.

Для выбора порта и номера вывода, от которых хотим получать прерывания, пишем:

SYSCFG\_EXTILineConfig(EXTI\_PortSourceGPIOA, EXTI\_PinSource0);

где EXTI\_PortSourceGPIOA – порт, к которому подключена кнопка;

EXTI\_PinSource0 – номер вывода, к которому подключена кнопка.

Дальше инициализируем структуру:

EXTI\_InitTypeDef EXTI\_InitStruct;

EXTI\_StructInit(&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 (когда на выводе происходит переход состояния из логического нуля в логическую единицу, в данном случае при нажатии на кнопку);
  + EXTI\_Trigger\_Falling – фронт спада (когда на выводе происходит переход состояния из логической единицы в логический нуль, в данном случае 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).

1. **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 помотреть в документации, либо в заголовочном файле stm32f4xx.h.

– IRQChannelPreemptionPriority. Приоритет прерывания (чем ниже, тем значительнее)

– IRQChannelSubPriority. Подприоритет.

Допустим, что включено два прерывания с разным приоритетом: прерывание А (приоритет 1) и прерывание Б (приоритет 2). Если возникло прерывание А, и оно обрабатывается долго, и в момент обработки этого прерывания возникло прерывание Б, то сначала контроллер довыполнит обработчик прерывания А, а замет перейдёт к выполнению обработчика прерывания Б. Если же в начале возникло прерывание Б, и когда оно обрабатывалось возникло прерывание А, то контроллер прервёт выполнение обработчика прерывания Б, выполнит обработку прерывания А, а затем вернётся довыполнить обработку прерывания Б.

1. **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 флаг прерывания, для того чтобы прерывание снова произошло.

Создадим глобальную переменную для установки возникновения прерывания от кнопки:

uint8\_t interruptState = 0;

Затем опишем сам обработчик перывания:

void EXTI0\_IRQHandler(void)

{

if(EXTI\_GetITStatus(EXTI\_Line0) != RESET)

{

interruptState = 1;

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 устанавливаем переменную interruptState в 1-цу. Далее, вызывая функцию EXTI\_ClearITPendingBit(EXTI\_Line0) мы сбрасываем флаг прерывания от внешнего источника линии. Если этого не сделать, то, как только закончится обработка прерывания, она вызовется заново.

Опишем логику работы программы: при нажатии на кнопку происходит переключение режимов работы основной программы. В данном случае режимов будет два. Для реализации такой логики введём

Теперь в бесконечном цикле функции main() добавим слдующие строки:

if(interruptState == 1)

{

Delay(10000);

if (GPIO\_ReadInputDataBit(GPIOA, GPIO\_Pin\_0) == 1) {

GPIO\_ToggleBits(GPIOD, GPIO\_Pin\_13);

}

interruptState = 0;

}

В данном участке кода мы проверяем – было ли прерывание от кнопки, и если да – обрабатываем. В начале вставляем задержку. Это необходимо, чтобы избежать множественного срабатывания от дребезга контактов кнопки (Проверьте выполнение программы без защиты от дребезга и с ней).

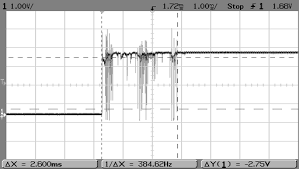


Рисунок 3 – Дребезг контактов на осциллографе

В данном случае используется самый простой способ защиты – выжидание некоторого фиксинованного времени, пока переходной процесс дребезга не закончится. Если по истечении этого времени кнопка нажата, то меняем значение на светодиоде функцией GPIO\_ToggleBits (если горел, то выключается, если не горел, то включаем). Проверяется значение кнопки считыванием значения при помощи функции GPIO\_ReadInputDataBit(GPIOA, GPIO\_Pin\_0). Если значение, возвращаемое функцией равно 1, то кнопка в данный момент времени нажата. В конце устанавливаем переменную interruptState в 0.

Большая часть логики вынесена из обработчика прерывания в бесконечный цикл программы. Это сделано умышленно, потому что обработка прерываний должно осуществляться максимально быстро. В обработчике лишь происходит установка переменной состояния.

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