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.

2. Теоретические сведения

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

Рисунок 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 (A, B, C…). На рисунке 2 такие выводы обозначаются буквой P, затем буквой порта и в конце номером вывода (от 0 до 15).

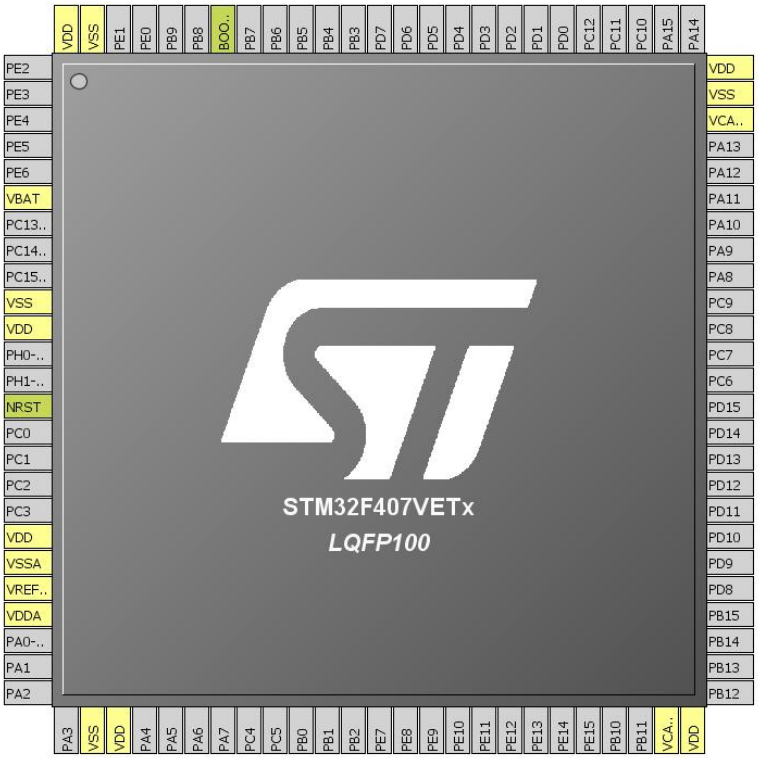

Рисунок 2 – Схематичное изображение микроконтроллера STM32F407VET6

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.

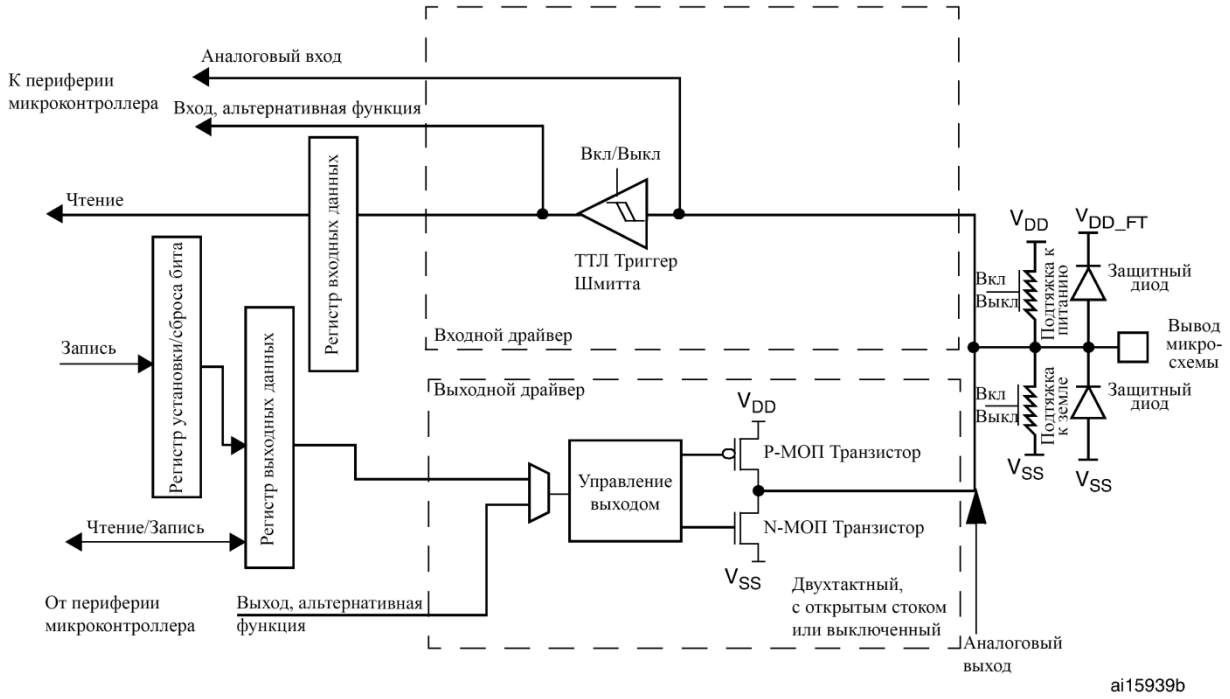

Рисунок 3 – Структурная схема интерфейса ввода/вывода общего назначения

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.

Библиотека CMSIS

– **M4 CMSIS Core**. Этот компонент предоставляет стандартизированный интерфейс для работы с Cortex-M4;

– **CMSIS BOOT**. Производит начальную настройку микроконтроллера при запуске, и вызывает функцию `main()`;

– **RCC**. Отвечает за сброс, настройку частоты тактирования микроконтроллера, а также за включение и настройку тактирования различных частей микроконтроллера;

– **GPIO**. Предоставляет функции и структуры для настройки и управления портами ввода вывода;

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).

Написание программы

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.

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**).

Для настройки портов ввода вывода используется стандартная структура из библиотеки **GPIO**, с названием **`GPIO_InitTypeDef`**. Для этого создаем переменную этой структуры:

```
GPIO_InitTypeDef GPIO_InitStruct;
```

Далее структуру нужно инициализировать значениями по умолчанию:

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

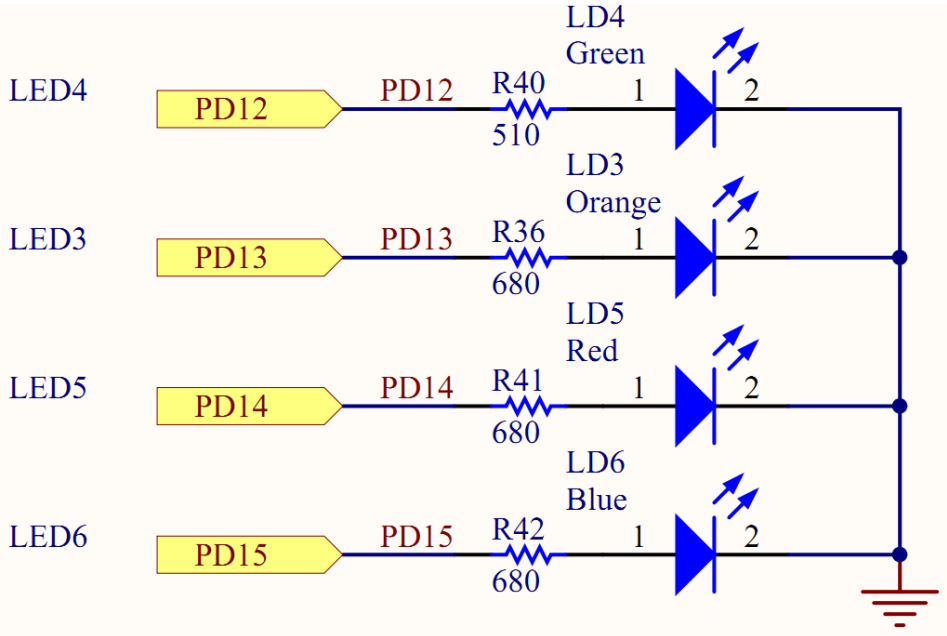

Рисунок 4 – Схема принципиальная подключения светодиодов на отладочной плате STM32F4DISCOVERY

Это необходимо для предотвращения инициализации структуры ошибочными значениями. В языке С при инициализации новой структуры под неё выделяется память, но никто не гарантирует, что эта память будет очищена. Так получается, что в этом участке памяти остались предыдущие

значения совершенно других данных. Функция GPIO_StructInit позволяет заполнить структуру стандартными значениями. Символ «&» перед названием структуры указывает на то, что мы передаем указатель на структуру.

В структуре нужно указать настройки порта, для этого пишем:

```
GPIO_InitStruct.GPIO_Pin = GPIO_Pin_13 | GPIO_Pin_14;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_NOPULL;
```

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

В параметре GPIO_Mode указывается направление настраиваемых выводов. Возможные значения:

- GPIO_Mode_IN Выход (МК устанавливает данные)
- GPIO_Mode_OUT Вход (МК считывает данные)
- GPIO_Mode_AF Альтернативная функция (Интерфейсы и др.)
- GPIO_Mode_AN Аналоговый (ЦАП, АЦП)

В данном случае направление выводов GPIO_Mode_OUT, т.к. необходимо подавать сигналы на светодиоды.

В параметре **GPIO_Speed** указываем скорость тактирования настраиваемых выводов. От скорости тактирования зависит энергопотребление (чем выше частота 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.

Мигание светодиодом

Для мигания используется две функции:

```
GPIO_SetBits(GPIOD, GPIO_Pin_14); //Подать лог. 1
GPIO_ResetBits(GPIOD, GPIO_Pin_14); // Подать лог. 0
```

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.

Обработка внешних прерываний

Настройка библиотек

В проекте нужны следующие компоненты:

- 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 5);

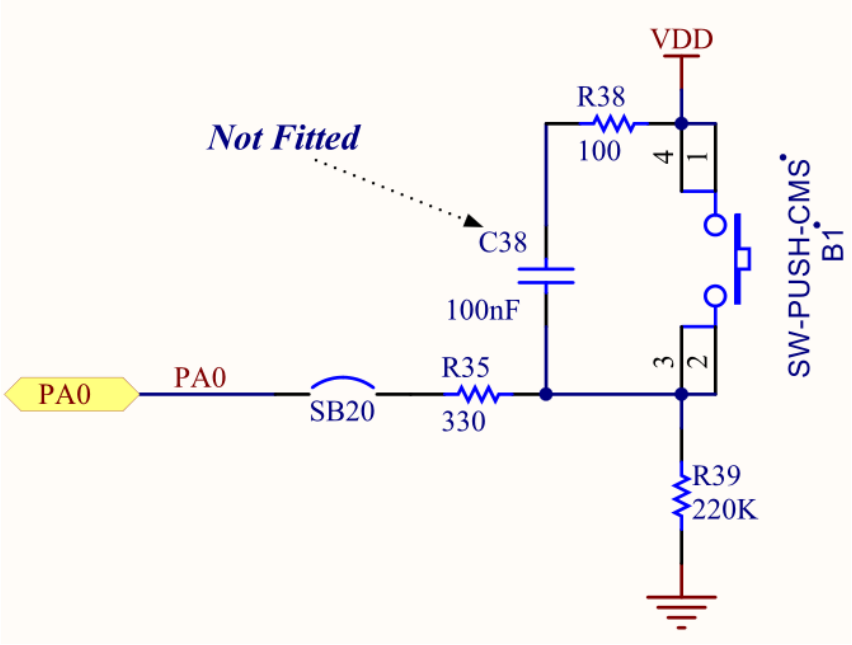

Рисунок 5 – Схема кнопки на отладочной плате (Всю схему смотрите в документации к плате **stm32f4Discovery_UserManual.pdf**)

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**, 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 то же 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 6.

Т.е. имеется 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.

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.

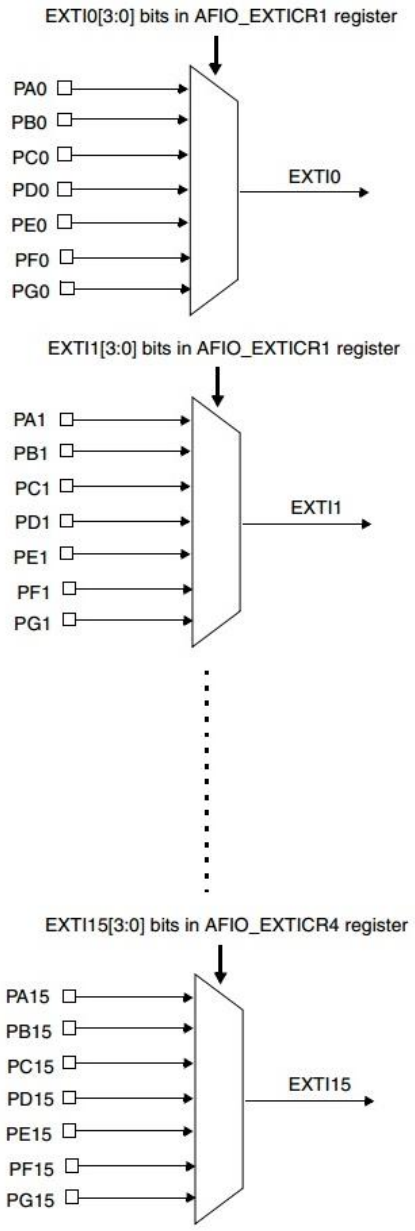

Рисунок 6 – Структура связи внешних прерываний **EXTI**

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.

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.