<https://habrahabr.ru/post/218825/>

Начинаем изучать Cortex-M на примере STM32 (Часть 2)

Данная статья является продолжением цикла по программированию микроконтроллеров на базе ядра Cortex-M.  
Первую статью можно прочитать здесь:  
[Начинаем изучать Cortex-M на примере STM32](http://habrahabr.ru/post/216843)  
Задачей статей является подробное описание особенностей, возникающих при программировании МК. Материал не предназначен для желающих за 10 минут запустить пример мигания светодиодом. Я постараюсь подробно описать то, что часто скрывают от новичков, чтобы их не напугать.  
  
Мне очень хочется, чтобы программисты использующие стандартные библиотеки, шаблоны, примеры и т.д. понимали как все это работает. А при отсутствии этих самых библиотек и примеров могли самостоятельно решить свою задачу.  
  
Основной акцент сделан на изучение документации на ядро Cortex-M и документации на конкретный контроллер.  
На этот раз речь пойдет про прерывания, а так же будут затронуты некоторые вопросы архитектуры памяти и структуры прошивки МК.

Несколько слов про документацию ARM

По не совсем ясным для меня причинам, нельзя зайти на сайт [ARM](http://www.arm.com/) и скачать полную документацию на ядро Cortex-M4. Да и на Cortex-M3 тоже нельзя.   
Придется почитать несколько документов.  
1. Изучение придется начать с [Cortex ™-M3 TechnicalReference Manual Revision: r1p1](http://infocenter.arm.com/help/topic/com.arm.doc.ddi0337e/DDI0337E_cortex_m3_r1p1_trm.pdf) — самой первой ревизии технической спецификации на ядро Cortex-M3  
2. Во всех дальнейших ревизиях и описании [Cortex ™-M4 TechnicalReference Manual](http://infocenter.arm.com/help/topic/com.arm.doc.ddi0439d/DDI0439D_cortex_m4_processor_r0p1_trm.pdf) описаны лишь общие данные и изменения относительно предыдущего документа.  
Так что прошу не удивляться ссылкам на спецификации другого ядра.

Interrupt and Events

Прежде всего необходимо разобраться с тем, что такое прерывания.  
В МК Cortex-M есть два понятия, которые часто путают *Interrupt* и *Event*.  
*Event* — это событие (аппаратное или программное), на которое могут реагировать ядро или периферийные блоки. Одним из вариантов реакции может быть — прерывание.  
*Interrupt* — это прерывание работы программы и переход управления в специализированный участок обработчик прерывания.  
  
Взаимосвязь между *Event* и *Interrupt* заключается в следующем:  
*Каждый Interrupt вызывается Event, но не каждый Event вызывает Interrupt.*  
Помимо прерываний, события могут активировать и другие возможности МК.

NVIC

Управление и обработка прерываниями производится *контроллером приоритетных векторных прерываний NVIC (Nested Vectored Interrupt Controller)*. Контроллер прерываний часть ядра Cortex-M. Документацию на этот контроллер необходимо начинать изучать с [Cortex ™-M3 TechnicalReference Manual Revision: r1p1](http://infocenter.arm.com/help/topic/com.arm.doc.ddi0337e/DDI0337E_cortex_m3_r1p1_trm.pdf)  
  
При возникновении, некоторого события контроллер прерываний автоматически прерывает выполнение основной программы, и вызывает соответствующую функцию обработки прерываний. После выхода из функции обработчика прерываний программа продолжает выполнение с того места, где произошло прерывание. Все происходит автоматически (при правильной настройке NVIC, но об этом ниже).  
  
Из самого названия видно, что контроллер NVIC поддерживает вложенность прерываний и приоритеты. Каждому прерыванию при настройке NVIC присваивается свой приоритет. Если во время обработки низкоприоритетного прерывания возникает высокоприоритетное, то оно, в свою очередь, прервет обработчик низкоприоритетного прерывания.

Как это работает?

Данный пост не претендует на абсолютную полноту, я советую изучить раздел прерываний в [Cortex™-M3 Technical Reference Manual](http://infocenter.arm.com/help/topic/com.arm.doc.ddi0337e/DDI0337E_cortex_m3_r1p1_trm.pdf). Поскольку эта часть ядра не претерпела изменений, ее описание дано в первой ревизии r1p1 на ядро Cortex-M3.

Вход в прерывание и выход из него

При инициации прерывания NVIC переключает ядро в режим обработки прерывания. После перехода в режим обработки прерывания регистры ядра помещаются в стек. Непосредственно во время записи значения регистров в стек осуществляется выборка начального адреса функции обработки прерывания.  
  
В стек перемещается регистр регистр статуса программы (*Program Status Register (PSR)*), счетчик программы (*Program Counter (PC)*) и регистр связи (*Link Register (LR)* ). Описание регистров ядра приведено в [Cortex-M4 Generic User Guide](http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf). Благодаря этому, запоминается состояние, в котором находилось ядро перед переходом в режим обработки прерываний.   
  
Также сохраняются регистры R0 — R3 и R12. Эти регистры используются в инструкциях для передачи параметров, поэтому, помещение в стек делает возможным их использование в функции обработки прерывания, а R12 часто выступает в роли рабочего регистра программы.  
  
По завершении обработки прерывания все действия выполнятся в обратном порядке: извлекается содержимое стека и, параллельно с этим, осуществляется выборка адреса возврата.  
  
С момента инициации прерывания до выполнения первой команды обработчика прерывний проходит 12 тактов, такое же время необходимо для возобновления основной программы после завершения обработки прерывания.

Вложенность прерываний

Как было сказано выше NVIC поддерживает прерывания с различными приоритетами, которые могут прерывать друг друга. При этом, могут возникнуть различные ситуации, обработка которых по разному оптимизирована.  
  
*1. Приостановка низкоприоритетного прерывания*  
В этой ситуации, обработка низкоприоритетного прерывания прекращается. Следующие 12 циклов выполняется сохранение в стек нового набора данных и запускается обработка высокоприоритетного прерывания. После его обработки, содержимое стека автоматически извлекается и возобновляется обработка низкоприоритетного прерывания.  
Больших отличий от прерывания основной программы не наблюдается.   
  
*2. Непрерывная обработка прерываний*  
Эта ситуация может возникнуть в двух случаях: если два прерывания имеют одинаковый приоритет и возникают одновременно, если низкоприоритетное прерывание возникает во время обработки высокоприоритетного.  
В этом случае, промежуточные операции над стеком не производятся. Происходит только загрузка адреса обработчика низкоприоритетного прерывания и переход к его выполнению. Отказ от операций над стеком экономит 6 тактов. Переход к следующему прерыванию происходит не за 12 тактов, а всего за 6.  
  
*3. Запаздывание высокприоритетного прерывания*  
Ситуация возникает, если высокоприоритетное прерывание происходит во перехода к обработке низкоприоритетного (за те самые 12 тактов). В этом случае переход к высокоприоритетному прерыванию будет происходить не менее 6 тактов с момента его возникновения (время необходимое для загрузки адреса обработчика прерывания и перехода к нему). Возврат в низкоприоритетное уже описан выше.

Приоритеты прерываний

Помимо простой установки приоритета прерываний, NVIC реализует возможность группировки приоритетов.  
Прерывания в группе с более высоким приоритетом могут прерывать обработчики прерываний группы с более низким приоритетом. прерывания из одной группы, но с разным приоритетом внутри группы не могут прерывать друг друга. Приоритет внутри группы определяет только порядок вызова обработчика, когда были активизированы оба события.  
  
Значение приоритета прерывания задается в регистрах *Interrupt Priority Registers* (см. [Cortex-M4 Generic User Guide](http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf)). При этом, часть бит отвечает за приоритет группы, в которой находится прерывание, а часть — за приоритет внутри группы.   
Настройка распределение бит на приоритет группы или приоритет внутри группы осуществляется с помощью регистра *Application Interrupt and Reset Control Register* (ВНИМАТЕЛЬНО!!! см. [Cortex-M4 Generic User Guide](http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf)).  
  
Как вы, наверно, заметили, в [Cortex-M4 Generic User Guide](http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf) сказано, что настройка приоритетов и группировки приоритетов зависят от конкретной реализации *implementation defined*.  
А вот дальше не очень приятная вещь. В [Reference manual](http://www.st.com/st-web-ui/static/active/en/resource/technical/document/programming_manual/DM00046982.pdf) к МК STM32F407 про NVIC почти нет информации. Но есть ссылка на отдельный документ. Для того, чтобы разобраться с реализацией NVIC в STM32 придется прочитать еще один документ — [STM32F3xxx and STM32F4xxx Cortex-M4 programming manual](http://www.st.com/st-web-ui/static/active/en/resource/technical/document/programming_manual/DM00046982.pdf). Вообще говоря, я советую внимательно изучить данный документ и по всем другим вопросам, в нем работа ядра расписана более подробно, чем в документации от ARM.  
В нем, уже можно найти:

A programmable priority level of 0-15 for each interrupt. A higher level corresponds to a   
lower priority, so level 0 is the highest interrupt priority

Из возможных 8 бит приоритета используются только 4. Но этого вполне достаточно для большинства задач.

Маскирование прерываний

Предположим, что у нас стоит задача запуска ракеты-носителя при нажатии на красную кнопку, но только при условии, что повернут ключ.  
Нет совершенно ни какого смысла генерировать прерывание на поворот ключа. А вот прерывание на нажатие красной копки нам понадобится. Для того, чтобы включать/выключать различные вектора прерываний, существует маскирование прерываний.  
Маскирование прерывания осуществляется с помощью регистров [*Interrupt Set-enable Registers*](http://infocenter.arm.com/help/topic/com.arm.doc.ddi0337e/DDI0337E_cortex_m3_r1p1_trm.pdf).  
Если прерывание замаскировано, это не означает, что периферия не генерирует события! Просто NVIC не вызывает обработчик этого события.

Таблица векторов прерываний

Все возможные прерывания, поддерживаемые NVIC, записываются в таблицу векторов прерываний. По сути своей, таблица векторов прерываний есть ни что иное, как список адресов функций обработчиков прерываний. Номер в списке соответствует номеру прерывания.  
  
Как написано в [Cortex-M4 Generic User Guide](http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf), NVIC поддерживает до 240 различных векторов прерываний. Но реализация уже зависит от конкретного производителя.  
В описании ядра стандартизованы только прерывания исключений ядра (см. раздел *2.3.2 Exception types* в [Cortex-M4 Generic User Guide](http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf)):

* Reset
* NMI
* HardFault
* MemManage
* BusFault
* UsageFault
* SVCall
* PendSV
* SysTick

С описанием этих исключений я предлагаю вам ознакомиться самостоятельно. Некоторые из них будут затронуты в следующих статьях.  
Остальные прерывания уникальны для МК. Описание таблицы векторов вашего МК вы можете найти в соответствующем [Reference manual](http://www.st.com/st-web-ui/static/active/en/resource/technical/document/reference_manual/DM00031020.pdf) (см. Vector table for STM32F405xx/07xx and STM32F415xx/17xx). Контроллеры STM32F4xx поддерживают 81 вектор прерываний. Можно заметить, что в этой таблице перечислены все периферийные блоки (а некоторые не единожды).  
Практически все периферийные блоки генерируют прерывания, чтобы ядро отвлекалось на работку с ним только по наступлению какого-либо значимого события (например, получении данных по UART).

Расположение векторов прерываний и загрузка МК

Разобравшись с принципами работы прерываний в Cortex-M осталось понять только где хранится таблица прерываний.  
Для этого стоит рассмотреть процесс загрузки и структуру прошивки контроллера. На этот раз мы будем рассматривать только загрузку из встроенной флеш памяти.  
В таблице векторов, находящейся в начале адресного пространства флеш памяти должны находиться по крайней мере (см. [Cortex-M3 Technical Reference Manual](http://infocenter.arm.com/help/topic/com.arm.doc.ddi0337e/DDI0337E_cortex_m3_r1p1_trm.pdf):

• stack top address  
• reset routine location  
• NMI ISR location  
• Hard Fault ISR location.

Из начала флеш памяти ядро считывает значение SP (stack top addres) и PC (reset routine location). Таким образом, автоматически начинает выполняться функция, с адресом считанным в регистр PC. Это может быть, например main.  
  
После обязательных четырех компонентов, может находиться дальнейшая таблица векторов прерываний. Главное сохранить порядок.  
При желании, можно разместить таблицу векторов прерываний в другой области памяти, но тогда, необходимо сообщить NVIC, куда мы передвинули таблицу. За это смещение таблицы векторов отвечает регистр *Vector Table Offset Register* (см. [Cortex-M4 Technical Reference Manual](http://infocenter.arm.com/help/topic/com.arm.doc.ddi0439d/DDI0439D_cortex_m4_processor_r0p1_trm.pdf). Это может понадобиться для написание встроенного загрузчика нового ПО (bootloader), но об этом как-нибудь в другой раз.

От теории к практике

ТЗ второго проекта

Пример создается для отладочной платы [STM32F4Discovery](http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF252419).  
При нажатии на кнопку должен загореться светодиод LED3. При замыкании контактов PC6 и GND загорается светодиод LED5.  
В процессе программирования поиграемся с приоритетами прерываний и посмотрим к чему это приведет.

Железная часть

Найдем в [документации к плате](http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00039084.pdf) кнопку и светодиод:  
  
При ненажатой кнопке на пине PA0 будет логический ноль, при нажатии на кнопку на кнопке появится логическая 1 (3.3В).  
Светодиод LED3 подключен к пину PD13.  
Светодиод LED5 подключен к пину PD14.  
Интересней всего с контактом PC6 — он напрямую выведен на штыревой разъем. Нам будет необходимо обеспечить регистрацию логической 1, когда он не закорочен с контактом GND. О том, как это сделать пойдет речь ниже.

Настройка GPIO

Для нашей задачи необходимо настроить пины PD13 и PD14 как выходные. О том, как это делать можно прочитать в [предыдущей статье](http://habrahabr.ru/post/216843).  
С настройкой пина PA0 тоже все достаточно просто — его нужно настроить на вход. Не смотря на то, что после ресчета МК почти все пины настроены на вход, крайне желательно явно прописать эту инициализацию.  
С пином PC7 все несколько интереснее. Поскольку он «висит в воздухе», его состояние не определено. Нам же необходимо, чтобы при этом его состояние всегда было «1». Для этого, в блоке GPIO активировать подтяжку. В нашем случае, необходима подтяжка к питанию — PULL UP.   
Активация подтяжки осуществляется с помощью регистра *GPIO port pull-up/pull-down register*.

Прерывания EXTI

Для выполнения нашего «ТЗ» с использованием прерываний, нам необходимо настроить прерывания, которые будут срабатывать при переходе контакта PA0 из состояния «0» в состояние «1», и прерывание при переходе контакта PC6 из состояния «1» в состояние «0».  
  
В МК STM32F4xx для этой цели служит контроллер внешних прерываний/событий -*EXTI* (External interrupt/event controller). Я настоятельно рекомендую ознакомиться с его функционалом в [Reference manual](http://www.st.com/st-web-ui/static/active/en/resource/technical/document/reference_manual/DM00031020.pdf). Нам необходимо поступить в соответствии с описанным:

Hardware interrupt selection  
To configure the 23 lines as interrupt sources, use the following procedure:  
• Configure the mask bits of the 23 interrupt lines (EXTI\_IMR)  
• Configure the Trigger selection bits of the interrupt lines (EXTI\_RTSR and EXTI\_FTSR)  
• Configure the enable and mask bits that control the NVIC IRQ channel mapped to the external interrupt controller (EXTI) so that an interrupt coming from one of the 23 lines can be correctly acknowledged.

Нам понадобятся 0 и 6 линии EXTI. Для размаскирования соответствующих линий прерываний необходимо записать в регистр EXTI\_IMR значение 0x9.  
Для линии PA0, необходима генерация события прерывания по переходу из состояния «0» в состояние «1» — по возрастающему фронту. То есть, необходимо записать 1 в нулевой бит регистра EXTI\_RTSR.  
Для линии PC6, наоборот, необходима генерация события прерывания по переходу из состояния «1» в состояние «0» — по падающему фронту. То есть, необходимо записать 1 в шестой бит регистра EXTI\_FTSR.  
На этом настройка блока EXTI закончена. Последний пункт будет реализован при настойке NVIC.  
  
По мимо этого, необходимо определиться, пин с какого порта подключается к определенной линии EXTI. Делается это с помощью регистров *SYSCFG external interrupt configuration register* ([Reference manual](http://www.st.com/st-web-ui/static/active/en/resource/technical/document/reference_manual/DM00031020.pdf)). Эти регистры находятся в *System configuration controller*, что мне кажется не очень логичным (почему было не включить эту насторойку в EXTI?), но оставим сей факт на совести ST.

Настройка NVIC

Активация обработки определенного вектора прерывания осуществляется с помощью регистров*Interrupt set-enable registers (NVIC\_ISERx)*. Описание регистров приведено в [Cortex-M4 Generic User Guide](http://infocenter.arm.com/help/topic/com.arm.doc.dui0553a/DUI0553A_cortex_m4_dgug.pdf). Сама таблицу векторов прерываний для нашего МК приведена в [Reference manual](http://www.st.com/st-web-ui/static/active/en/resource/technical/document/reference_manual/DM00031020.pdf) (см. Table 61).   
  
Из таблицы можно увидеть, что для 0 линии есть отдельное прерывание, а вот линии с 5 по 9 генерируют одно прерывание на всех.  
Кроме того, из таблицы мы узнали номера векторов, необходимых нам прерываний. Теперь нужно записать «1» в 6 бит (активация прерываний линии 0 EXTI) регистра NVIC\_ISER0 (адрес 0xE000E100) и в 23 бит того же регистра (активация прерываний линий 5-9). 

Настройка приоритетов

Для того, чтобы можно было побаловаться с приоритетами прерываний настроим группы приоритетов так, чтобы 2 бита отвечали за приоритет внутри группы, и 2 бита — за приоритет самой группы. Для этого необходимо записать значение 0х05FA0500 в регистр *Application interrupt and reset control register* ([STM32F3xxx and STM32F4xxx Cortex-M4 programming manual](http://www.st.com/st-web-ui/static/active/en/resource/technical/document/programming_manual/DM00046982.pdf)).  
Настройка приоритетов осуществляется с помощью регистров *Interrupt Priority Registers* ([STM32F3xxx and STM32F4xxx Cortex-M4 programming manual](http://www.st.com/st-web-ui/static/active/en/resource/technical/document/programming_manual/DM00046982.pdf)). Нас будут интересовать регистры Interrupt Priority Register 2 (0xE000E4008) и регистр Interrupt Priority Register 5(0xE000E401C).  
Пока не будем изменять приоритеты. Пусть будут одинаковы для обоих прерываний.

Обработка прерываний

Функции обработчики прерываний — ни что иное, как просто функции языка C, который ни чего не получают и не возвращают (и правильно — не от кого и не кому).  
*Главное правило — обработка прерываний должна осуществляться как можно быстрее!!!* Иначе низкоприоритетные прерывания могут слишком долго ждать.  
  
После окончания обработки прерывания необходимо сбросить активность события, вызвавшего прерывание — «очистить прерывание». Очистка прерывания EXTI производится с помощью регистра EXTI\_PR . Обратите внимание: запись «1» очищает прерывание, запись «0» не имеет ни какого воздействия.  
  
Если с обработкой прерывания линии 0 EXTI все достаточно просто, то с группой линий 5-9 возникает вопрос — как определить какая линия вызвала прерывание. Узнать это можно проверкой бит регистра Pending register (EXTI\_PR) — [Reference manual](http://www.st.com/st-web-ui/static/active/en/resource/technical/document/reference_manual/DM00031020.pdf).

Создаем таблицу векторов и располагаем ее в правильном месте

Для тог, чтобы таблица векторов с правильными адресами функций обработчиков прерываний располагались в начале флеш памяти МК, создадим и подключим к проекту файл startup.c.

Содержимое файла

Использование 

@ ".intvec"

Располагает таблицу \_\_vector\_table в начале секции, объявленной в файле линкера. Сам файл можно посмотреть тут:  
  
  
Сама секция задается в начале ROM памяти. Адреса можно посмотреть [тут](http://www.st.com/st-web-ui/static/active/en/resource/technical/document/programming_manual/DM00023388.pdf) (документ, в котором описана адресация флеш памяти STM32):  
  
  
  
Комбинация директивы IAR и спецфункции IAR:

**#pragma segment="CSTACK"**

\_\_sfe( "CSTACK" )

Записывает в начале флеша указатель на верхушку стека.  
  
Саму таблицу заполняют адреса функций, реализующий вечный цикл. Исключение сделано только для интересующих нас функций:

**extern** **void** **EXTI\_Line0\_IntHandler**(**void**);

**extern** **void** **EXTI\_Line6\_IntHandler**(**void**);

В функции, вызываемой при старте, просто производится переход к 

**extern** **void** \_\_iar\_program\_start(**void**);

Это функция — main(). Сам символ можно переопределить, если возникнет желание:

Переходим к основному файлу

Для начала выпишем и переопределим все адреса и битовые поля, которые нам понадобятся.

Листинг

Обратите внимание на то, что значения спецрегистров МК объявлены как *volatile*. Это необходимо, чтобы компилятор не пытался оптимизировать операции обращения к ним, поскольку это не просто участки памяти и их значения могут изменяться без участия ядра.

Настраиваем группирование приоритетов

В первую очередь стоит настроить группировку приоритетов прерываний:

SCB\_AIRCR = SCB\_AIRCR\_GROUP22;

.Данное действие должно выполняться только один раз. В сложных проектах, использующих сторонние библиотеки стоит проверять данный факт. Изменение разбиения приоритетов на группы может привести к некорректной работе прошивки.

Включение тактирование используемой периферии

Напомню, что перед началом работы с периферийными блоками необходимо включить их тактирование:

*//Enable SYSCFG , GPIO port A and D clocking*

RCC\_AHB1\_ENR |= RCC\_AHB1\_ENR\_GPIOA|RCC\_AHB1\_ENR\_GPIOC|RCC\_AHB1\_ENR\_GPIOD;

RCC\_APB2\_ENR |= RCC\_APB2\_ENR\_SYSCFG;

Работать сразу с SYSCFG нельзя, нужно подождать несколько тактов. Но мы и не будем. Займемся инициализацией GPIO.

Инициализация GPIO

Светодиоды инициализируются так же как и в прошлый раз:

*//LED3 and LED5 initialization*

GPIOD\_MODER = (GPIOD\_MODER & (~GPIO\_MODER\_13BITS)) | GPIO\_MODER\_13OUT;

GPIOD\_MODER = (GPIOD\_MODER & (~GPIO\_MODER\_14BITS)) | GPIO\_MODER\_14OUT;

Кнопка PA0 и контакт PC7 инициализируются как входные:

*//PA0 and PC6 pins initialization*

GPIOA\_MODER = (GPIOA\_MODER & (~GPIO\_MODER\_0BITS)) | GPIO\_MODER\_0IN;

GPIOC\_MODER = (GPIOC\_MODER & (~GPIO\_MODER\_6BITS)) | GPIO\_MODER\_6IN;

Вот только для контакта PC6 необходимо включить подтяжку питания. Активация подтяжки производится с помощью регистра GPIOC\_PUPDR:

*//Enable PC6 pull-up*

GPIOC\_PUPDR = (GPIOC\_PUPDR & (~GPIOC\_PUPDR\_7BITS)) | GPIOC\_PUPDR\_6PU;

Настройка EXTI

И так, на нужно настроить следующие параметры — включить прерывания для линий 0 и 6, для линии 0 прерывание по растущему фронту, для линии 6 — прерывание по падающему фронту:

*//Set up EXTI*

EXTI\_RTSR |= EXTI\_LINE0;

EXTI\_FTSR |= EXTI\_LINE6;

EXTI\_IMR = EXTI\_LINE0|EXTI\_LINE6;

Осталось настроить пины каких портов подключены к линии EXTI (странное решение, например МК stellaris могут генерировать прерывание при любой комбинации пинов, у STM32 с этим сложнее):

*//EXTI to port connection*

SYSCFG\_EXTICR1 = (SYSCFG\_EXTICR1&(~SYSCFG\_EXTICR1\_0BITS)) | SYSCFG\_EXTICR1\_0PA;

SYSCFG\_EXTICR2 = (SYSCFG\_EXTICR2&(~SYSCFG\_EXTICR2\_6BITS)) | SYSCFG\_EXTICR2\_6PC;

Настройка NVIC

Осталось настроить приоритеты прерываний и маскировать их для инициации обработки. Обратите внимание, что регистры *NVIC\_IPR* доступны для побайтового обращения, что значительно упрощает доступ только к необходимым байтам приоритетов отдельных векторов прерываний. Достаточно только сделать сдвиг на величину номера вектора прерывания (см. листинг определений). Еще раз напомним, что EXTI Line 0 имеет 6 номер в таблице векторов, а EXTI line 5\_9 — номер 23. У STM32 значение имеют только старшие 4 бита приоритета:

*//Set interrupts priority*

NVIC\_IPR6\_REG = 0xF0;

NVIC\_IPR23\_REG = 0x0;

Для демонстрации приоритеты установлены различными.  
Теперь можно включить прерывания:

*//Enable interrupts*

NVIC\_ISER0\_REG |= NVIC\_ISER0\_6VECT | NVIC\_ISER0\_23VECT;

С этого момента нажатие на кнопку и закоротки PC6 и GND будет приводить к вызову функций обработчиков прерываний EXTI\_Line0\_IntHandler и EXTI\_Line6\_IntHandler соответственно.

Обработка прерываний

В функциях обработки прерываний в первую очередь необходимо очистить прерывание, после этого можно зажечь светодиоды. Для демонстрации приоритетов прерываний в один из обработчиков добавлен вечный цикл. Если приоритет прерывания с вечным циклом ниже приоритета второго — то оно не сможет быть вызвано. Иначе, оно сможет прервать первое. Я предлагаю вам самим попробовать различные знчения приоритетов прерываний и наглядно увидеть к чему это приводит (*ВНИМАНИЕ — не забудьте про группы прерываний!*).

**void** **EXTI\_Line0\_IntHandler**(**void**)

{

*//Clear interrupt*

EXTI\_PR = EXTI\_LINE0;

*//Turn on LED 3*

GPIOD\_ODR |= GPIO\_ODR\_13PIN;

}

**void** **EXTI\_Line6\_IntHandler**(**void**)

{

*//Clear interrupt*

EXTI\_PR = EXTI\_LINE6;

*//Turn LED4*

GPIOD\_ODR |= GPIO\_ODR\_14PIN;

**while**(1);

}

Вместо заключения

На всякий случай приведу полный листинг получившейся программы.

Листинг

Для проверки влияния приоритетов прерываний и приоритетов групп прерываний попробуйте менять приоритеты и наблюдать, что будет происходить (два бита — приоритет внутри группы, 2 бита — приоритет группы).