

Оглавление

STM32F4 это же просто и на русском языке	1
Часть 1. Создание проекта в Keil uVision	1
Часть 2. Мигаем светодиодом (начало).....	6
Часть 2. Мигаем светодиодом (продолжение).....	10
Часть 3. Первое знакомство с отладкой	16
Часть 4. Прерывания.....	19

STM32F4 это же просто и на русском языке

Автор – Андронников Игорь; ссылка на проект: <http://electro.luxmentis.ru/articles/stm32>

Часть 1. Создание проекта в Keil uVision

Оригинал статьи: <http://electro.luxmentis.ru/articles/23-stm32f4-eto-zhe-prosto-i-na-russkom-jazyke-chast-1-sozdanie-proekta-v-keil-uvision.html>

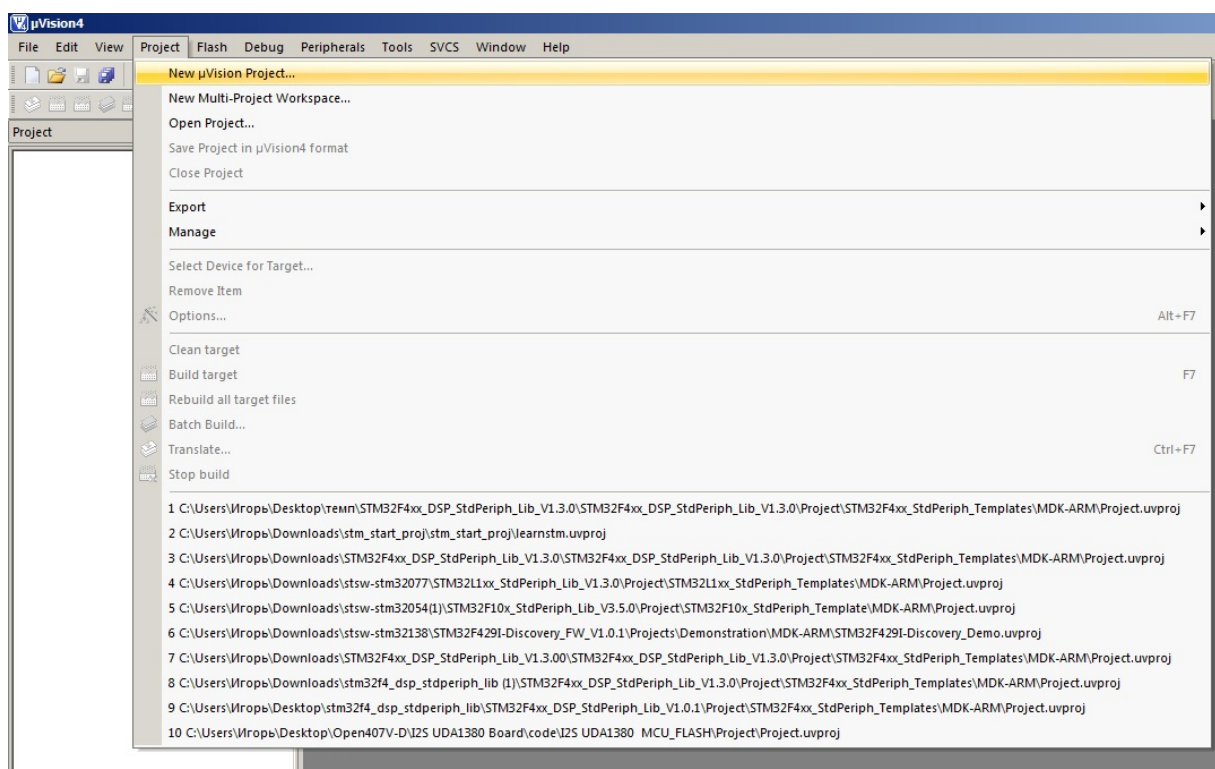
Доброго времени суток, друзья и коллеги, следующий цикл статей пишется для тех, кто имеет небольшой опыт работы с микроконтроллерами, имеет желание освоить STM32, но не знает, как и с чем к ним подойти, а также пугается англоязычной документации. Цель этих статей — научить основам работы с STM32 и дать необходимую информацию для того, чтобы читатель смог продолжить самостоятельно изучение данных микроконтроллеров. Хочу сразу сказать, если у кого-то что-то не получается или нашли какие-то ошибки и неточности — пишите сразу в комментариях, я постараюсь подсказать или оперативно внести подсказку.

Итак, приступим, а начнем мы с создания проекта в среде разработки, которую я на свой субъективнейший взгляд считаю лучшей вообще для ARM микроконтроллеров — Keil uVision.

Для начала необходимо скачать библиотеку (SPL далее) вот отсюда:

<http://www.st.com/web/en/catalog/tools/PF257901>

1. Запускаем Keil.
2. Заходим в верхнее меню Project -> New uVision project



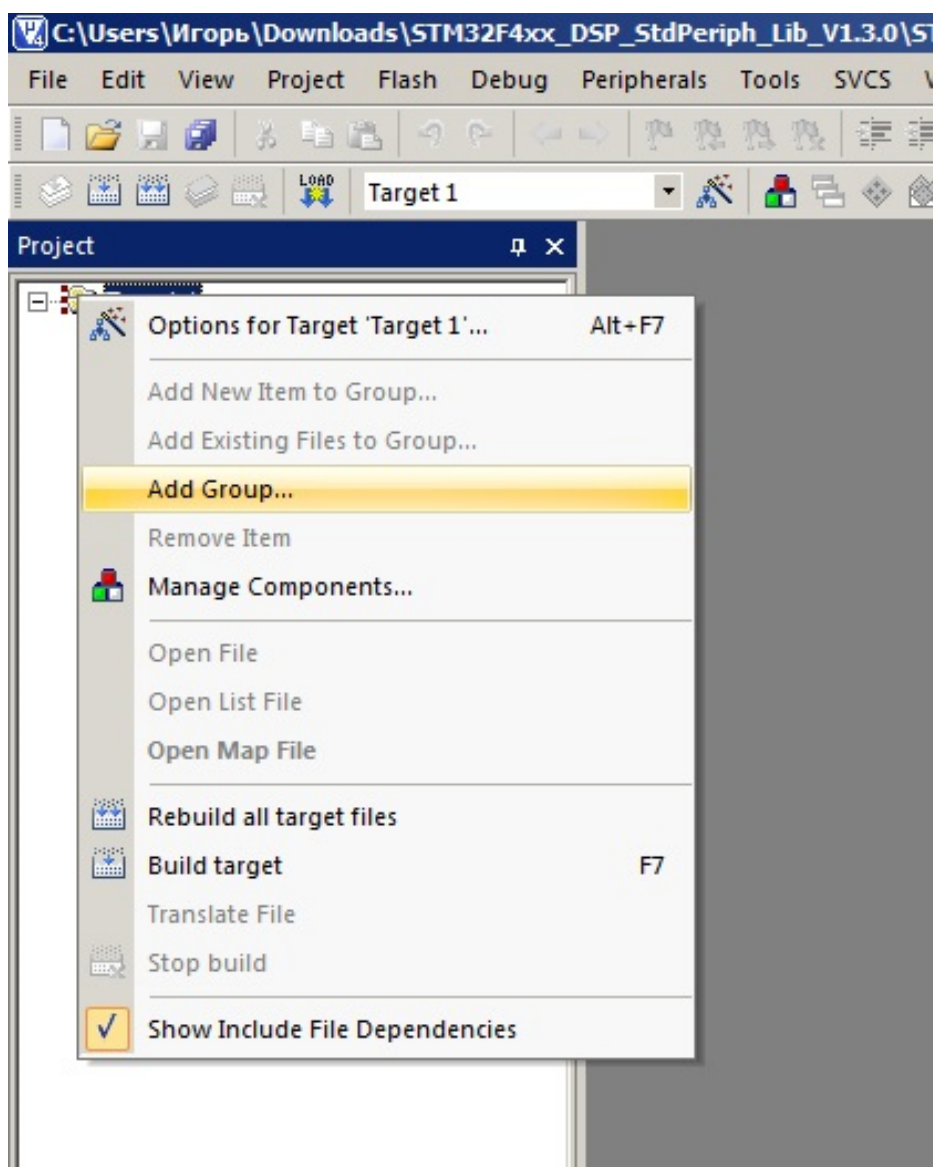
3. В открывшемся диалоговом окне заходим в папку вашей SPL библиотеки STM32F4xx_DSP_StdPeriph_Lib_V1.3.0\Project

и создаем там папку с названием вашего проекта. Затем заходим в эту папку и уже там создаем файл нового проекта. Постарайтесь не использовать русских символов.

4. В открывшемся списке выбираем микроконтроллер — в нашем случае STM32F407VG от STMicroelectronics.

5. На вопрос о копировании файла startup_stm32f...s отвечаем "Нет", мы это сделаем позже. Проект создан, осталось дело за малым — его сконфигурировать))))

6. Кликаем правой кнопкой мыши слева в дереве проекта на корень "Target 1" и в развернувшемся меню выбираем Add group... и называем новую группу SRC (не константа), проделываем ещё два раза для групп "User" и CMSIS.



7. Папки, структурирующие наш проект, созданы. Что надо сделать дальше? Совершенно верно — заполнить их файлами. Тут то начинается самое интересное (и чуточку рутинное).

а) Кликаем двойным щелчком по SRC и добавляем файлы библиотеки (STM32F4xx_DSP_StdPeriph_Lib_V1.3.0\Libraries\STM32F4xx_StdPeriph_Driver\src)

для тех периферийных модулей, которые вам могут потребоваться в проекте, если вы не знаете, что такое периферийные модули, то добавляете все файлы из этой папки кроме stm32f4xx_fmc.c(его не добавлять) — лишние удалить никогда не поздно))))).

б) Далее кликаем так же на CMSIS и добавляем startup_stm32f40_41xxx.s (из папки \STM32F4xx_DSP_StdPeriph_Lib_V1.3.0\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates\arm) и system_stm32f4xx.c (из папки

\STM32F4xx_DSP_StdPeriph_Lib_V1.3.0\Libraries\CMSIS\Device\ST\STM32F4xx\Source\Templates). Когда будете добавлять первый файл

не забудьте указать тип файла с расширением .s, а то тупо не увидите что добавлять.

в) Осталась папка User. С ней чуть сложнее — файлы, которые туда надо добавить, для начала нужно создать. Это файлы

main.c и, как вы уже догадались, main.h (его добавлять пока не надо, дальше поймете почему). Но не создавать же их пустыми, верно?

Жмем сверху меню file->new, добавляем в текст файла:

```
#include "main.h"
```

```
int main(void)
{
    while (1)
    {

    }
}
```

И сохраняем под именем main.c.

То же самое делаем с main.h, только текст должен выглядеть следующим образом:

```
#include "stm32f4xx.h"
```

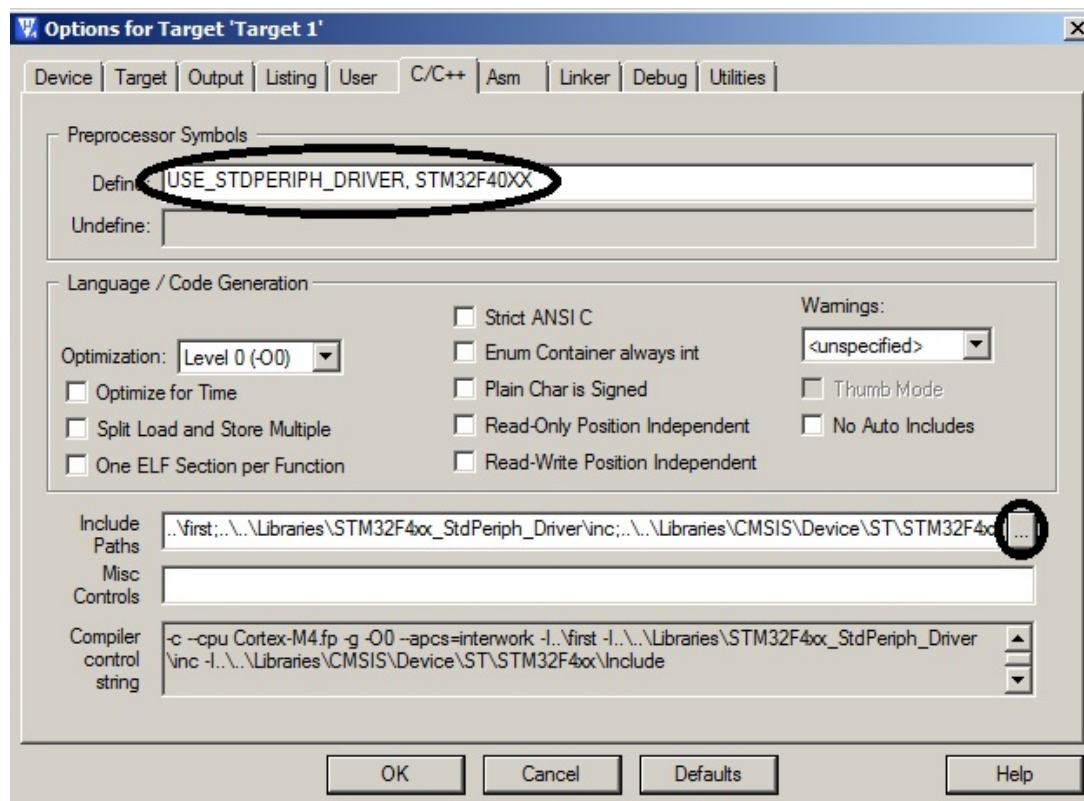
И сохраняем как main.h.

После чего добавляем уже существующий файл main.c в User

8. Идем в папку

\\STM32F4xx_DSP_StdPeriph_Lib_V1.3.0\\Project\\STM32F4xx_StdPeriph_Templates\\ и копируем оттуда файл «stm32f4xx_conf.h» и «system_stm32f4xx» в папку своего проекта.

9. Вроде с файлами разобрались... переходим к настройкам проекта. Жмем правой кнопкой на корень вашего проекта слева. Далее выбираем Options for Target... и переходим на вкладку C/C++, в поле define: прописываем USE_STDPERIPH_DRIVER, STM32F40XX.



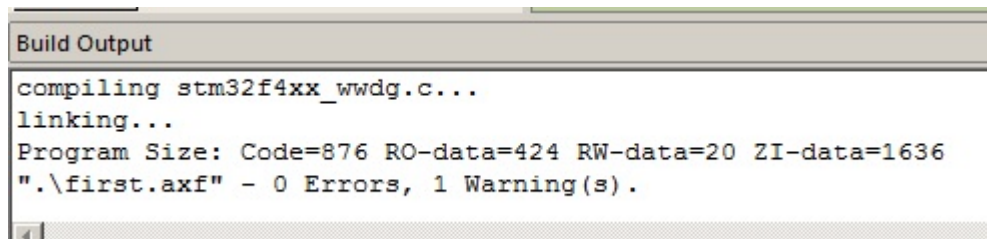
10. Далее нужно прописать пути к хидерам, для этого жмем в этой же вкладке на кнопку справа от поля Include Paths, и указываем путь к папкам, в которых хранятся все инклюды:

..\\..\\Libraries\\CMSIS\\Device\\ST\\STM32F4xx\\Include

..\..\Libraries\STM32F4xx_StdPeriph_Driver\inc

..\first – это сама папка проекта, поэтому, тут у каждого будет та папка, которую он создал в начале.

11. Ну, все, после этого жмем F7, и если, все сделали правильно, проект должен скомпилироваться без ошибок.



```
Build Output
compiling stm32f4xx_wwdg.c...
linking...
Program Size: Code=876 RO-data=424 RW-data=20 ZI-data=1636
"..\first.axf" - 0 Errors, 1 Warning(s).
```

Тех, у кого все получилось — поздравляю! Если не получилось, то ищите ошибку либо задавайте вопросы в комментариях.

Всё ещё только начинается — продолжение следует!

Часть 2. Мигаем светодиодом (начало)

Оригинал статьи: <http://electro.luxmentis.ru/articles/26-stm32f4-eto-zhe-prosto-i-na-russkom-jazyke-chast-2-migaem-svetodiodom-nachalo.html>

Итак, теперь у вас есть, надеюсь, сконфигурированный проект, если у кого-то нет, то прошу в предыдущую статью, где описано, как его получить. А мы, тем временем, продолжим осваивание STM32. Наш проект сконфигурирован для работы с микроконтроллером STM32F407 с помощью библиотеки стандартной периферии (SPL), все примеры и задания, которые будут указаны в начальной стадии обучения, адаптированы для использования с отладочной платой STM32F4-DISCOVERY.

Для начала, кто не в курсе, объясню: SPL — это библиотека стандартных периферийных модулей, создана для упрощения процесса освоения микроконтроллера и ускорения процесса разработки. В двух словах, библиотека устроена так, что всю работу с регистрами на низком уровне уже за вас сделали, интерпретировав обращение к регистрам и конфигурацию микроконтроллеру, к интуитивно понятным структурам и функциям. А для того, чтобы понять что записывать в поля структур, и какие параметры нужно передавать в функцию, чтобы добиться желаемого результата, в корневой папке библиотеки есть файл помощи `stm32f4xx_dsp_stdperiph_lib_um`, в котором есть вся необходимая документация для работы с библиотекой. Далее на простых примерах я постараюсь дать вам необходимые навыки для дальнейшего самостоятельного изучения SPL.

Приступим:

1. Откроем проект, который создали на прошлом занятии – необходимо просто открыть тот файл проекта, который вы создавали в самом начале.
2. Открыть `main.c` – для этого надо найти его слева в дереве проекта и вставить в него следующий код (пока просто вставьте, разберем его подробно на следующем уроке):

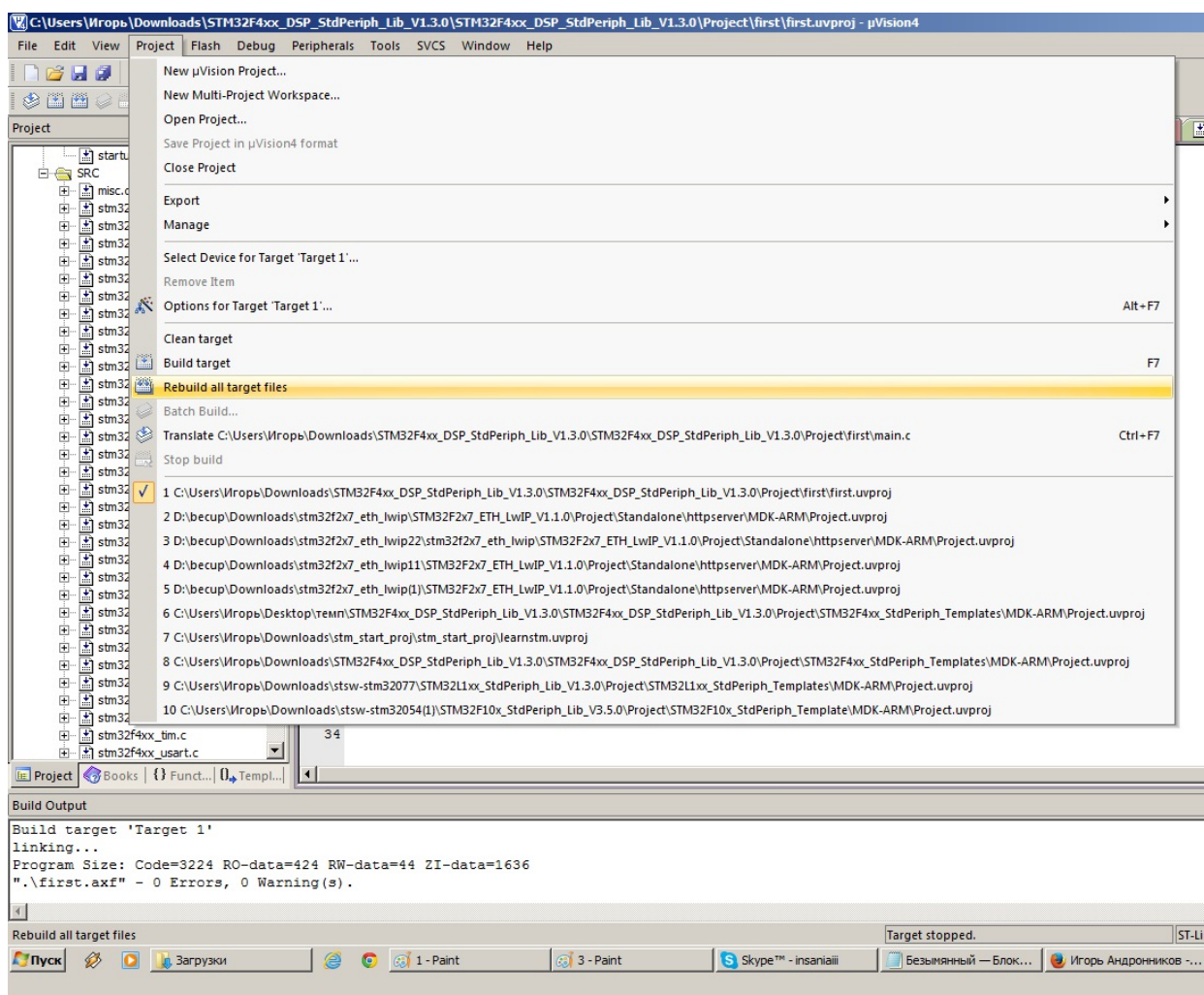
```
#include "main.h"

GPIO_InitTypeDef  GPIO_InitStructure;

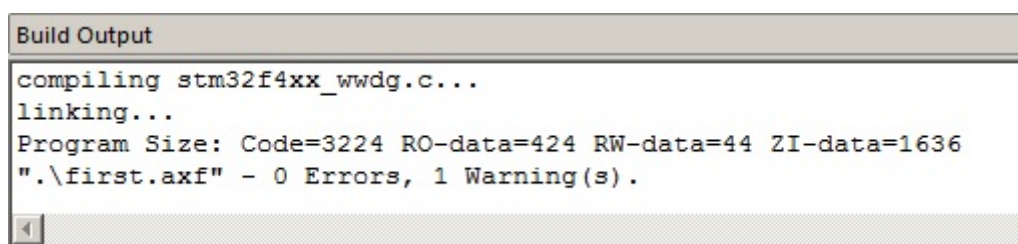
void delay (int t);

int main(void)
{
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOID, ENABLE);
    GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;
    GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
    GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
    GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
    GPIO_Init(GPIOID, &GPIO_InitStructure);
    while (1)
    {
        GPIO_WriteBit(GPIOID, GPIO_Pin_12, Bit_SET);
        delay(9999);
        GPIO_WriteBit(GPIOID, GPIO_Pin_12, Bit_RESET);
        delay(9999);
    }
}

void delay (int t)
{
    int tt = 0;
    for (tt = 0; tt<t; tt++);
}
```

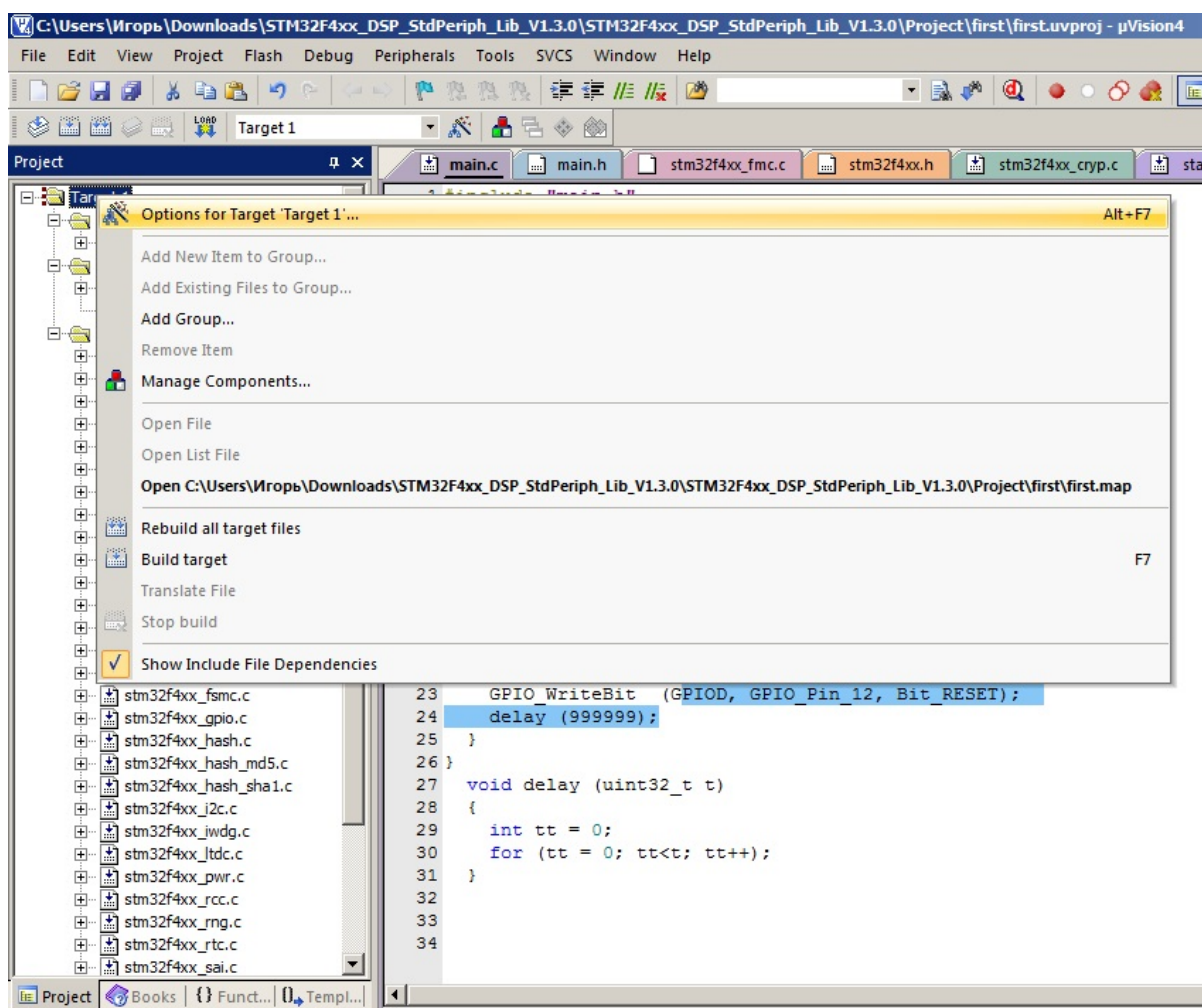
3. Сохраняем изменения, заходим в верхнем меню во вкладку Project и нажимаем Rebuild all target files. Если все правильно сделали, то проект должен скомпилироваться без ошибок.



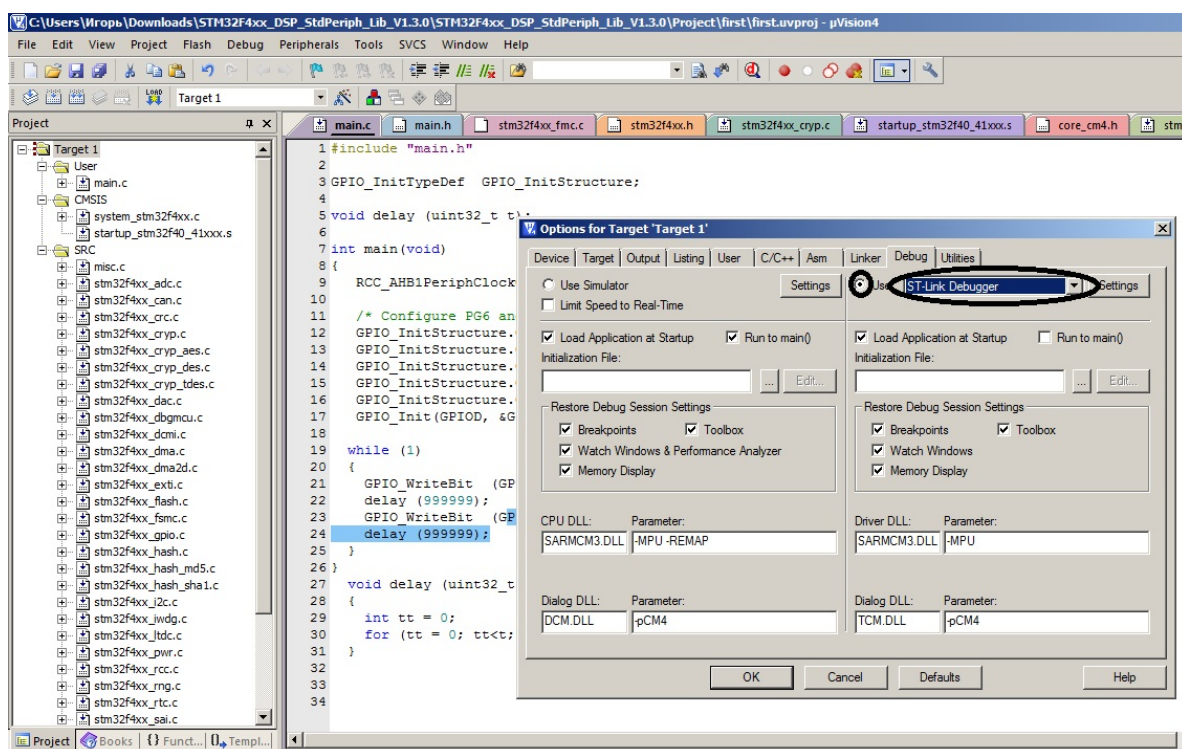
4. Ну, теперь самое интересное в этом уроке – проверить все на практике. Для этого берем STM32F4-DISCOVERY и подключаем к USB. Если драйвер ST-LINK ещё не установлен, это надо сделать – скачиваем отсюда драйвер для своей операционной системы:

http://www.st.com/web/catalog/tools/FM146/CL1984/SC720/SS1450/PF251168?s_searchtype=partnumber и устанавливаем его.

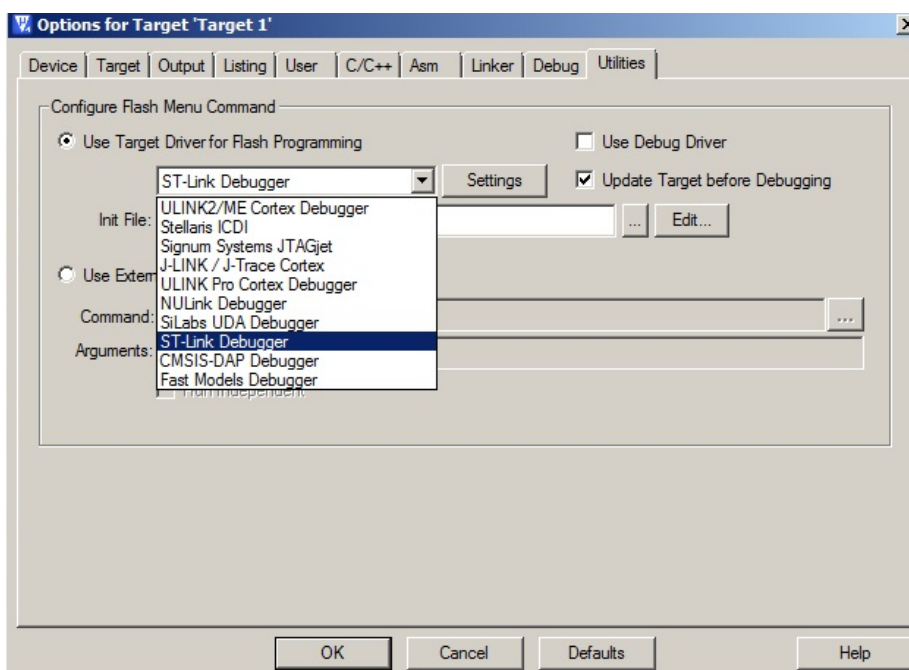
5. Жмем правой кнопкой слева на название нашего проекта и выбираем Option for target... — будем настраивать программатор и отладчик.



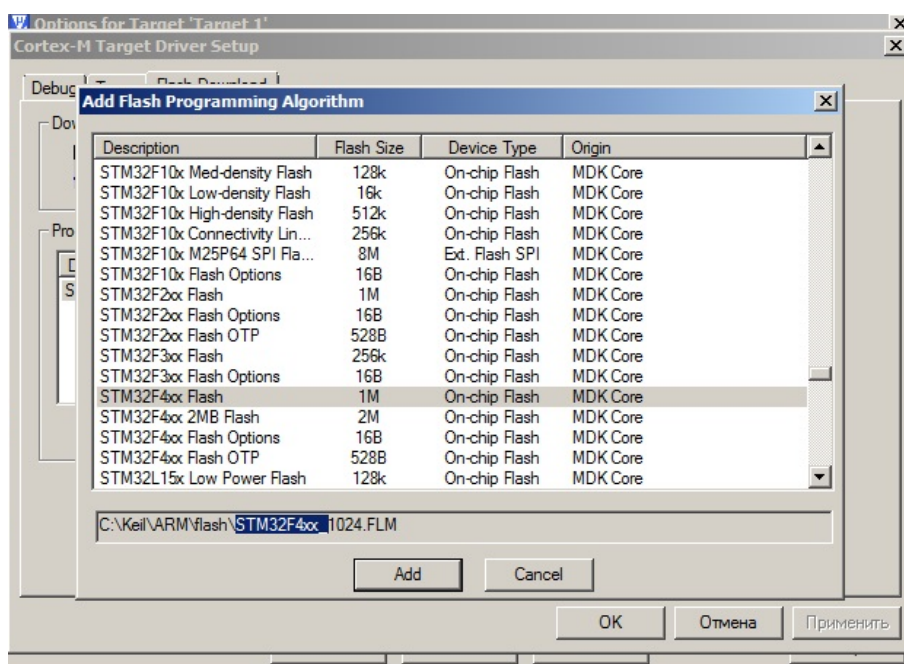
6. В открывшемся уже знакомом на прошлом уроке окне нажимаем на вкладку Debug (отладка). В правой стороне ставим галочку в поле рядом с USE: и выбираем из списка ST-LINK Debugger.



7. Переходим на вкладку Utilites, ставим галочку в поле возле поля Use Target driver for flash programming, в списке выбираем что? Правильно ST-Link Debbuger.



8. Жмем кнопку Settings. Жмем кнопку ADD и из списка выбираем STM32F4xx Flash 1M – это алгоритм для программирования флэш нашего МК. Жмем кнопку ADD. Жмем OK и выходим из настроек проекта.



9. Жмем CTRL+F5, тем самым запускаем режим отладки, попутно прошивая наш МК. После того как прошивка завершилась, вы попадаете в режим отладки, жмем F5, и если все сделали правильно, то светодиод зелененький будет мигать. Могу поздравить вас с первой прошивкой, зашитой в МК.

На следующем уроке мы подробно рассмотрим и изучим код этой простенькой программы, чтобы на его примере понять принципы работы SPL.

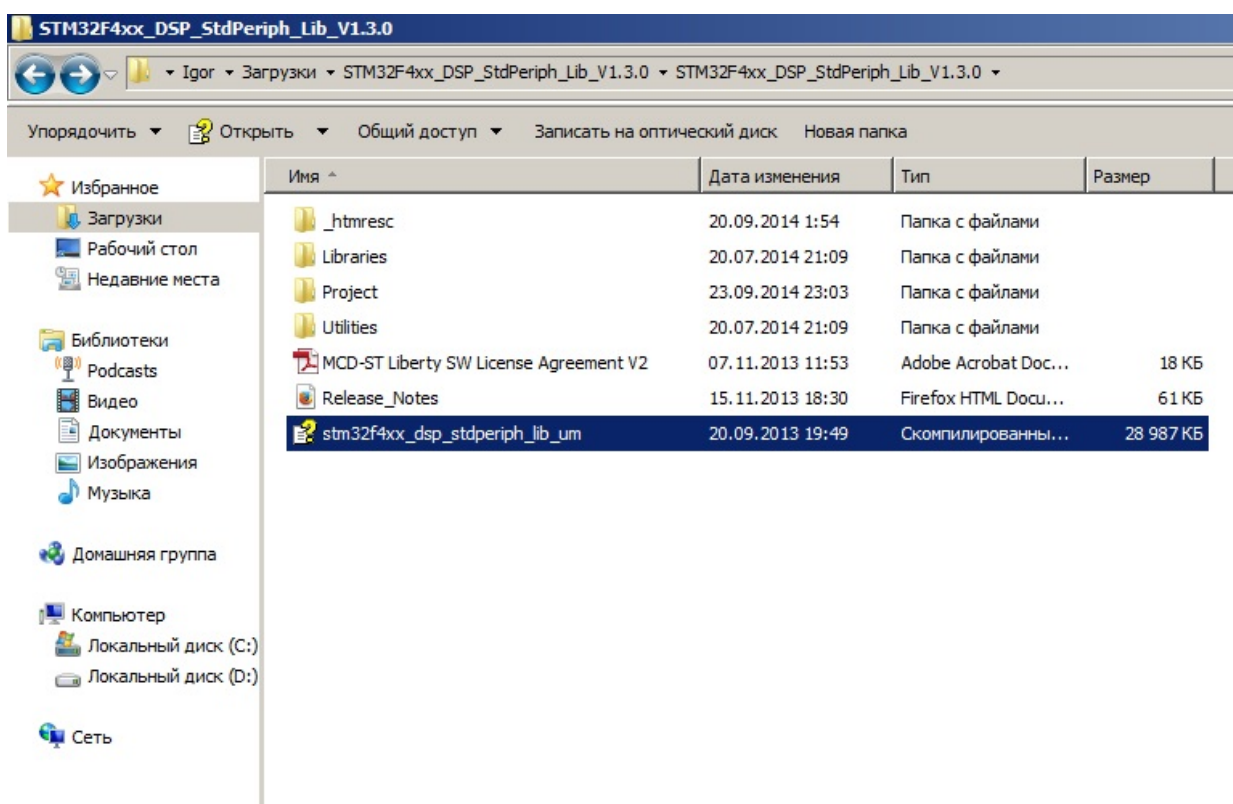
Если у кого-то что-то не получилось, пишите в комментариях, я разберусь индивидуально с каждым вопросом и, если потребуется, исправлю руководство.

Часть 2. Мигаем светодиодом (продолжение)

Оригинал статьи: <http://electro.luxmentis.ru/articles/27-stm32f4-eto-zhe-prosto-i-na-russkom-jazyke-chast-2-migaem-svetodiodom-prodolzhenie.html>

О том, как поморгать светодиодом, мы писали в предыдущей статье. В этой статье мы разберем механизм работы и использования SPL на примере нашей коротенькой программы мигания светодиодом.

Для начала, вкратце я расскажу, как работает и устроена SPL. А устроена она очень просто – обращение к регистрам STM32 уже написано разработчиками библиотеки и представлено в удобном для пользователя виде через структуры и функции. Те файлы, что на первом уроке вы добавляли в папку SRC, как раз содержат эти функции, которые вам придется вызывать для конфигурирования и использования периферии STM32. В корневой папке библиотеки SPL есть файл “stm32f4xx_dsp_stdperiph_lib_um” с описанием структур и функций библиотеки с подробными примерами – он будет необходим вам для дальнейшего изучения и понимания возможностей библиотеки.



Итак, начнем по частям разбирать, что же мы написали на прошлом уроке:

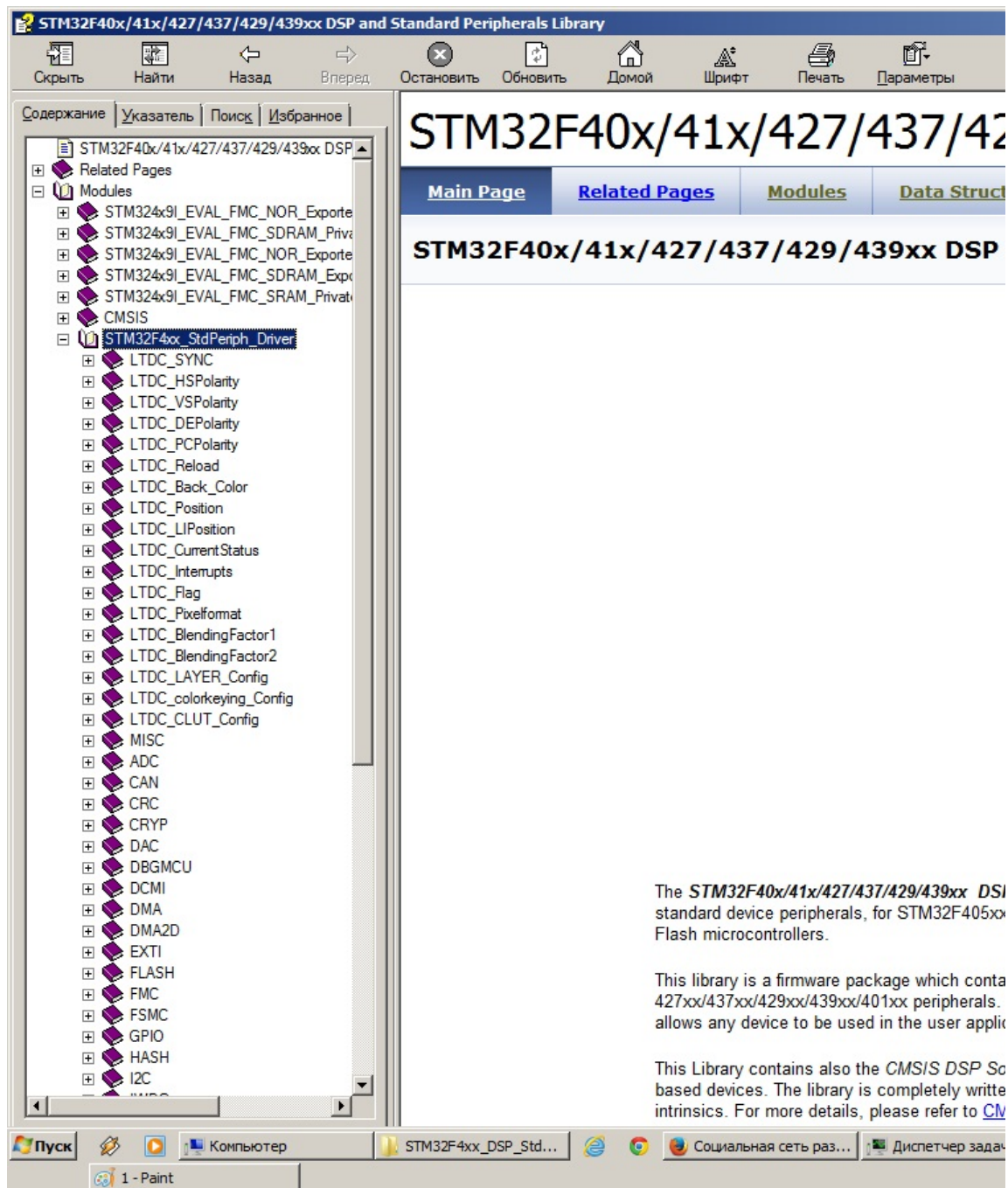
```
GPIO_InitTypeDef  GPIO_InitStructure;
```

Те, кто неплохо знаком с языком C, наверное, уже поняли, что это - объявление структуры с именем GPIO_InitStructure (можно указать любое имя). Данная структура нам необходима для конфигурации GPIO (порты ввода\вывода) STM32.

RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE) — данная функция включает тактирование для периферийного модуля, в данном случае, для GPIOD. Сразу хочу добавить, что у ARM своя философия, и необходимо отдельно включать тактирование для каждого периферийного модуля перед его использованием, если тактирование не включено, то даже не сможете его сконфигурировать. Для того, чтобы изучить, какие параметры можно передавать в функцию и что они значат – запускаем stm32f4xx_dsp_stdperiph_lib_um:

Слева в окне содержимое находим вкладку Modules, затем, развернув её, находим STM32F4xx_StdPeriph_Driver, далее находим RCC – тут есть вся информация для управления тактированием при помощи SPL.

Открываем вкладку Functions и находим там огромное количество непонятных нам функций.



Находим в этом огромном списке нужную нам функцию, а именно RCC_AHB1PeriphClockCmd и открываем.

Открываем и видим кучу параметров:

RCC_AHBPeriph,:	RCC_AHB1Periph_GPIOA: GPIOA clock RCC_AHB1Periph_GPIOB: GPIOB clock RCC_AHB1Periph_GPIOC: GPIOC clock RCC_AHB1Periph_GPIOD: GPIOD clock RCC_AHB1Periph_GPIOE: GPIOE clock RCC_AHB1Periph_GPIOF: GPIOF clock RCC_AHB1Periph_GPIOG: GPIOG clock RCC_AHB1Periph_GPIOG: GPIOG clock RCC_AHB1Periph_GPIOI: GPIOI clock RCC_AHB1Periph_GPIOJ: GPIOJ clock (STM32F42xxx/43xxx devices) RCC_AHB1Periph_GPIOK: GPIOK clock (STM32F42xxx/43xxx devices) RCC_AHB1Periph_CRC: CRC clock RCC_AHB1Periph_BKPSRAM: BKPSRAM interface clock RCC_AHB1Periph_CCMDATARAMEN CCM data RAM interface clock RCC_AHB1Periph_DMA1: DMA1 clock RCC_AHB1Periph_DMA2: DMA2 clock RCC_AHB1Periph_DMA2D: DMA2D clock (STM32F429xx/439xx devices) RCC_AHB1Periph_ETH_MAC: Ethernet MAC clock RCC_AHB1Periph_ETH_MAC_Tx: Ethernet Transmission clock RCC_AHB1Periph_ETH_MAC_Rx: Ethernet Reception clock RCC_AHB1Periph_ETH_MAC_PTP: Ethernet PTP clock RCC_AHB1Periph_OTG_HS: USB OTG HS clock RCC_AHB1Periph_OTG_HS_ULPI: USB OTG HS ULPI clock
NewState,:	ENABLE or DISABLE.

Как вы поняли, функция `RCC_AHB1PeriphClockCmd` может принимать два параметра — `RCC_AHBPeriph` и `NewState`. `RCC_AHBPeriph` — принимает значение из списка (выбираем тот периферийный модуль, для которого надо включить\выключить тактирование), а `NewState` — принимает значение либо `ENABLE`, либо `DISABLE`, в зависимости от того, что нужно, включить или выключить тактирование на выбранный модуль (`ENABLE` — включить).

Если вам захочется посмотреть внутренности функции, то ищем снизу строчку `Definition at line 1460 of file stm32f4xx_rcc.c`. Кликаем на 1460 — и вас переправляют к сорсу функции.


```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOD, &GPIO_InitStructure);
```

Для того, чтобы посмотреть, какие значения можно записывать в поля структуры и что они будут означать, обращаемся снова к файлу `stm32f4xx_dsp_stdperiph_lib_um`:

Открываем Modules -> STM32F4xx_StdPeriph_Driver -> GPIO -> Data Structures -> GPIO_InitTypeDef.

Справа в открывшемся информационном окне находим следующий блок:

Data Fields	
GPIOMode_TypeDef	GPIO_Mode
GPIOOType_TypeDef	GPIO_OType
uint32_t	GPIO_Pin
GPIOPuPd_TypeDef	GPIO_PuPd
GPIOSpeed_TypeDef	GPIO_Speed

Это и есть все поля данных для настройки в структуре GPIO_InitStructure.

Нажимаем на [GPIOMode_TypeDef](#) и видим следующую информацию:

<i>GPIO_Mode_IN</i>	GPIO Input Mode
<i>GPIO_Mode_OUT</i>	GPIO Output Mode
<i>GPIO_Mode_AF</i>	GPIO Alternate function Mode
<i>GPIO_Mode_AN</i>	GPIO Analog Mode

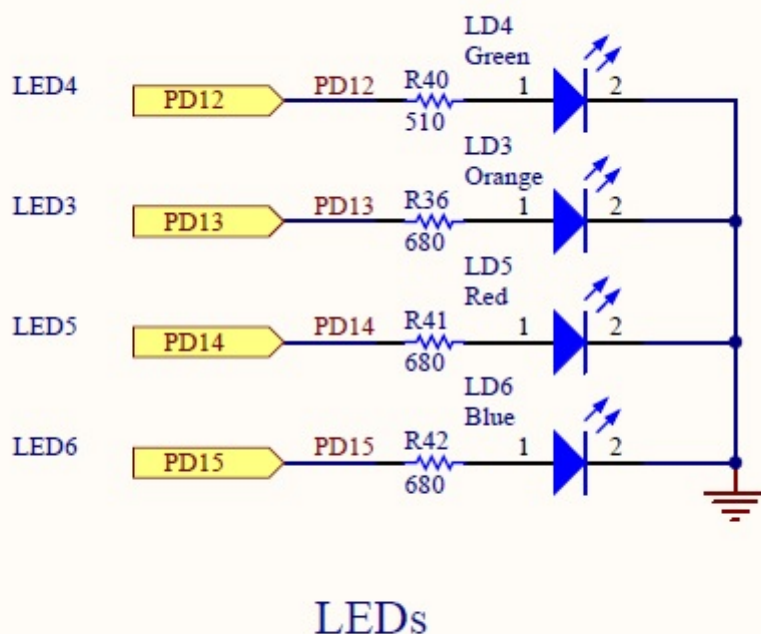
Здесь перечислены все доступные варианты использования GPIO в STM32 – то есть, ножка может быть сконфигурирована как выход, вход, аналоговый режим, и как альтернативная функция (в даташите в таблице назначения пинов можно увидеть альтернативные функции для каждой ножки).

Подобным способом найдите описание для каждого поля в данной структуре и тщательно разберитесь с их значением, и так, в дальнейшем, поступайте с каждой структурой и функцией, которую найдете в библиотеки, и которая вам понадобится. Это необходимо для максимального понимания возможностей SPL. Если есть трудности с пониманием каких-либо полей, структур или значений функций – пишите тут, в комментариях, и мы ответим на ваши вопросы.

GPIO_WriteBit(GPIOD, GPIO_Pin_12, Bit_SET); — собственно эта функция устанавливает (или, в зависимости от последнего параметра, сбрасывает) нужный бит в нужном GPIO. Описание можно найти так же в stm32f4xx_dsp_stdperiph_lib_um -> Modules -> STM32F4xx_StdPeriph_Driver -> GPIO -> Functions, так же, как и других функций GPIO, которые могут быть полезны, поэтому, советую досконально изучить их все.

Собственно, предлагаю вашему вниманию небольшое домашнее задание – по ссылке качаем схему STM32F4-DISCOVERY: http://www.st.com/st-web-ui/static/active/en/resource/technical/layouts_and_diagrams/schematic_pack/stm32f4discovery_sch.zip – ищем на схеме 4 светодиода, доступные для использования, и реализуем на них

бегущие огни. Задание достаточно простое, но имеет несколько разные варианты реализации, поэтому, предлагаю писать варианты в комментариях.



На следующем уроке разберем один из самых важных аспектов в любом микроконтроллере или процессоре — это отладка. Без неё невозможно написание никакой сколько-нибудь сложной программы.

Часть 3. Первое знакомство с отладкой

Оригинал статьи: <http://electro.luxmentis.ru/articles/28-stm32f4-eto-zhe-prosto-i-na-russkom-jazyke-chast-3-pervoe-znakomstvo-s-otladkoi.html>

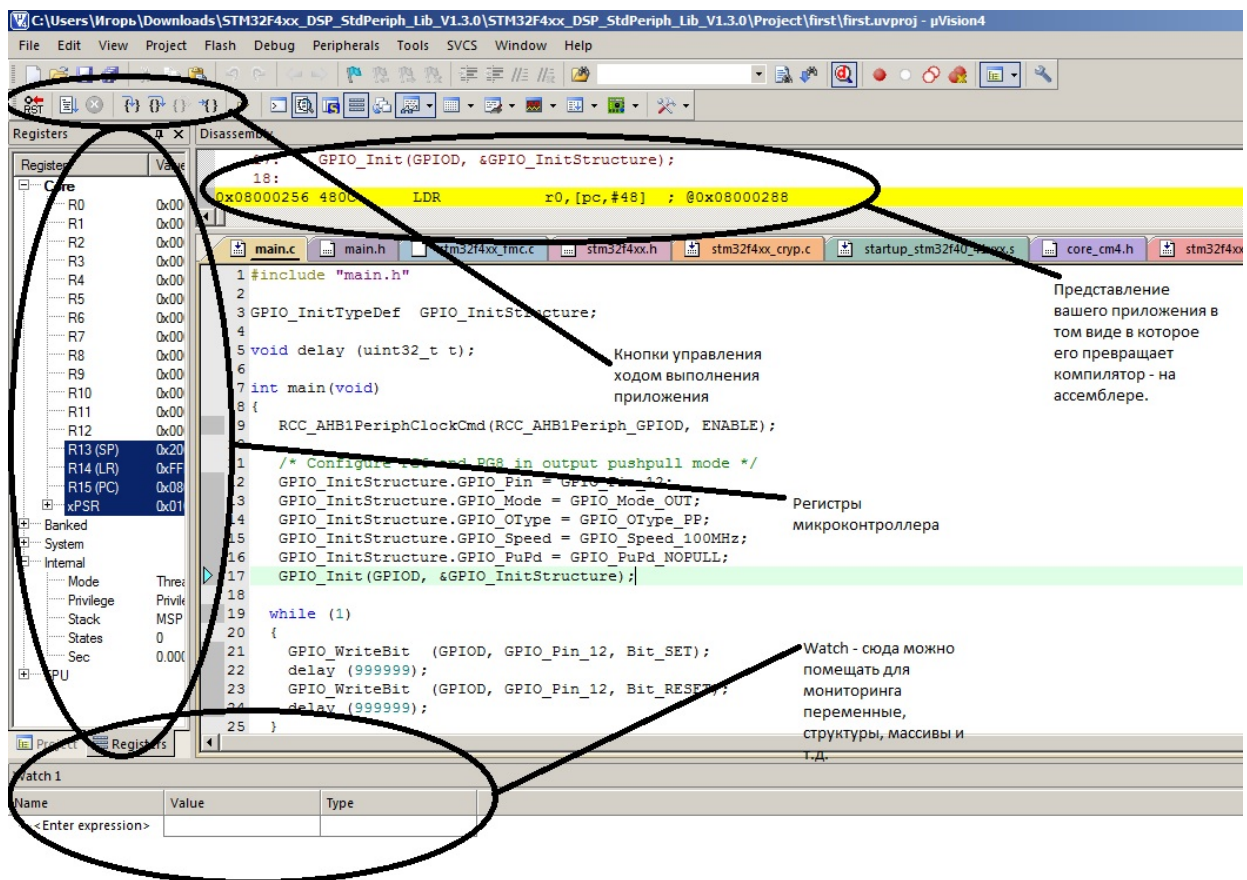
Теперь настало время поговорить о такой полезной вещи как отладка. Если вы уже не новичок в написании кода и работы с микроконтроллерами, то данный урок можно пропустить – скорее всего, не найдете ничего в нём нового. На данном уроке я постараюсь познакомить вас с начальными принципами отладки вашего приложения при помощи интерфейса JTAG/SWD для микроконтроллеров STM32. Конечно, в рамках одного урока невозможно полностью охватить все возможности такого мощного инструмента, поэтому, на последующих уроках будем постепенно, по мере необходимости, осваивать различные возможности и средства для отладки вашего программного кода и функционирования устройства.

Приступим:

Запустите уже созданный на прошлых уроках проект мигания светодиодом в среде Keil uVision.

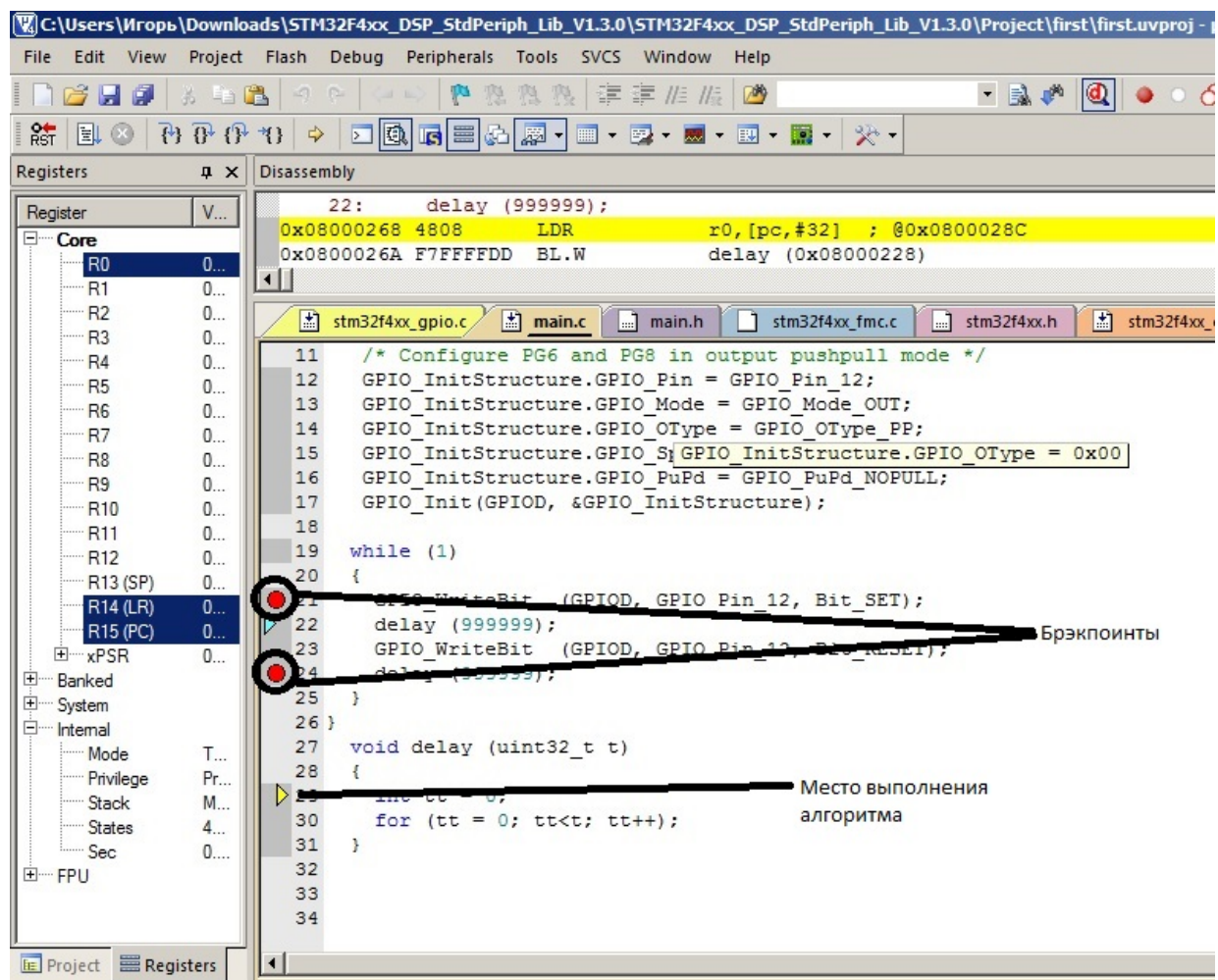
Подключите STM32F4Discovery к USB компьютера. (подразумевается, что вы выполняли все предыдущие уроки, и у вас установлены необходимые драйвера, и настроен проект для отладки с использованием ST-Link).

Нажмите CTRL+F5 и вы увидите, как ваш скомпилированный код заливается во флэш микроконтроллера (да, отладка в данном случае может выступать синонимом программирования МК), и вы переходите в режим отладки. Остановитесь и рассмотрите внимательно, что вы видите в этом режиме.



Начнем по порядку. Кнопки управления ходом выполнения приложения – основной ваш инструмент во время отладки, пожалуй, единственный инструмент, без которого нельзя обойтись. Расскажу о каждой кнопке, начиная с левой RST – тут все просто, при нажатии на эту кнопку микроконтроллер сбрасывается в начальное состояние, и вы можете запустить программу заново. Следующая кнопка RUN – запускает выполнение программы. Далее с крестиком кнопка STOP – останавливает выполнение программы. STEP – выполнять программу по одной строчке вашего кода. STEP OVER – выполнять по функции (не дискретизируя алгоритм функции). STEP OUT – выполнить участок кода до конца функции, в которой находитесь в настоящее время. Run to Cursor line – тут, думаю, поняли, что программа будет выполняться до той линии, на которой установлен курсор. Я думаю, что вы поняли, что можете выполнять код вашей программы любыми кусочкам, как будет удобно для вашей цели. Теперь рекомендую вам попробовать разные комбинации этих кнопок для того, чтобы понять, как это работает на практике при отладке не сложного вашего приложения.

Сразу под кнопками управления вы можете увидеть область с регистрами микроконтроллера и посмотреть их состояние. В окне, где код представлен в виде ассемблерных команд, можно так же наблюдать ход выполнения программы, только более детально – очень мощный инструмент для поиска неполадок, особенно в том случае, когда все остальные средства не помогли))) Ну и в самом низу располагается область Watch, которая служит для мониторинга состояния и изменения переменных и прочего формата содержания ваших данных.



Брекипункты – это точки останова выполнения кода, их можно ставить в любом месте вашей программы, для этого достаточно нажать правой кнопкой в серую область слева от

нумерации строк, повторное нажатие на брекпоинт его отменит. Максимальное количество брекпоинтов зависит от возможностей вашего отладчика, как правило, чем дороже ваш отладчик, тем больше возможно поставить брекпоинтов. Я предлагаю вам провести несколько экспериментов с установкой брекпоинтов в разных частях программы и понаблюдать выполнение, при этом используя кнопки управления ходом выполнения программы.

На этом наш мини урок можно считать завершенным. Выход из состояния отладки такой же, как и вход — CTRL+F5.

На следующем уроке познакомимся с обязательным элементом построения систем реального времени – прерываниями.

По всем замечаниям, вопросам и предложениям по материалу – пишите в комментариях.

Часть 4. Прерывания

Оригинал статьи: <http://electro.luxmentis.ru/articles/29-stm32f4-eto-zhe-prosto-i-na-russkom-jazyke-chast-4-preryvaniya.html>

И так, наконец-то прерывания, куда же без них... Да, на самом деле, нельзя построить ни одного серьезного устройства на микроконтроллерах, не владея системой прерываний, тем более, у STM32 в плане прерываний выбор ой как широк – забудьте про 16 прерываний, доступных в МК AVR. У STM32 система прерываний состоит из 240 плавно настраиваемых прерываний с возможностью динамического регулирования приоритета прерываний и 15 дополнительных или системных прерываний, которые обслуживают различные исключения и ситуации, вызванные в результате работы ядра МК. Стоит заметить, что вход в прерывания занимает всего 11 тактов, при том, что переход из одного прерывания займет у процессора всего 6 тактов.

Ну, на самом деле, это все теория, лирика и прочая малозначащая информация.))) Надо плавно переходить к практической части.

Для управления и контроля прерываний в STM32 есть модуль NVIC (Контроллер вложенных векторизированных прерываний), и, если вы внимательно изучили устройство SPL, то, наверное, заметили там структуры и функции, связанные с NVIC. На данном уроке мы модифицируем нашу программу так, чтобы уже второй светодиод мигал по нажатию кнопки, но не через стандартный опрос кнопки, а при помощи внешних прерываний.

Приступим:

Откроем наш проект, где мы мигали светодиодом. Так уж повелось, что этот проект будет с нами на протяжении всех лабораторных работ терпеть наши с вами издевательства.

Скопируем файлы `stm32f4xx.c` и `stm32f4xx.h` из папки `STM32F4xx_DSP_StdPeriph_Lib_V1.3.0\Project\STM32F4xx_StdPeriph_Templates`, в папку со своим проектом. Добавим файл `stm32f4xx.c` в папку `User` структуры вашего проекта в среде Keil.

Откроем этот файл для редактирования и заменим все содержимое в нем на

```
#include "stm32f4xx_it.h"

#include "main.h"

void NMI_Handler(void)
{
}

void HardFault_Handler(void)
{
    while (1)
    {
    }
}

void MemManage_Handler(void)
{
    while (1)
    {
    }
}
```

```

void BusFault_Handler(void)
{
    while (1)
    {
    }
}

void UsageFault_Handler(void)
{
    while (1)
    {
    }
}

void SVC_Handler(void)
{
}

void DebugMon_Handler(void)
{
}

void PendSV_Handler(void)
{
}

void SysTick_Handler(void)
{
}

```

И, соответственно, сохраним этот файл. В него мы будем помещать обработчики прерываний. Для тех, кто привык писать все в `main.c`, могу сказать, что это плохая практика – плохо структурированный текст программы мешает восприятию его и пониманию, и при каком-либо прилично сложном проекте вы рискуете сильно запутаться.

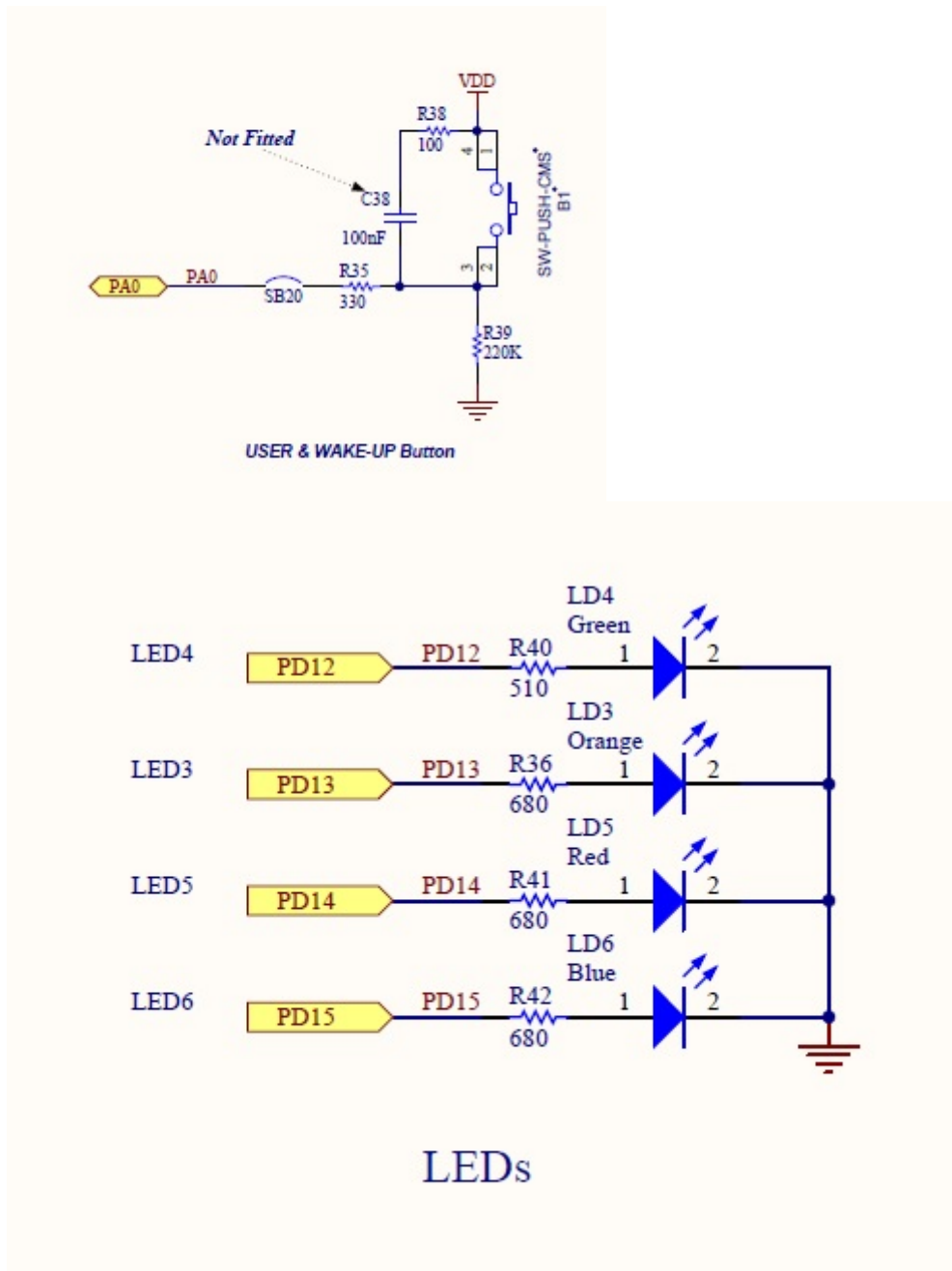
4. Возвращаемся в `main.c` и добавляем следующие фрагменты кода:

Там, где объявляли структуру

```

EXTI_InitTypeDef  EXTI_InitStructure;
NVIC_InitTypeDef  NVIC_InitStructure;

```



Сразу после конфигурации ножки для мигания светодиодом добавляем конфигурацию для ножки, к которой подключена кнопка и, которая будет сообщать нам сигнал для запуска

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0;
GPIO_Init(GPIOA, &GPIO_InitStructure);
SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0);
```

Ну и следует сконфигурировать ещё одну ножку для работы с ещё одним светодиодом, для этого ищем у себя в программе следующую строчку

```
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12;
```

и меняем её на `GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13;` — ножки со светодиодами все на одном порту, поэтому сконфигурируем сразу две.

И чуть ниже добавляем самое интересное - это конфигурацию самого прерывания:

```
EXTI_InitStructure.EXTI_Line = EXTI_Line0;
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
EXTI_InitStructure.EXTI_LineCmd = ENABLE;
EXTI_Init(&EXTI_InitStructure); //Конфигурируем модуль внешних прерываний

NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure); //Конфигурируем NVIC - указываем какой канал
//прерываний, приоритеты, ну и включаем всё это "веселье"
```

С конфигурацией вроде закончили – кому что не понятно или хотите разобраться со всеми параметрами, то загляните в файл помощи от SPL, если, даже, там будет не понятно, то пишите вопросы в комментариях.

5. Открываем файл stm32f4xx_it.c – как раз тот файл, который мы недавно скопировали и, в котором будут жить обработчики прерываний.
6. Добавляем функцию обработчика внешних прерываний для линии EXTI_Line0

```
void EXTI0_IRQHandler(void)
{
}
```

Эта функция будет вызываться всякий раз, когда срабатывает внешнее прерывание EXTI0 – в нашем случае по нажатию кнопки.

7. Теперь для полного счастья необходимо добавить в эту функцию код, чтобы можно было зажигать светодиод:

```
GPIO_ToggleBits (GPIOC, GPIO_Pin_13); //Будем мигать другим способом)
EXTI_ClearITPendingBit(EXTI_Line0); //сбрасываем флаг прерывания
```

8. Постарайтесь сами обработать дребезг контактов, чтобы зажегся светодиод стабильно.

Ну всё, компилируем, зашиваем, смотрим как работает.

В следующей статье поговорим о таймерах.

Если у вас возникли вопросы, замечания или предложения — пишите в комментариях, я с удовольствием на них отвечу.