Статья ориентирована на одного моего товарища, вполне перспективного программиста, надеюсь она ему поможет. Если вы не он, ~~закройте страницу~~ тоже можете читать.

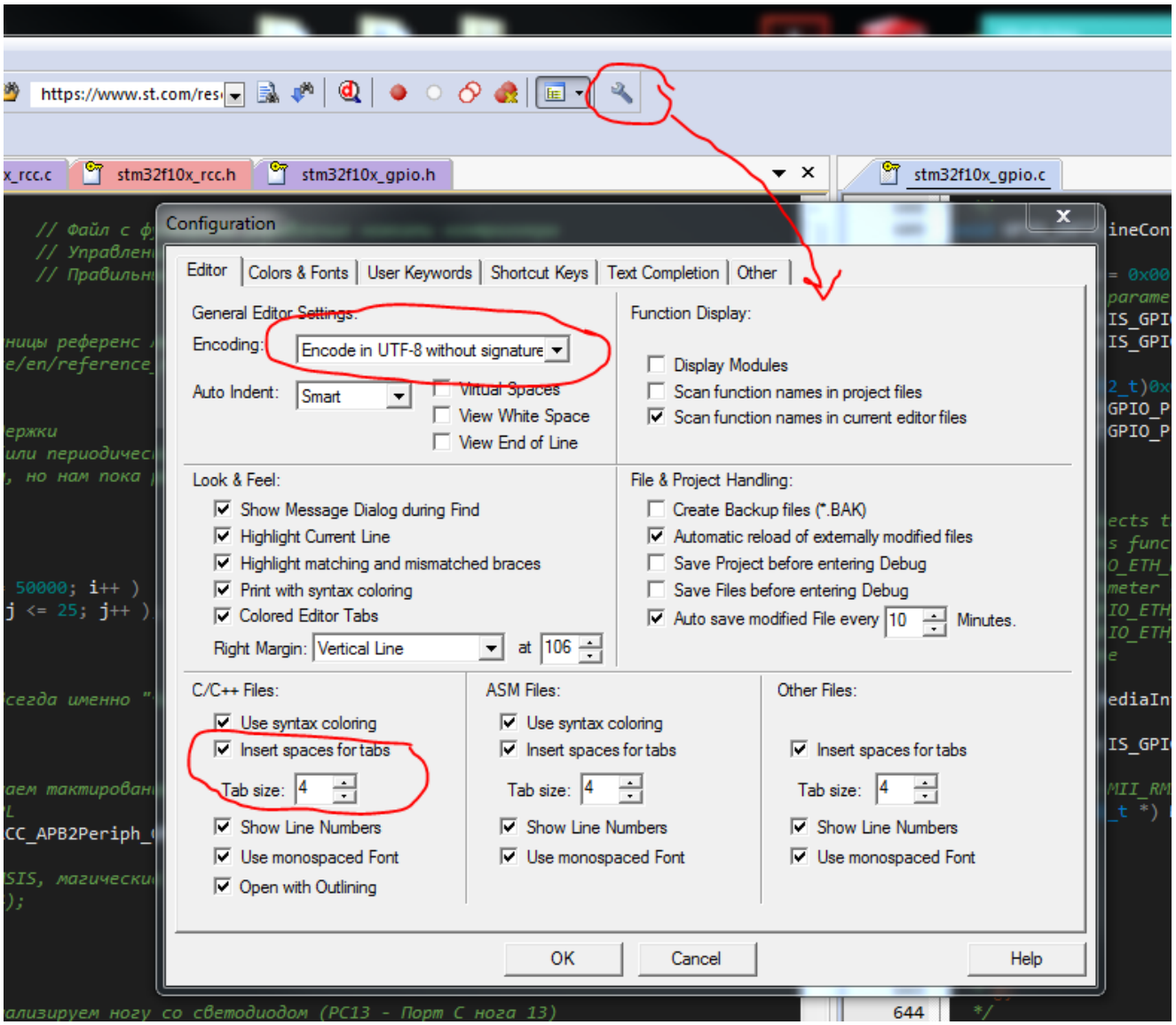
Основные вопросы этой заметки:

- как начать разрабатывать ПО для **`stm32`** в **`Keil`** ;
- что для этого необходимо.

Начать необходимо со второго вопроса. Необходимый минимум это среда программирования (**`Keil`**), программатор и отладочная плата. Можно конечно и в симуляторе запускать программу, но это не так инетресно, так что плата и программатор нужны. Где брать писать не буду, не моя это забота.

Для начала необходимо установить keil, его можно взять на официальном сайте, на данный момент последняя версия **`5.26`** , если у вас **`5.25`** не отчаивайтеся, там различий не много, есть весьма полезные функции, но они вам не нужны.

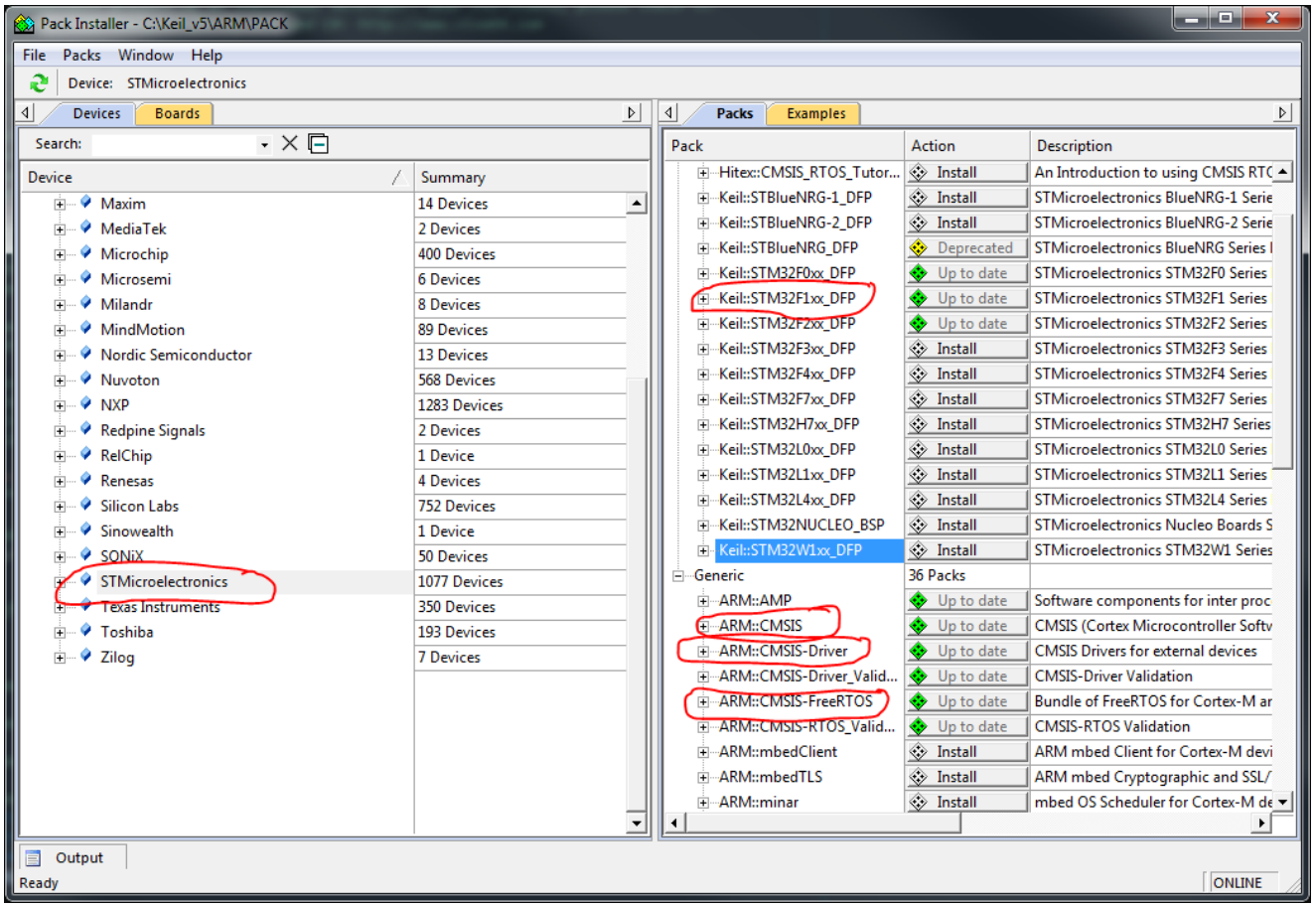
Я сразу же после установки на новую машину произвожу настройку среды программирования:



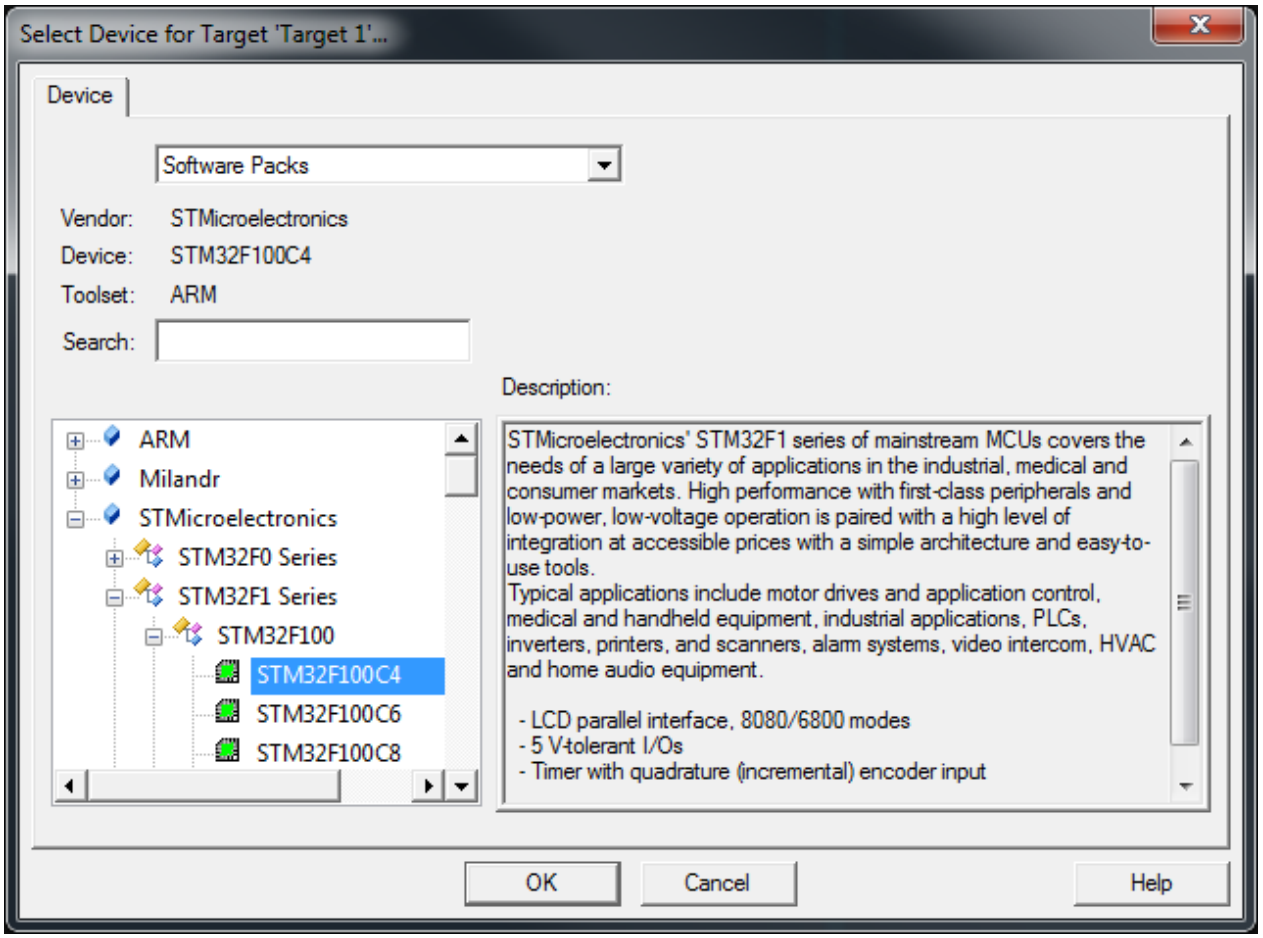
Вот так они выглядят. Теперь по пунктам, что и где настроено.

- Изменена кодировка файлов на юникод, так у вас гарантированно будут отображаться русские символы в комментариях
- Настраиваю замену табуляции на 4 пробела (это довольно холиварная тема, но я любю так), с такими настройками код гарантированно одинаково будет выглядить в любом редакторе.

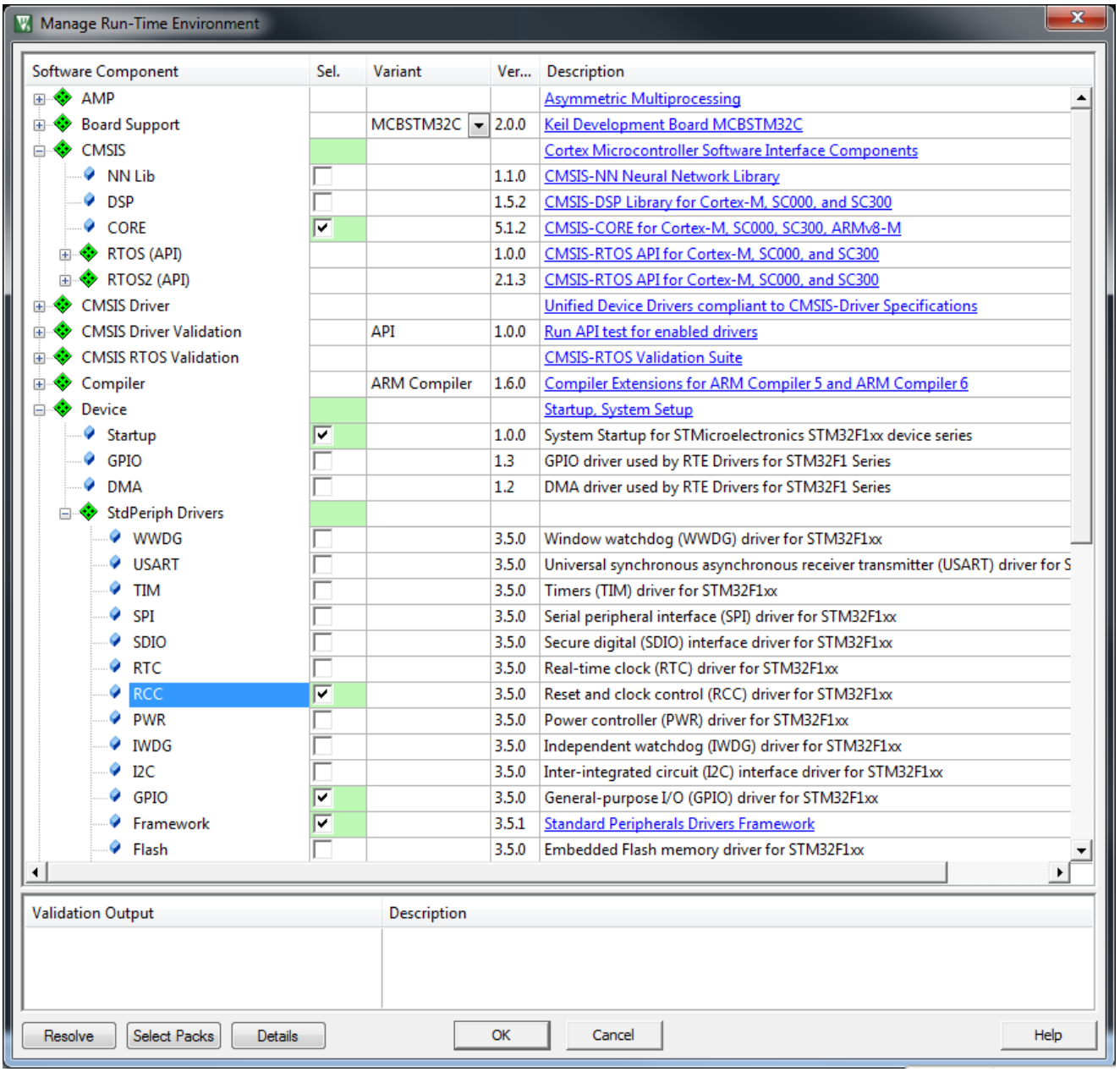
Теперь установим паки (различные библиотеки) необходимые для разработки под наш чип. Минимум установлен на картинке. FreeRTOS на самом деле не обязательно устанавливать, но пусть будет, ведь после игр с различной переферией будет пытаться оседлать её.



И так, все настройки сделаны, всё установлено, пробудем создать новый проект. Делается это через меню **`Project->New µVision Project...`** Сразу после нажатия откроется окно с предложением выбрать используемый чип, его название можно посмотреть на корпусе =)



Следующим этапом нам будет предложено выбрать необходимые в нашей работе паки. Отмечаем всё как на картинке.



- **`CMSIS`** (Cortex Microcontroller Software Interface Standard);
- **`Startup`** - это ассемблерный файл для запуска нашей программы, с содержимым можно будет ознакомиться самостоятельно;
- **`STDPeriph`** - Библиотека переферийных устройств:
  - **`Fraemwork`** - оснойной файл, подлючающий бибилотеку;
  - **`GPIO`** - для управления портами ввода-вывода;
  - **`RCC`** - для управления тактированием.

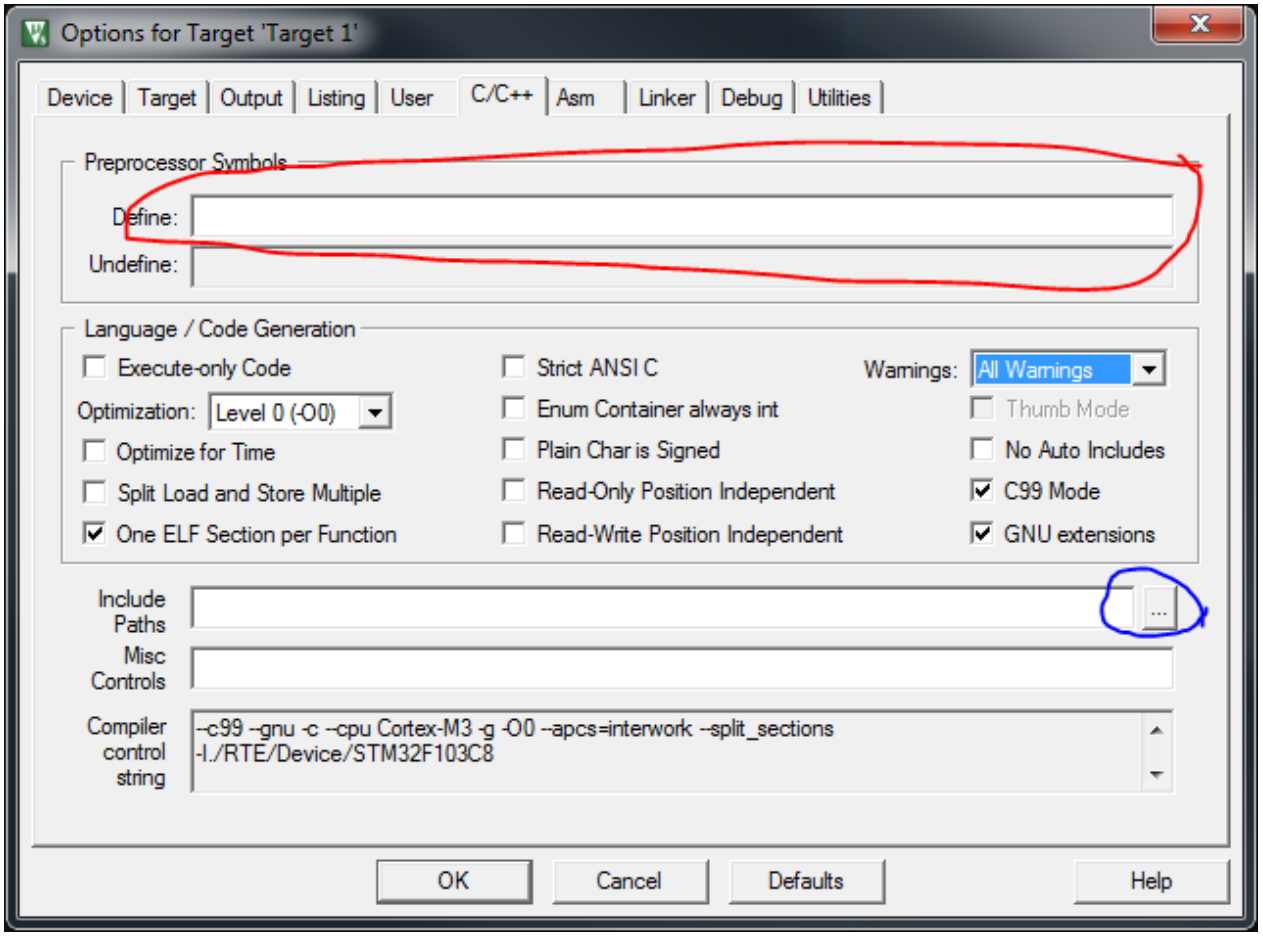
Всё выбрано, значить пожно приступать к следующему шагу. Настройка проекта. открывается через меню **`Project->Option for Target...`**

Рассмотрим наиболее интересные и необходимые вкладки.

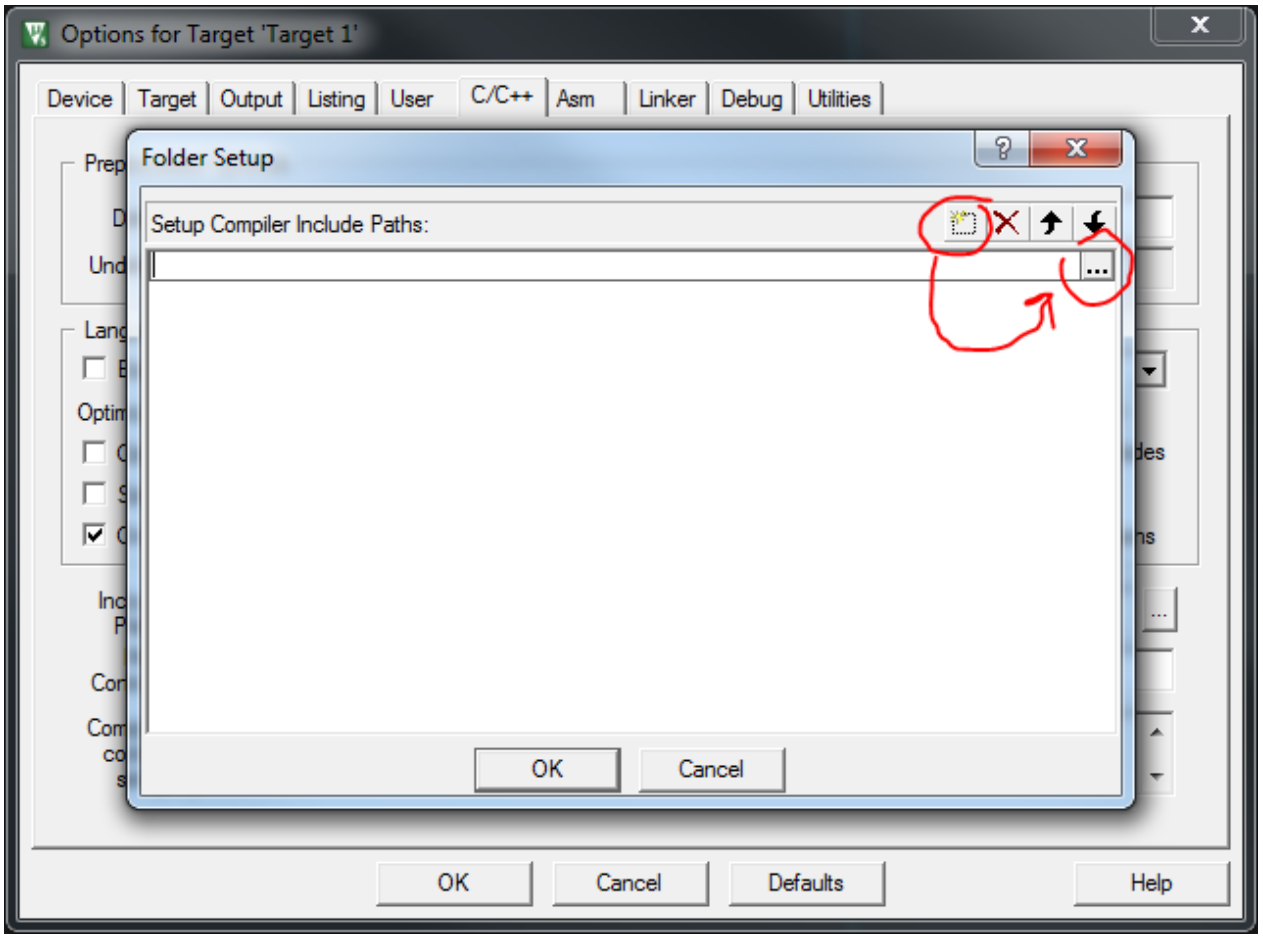
Здесь можно выбрать версию компилятора, для простоты будем использовать версию 5.

Описание изображения

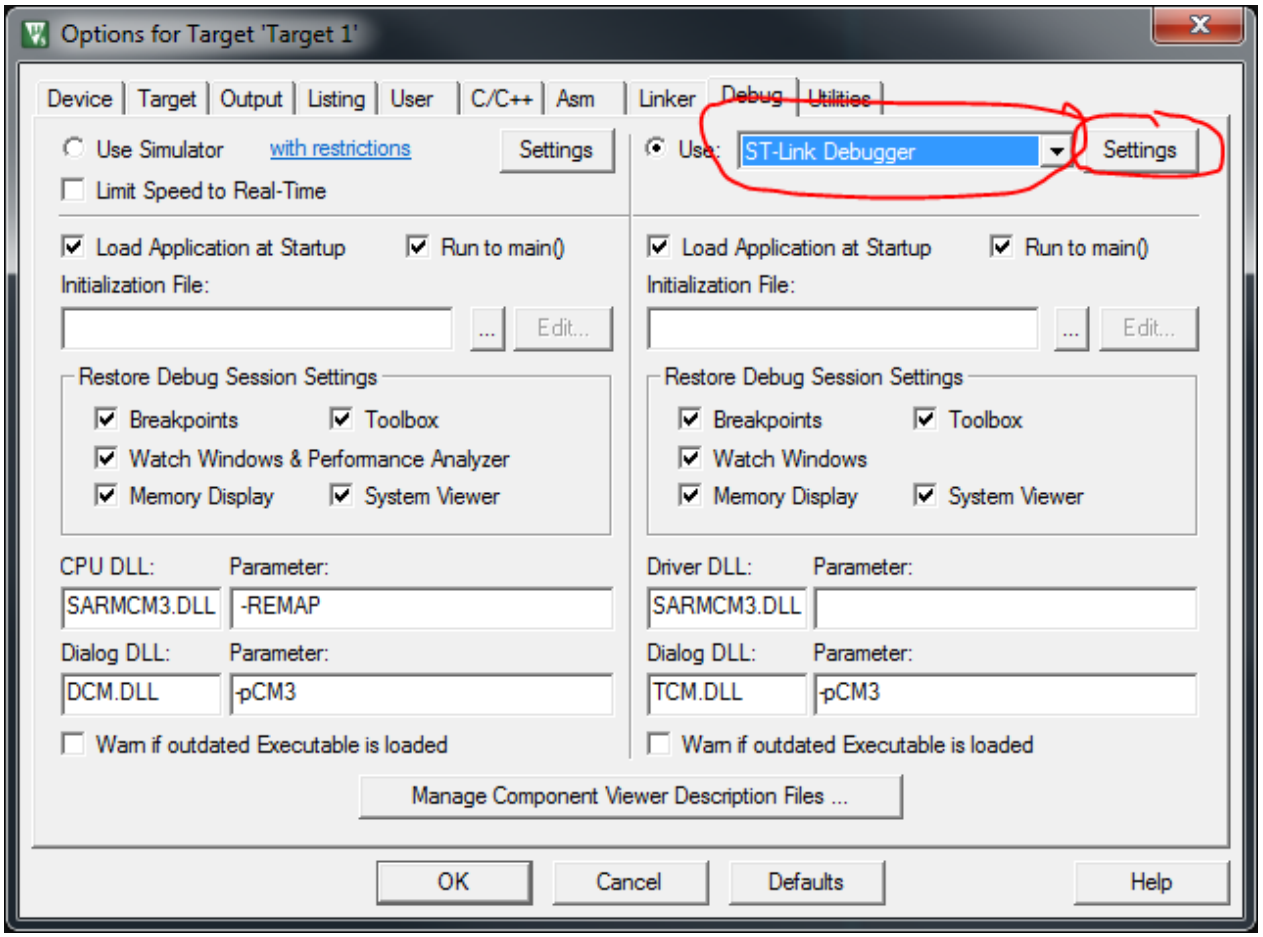
Вкладка для настройки компилятора. Здесь можно включить оптимизацию, выбрать уровень предупреждений, установить список инклудов и предопределенных макросов, и многое другое.



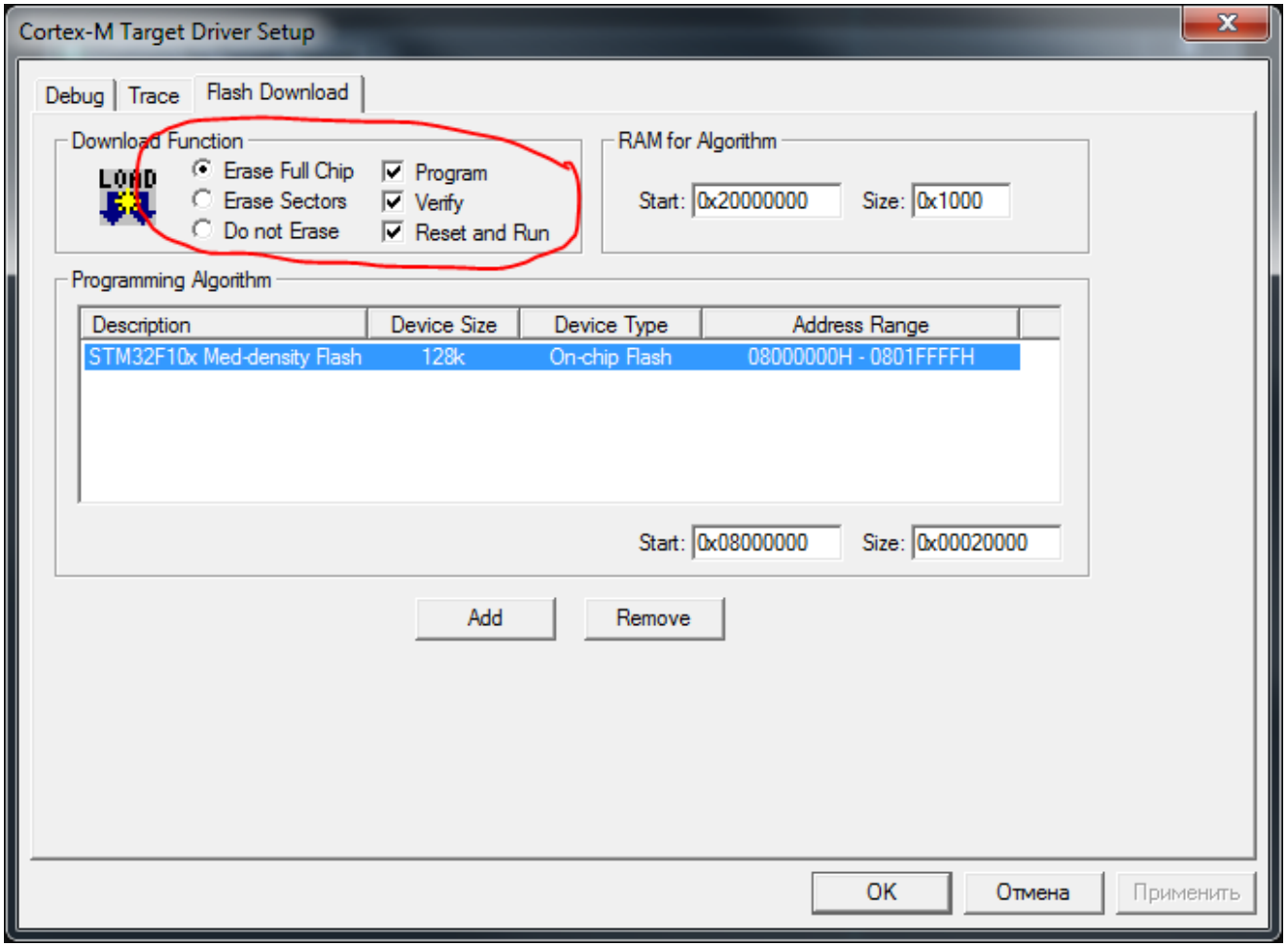
Нажатие на кнопку обведенную синим открывает диалоговое окно для добавления путей к используемым зголовочным файлам.



Следующая вкладка для настройки отладки, здесь можно выбрать используемый программатор и настроить его.



Настройки программатора. При загрузке прошивки будем стирать весь чип, программировать, верифицировать и запускать программу на исполнение.



Ну вот с настройка мы разобрались.

Приступим к написанию кода.

Программа для микроконтроллера выглядит в виде бесконечного цикла. Что бы это понять предлагаю ознакомиться с кодом:

```
/* Точка входа в программу */
int main( void )
{
    /*
    Здесь производится инициализация основных узлов программы и микроконтроллера
    */

    while( 1 )
    {
        /*
        Вечный цикл необходим для работы, иначе программа остановиться
        Сюда обычно помещают код который выполняется постоянно
        или оставляют цикл пустым если работают на прерываниях, можно совмещать.
        */
```

```
    }
}
```

Что нужно сделать чтоб ы помигать светодиодом?

1. Включить тактирование порта к которому подключен светодиод;
2. Настроить порт на выход;
3. в вечном цикле переключать вывод с небольшой задержкой то в `0` , то в `1` .

Вот и наш учебный код:

```
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"     // Файл с функциями управления ножками контроллера
#include "stm32f10x_rcc.h"      // Управление тактированием
#include <stdint.h>              // Правильные типы данных, вместо всяких int, char и
тому подобных

// Далее будут ссылки на страницы референс мануала, его можно сказать по ссылке:
// https://www.st.com/resource/en/reference_manual/CD00171190.pdf

// Примитивнейшая функция задержки
// Для орагнизации задержек (или периодических событий)
// обычно применяются таймеры, но нам пока рано, так что
// тупо мотаем такты

void Delay( void )
{
    for( uint16_t i = 0; i <= 50000; i++ )
        for( uint16_t j = 0; j <= 25; j++ );
}

// точка входа в программу, всегда именно "int main ( void )"
int main( void )
{

    // ---------------- Включаем тактирование порта GPIOC
    // 1. С использованием SPL
    RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOC, ENABLE );

    // 2. С использованием CMSIS, магические числа
//    RCC->APB2ENR |= (1 << 4);                       // см 146 стр.
    // ----------------

    // ---------------- Инициализируем ногу со светодиодом (PC13 - Порт С нога 13)
    // 1. С использованием SPL
    GPIO_InitTypeDef PortC;                     // Структура с необходимыми полями
```

```
    PortC.GPIO_Mode         = GPIO_Mode_Out_PP;     // Выход пуш-пул, см 164 стр.
     PortC.GPIO_Speed           = GPIO_Speed_10MHz;       // По сути это ток который
сможет обеспечить вывод
    PortC.GPIO_Pin          = GPIO_Pin_13;          // Номер ноги
    GPIO_Init(GPIOC, &PortC);                       // Применяем настройки

    // 2. С использованием CMSIS
//    GPIOC->CRH |= (0x00 << 22) | (0x01 << 20);      // см 172 стр.
    // -----------------

    // Основной цикл, программа ВСЕГДА должна зацикливаться!!!
    // Не всегда наполнен чем-то вразумительным, иногда может быть пустым,
    // например когда вся логика реализована в прерываниях.
    while( 1 )
    {
        // ---------------- Устанавливае ногу со светодиодом (PC13 - Порт С нога 13)
        // 1. С использованием SPL
        GPIO_WriteBit(GPIOC, GPIO_Pin_13, Bit_SET);
        // 2. С использованием CMSIS и ODR регистра, см 173 стр.
//        GPIOC->ODR |= (1 << 13);
        // 3. С использованием CMSIS и BSRR регистра, см 173 стр.
//        GPIOC->BSRR = (1 << 13);
        // ----------------

        Delay();

        // ---------------- Сбрасываем ногу со светодиодом (PC13 - Порт С нога 13)
        // 1. С использованием SPL
        GPIO_WriteBit(GPIOC, GPIO_Pin_13, Bit_RESET);
        // 2. С использованием CMSIS и ODR регистра, см 173 стр.
//        GPIOC->ODR &= ~(1 << 13);
        // 3. С использованием CMSIS и BRR регистра , см 174 стр.
//        GPIOC->BRR = (1 << 13);
        // 4. С использованием CMSIS и BSRR регистра , см 173 стр.
//        GPIOC->BSRR = (1 << 29);
        // ----------------

        Delay();
    }
}
// В конце файла для Кейла обязательна пустая строка! Хз зачем, просто нужна.
```

Код мигалки специально реализован несколькими возможными способами, используя **`STDPeriph`** и без него (только **`CMSIS`**)

Проект с этим кодом можно скачать здесь. Но лучше научиться создавать проект с нуля.

Будут вопросы пишите в комментариях, я обычно быстро отвечаю.

Как всегда - Спасибо за внимание и хорошего кодинга! =)

‹ Миландр. Неожиданное поведение CAN.
Библиотека CMSIS DSP. Так ли быстр целочисленный квадратный корень? ›