1) Создание проекта в Keil

В главном меню выбираем Project ->Create uVision Project… В открывшемся диалоговом окне выбираем папку, где будут храниться файлы проекта. После этого отрывается окно выбора микроконтроллера (рисунок 1). Выбираем STM32F107VC (такой контроллер установлен на отладочном стенде Mikroelektronika EasyMX PRO v7 for STM32).

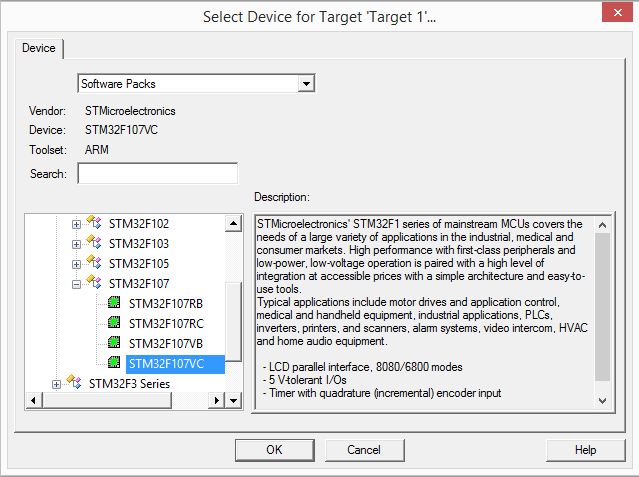


Рисунок 1.

Далее необходимо выбрать библиотеки. В разделе CMSIS выбираем CORE, в разделе Device выбираем пункт Startup, в подразделе StdPeriph Drivers выбираем пункты Framework, GPIO, RCC (рисунок 2). Если какой-то пункт требуется, но не выбран, внизу окна будет соответствующее сообщение.

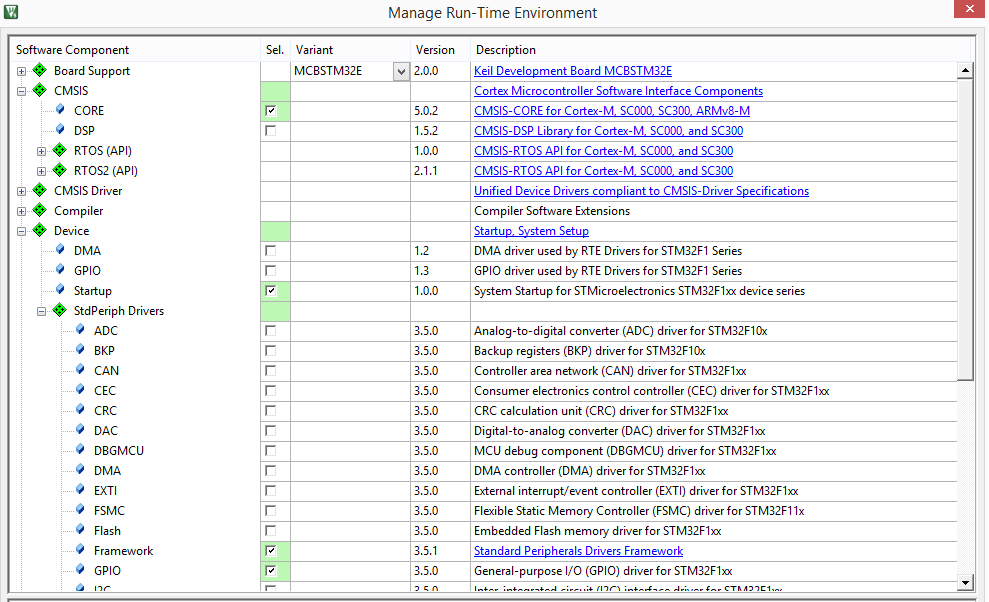


Рисунок 2

После нажатия кнопки ОК открывается окно редактора исходного кода (рисунок 3).

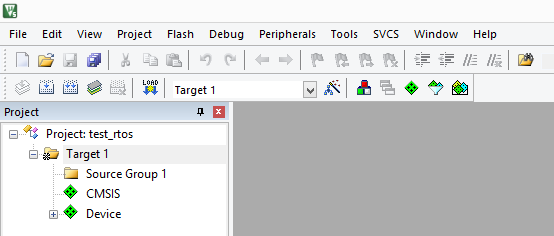


Рисунок 3

В Keil широко применяется использование групп файлов. В частности, одной из единиц является Target (цель). Использование цели позволяет в одном проекте собирать разные прошивки, содержащие некоторые общие файлы исходного кода. Фактически, разные target’ы могут содержать разные настройки проекта (параметры компиляции, тип микроконтроллера, конфигурационные файлы и т.д.). Файловые группы существуют только в рамках проекта Keil и не обязаны совпадать с папками операционной системы.

Напишем простейшую программу – мигание светодиодами. Выбрав в дереве проекта Source Group 1, в контекстном меню выбираем пункт «Add New Item to Group ‘Source group 1’». В открывшемся окне (рисунок 4) выбираем тип файла C File(.c), вводим название main, в пути к файлу дописываем \src\main\, чтобы создаваемый файл исходного кода хранился в отдельной папке. В папке верхнего уровня \src в дальнейшем будем хранить файлы исходного кода, разбитые по группам (FreeRTOS, Drivers и т.д.)

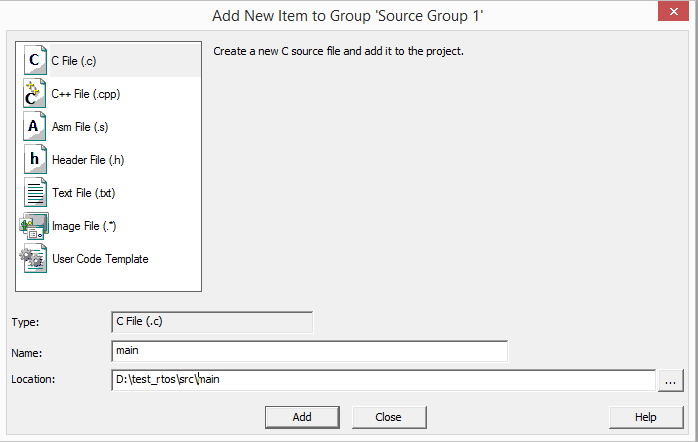


Рисунок 4

Напишем код, который управляет миганием двух светодиодов, причем один из них мигает вдвое чаще другого.

#include <stm32f10x\_gpio.h>

#include <stm32f10x\_rcc.h>

int main()

{

// подаем тактирование на порт D

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOD, ENABLE);

// настраиваем линии 0 и 1 порта D на выход

GPIO\_InitTypeDef gpioInit;

gpioInit.GPIO\_Mode = GPIO\_Mode\_Out\_PP;

gpioInit.GPIO\_Speed = GPIO\_Speed\_50MHz;

gpioInit.GPIO\_Pin = GPIO\_Pin\_0 | GPIO\_Pin\_1;

GPIO\_Init(GPIOD, &gpioInit);

while (1)

{

GPIOD->ODR ^= GPIO\_Pin\_0;

for (int i = 0; i < 1000000; i++);

GPIOD->ODR ^= GPIO\_Pin\_0;

GPIOD->ODR ^= GPIO\_Pin\_1;

for (int i = 0; i < 1000000; i++);

}

return 0;

}

Перед компиляцией необходимо в настройках проекта (рисунок 5) в разделе С/С++ необходимо поставить флажок напротив пункта C99 Mode (рисунок 6), а в разделе Debug установить в качестве отладчика ST-Link Debugger (рисунок 7). В настройках отладчика во вкладке Flash Download следует отметить флажок напротив пункта Reset and run (рисунок 8).

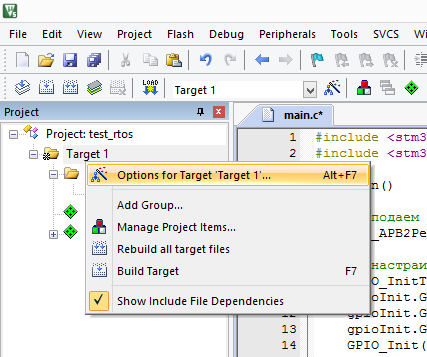


Рисунок 5

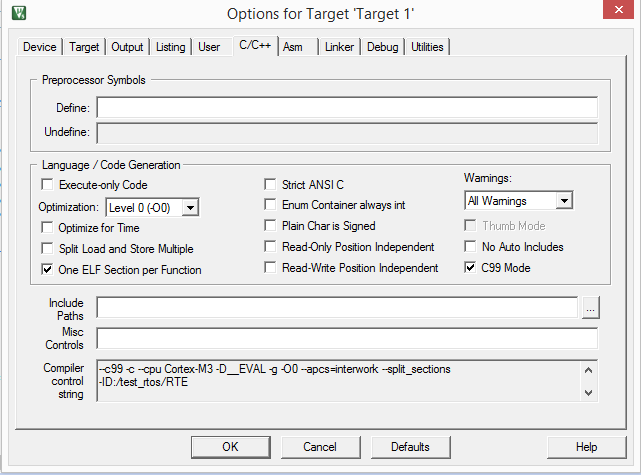


Рисунок 6

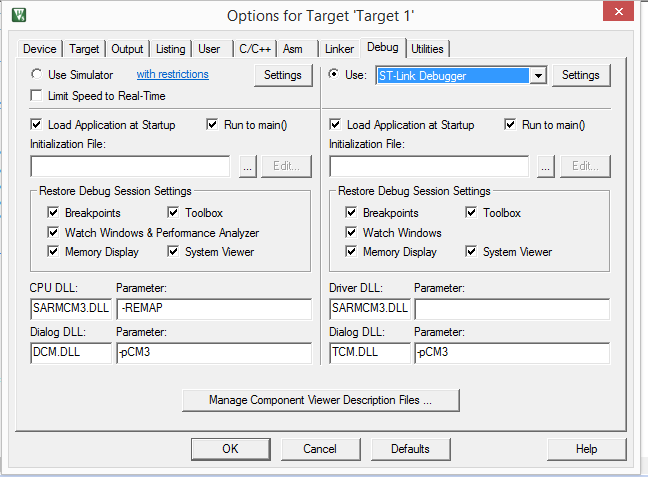


Рисунок 7

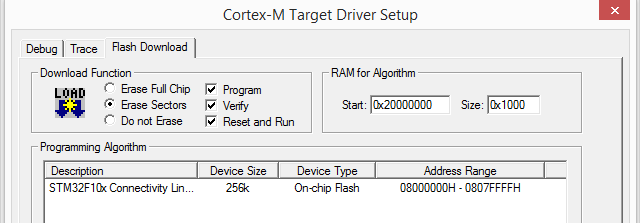


Рисунок 8

После изменений в настройках проекта необходимо нажать кнопку «Сохранить все».

Собираем проект (Project ->Build target), прошиваем в плату (Flash -> Download). Видим, что светодиод, подключенный к линии PD0, мигает в два раза чаще, чем светодиод, подключенный к линии PD1. Основной минус программы – задержка формируется за счет выполнения ядром процессора пустых операций, что нерационально. Разумнее в промежутках, когда состояние вывода контроллера ни меняется, отдавать процессорное время на полезные задачи. Для этих целей, в частности, применяются операционные системы реального времени, к которым относится FreeRTOS.

2 Добавляем FreeRTOS

Дистрибутив FreeRTOS можно получить на сайте <http://www.freertos.org/>.

Весь исходный код ядра находится в директории /Source. Его составляют следующие файлы:

1. tasks.c — планировщик, реализация механизма задач.

2. queue.c — реализация очередей.

3. list.c — внутренние нужды планировщика, однако функции могут использоваться и в прикладных программах.  
4. croutine.c — реализация сопрограмм (может отсутствовать в случае, если сопрограммы не используются).  
Заголовочные файлы, которые находятся в директории Source/Include:  
1. tasks.h, queue.h, list.h, croutine.h — заголовочные файлы соответственно для одноименных файлов с кодом.  
2. FreeRTOS.h — содержит препроцессорные директивы для настройки компиляции.  
3. mpu\_wrappers.h — содержит переопределения функций программного интерфейса (API‑функций) FreeRTOS для поддержки модуля защиты памяти (MPU).  
4. portable.h — платформенно‑зависимые настройки.  
5. projdefs.h — некоторые системные определения.  
6. semphr.h — определяет API‑функции для работы с семафорами, которые реализованы на основе очередей.  
7. StackMacros.h — содержит макросы для контроля переполнения стека.  
Каждая аппаратная платформа требует небольшой части кода ядра, которая реализует взаимодействие FreeRTOS с этой платформой. Весь платформенно‑зависимый код находится в поддиректории /Source/Portable, где он систематизирован по средам разработки (IAR, GCC и т.д.) и аппаратным платформам (например, ARM\_CM3, ARM\_CM4\_MPU). К примеру, поддиректория /Source/Portable/ RVDS/ARM\_CM3 содержит файлы port.c и portmacro.h, реализующие сохранение/восстановление контекста задачи, инициализацию таймера для создания временной базы, инициализацию стека каждой задачи и другие аппаратно‑зависимые функции для микроконтроллеров семейства STM32F10x, построенных на ядре Cortex M3 и компилятора, входящего в состав Keil. Отдельно следует выделить поддиректорию /Source/Portable/MemMang, в которой содержатся файлы heap\_1.c, heap\_2.c, heap\_3.c, heap\_4.c, heap\_5.c, реализующие различные механизмы выделения памяти для нужд FreeRTOS.

В директории /Demo находятся готовые к компиляции и сборке демонстрационные проекты. Общая часть кода для всех демонстрационных проектов выделена в поддиректорию /Demo/Common.

Чтобы использовать FreeRTOS в своем проекте, необходимо включить в него файлы исходного кода ядра и сопутствующие заголовочные файлы. Нет необходимости модифицировать их или понимать их реализацию.

Мы планируем использовать порт для микроконтроллеров STM32F107 и среду разработки Keil. В папке /src создаем папку FreeRTOS, в которую копируем все файлы и подпапку include из папки Source дистрибутива FreeRTOS. В эту же папку копируем файлы из папки portable\RVDS\ARM\_CM3. Файл portmacro.h переносим в подпапку include. Из папки /Source/Portable/MemMang необходимо скопировать файл heap\_2.c. Следует упомянуть также о заголовочном файле FreeRTOSConfig.h, который находится в каждом демонстрационном проекте (в нашем случае подойдет файл из директории Demo\CORTEX\_STM32F103\_Keil). FreeRTOSConf ig.h содержит определения (#define), позволяющие произвести настройку ядра FreeRTOS:  
1. Набор системных функций.  
2. Использование сопрограмм.  
3. Количество приоритетов задач и сопрограмм.  
4. Размеры памяти (стека и кучи).  
5. Тактовая частота МК.  
6. Период работы планировщика — квант времени, выделяемый каждой задаче для выполнения, который обычно равен 1 мс.  
Отключение некоторых системных функций и уменьшение количества приоритетов позволяет уменьшить расход памяти программ и данных.  
Необходимо скопировать файл FreeRTOSConfig.h в папку include.

Теперь необходимо подключить скопированные файлы в проекте Keil. Внутри Target1 создадим новую группу, которую назовем FreeRTOS и добавим в неё все скопированные C-файлы (рисунок 9)

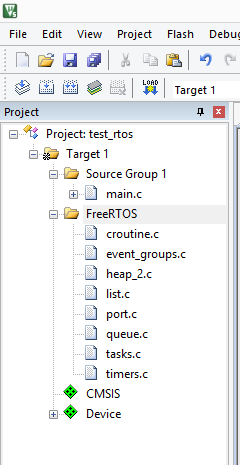
******

Рисунок 9

Далее в настройках цели во вкладке С/С++ необходимо указать пути для подключения заголовочных файлов (рисунок 10)

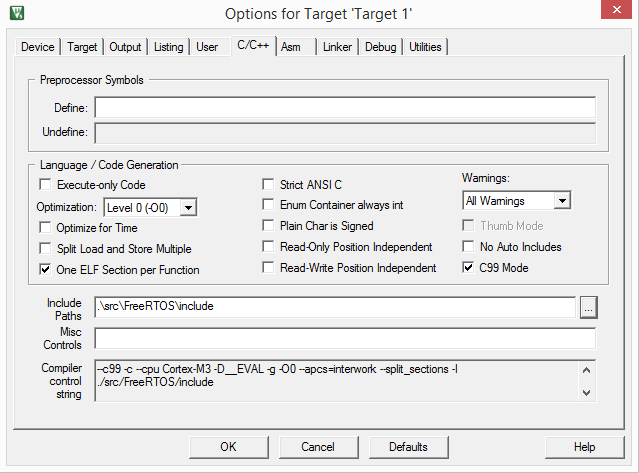


Рисунок 10

Для ARM Cortex-M3, ARM Cortex-M4 необходимо установить обработчики на векторы прерываний SysTick, PendSV и SVCCall.

Для этого в файл FreeRTOSConfig.h необходимо добавить следующие строчки:

// Setup Interrupt Vectors

#define xPortSysTickHandler SysTick\_Handler

#define xPortPendSVHandler PendSV\_Handler

#define vPortSVCHandler SVC\_Handler

Модифицируем код в файле main.с следующим образом.

#include <stm32f10x\_gpio.h>

#include <stm32f10x\_rcc.h>

#include <FreeRTOS.h>

#include <task.h>

void task1(void\* params);

void task2(void\* params);

int main()

{

RCC\_APB2PeriphClockCmd(RCC\_APB2Periph\_GPIOD, ENABLE);

GPIO\_InitTypeDef gpioInit;

gpioInit.GPIO\_Mode = GPIO\_Mode\_Out\_PP;

gpioInit.GPIO\_Speed = GPIO\_Speed\_50MHz;

gpioInit.GPIO\_Pin = GPIO\_Pin\_0 | GPIO\_Pin\_1;

GPIO\_Init(GPIOD, &gpioInit);

xTaskCreate(task1, "task1", configMINIMAL\_STACK\_SIZE, NULL, 2, NULL);

xTaskCreate(task2, "task2", configMINIMAL\_STACK\_SIZE, NULL, 2, NULL);

vTaskStartScheduler();

while (1)

{

}

return 0;

}

void task1(void\* params)

{

while (1)

{

GPIOD->ODR ^= GPIO\_Pin\_0;

for(int i = 0; i < 1000000; i++);

}

}

void task2(void\* params)

{

while (1)

{

GPIOD->ODR ^= GPIO\_Pin\_1;

for(int i = 0; i < 2000000; i++);

}

}

Скомпилируйте и про