Содержание

1 работа №1. Знакомство со средой разработки IAR Embedded Workbench for ARM (EWARM). Разработка программы с использованием библиотеки Standard Peripherals Library (SPL)

Цель работы:

- Знакомство со средой разработки IAR.
- Изучение принципов работы со светодиодами, знакомство со Standard Peripherals Library (SPL).
- Изучение принципов отладки программы в среде IAR.

1.1 Обзор платы STM32L-Discovery

В данной работе используется отладочная плата STM32L - Discovery на базе 32 МГц микроконтроллера STM32L152RB с 128 КБ Flash, 16 КБ RAM и 4 КБ EEPROM от STMicroelectronics. Микроконтроллер построен на основе ядра Cortex-M3. Данные микроконтроллеры отличаются ультранизким энергопотреблением (порядка 270 нА в спящем режиме). STM32L-Discovery — полноценный инструментарий, включающий в себя отладочную плату, программатор и отладчик с поддержкой самых популярных программных средств разработки от таких фирм как IAR, Keil и Atollic. Сигналы встроенного программатора-отладчика ST-Link выведены на внешний разъем, что позволяет в дальнейшем использовать STM32L-Discovery в качестве программатора-отладчика для своих собственных разработок [**?**].

Основные характеристики STM32L-Discovery:

- Микроконтроллер STM32L152RBT6
- Ядро Cortex-M3, 128 KB Flash, 16 KB RAM, 4 KB EEPROM
- Интерфейсы USB 2.0 FS, 3xUSART, 2xSPI, 2xI2C, 8 таймеров
- 24-канальный 12-бит АЦП 1мкс, компараторы, 2x12-бит ЦАП
- Полноценные часы реального времени
- Встроенный контроллер LCD 8x40

– Встроенный программатор ST-Link с возможностью программировать другие микроконтроллеры STM32.
– LCD дисплей 24x8 в форм-факторе DIP28
– Возможность измерения потребляемого тока
– Четыре светодиода:

    LD1 (красный/зеленый) для сигнализации обмена данных по USB

    LD2 (красный) для питания 3.3В

    Два пользовательских диода LD3 (зеленый) и LD4 (синий)

– Две кнопки (user и reset)
– Сенсорная клавиатура (четыре сенсорных кнопки или один слайдер)
– Все свободные выводы STM32L152RBT6 выведены на контактные площадки

1.2 Общие сведения о ядре Cortex

Семейство ARM Cortex — новое поколение процессоров, которые выполнены по стандартной архитектуре. В отличие от других процессоров ARM, семейство Cortex является завершенным процессорным ядром.

Семейство Cortex доступно в трех основных профилях:

– *Cortex-A* — для высокопроизводительных применений. Это полноценные процессоры общего назначения для самых различных задач. Процессор Apple A5, используемый в iPhone 4S и iPad 2, построен на основе ядра Cortex-A9.
– *Cortex-R* — профиль для операционных систем реального времени (ОСРВ, англ. Real-Time Operating System).
– *Cortex-M* — для чувствительных к стоимости и микроконтроллерных применений.

Для упрощение разработки под микроконтроллер используется *CMSIS (Cortex Microcontroller Software Interface Standard). CMSIS* — уровень абстракции аппаратного обеспечения для Cortex-M, обеспечивающий последовательный и простой интерфейс программного обеспечения для процессора и периферийных

устройств. *CMSIS* стандартизирует программное обеспечение, позволяя переносить его на другие устройства Cortex-M.

CMSIS состоит из нескольких файлов:

- *core_cm3.c, core_cm3.h*[1] — описание ядра, стандартизировано для всех Cortex-M3.
- *stm32l1xx.h* — файл описание периферии, а также структуры доступа к ним.
- *system_stm32l1xx.c* — функции CMSIS.
- *system_stm32l1xx.h* — заголовочные файлы для функций CMSIS.

CMSIS доступен на сайте производителя микроконтроллеров.

1.3 Создание проекта в IAR

IAR EWARM — интегрированная среда разработки, включающая в себя компилятор языка Си, отладчик (debugger) и компоновщик (linker) [**?**]. Создание программного обеспечения для микроконтроллера подразумевает создание проекта, который будет объединять CMSIS, сторонние библиотеки и исходные коды на языке Си. Для создание нового проекта выбрать:

Project => Create New Project

В появившемся окне можно выбрать язык программирования. В данной лабораторной это язык Си [**?**]. Подпункт *m*ain позволяет создать файл главной программы *main.c* вместе с каркасом функции *main()*.

```
int main()
{
  return 0;
}
```

Файлы внутри проекта можно объединять в группы (папки, подпапки). Для упрощения работы с периферией существуют библиотеки *Standard Peripherals Library (SPL)*. В данной работе потребуются библиотека для работы

[1]IAR, начиная с версии 6.2, использует собственные файлы core_cm3.c, core_cm3.h

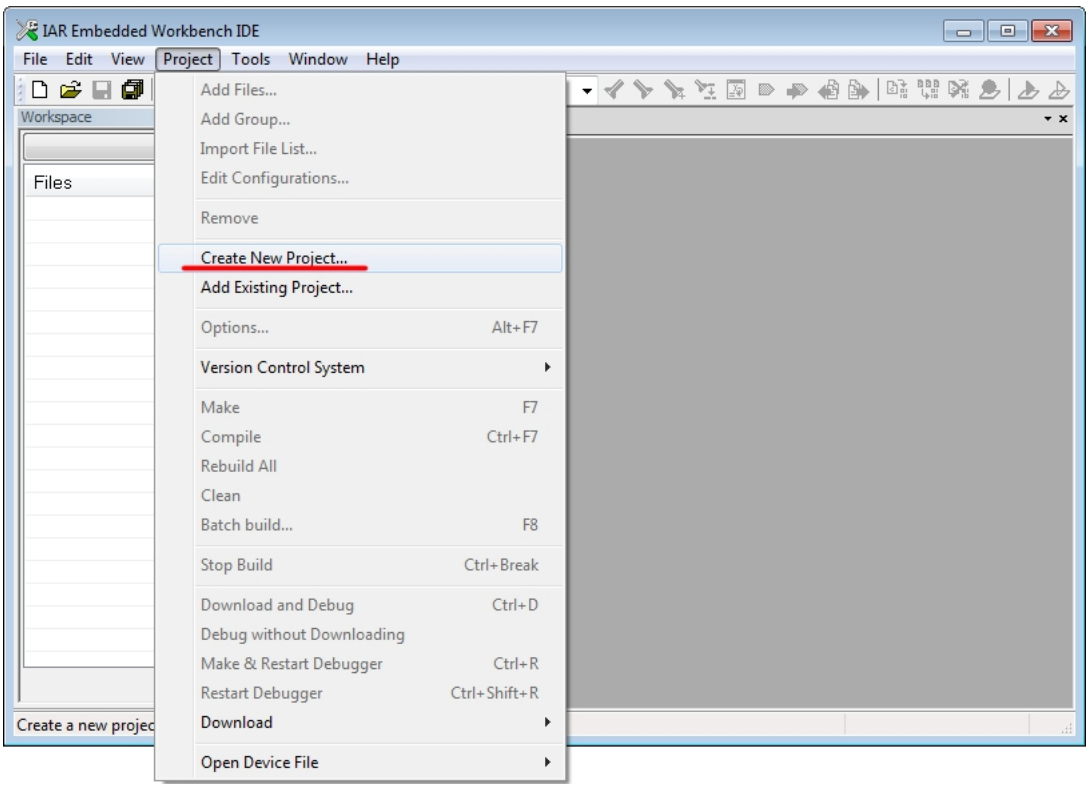

Рисунок 1.1 — Создание нового проекта

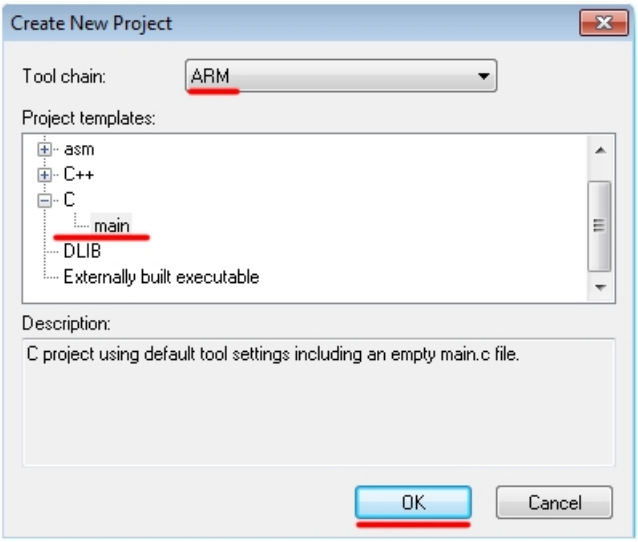

Рисунок 1.2 — Выбор языка программирования

с *GPIO (General Purpose Input-utput)* – портами ввода-вывода общего назначения и библиотека для работы с *RCC (Reset and Clock Control)* – системой тактирования и сброса. Создайте группу *SrdPereph*, в ней подпапки *inc* – для заголовочных файлов, *src* – для исходных файлов.

Добавьте в созданные папки файлы библиотек *stm32l1xx_gpio.h, stm32l1xx_gpio.c, stm32l1xx_rcc.c, stm32l1xx_rcc.h.*

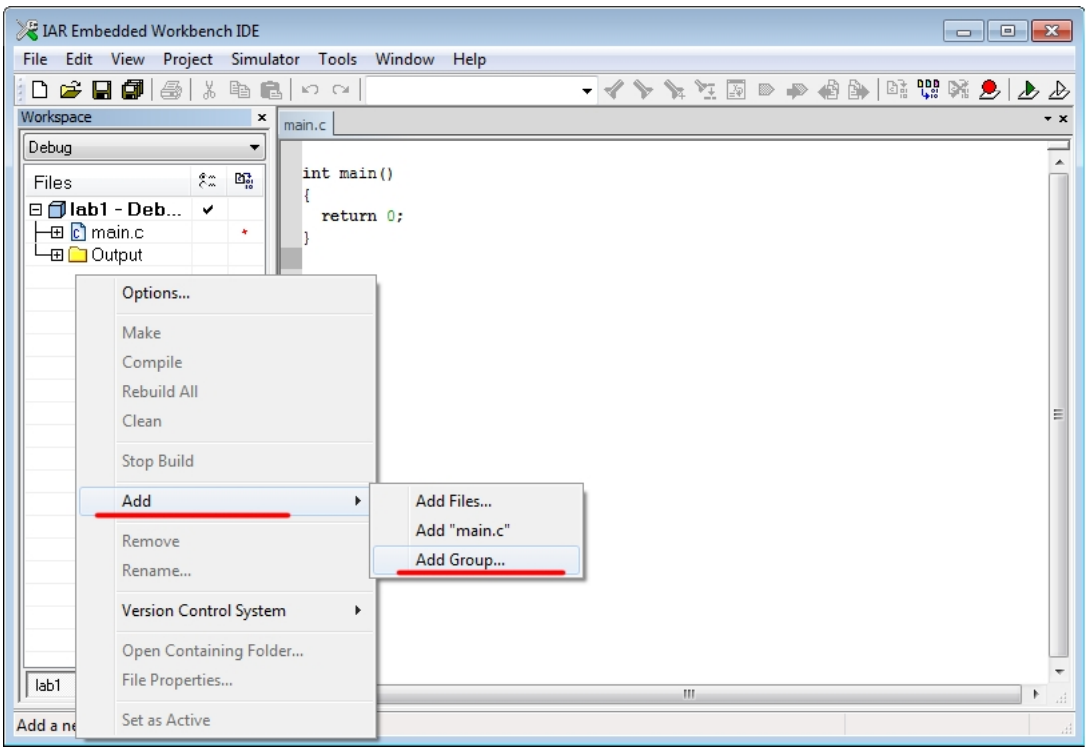

Рисунок 1.3 — Создание новой группы

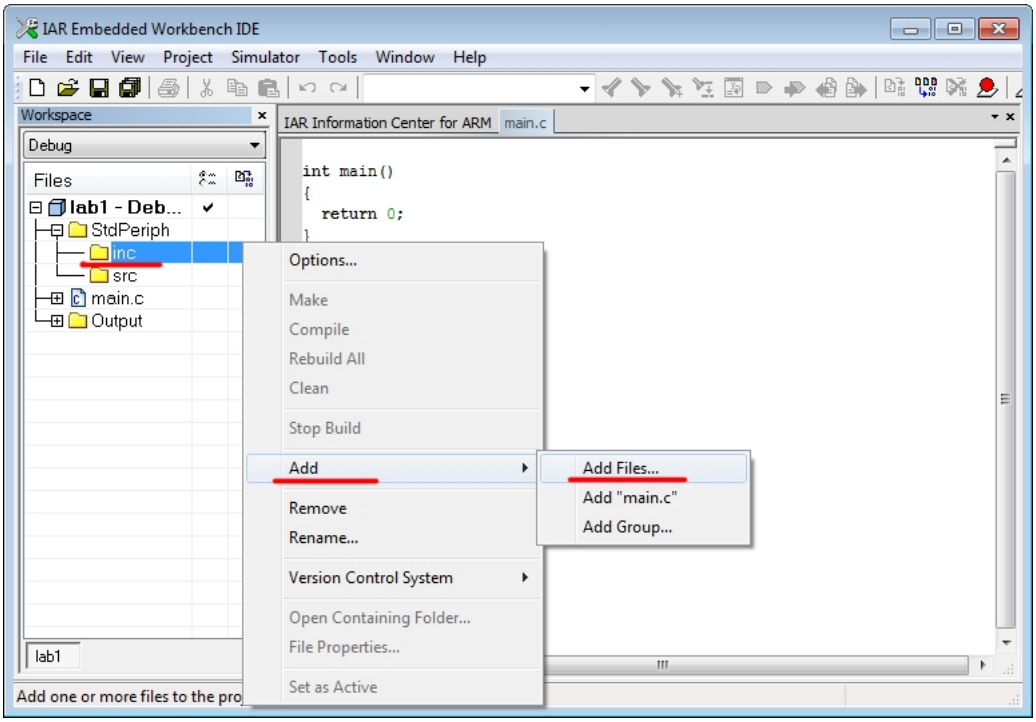

Рисунок 1.4 — Добавление файлов библиотек

Для дальнейшей работы необходимо настроить проект под заданный микроконтроллер.

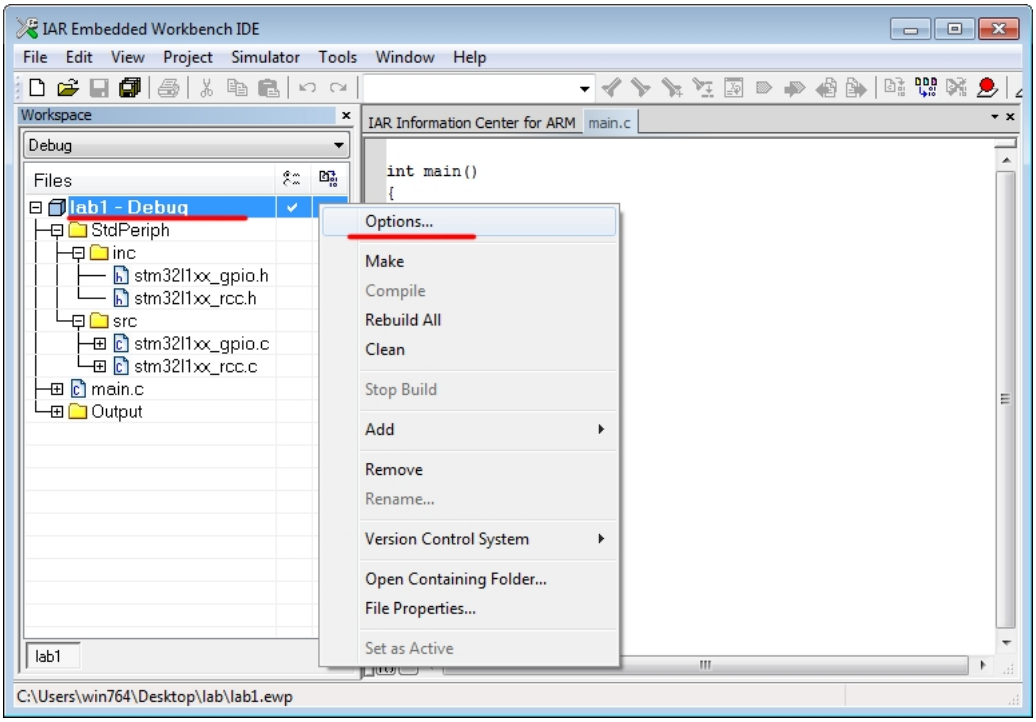

Рисунок 1.5 — Настройка проекта

В категории *General Options*, во вкладке *Target*, отметьте пункт *Device* и в выпадающем списке выберите микроконтроллер

$$ST => ST\ ST32L52xB$$

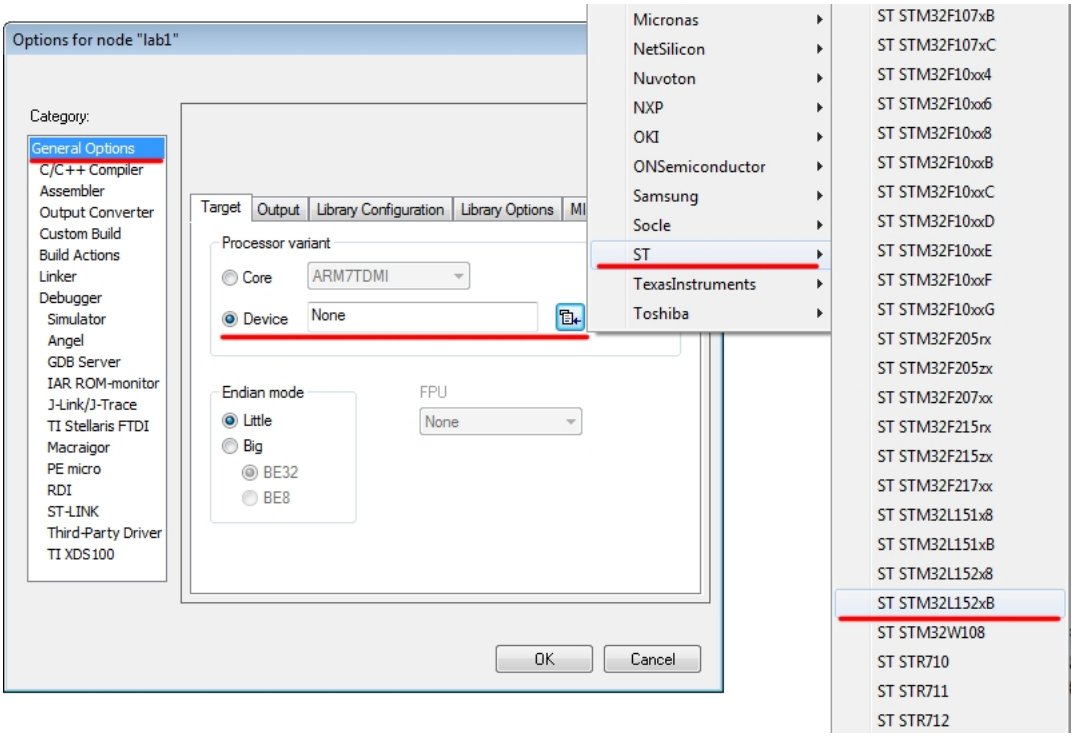

Рисунок 1.6 — Выбор микроконтроллера

В категории *General Options*, во вкладке *Library Configuration*, в выпадающем списке *Library* выберите *Full* – для полного использования библиотеки времени выполнения (runtime library). Отметьте пункт *Use CMSIS* для использования файлов описания ядра, разработанных IAR.

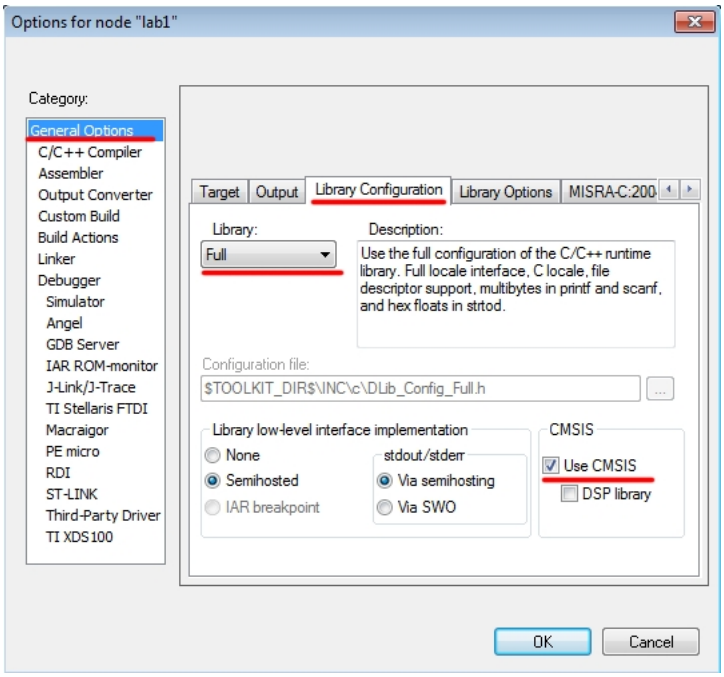

Рисунок 1.7 — Настройка библиотек

В категории *C/C++ Compiler*, во вкладке *Preprocessor* необходимо указать компилятору пути до заголовочных и исходных файлов. Для относительного описания пути можно использовать переменную `$PROJ_DIR$` – директория проекта [?].

```
$PROJ_DIR$\STM32L1xx_StdPeriph_Driver\inc
$PROJ_DIR$\STM32L1xx_StdPeriph_Driver\src
$PROJ_DIR$\CMSIS\CM3\DeviceSupport\ST\STM32L1xx
$PROJ_DIR$\
```

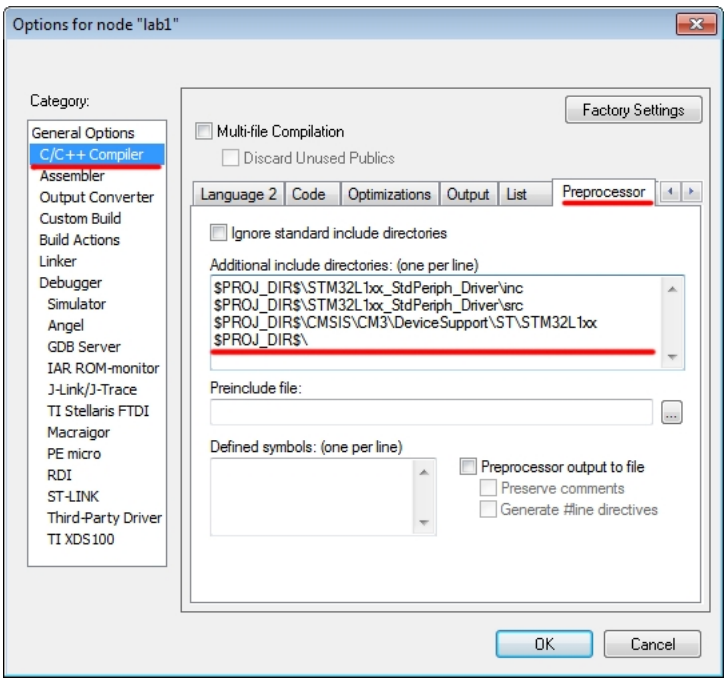

Рисунок 1.8 — казание дополнительных путей до файлов

По умолчанию IAR генерирует исполняемый файл в формате *ELF (Executable and Linkable Format)* – формат исполняемых и компонуемых файлов, используемый во многих UNIX-подобных операционных системах. В категории *Output Converter* отметьте пункт *Generate additional output*, а в выпадающем списке *Output format* выберете *binary*.

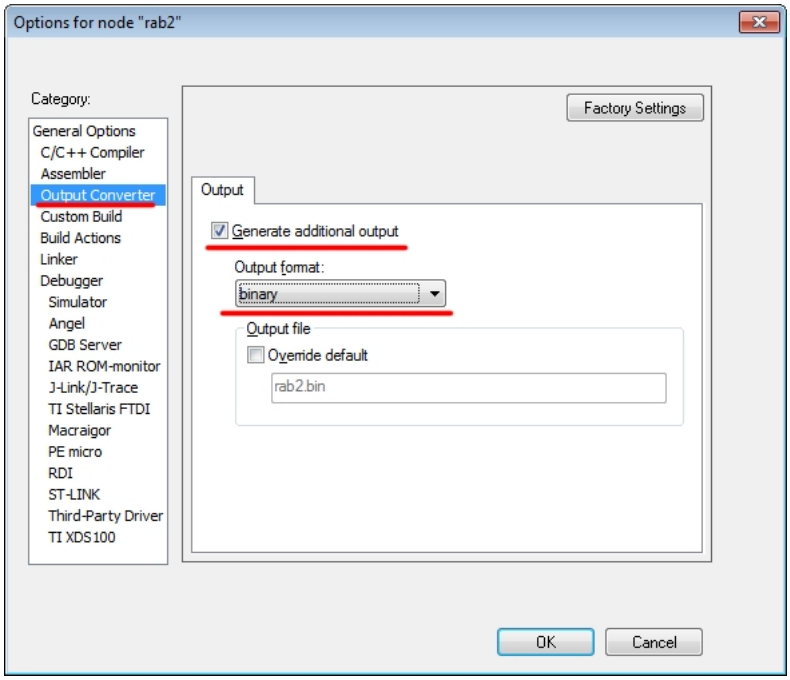

Рисунок 1.9 — Выбор формата исполняемого файла

В категории *Linker*, отметьте пункт *Override default* (отменить настройки по умолчанию), нажмите *Edit* для настройки конфигурации компоновщика.

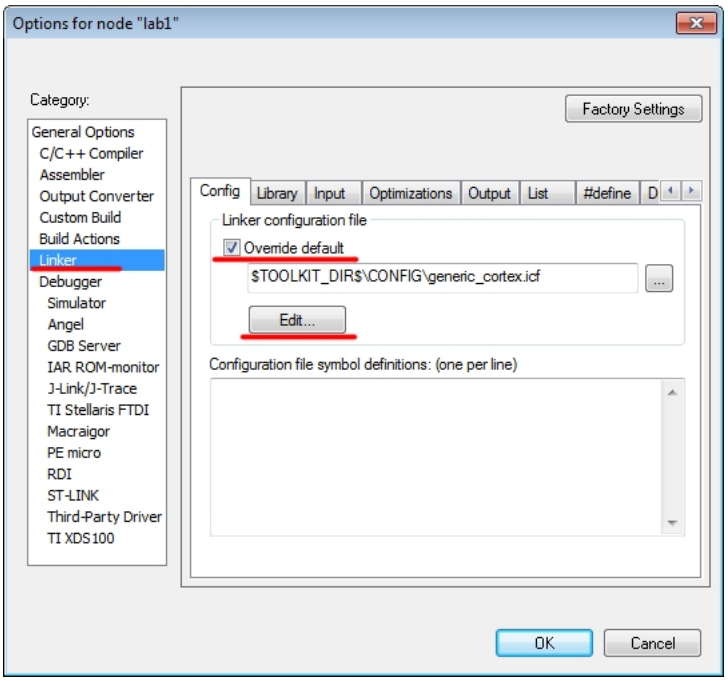

Рисунок 1.10 — Указание дополнительных путей до файлов

Во вкладке *Vector Table* необходимо указать начало таблицы векторов прерываний. Адрес может быть 0x00000000 или 0x08000000. Адрес 0x08000000 указывает на начало внутренней Flash памяти, так же этот адрес рекомендует использовать STMicroelectronics в своем руководстве UM1451 User manual [?].

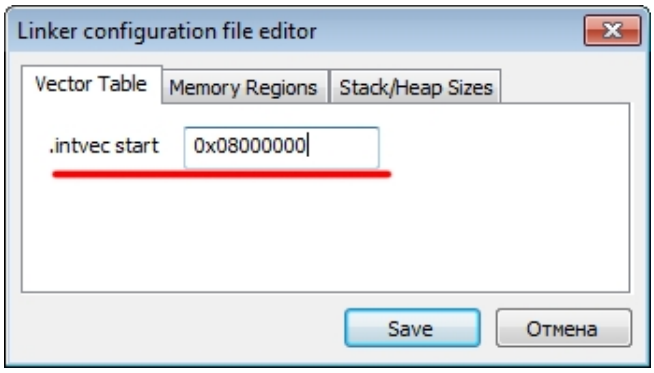

Рисунок 1.11 — Указание начала таблицы векторов прерываний

Во вкладке *Memory Regions* задаются адреса начала и окончания ROM (ПЗУ) и RAM (ОЗУ). ROM - внутренняя Flash память, начинается с адреса 0x08000000. Адрес окончания у каждого микроконтроллера разный и зависит от объема Flash памяти. Адрес окончания рассчитывается по формуле:

$$\text{Адрес}_{16} = \text{адрес начала}_{16} + \text{размер } Flash \text{ памяти}_{16} - 1_{16}$$

В микроконтроллере STM32L152RB 128 КБ Flash памяти, 16 КБ RAM.

$$0x08000000 + (128 \cdot 1024)_{10} - 1_{16} = 0x0801FFFF$$

RAM память начинается с адреса 0x20000000. Адрес окончания рассчитывается аналогично:

$$0x20000000 + (16 \cdot 1024)_{10} - 1_{16} = 0x20003FFF^{1}$$

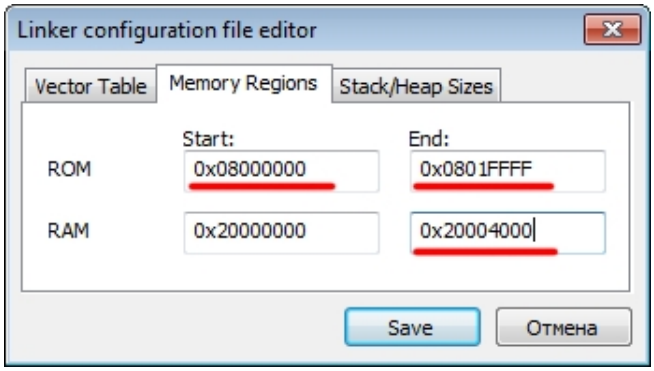

Рисунок 1.12 — Указание адреса начала и окончания ROM (ПЗУ) и RAM (ОЗУ)

В категории *Debugger*, во вкладке *Setup*, в выпадающем списке *Driver* выберете *ST-LINK*. Отметьте пункт *Run to* и укажите значение *main*, указывающее, что программа должна начинать работать с функции *main*.

Во вкладке *Download*, отметьте пункт *Use flash loader(s)*, позволяющий программировать Flash непосредственно из среды IAR.

Для настройки встроенного программатора-отладчика в категории *ST-LINK*, в качестве интерфейса, выберете пункт *SWD*.

Программатор — аппаратно-программное устройство, предназначенное для записи/считывания информации в постоянное запоминающее устройство.

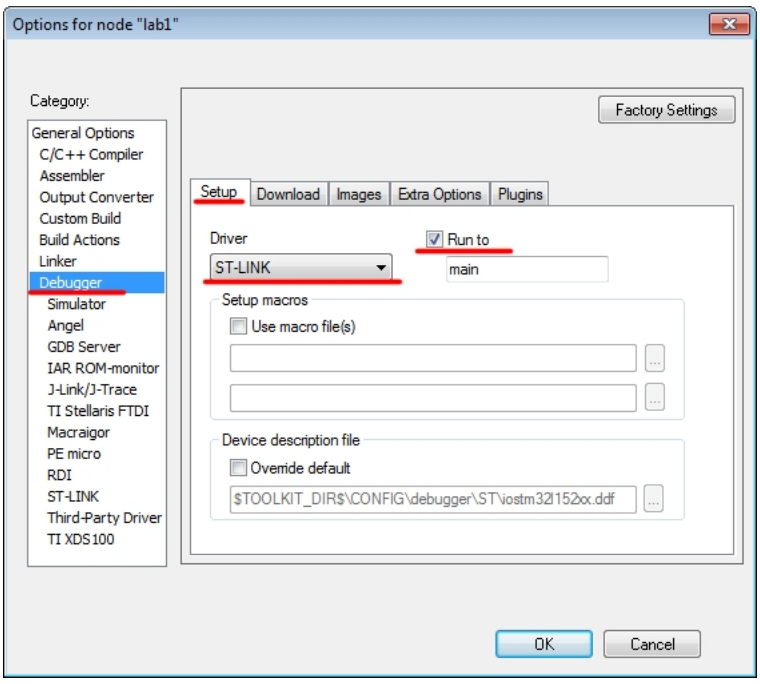

Рисунок 1.13 — Настройка отладчика. Часть 1

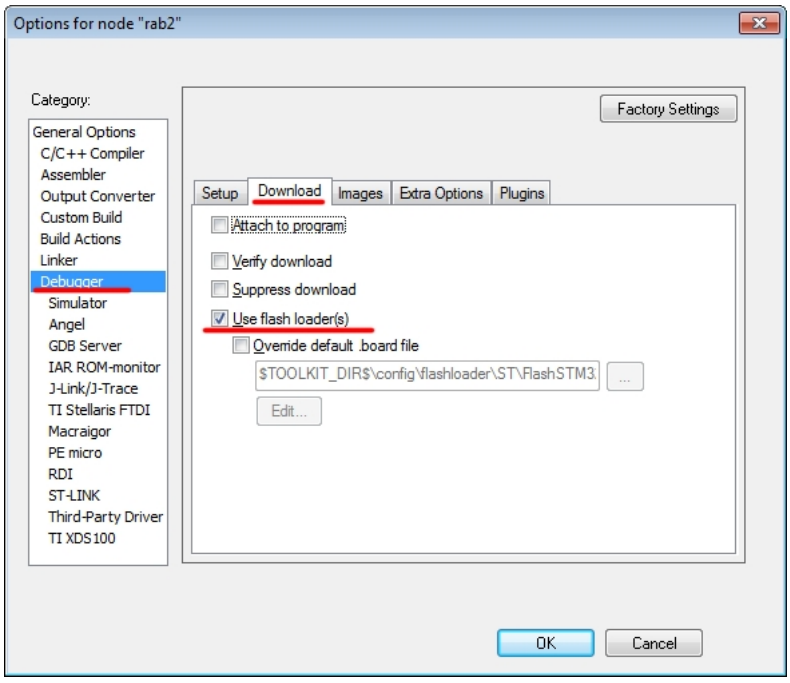

Рисунок 1.14 — Настройка отладчика. Часть 2

При попытке собрать файлы проекта будет появляться ошибка:

```
Warning[Pe223]: function "assert_param" declared implicitly
C:\Users\User\Desktop\lab\stm32l1xx_gpio.c 124
```

[1]STMicroelectronics, в своем руководстве UM1451 User manual, рекомендует использовать адрес 0x20004000, т.е. $0x20003FFF+1_{16}$

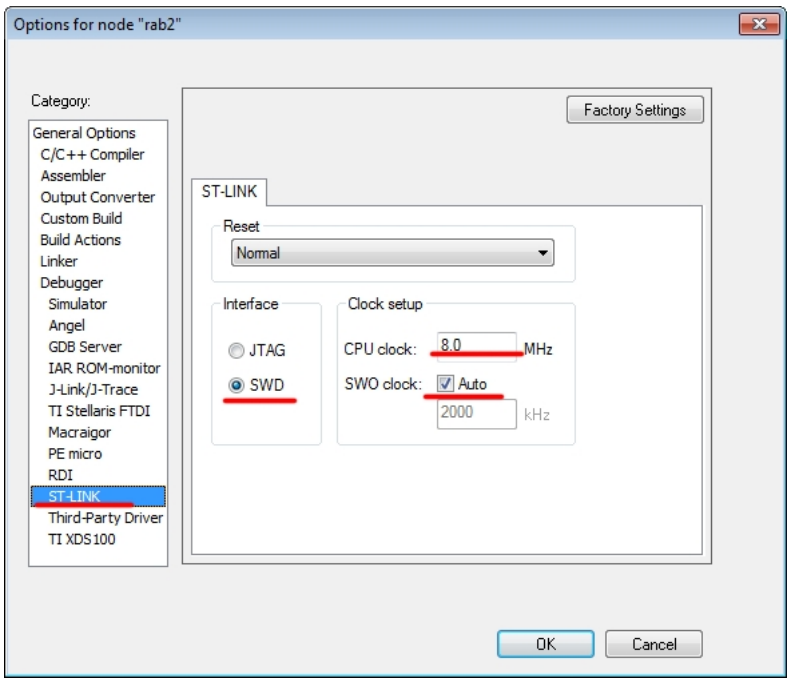

Рисунок 1.15 — Настройка программатора

Ошибка указывает на то, что функция *assert_param()*, которая используется в библиотеке проверки своих аргументов, объявлена неявно. Для устранения ошибки создайте файл *stm32l1xx_conf.h* содержащий:

```
#ifndef STM32L1XX_CONF_H_
#define STM32L1XX_CONF_H_
/* ------------------- */
#ifndef USE_FULL_ASSERT
 #define assert_param(x)
#endif
/* ------------------- */
#endif
```

Поместите файл в директорию проекта и добавьте его в проект. В исходные файлы подключаемых библиотек (*stm32l1xx_rcc.c, stm32l1xx_gpio.c*) добавьте строку:

```
#include <stm32l1xx_conf.h>
```

Для сборки файлов проекта нажмите *Make* (F7). Будут созданы исполняемые файлы, в окне сообщений отобразиться количество ошибок и предупреждений.

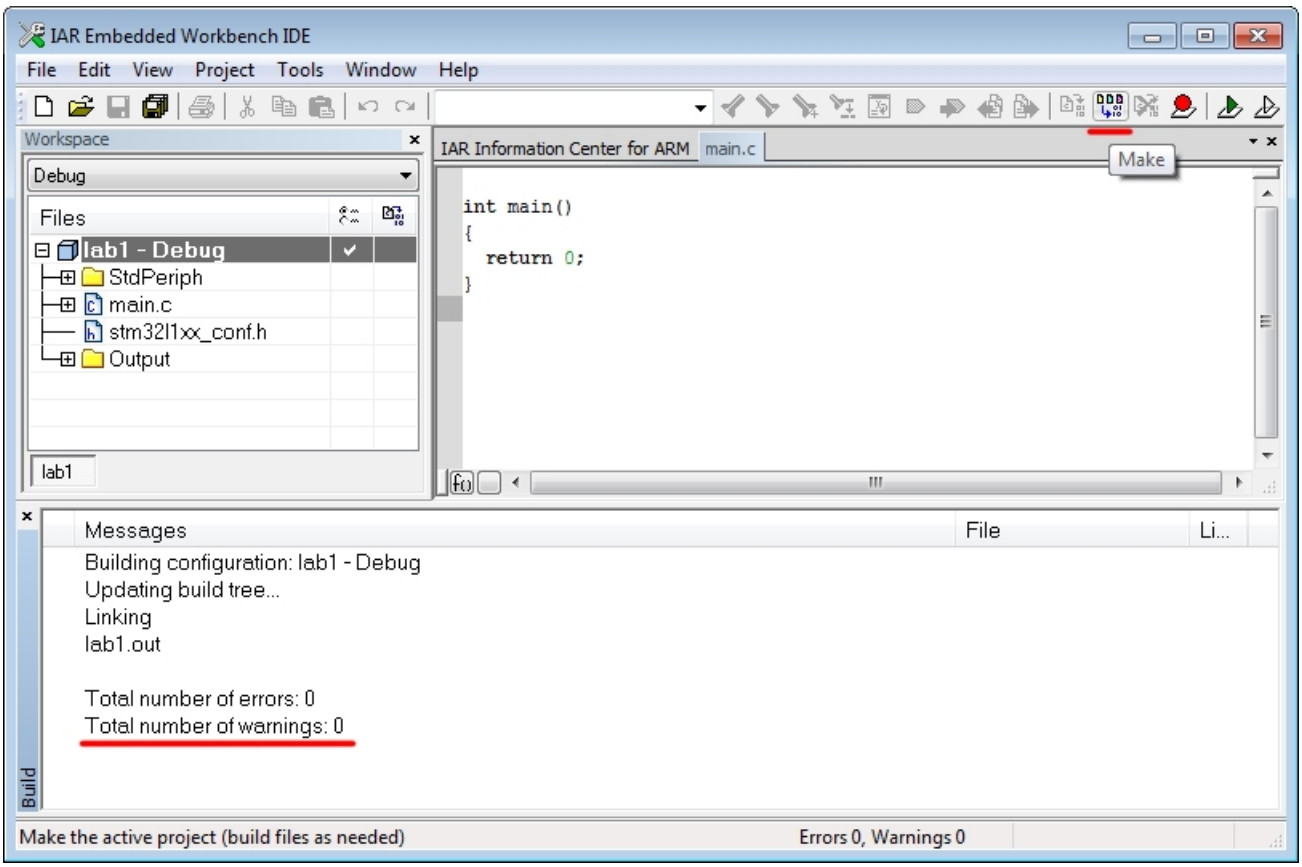

Рисунок 1.16 — Сборка файлов проекта

1.3.1 Работа с периферией. Функции и структуры

Для работы с периферией существуют два способа: Первый способ заключается в использовании регистров определенных в заголовочном файле *stm32l1xx.h* библиотеки CMSIS, расположенному в `CMSIS/DeviceSupport/ST/STM32L1xx`. Для записи в регистр используются битмаски или шестнадцатеричные коды. Программирование через регистры достаточно трудно, но позволяет добиться большей производительности. Программирование с использованием регистров будет описано во второй лабораторной работе

Второй способ заключается в использовании готовых библиотек, например, *Standard Peripherals Library (SPL)* или *Touch-Sensing Library (TSL)*. Библиотеки предоставляют готовые функции для работы с периферией, однако, скорость выполнения программы снижается, поскольку в конечном итоге все библиотечные функции управляют периферией через регистры.

В данной лабораторной работе управление периферией (светодиодами) будет происходить через библиотеку *Standard Peripherals*

Library (SPL), все необходимые для этого функции находятся в файлах *stm32l1xx_gpio.c, stm32l1xx_gpio.h, stm32l1xx_rcc.c, stm32l1xx_rcc.h* в папке `Лабораторные работы по микроконтроллерам /Materials/STM32L1xx_StdPeriph_Driver`.

Светодиод LD3 подключен к порту ввода-вывода PB7, светодиод LD4 подключен к порту ввода-вывода PB6. По умолчанию периферия STM32 не работает, пока на нее не подан тактирующий сигнал. Для подачи тактирующего сигнала используется функция, прототип которой объявлен в заголовочном файле *stm32l1xx_rcc.h*:

```
void RCC_AHBPeriphClockCmd(uint32_t RCC_AHBPeriph,
FunctionalState NewState);
```

Где *AHB* — шина, от которой происходит тактирование. Определить нужную шину можно по названию именованной константы в файле *stm32l1xx_rcc.h*:

```
...
#define RCC_AHBPeriph_GPIOA       RCC_AHBENR_GPIOAEN
#define RCC_AHBPeriph_GPIOB       RCC_AHBENR_GPIOBEN
#define RCC_AHBPeriph_GPIOC       RCC_AHBENR_GPIOCEN
#define RCC_AHBPeriph_GPIOD       RCC_AHBENR_GPIODEN
#define RCC_AHBPeriph_GPIOE       RCC_AHBENR_GPIOEEN
...
```

В качестве первого аргумента функции используется приведенная выше именованная константа *RCC_AHBPeriph_GPIOB* (В случае портов PB6, PB7). В качестве второго аргумента используется константа *ENABLE* или *DISABLE*. На это указывает комментарий в файле *stm32l1xx_rcc.c* перед объявлением функции *RCC_AHBPeriphClockCmd()*:

```
NewState: new state of the specified peripheral clock.
  *         This parameter can be: ENABLE or DISABLE.
```

Таким образом, функция для подачи тактирующего сигнала на порты GPIOB примет вид:

```
RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOB, ENABLE);
```

Для описания режимов работы порта необходимо объявить инициализированную структуру *GPIO_InitTypeDef*, которая содержит настройки порта в качестве элементов структуры. Структура описывается в заголовочном файле периферии, которую необходимо инициализировать. В данной работе используются GPIO, следовательно, информация о структуре находиться в файле *stm32l1xx_gpio.h*:

```
typedef struct
{
uint32_t GPIO_Pin;            /*!< Specifies the GPIO pins to be
                               configured. This parameter can be any
                               value of @ref GPIO_pins_define */

GPIOMode_TypeDef GPIO_Mode;   /*!< Specifies the operating mode for
                               the selected pins. This parameter can
                               be a value of @ref GPIOMode_TypeDef */

GPIOSpeed_TypeDef GPIO_Speed; /*!< Specifies the speed for the
                                selected pins. This parameter can be a
                                value of @ref GPIOSpeed_TypeDef */

GPIOOType_TypeDef GPIO_OType; /*!< Specifies the operating output
                                type for the selected pins. This
                                parameter can be a value of @ref
                                GPIOOType_TypeDef */

GPIOPuPd_TypeDef GPIO_PuPd;   /*!< Specifies the operating Pull-
                                up/Pull down for the selected pins.
                                This parameter can be a value of @ref
                                GPIOPuPd_TypeDef */
}GPIO_InitTypeDef;
```

Элементы структуры имеют тип схожий с названием элемента с добавлением суффикса _ *TypeDef*. Типы определены в качестве перечислений (enum) в файле *stm32l1xx_gpio.h*:

```
typedef enum
{
  GPIO_Speed_400KHz = 0x00, /*!< Very Low Speed */
  GPIO_Speed_2MHz   = 0x01, /*!< Low Speed */
  GPIO_Speed_10MHz  = 0x02, /*!< Medium Speed */
  GPIO_Speed_40MHz  = 0x03  /*!< High Speed */
}GPIOSpeed_TypeDef;
...
typedef enum
{
  GPIO_Mode_IN   = 0x00, /*!< GPIO Input Mode */
  GPIO_Mode_OUT  = 0x01, /*!< GPIO Output Mode */
  GPIO_Mode_AF   = 0x02, /*!< GPIO Alternate functionMode */
  GPIO_Mode_AN   = 0x03  /*!< GPIO Analog Mode */
}GPIOMode_TypeDef;
...
```

В данной работе структура будет иметь следующий вид:

```
GPIO_InitTypeDef Имя_структуры;
Имя_структуры.GPIO_Pin = GPIO_Pin_7 | GPIO_Pin_6 ;
Имя_структуры.GPIO_Mode = GPIO_Mode_OUT;
Имя_структуры.GPIO_Speed = GPIO_Speed_2MHz;
Имя_структуры.GPIO_OType = GPIO_OType_PP;
```

GPIO_ Mode_ OUT – указывает, что порт является выходом. В случае если не указать тип в структуре, то при подаче питания на отладочную плату порт будет находиться в произвольном состоянии. *GPIO_ OType_ PP* – указывает, что выход имеет два состояния (Push-Pull).

Для инициализации полученной структуры используется функция, прототип которой объявлен в заголовочном файле *stm32l1xx_ gpio.h*:

```
void GPIO_Init(GPIO_TypeDef* GPIOx, GPIO_InitTypeDef* GPIO_InitStruct);
```

В качестве аргументов функции передаются порт и указатель на сформированную структуру:

```
GPIO_Init(GPIOB, &Имя структуры);
```

Для подачи питания на светодиод необходимо использовать функцию, прототип которой объявлен в заголовочном файле *stm32l1xx_gpio.h*:

```
void GPIO_SetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
```

В качестве аргументов функции передаются порт и именованная константа, определенная в заголовочном файле *stm32l1xx_gpio.h*, отвечающая за необходимый пин:

```
...
#define GPIO_Pin_2 ((uint16_t)0x0004)/*!< Pin 2 selected */
#define GPIO_Pin_3 ((uint16_t)0x0008)/*!< Pin 3 selected */
#define GPIO_Pin_4 ((uint16_t)0x0010)/*!< Pin 4 selected */
#define GPIO_Pin_5 ((uint16_t)0x0020)/*!< Pin 5 selected */
#define GPIO_Pin_6 ((uint16_t)0x0040)/*!< Pin 6 selected */
#define GPIO_Pin_7 ((uint16_t)0x0080)/*!< Pin 7 selected */
#define GPIO_Pin_8 ((uint16_t)0x0100)/*!< Pin 8 selected */
#define GPIO_Pin_9 ((uint16_t)0x0200)/*!< Pin 9 selected */
...
```

Для того, что бы подать питания на светодиод LD3, подключенному к порту B, пину 7 (PB7), необходимо использовать функцию следующего вида:

```
PIO_SetBits(GPIOB, GPIO_Pin_7);
```

Что бы отключить питание используется функция с аналогичными свойствами:

```
void GPIO_ResetBits(GPIO_TypeDef* GPIOx, uint16_t GPIO_Pin);
GPIO_ResetBits(GPIOB, GPIO_Pin_7);
```

В качестве программной задержки между подачей питания и отключением питания используется цикл for без инструкций:

```
for (i=0;i<=100000;i++)
;
```

1.3.2 Загрузка программы в микроконтроллер и ее отладка

После создание программы, нужно создать исполняемые файлы, для этого в главном меню выберите пункт

Project => Make

или на панели инструментов *Make* (F7). При последующем изменении программы, инструмент *Make* скомпилирует файлы, которые подверглись изменению с момента первоначального создания исполняемых файлов. Функция *Compile* (Ctrl+F7) компилирует только выбранные файлы.

Рисунок 1.17 — Создание исполняемых файлов

Подключите отладочную плату STM32l-Discovery через разъем Mini USB к USB разъему компьютера. Для загрузки программы в микроконтроллер в главном меню выберите пункт

Project => Download and Debug

или на панели инструментов *Download and Debug* (Ctrl+D).

Рисунок 1.18 — Загрузка программы в микроконтроллер

После завершения загрузки программы в микроконтроллер, в IAR появиться панель отладки [?].

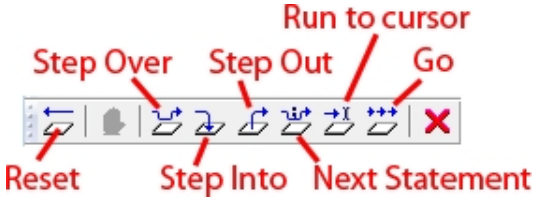

Рисунок 1.19 — Панель отладки

Step Over – выполняется следующая инструкция, оператор или функция, без входа в саму функцию.

Step Into – выполняется следующая инструкция, оператор или функция с входом в саму функцию.

Step Over – выполняется следующая инструкция, оператор или функция с выходом из функции.

Next Statement – выполняется следующая инструкция, оператор или функция без остановки вызовов функций.

Run to cursor – выполняется следующая инструкция, оператор или функция до выделенного фрагмента кода.

Reset – отладка выполняется с начала программы.

Go – выполняется следующая инструкция, оператор или функция до метки или окончания программы.

Для контроля значений переменных и функций используется инструмент *Live Watch*.

View => Live Watch

Live Watch позволяет отслеживать значение выбранных переменных во время выполнения программы. Для запуска загруженной программы в микроконтроллер нажмите на панели отладки на *Go* (F5). Все инструменты отладки дополнительно приведены в пункте *Debug* в главном меню.

1.4 Порядок выполнения лабораторной работы

1) Выбрать вариант задания в приложении 3 в соответствии с номером в журнале.
2) Создать в папке с номером группы на рабочем столе папку `Lab1` для файлов проекта.
3) копировать в созданную папку `Lab1`, папку `CMSIS` из папки `Лабораторные работы по микроконтроллерам/Materials` на рабочем столе.

4) Добавить в созданную папку с проектом папку `STM32L1xx_StdPeriph_Driver` из папки `Лабораторные работы по микроконтроллерам/Materials`.

5) Добавить в созданную папку с проектом файл *stm32l1xx_conf.h* из папки `Лабораторные работы по микроконтроллерам/Materials`, добавить его в проект.

6) Добавьте в файлы *stm32l1xx_gpio.c, stm32l1xx_rcc.c* строку:

```
#include <stm32l1xx_conf.h>
```

7) Создать и настроить проект в среде разработки IAR. В качестве имени проекта указать *lab1*, все файлы настроек проекта сохранить в папке `Lab1`.

8) Добавить в проект IAR файлы *stm32l1xx_gpio.c, stm32l1xx_gpio.h, stm32l1xx_rcc.c, stm32l1xx_rcc.h.*

9) Построить блок-схему алгоритма главной программы.

10) Написать программу для микроконтроллера на языке Си.

11) Подключить отладочную плату STM32L-Discovery к компьютеру.

12) Загрузить программу в микроконтроллер и произвести ее отладку.

13) Отчет.