№2 / 2011

Статья 4

Микроконтроллеры STM32 «с нуля»

Роман Попов (КОМПЭЛ)

Компания STMicroelectronics одной из первых вывела на рынок семейство микроконтроллеров на ядре ARM Cortex-M3 и на сегодняшний день по праву занимает лидирующее место среди производителей микроконтроллеров на этом ядре. Все началось в 2007 году с двух семейств – Performance Line (STM32F103) и Access Line (STM32F101). Компания постоянно работает как над расширением номенклатуры семейства, так и над улучшением характеристик, не забывая при этом также пополнять программную составляющую продукта. На сегодняшний момент STM32 уже состоит из 10 линеек (рис. 1) для всевозможных применений – микроконтроллеры с высокой производительностью, недорогие микроконтроллеры общего применения, микроконтроллеры с ультранизким энергопотреблением, микроконтроллеры со встроенным радиомодулем для беспроводных решений, и все это – на одном ядре ARM Cortex-M3! Нельзя не отметить pin-to-pin и программную совместимость по всем линейкам. Для более подробной информации по семейству STM32 следует посетить официальный сайт компании [1].

Рис. 1. Семейство STM32

Для начала работы и изучения любого микроконтроллера разработчику необходимы три инструмента –

программная среда разработки, программатор-отладчик и оценочная плата от производителя или от сторонних производителей.

Выбор программного инструментария для разработки

Под ARM-архитектуру существует довольно широкий выбор программных средств разработки. Приведем лишь основные и самые популярные программные пакеты на российском рынке (табл. 1)

Таблица 1. Среды разработки программного обеспечения

Инструментарий	Среда разработки	C/C++ компилятор	Ограничение Си-иструментария	Программатор-отладчик	Ссылка
IAR Systems	Embedded Workbench	IAR C/C++	32 Кбайт или полная с ограничением на 30 дней	J-Link ST-Link	*www.iar.com*
Keil	uVision (MDK-ARM)	Keil C/C++	32 Кбайт	ULink ST-Link	*www.keil.com*
Raisonance	Ride7 + RKIT-ARM	GCC C/C++	Нет, ограничения по отладке	RLink	*www.raisonance.com*
Atollic	TrueSrudio	GCC C/C++	Нет, ограничения по функционалу	ST-Link STICE	*www.atollic.com*
Open source	Eclipse	GCC C/C++	Без ограничений	ARM-Link	*www.eclipse.org [2]*

Наиболее популярными (но и самыми дорогими) среди разработчиков для разработки ПО под ARM архитектуру являются инструментарии от компаний Keil и IAR Systems. Это обусловлено наиболее продвинутыми С-инструментариями с точки зрения оптимизации и компактности кода. Также, помимо лидирующих позиций в С-инструментариях, данные компании предоставляют широкие наборы дополнительного ПО – операционные системы реального времени, USB-стеки, TCP/IP-стеки и многое другое, но за дополнительную плату. К тому же компания Keil принадлежит ARM, и при пользовании услугами этих двух компаний вы получаете очень хорошую техническую поддержку. Но мы все же остановимся на инструментарии от IAR Systems. Выбор обусловлен универсальностью инструментария, поддерживающего большинство известных нам архитектур микроконтроллеров таких производителей как STMicroelectronics, Texas Instruments, Microchip, Atmel и т.д.

Также следует отметить популярность средств на основе компилятора GCC. Существуют как платные их варианты, так и бесплатные. Помимо всего, GCC является лидером по количеству поддерживаемых процессоров и операционных систем. Как пример варианта платных средств в сводной таблице мы привели инструментарии от компаний Raisonance и Atollic. По сравнению с двумя ранее описанными вариантами вы получаете за гораздо меньшие деньги полноценный С-инструментарий со средой разработки и технической поддержкой. Также существует вариант полностью бесплатного инструментария, например, среда разработки Eclipse и компилятор GCC. Более подробно по бесплатному варианту вопросу следует обратиться к источнику [2].

Выбор оценочной платы для разработки

На рынке существует огромный выбор оценочных плат для STM32 как от STMicroelectronics, так и от сторонних производителей. Например, недорогие и оригинальные модули «Махаон» и «Барракуда» предлагает компания Терраэлектроника. Но наша основная цель – использовать для ознакомления и изучения микроконтроллеров семейства STM32 доступные и по возможности недорогие модули. Именно для таких целей компания STMicroelectronics разработала линейку оценочных плат «Discovery»: для восьмибитных микроконтроллеров – **STM8S-Discovery** и **STM8L-Discovery,** для 32-битных – **STM32VLDiscovery.** Особенность данных оценочных плат заключается в завершенном решении для старта разработки программного обеспечения на микроконтроллерах – сам микроконтроллер с необходимой обвязкой и внешними компонентами, а также интегрированный программатор-отладчик ST-Link. Это

полноценное решение, не требующее дополнительных затрат, а рыночная цена плат «Discovery» составляет 10…15$. Используя эти платы в собственных разработках, можно применять для программирования и отладки собственных приложений встроенный ST-Link через выведенный внешний разъем. С учетом всего вышеописанного, для широкого круга радиолюбителей и разработчиков коммерческих компаний отпадает необходимость в самостоятельном изготовлении отладочных плат и программаторов.

В журнале «Новости Электроники» №6 за 2010 год была опубликована статья «STM8 с нуля», в которой рассматривалась отладочная плата STM8S-Discovery, нас же интересуют микроконтроллеры семейства STM32. В связи с этим мы рассмотрим более подробно отладочную плату STM32VLDiscovery (рис. 2).

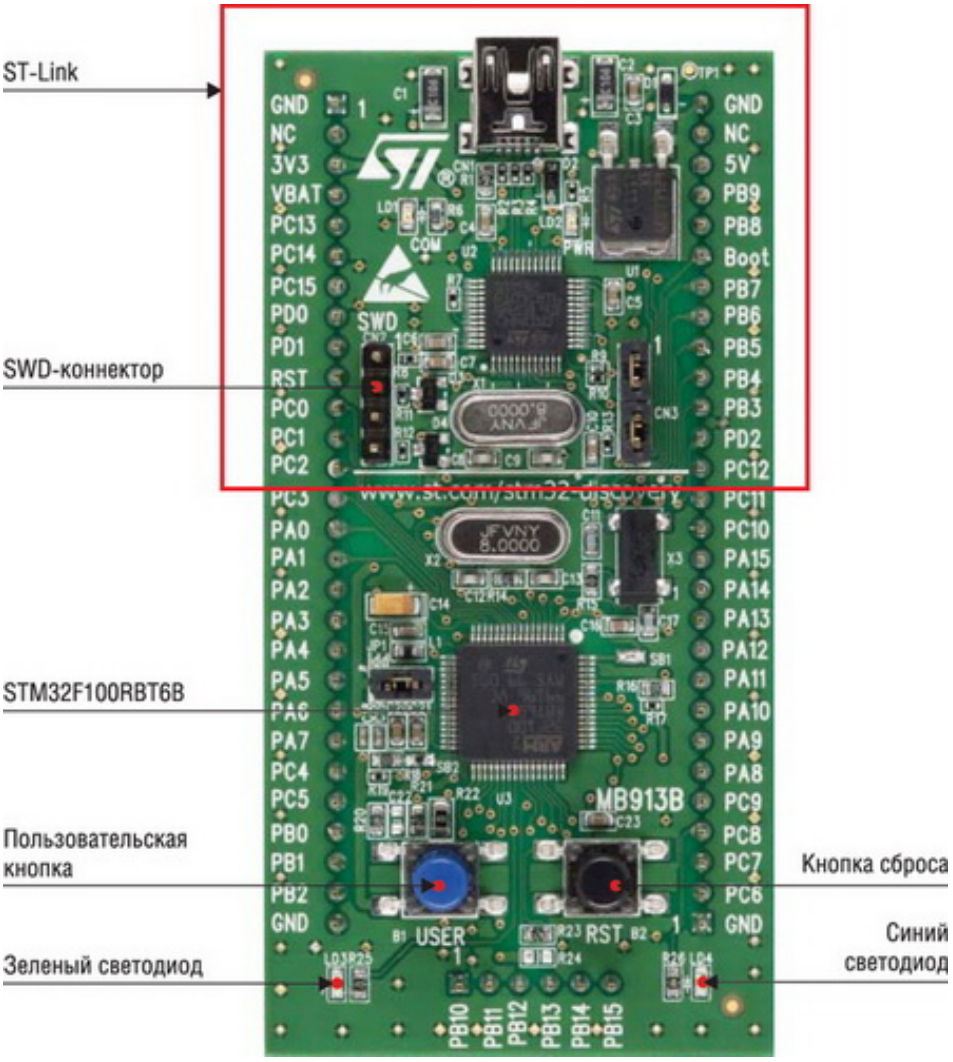

Рис. 2. Оценочный набор STM32VLDiscovery

В основе платы – микроконтроллер линейки «Value Line» STM32F100RBT6, программатор-отладчик ST-Link с выведенным разъемом SWD, механические кнопки, светодиоды и обвязка. Свободные ножки микроконтроллера выведены на внешние разъемы.

Выбор программатора-отладчика для разработки

В качестве программатора-отладчика производитель каждой среды разработки, как правило, предоставляет собственное решение, но поддерживаются устройства и других производителей. В таблице 1 можно увидеть среды разработки и поддерживаемые программаторы-отладчики. Большинство сред разработки поддерживают ST-Link – самый дешевый вариант на сегодняшний день. Нужно признать, что «родные» программаторы отладчика предоставляют большие возможности по отладке ПО, хотя разница в цене также ощутима. Добавим, что при выборе программаторов-отладчиков существует несколько вариантов от одного производителя: как более простые с поддержкой основных отладочных функций, так и профессиональные

версии с поддержкой полного спектра функций отладки и трассировки. Например, программаторы-отладчики для IAR Embedded Workbench – J-Link и J-Trace, для Keil uVision – ULink и ULink-Pro. В нашем же случае, как писалось выше, ST-Link интегрирован в отладочную плату, так что дальше мы будем пользоваться именно этим инструментом и отладочной платой STM32VLDiscovery.

Установка среды разработки

На данном этапе мы уже определились с программным и аппаратным обеспечением. Теперь переходим на официальный сайт IAR Systems [3], далее – на вкладку «Downloads». Должна появиться страница со списком всего доступного ПО. Нас интересует поле «Processor or core», строка «ARM», тут видно, что для загрузки доступны два варианта ПО. Первая версия – с ограничением по времени, но без ограничения по функциональности (30-day evaluation edition). Вторая версия – без ограничения по времени, но с ограничением 32Кб по загружаемому в микроконтроллер коду (Kickstart edition). Для загрузки необходимо заполнить форму на сайте, затем вам на почту должна прийти ссылка, и уже по этой ссылке можно скачать установочный файл ПО. Но при этом необходимо сохранить ключ активации и лицензионный номер, которые понадобятся при установке ПО. Загружаем второй вариант и устанавливаем. Процесс установки ПО ничем не отличается от стандартных программ, поэтому подробно на этом пункте мы заострять внимание не будем.

Стандартная библиотека периферии STM32

STMicroelectronics для облегчения труда разработчиков предоставляет бесплатные стандартные библиотеки периферии для своих микроконтроллеров и, в частности, для семейства STM32. Вначале мы рассмотрим библиотеку, разберемся, как с ней работать, и далее на основе этой библиотеки обсудим небольшой пример на отладочной плате STM32VLDiscovery. Структура библиотеки представлена на рисунке 3.

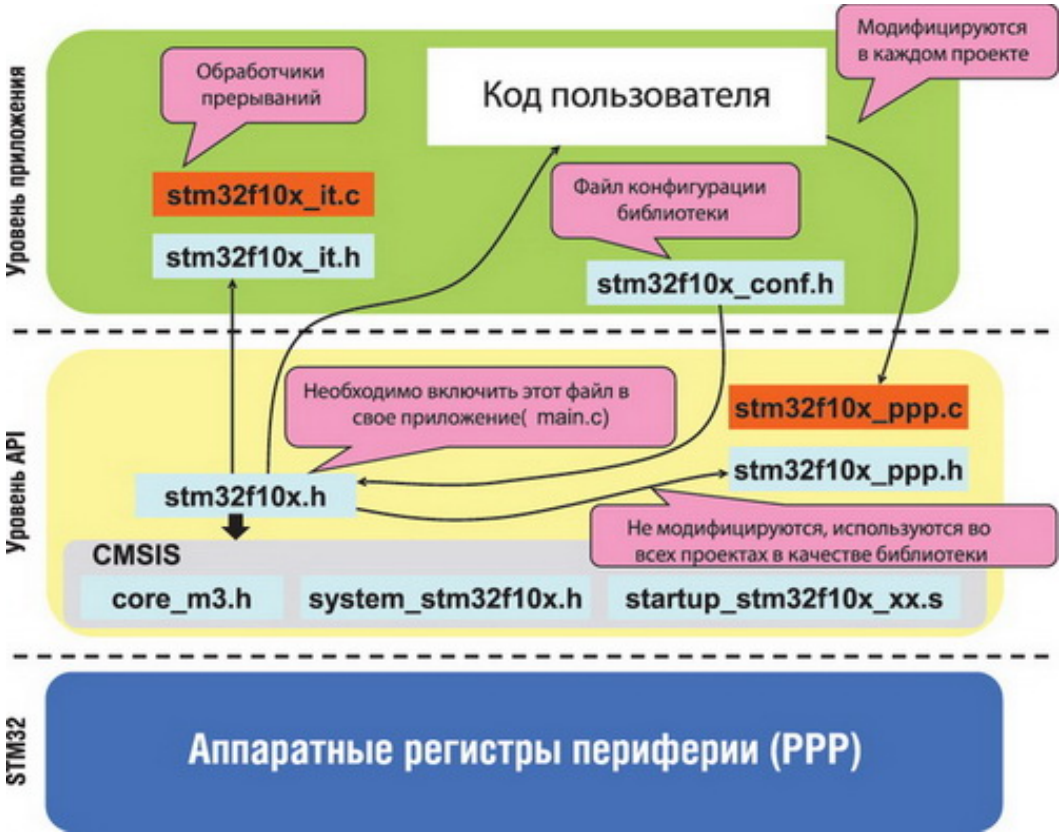

Рис. 3. Функциональная структура стандартной библиотеки периферии

Стандартная библиотека периферии написана в соответствии со стандартом ANSI C и может использоваться с любым компилятором. Структура библиотеки не так сложна, как кажется на первый взгляд, и состоит из двух взаимодополняющих составляющих.

Первая составляющая – заголовочные файлы и файлы реализации всей периферии микроконтроллеров STM32 – STM32F10x_StdPeriph_Driver. Вся функциональность периферийных модулей описана в заголовочных файлах и файлах реализации. Например, для портов ввода-вывода это два файла – stm32f10x_gpio.h и stm32f10x_gpio.c.

Вторая составляющая – заголовочные файлы и файлы реализации самого ядра ARM Cortex-M3 от компании ARM – CMSIS (ARM® Cortex™ Microcontroller Software Interface Standard). Ядро ARM Cortex-M3 выходит за рамки обычного понятия ядра микроконтроллера и представляет собой мини-микроконтроллер с периферией – встроенные системный таймер, контроллер прерываний и т.д. CMSIS предоставляет собой константы и определения, функции доступа к регистрам и периферийным модулям ядра, независимый интерфейс для операционных систем реального времени (RTOS). CMSIS состоит из трех файлов:

- core_m3.h – вспомогательные функции доступа к регистрам ядра;
- startup_stm32f10x_xx.s- набор файлов для каждой линейки семейства STM32, обеспечивающие инициализацию стека и таблицу векторов прерываний;
- system_stm32f10x.h- файл начальной инициализации тактовой частоты микроконтроллера.

Для использования стандартной библиотеки периферии необходимо в файл основной программы (обычно это main.c) включить файл #include “stm32f10x.h” и прописать определенные константы в свойствах проекта. Настройка проекта более подробно будет рассмотрена в приведенном ниже примере. Также библиотекой предоставляются три файла, доступные для модификации пользователем – файл конфигурации библиотеки stm32f10x_conf.h и файлы прерываний stm32f10x_it.h и stm32f10x_it.c. Для использования определенных модулей периферии в проект необходимо добавить файлы реализации и сконфигурировать файл stm32f10x_conf.h. Под конфигурацией файла stm32f10x_conf.h подразумевается раскомментирование строчек с названием периферийного модуля, предполагаемого для использования. В нашем случае это строчка #include “stm32f10x_gpio.h”. Обработка прерываний происходит в заголовочном файле и файле реализации stm32f10x_it.h и stm32f10x_it.c. Функции обработчиков прерывания не должны содержать параметров – void function(void). Если посмотреть файл stm32f10x_it.h, то можно увидеть, что в нем уже написаны пустые обработчики прерывания, но тут имеются не все функции. Дополнительные имена функций обработчиков прерывания – это фактически адреса обработчиков прерывания. Их придется прописывать вручную, и эти имена уже объявлены в файле начальной инициализации. Этот файл мы рассмотрим далее более подробно.

Вся периферия описана в структурах данных языка Си, которые и используются для конфигурации периферийного модуля. Функции и константы для периферийных модулей начинаются с префиксов, совпадающих с именем периферийного модуля. Например, имена функций для портов ввода-вывода – GPIO_Init(), GPIO_SetBits(), GPIO_ReadInputData() и константы – GPIOA, GPIO_Speed_50MHz, GPIO_Pin_0.

Для конфигурации периферийного модуля необходимо заполнить все поля структуры и далее передать структуру функции инициализации периферийного модуля. Например, для инициализации портов ввода-вывода необходимо объявить и заполнить поля структуры GPIO_InitTypeDef и передать объявленное имя функции инициализации GPIO_Init(… , …). Все доступные функции для работы с периферийным модулем можно посмотреть в справке на библиотеку или в заголовочном файле. Для портов ввода-вывода это файл stm32f10x_gpio.h. Большинство периферийных модулей имеют одинаковый набор функций, например (PPP – имя периферийного модуля):

- PPP_DeInit(…) – установка всех регистров в начальное(после сброса) состояние;
- PPP_Init(…) – установка параметров через структуры данных;
- PPP_Cmd(ENABLE/DISABLE) – разрешение/запрещение работы (не тактирование!);
- PPP_ITConfig(…) – конфигурация источников прерываний;
- PPP_GetFlagStatus(…) – чтение флагов периферийного модуля;
- PPP_ClearFlag(…) – очищение флагов периферийного модуля;
- PPP_ClearITPendingFlag(…) – сброс флага прерывания.

Для большей наглядности кода и сокращения его визуального размера стандартные типы данных предопределены в файле stm32f10x_type.h, например:

- u8 – unsigned char;
- u16 – unsigned short;
- RESET/SET;
- FALSE/TRUE;
- DISABLE/ENABLE.

Стандартную библиотеку периферии можно загрузить с официального сайта STMicroelectronics по ссылке [4]. Структура пакета библиотеки следующая:

- Libraries:
  - CMSIS – библиотека ядра ARM Cortex-M3;
  - STM32F10x_StdPeriph_Lib_V3.4.0LibrariesSTM32F10x_StdPeriph_Driver – библиотека периферии STM32.
- Project:
  - STM32F10x_StdPeriph_Lib_V3.4.0ProjectSTM32F10x_StdPeriph_Examples – исходные файлы примеров периферии STM32;
  - STM32F10x_StdPeriph_Lib_V3.4.0ProjectSTM32F10x_StdPeriph_Template – шаблон «пустого» проекта для оценочных плат STM3210xx-EVAL.
- Utilities – драйвера для отладочных плат STMicroelectronics.
- STM32F10x_StdPeriph_Lib_V3.4.0stm32f10x_stdperiph_lib_um.chm – файл справки.

Для более простого старта создания и конфигурации проекта на основе отладочных плат STM3210xx-EVAL предлагается «пустой» шаблон проекта для разнообразных сред разработки. В него можно внести ваш код, выбрать в конфигурации конкретную отладочную пату и начать работать. Шаблон проекта приведен для пяти сред разработки, можно начать работать с любой из них. Помимо этого, на первых порах можно посмотреть параметры конфигурации проектов.

Для прояснения картины пользования библиотекой перейдем к практической части статьи.

Практическая часть: пример проекта

На данном этапе мы загрузили и установили среду разработки IAR Embedded Workbench и получили общие знания по стандартной библиотеке периферии. Самое время приступить к практической части нашего материала.

Для всех своих отладочных плат STMicroelectronics предоставляет на официальном сайте примеры работы и описание. Полный список документации для STM32VLDiscovery можно посмотреть по ссылке [5], а кликнув по вкладке «Design Support», можно посмотреть всю доступную информацию по отладочной плате. С этой же страницы загружаем пакет «STM32VLDISCOVERY firmware package (AN3268)» по ссылке [3]. В состав пакета входит стандартная библиотека периферийных устройств и примеры проектов для отладочной платы. Рассмотрим структуру пакета более подробно:

- Libraries:
  - CMSIS – библиотека ядра ARM Cortex-M3;
  - STM32F10x_StdPeriph_Lib_V3.4.0LibrariesSTM32F10x_StdPeriph_Driver – библиотека периферии STM32.
- Project:
  - Demo – общий пример для STM32VLDiscovery;
  - Examples – исходные файлы примеров работы с периферией;
  - Master Workspace – проект на примерах из Examples.
- Utilities – драйвер для оценочной платы STM32VLDiscovery.

Заходим в директорию *ProjectMaster WorkspaceEWARMv5* и запускаем проект *Value_Line_Discovery.eww.* В

рабочей области проектов (рис. 4) можно увидеть 13 различных проектов для работы с портами ввода-вывода, внешними прерываниями, контроллером DMA, таймерами, режимами низкого энергопотребления и т.д. Для работы с каждым конкретным проектом его необходимо сделать активным – наводим на имя проекта курсор мыши, кликаем правую кнопку мыши и в выпадающем меню выбираем поле *Set as Active.* Далее можно вносить свои изменения, компилировать, загружать прошивку и отлаживать ПО.

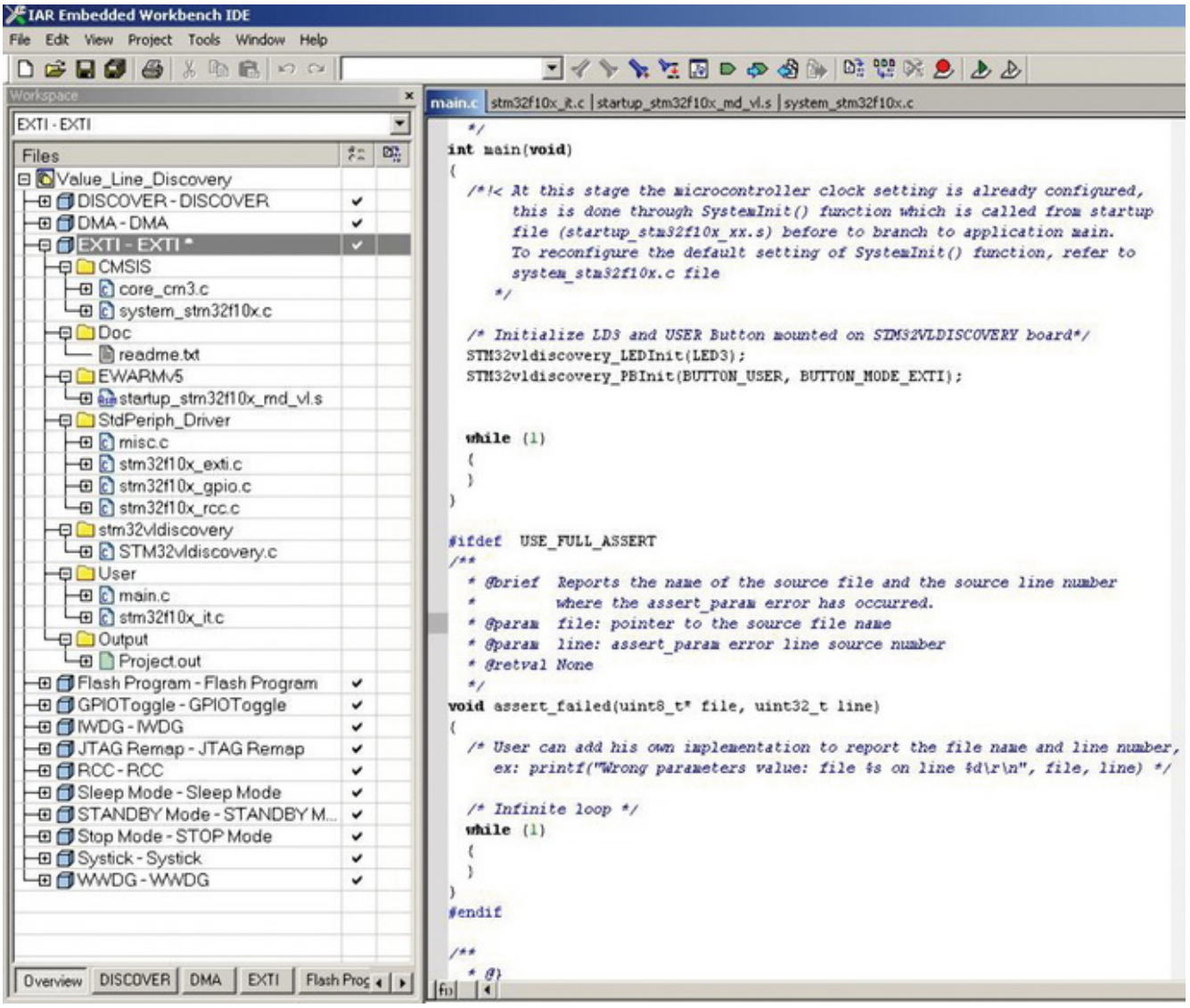

Рис. 4. Проект EXTI – EXTI

В качестве примера далее будем рассматривать проект **EXTI-EXTI.** В данном проекте происходит работа с портами ввода-вывода и внешними прерываниями. Для первого знакомства это вполне подходящий вариант. Делаем проект активным, как описывалось выше.

Как видно из рисунка, проект состоит из нескольких папок, разделяющих файлы проекта по смысловому значению:

- CMSIS – библиотека ядра ARM Cortex-M3,
- Doc – справка,
- EWARMv5 – начальная инициализация микроконтроллера,
- StdPeriph_Driver – библиотека периферии STM32,
- stm32vldiscovery – драйвер отладочной платы STM32VLDiscovery,
- User – пользовательские файлы,
- Output – файл прошивки.

Как уже описывалось выше, папка *CMSIS* содержит файл библиотеки ядра ARM Cortex-M3 и файл начальной инициализации. Папка *EWARMv5* содержит начальный файл инициализации, он выбирается для каждого объема памяти микроконтроллера. Все доступные файлы для всех микроконтроллеров STM32 можно, в зависимости от микроконтроллера, используемого в проекте, выбрать по следующему пути: *X:..stm32vldiscovery_packageLibrariesCMSISCM3DeviceSupportSTSTM32F10xstartuparm.* Необходимый файл выбирается по окончанию файла. Опишем все файлы:

- startup_stm32f10x_cl.s – линейка «Connectivity Line»;
- startup_stm32f10x_hd.s – линейки high-density «Access Line» и «Performance Line»;
- startup_stm32f10x_hd_vl.s – линейка high-density «Value Line»;
- startup_stm32f10x_ld.s – линейки low-density «Access Line» и «Performance Line»;
- startup_stm32f10x_ld_vl.s – линейка low-density «Value Line»;
- startup_stm32f10x_md.s – линейки medium-density «Access Line» и «Performance Line»;
- startup_stm32f10x_md_vl.s – линейка medium-density «Value Line»;
- startup_stm32f10x_xl.s – линейки xl-density «Access Line» и «Performance Line».

Узнать, какой микроконтроллер соответствует той или иной линейке и префиксу, можно из документации.

В папке *StdPeriph_Driver* содержатся исполняемые файлы периферийных устройств. В данную папку необходимо добавлять именно файлы, соответствующие используемой периферии в проекте. В нашем случае мы видим (рис. 4), что в папку добавлены четыре файла – файл приоритетов прерывания, внешних прерываний, портов ввода-вывода и системы тактирования. Соответственно, в проекте используются вышеперечисленные периферийные модули.

В папке *User* размещены пользовательские файлы – основной файл программы и файл обработчиков прерываний. Ну а в папке *Output* содержится выходной файл прошивки.

Для работы со стандартной библиотекой периферии необходимо настроить проект. У нас он уже настроен, но рассмотрим все необходимые настройки для создания будущего собственного проекта. Кликаем правой кнопкой мыши на имени проекта, в выпадающем поле выбираем вкладку *options,* далее в поле *Category* выбираем пункт *General Options* и, далее, в правом поле – вкладку *Target* (рис. 5). В данном меню выбирается тип микроконтроллера, в нашем случае это *STM32F100RBT6.*

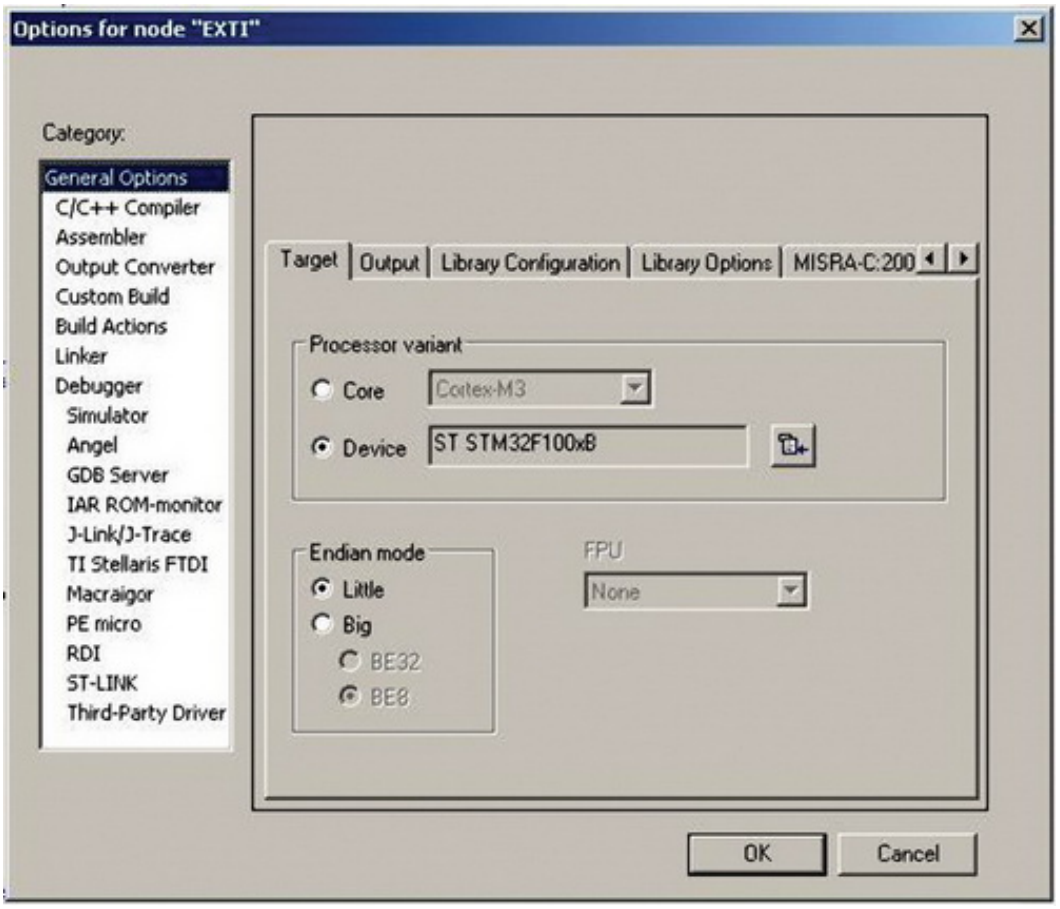

Рис. 5. Настройка проекта – выбор типа микроконтроллера

После выбора микроконтроллера необходимо прописать все пути к стандартной библиотеке периферии. В поле *Category* выбираем пункт *C/C++ Compiler* и далее в правом поле – вкладку *Preprocessor* (рис. 6)

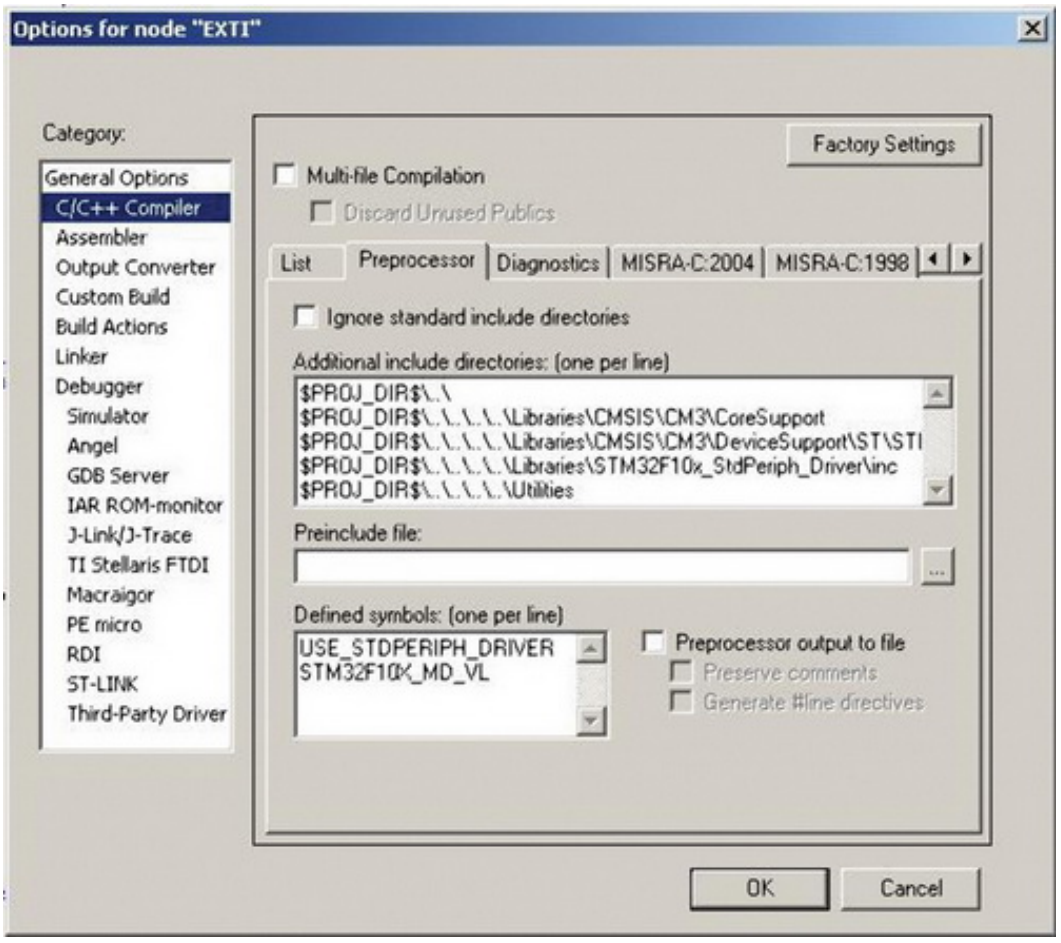

Рис. 6. Настройка проекта – настройка путей доступа к библиотеке

Над содержанием поля *Additional include directories* необходимо остановиться более подробно:

1. $PROJ_DIR$..
2. $PROJ_DIR$……..LibrariesCMSISCM3CoreSupport
3. $PROJ_DIR$……..LibrariesCMSISCM3DeviceSupportSTSTM32F10x
4. $PROJ_DIR$……..LibrariesSTM32F10x_StdPeriph_Driverinc
5. $PROJ_DIR$……..Utilities

Тег $PROJ_DIR$ является отправной точкой и олицетворяет собой путь до имени файла проекта. В нашем примере это путь *X:…an3268stm32vldiscovery_packageProjectMaster WorkspaceEWARMv5,* где X:… означает метку жесткого диска в системе и путь до пакета. Тег .. подразумевает переход на один уровень вверх по файловой директории. Таким образом, во всех строчках вышеприведенного поля, с помощью тегов $PROJ_DIR$ и .. задаются пути ко всем директориям файлов проекта. Например, во второй и третьей строке поля задаются пути к файлам библиотеки CMSIS, в четвертой – к стандартной библиотеке периферии, в пятой – к файлу драйвера отладочной платы STM32VLDiscovery.

Также необходимо обратить внимание на поле Defined symbols:(one per line). В данном поле содержатся два макроопределения. Макроопределение USE_STDPERIPH_DRIVER необходимо указывать, если вы собираетесь пользоваться стандартной библиотекой периферии. Макроопределение STM32F10X_MD_VL указывает библиотеке, какой микроконтроллер используется в проекте. Он должен быть аналогичен файлу, добавленному в папку *EWARMv5,* но без префикса startup_. В данное поле можно добавлять пользовательские макроопределения, используемые в проекте.

Следующий шаг – необходимо указать линковщику файл с расширением .icf. В поле *Category* выбираем пункт *Linker* и далее в правом поле – вкладку *Config* (рис. 7). Файлы этого типа используются для

управления линкером, указывают области памяти, определяют размер памяти и так далее. Файлы с расширением .icf для различных вариаций настройки проекта, например, для размещения кода только в SRAM, деления памяти на разные области и так далее, можно найти в шаблонах проектов от STMicroelectronics или написать самим. Более подробно про самостоятельное создание этих файлов можно прочитать в разделе «Помощь» среды разработки IAR Embedded Workbench, вкладка «Help – C/C++ Development Guide» на основной форме программы. В нашем примере данный файл находится в корне проекта.

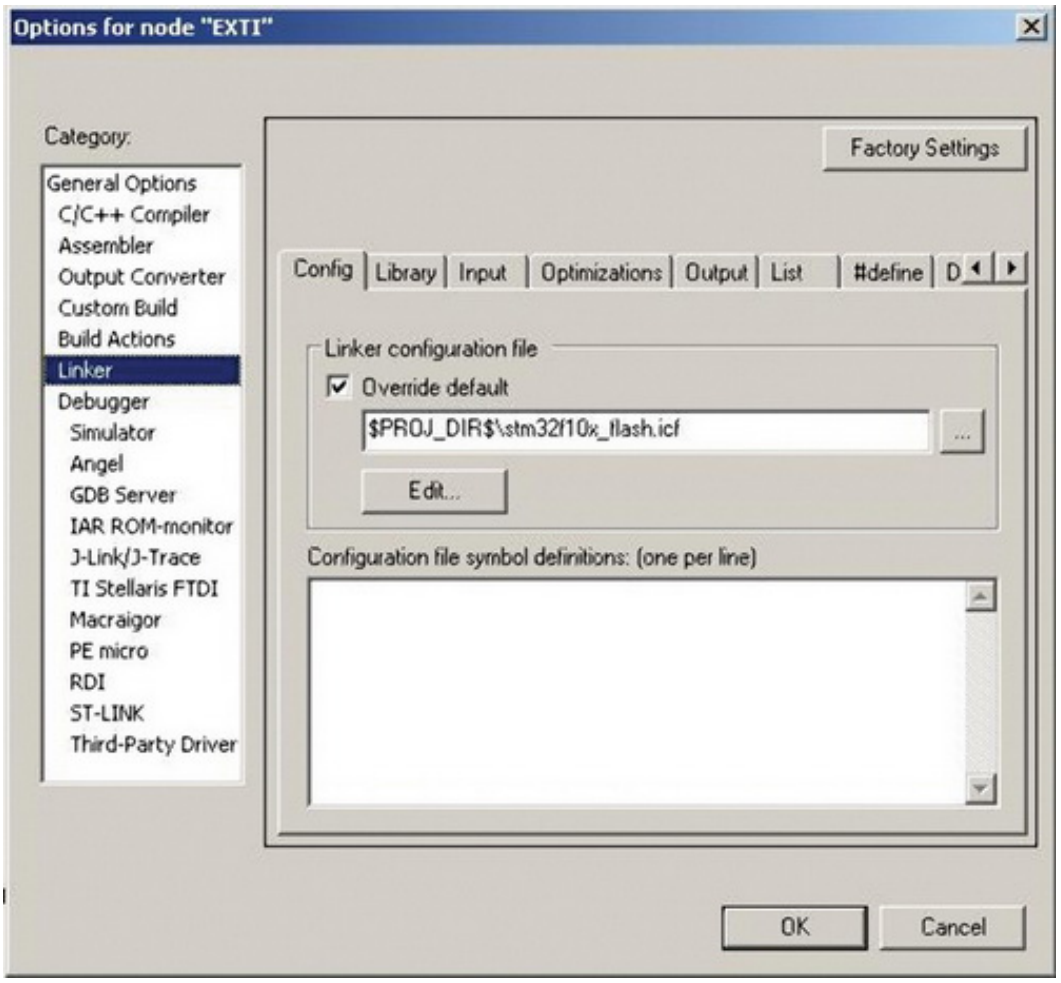

Рис. 7. Настройка проекта – настройка линкера (часть 1)

Также линковщику необходимо указать объемы памяти, стека и кучи. Данная операция осуществляется в этом же окне, но необходимо нажать кнопку *Edit…* (рис. 8).

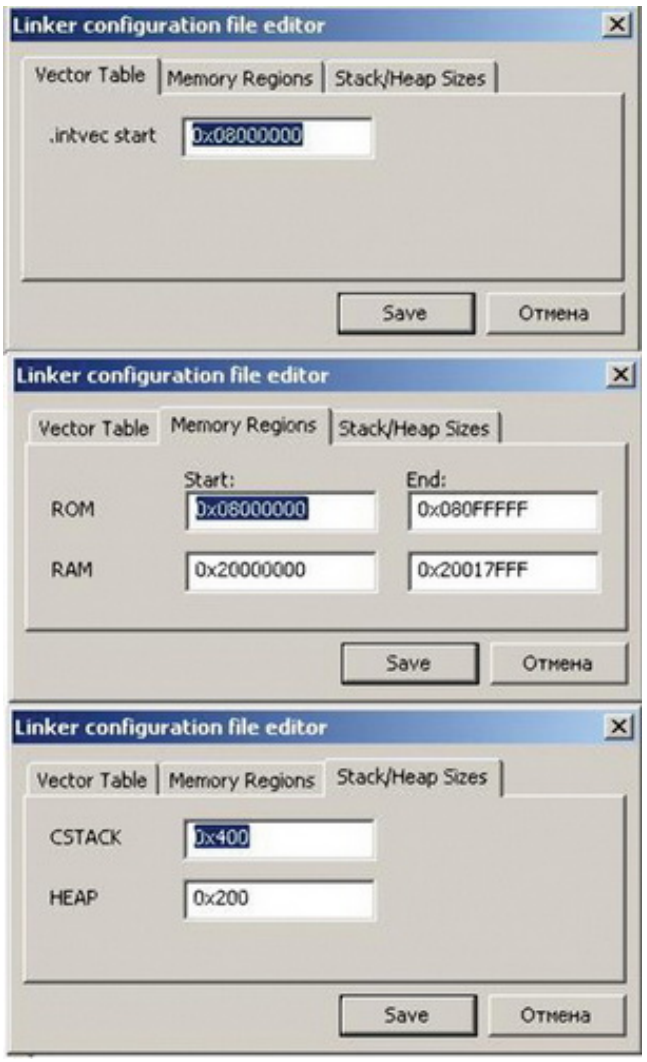

Рис. 8. Настройка проекта – настройка линкера (часть 2)

В первом окне рисунка 8 указывается адрес таблицы векторов прерывания. В нашем случае адрес равен 0x08000000, это начало внутренней flash-памяти. Во втором окне необходимо указать адреса flash- и SRAM-памяти. Начальные адреса начинаются с 0x08000000 и 0x02000000, это заложено при проектировании. А вот конечные адреса разные, т.к. микроконтроллеры имеют разные объемы памяти. В окнах необходимо указать правильные параметры, т.к. в случае выхода размера ПО за эти границы линкер должен выдать сообщение об ошибке во избежание непредсказуемых последствий. Для нашего микроконтроллера STM32F100RBT6 объем flash равен 128 Кбайт, а объем SRAM – 8 Кбайт. Соответственно, значение второго окна для поля ROM должно быть равно (128x1024) – 1 = 0x0801FFFF, и по такой же методике значение второго окна должно быть равно 0x02001FFF. На рисунке 8 отображены максимальные значения памяти для семейства STM32. Не менее важным является третье окно, отвечающее за размер стека и кучи. Стек необходим для сохранения контекста при вызове функций, передачи параметров функций и так далее. Куча необходима для функций работы с памятью C/C++. Оставим эти параметры такими же, как на рисунке. Добавим лишь, что память для стека и кучи выделяется из SRAM, и, например, в случае неиспользования функции работы с памятью поле *HEAP* можно сделать нулевым, таким образом сэкономив оперативную память для нужд приложения.

Следующим шагом является выбор программатора-отладчика. В поле *Category* выбираем пункт *Debugger* и далее в правом поле – вкладку *Setup* (рис. 9)

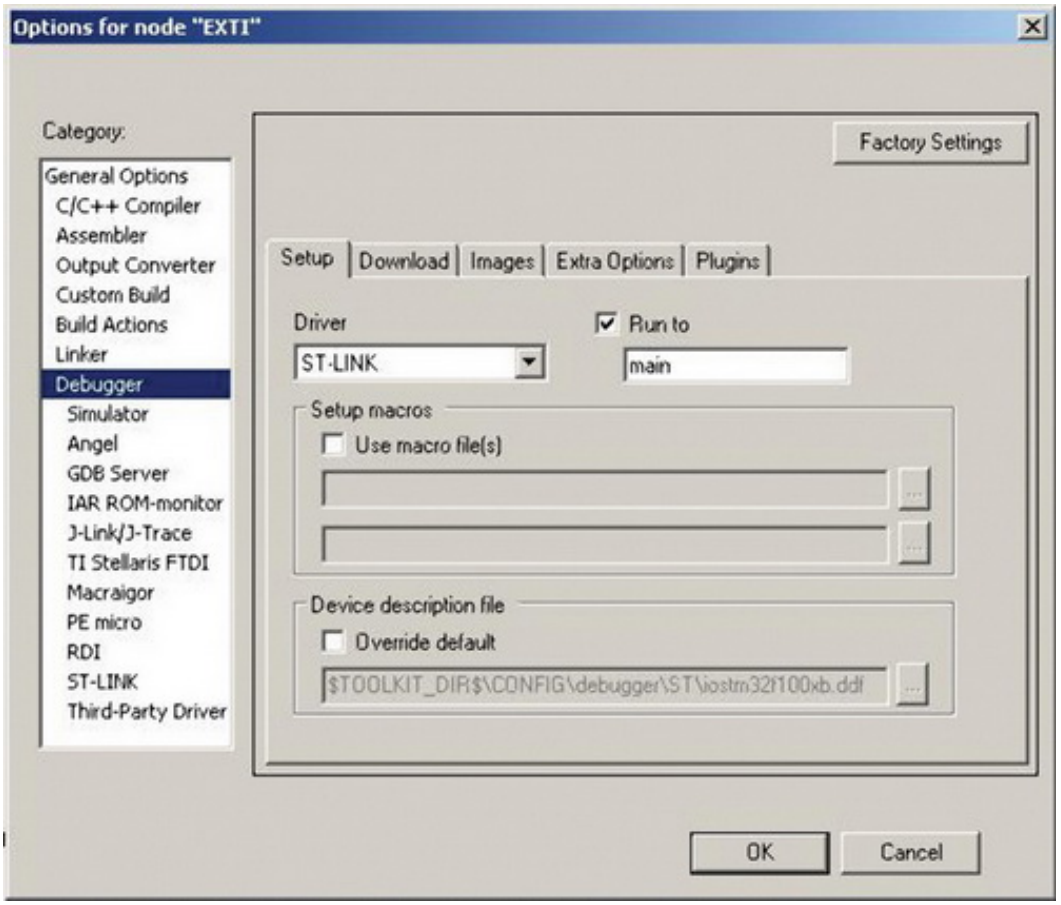

Рис. 9. Настройка проекта – выбор программатора

В нашем случае нам необходимо выбрать программатор-отладчик компании STMicroelectronics **ST-Link.** Для этого выбираем категорию *Linker.*

Следует обратить внимание на поле *Run to main.* Данный пункт указывает, с какого места будет начинаться отладка ПО. Если этот пункт активен, то процесс отладки начнется с функции main(), и первоначальная инициализация микроконтроллера пропускается (она будет выполнена автоматически). Если же данный пункт не активен, то процесс отладки начнется с самого начала, а именно – с вектора сброса *Reset Handler* файла startup_stm32f10x_md_vl.s. На первоначальном этапе для более подробного изучения инициализации микроконтроллера лучше сделать этот пункт неактивным и посмотреть по шагам всю инициализацию микроконтроллера.

И, наконец, последний пункт в настройках проекта – выбор интерфейса для отладки. В архитектуре ARM Cortex-M3 предусмотрено два альтернативных интерфейса для отладки микроконтроллера – двухпроводной интерфейс SWD и пятипроводной интерфейс JTAG. В STM32VLDiscovery реализован SWD, соответственно, убеждаемся в том, что выбран именно этот интерфейс – в поле *Category* выбираем пункт *ST-Link* и затем – *SWD* (рис. 10).

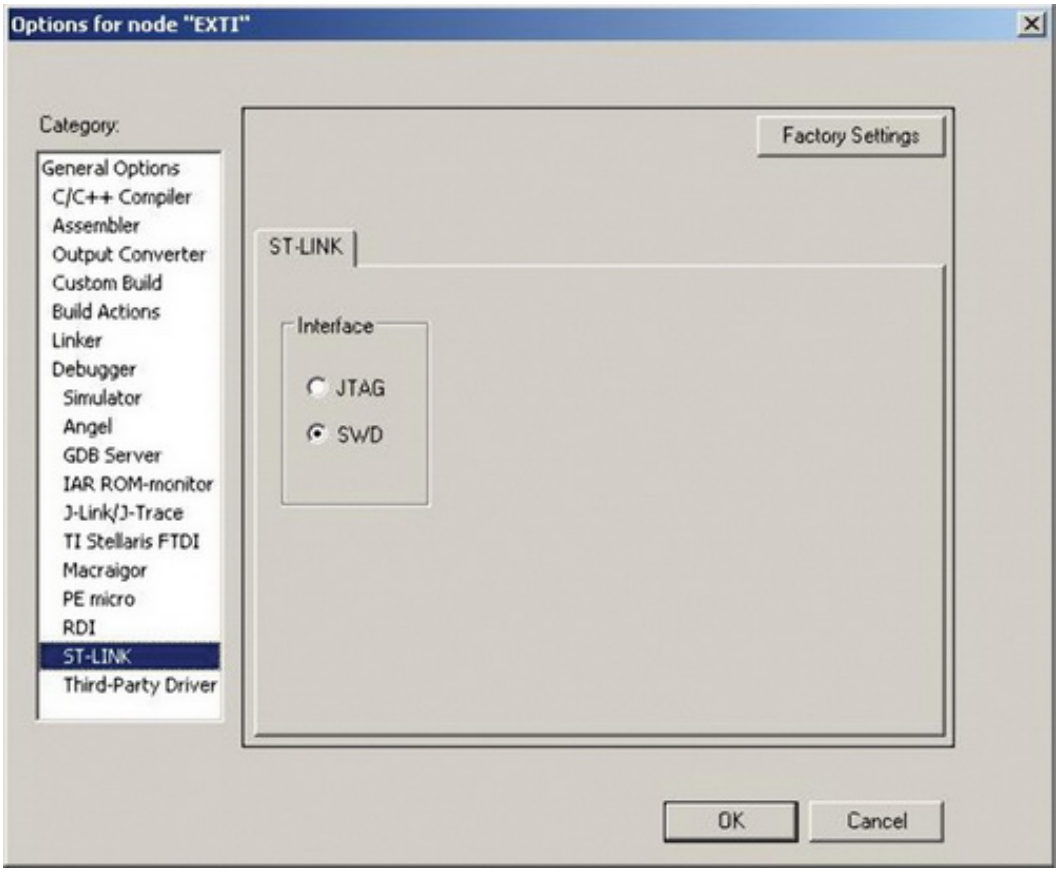

Рис. 10. Настройка проекта – выбор интерфейса программатора

После рассмотрения настроек проекта **EXTI-EXTI** переходим к более детальному рассмотрению самого проекта. Обычно точкой входа в программу является функция main.c, далее происходит инициализация периферии микроконтроллера. Но в случае использования стандартной библиотеки периферии при входе в функцию main.c уже произведена первоначальная инициализация микроконтроллера. Полную инициализацию микроконтроллера можно посмотреть по шагам в отладочном режиме, если снять пункт *Run to main* на рисунке 9. Старт работы ПО микроконтроллера после включения питания или сброса происходит с адреса сброса (Reset_Handler). Для начальной инициализации проекта мы уже добавили в проект файл начальной инициализации (в нашем случае это startup_stm32f10x_md_vl.s). При просмотре содержимого данного файла видно, что при первоначальном сбросе вызывается функция SystemInit(). Это, по сути, и является первоначальной инициализацией микроконтроллера. Далее управление передается этой функции, которая реализована в файле system_stm32f10x.c. Но отправной точкой работы любого микроконтроллера является его тактирование. В семействе STM32 доступны четыре источника тактирования:

- HSI – внутренний высокоскоростной источник тактирования,
- LSI – внутренний низкороскоростной источник тактирования,
- HSE – внешний высокоскоростной источник тактирования,
- LSE – внешний низкоскростной источник тактирования.

Для тактирования ядра микроконтроллера могут использоваться только высокочастотные источники HSI и HSE. После подачи или сброса питания микроконтроллеры STM32F1x по умолчанию тактируются от внутреннего источника, и системная частота тактирования после сброса равняется 8 МГц (SYSCLK = 8 МГц). В файле system_stm32f10x.c в макроопределениях задается первоначальная частота тактирования микроконтроллера до входа в функцию main.c при помощи раскомментирования строк (рис. 11):

```
#if defined (STM32F10X_LD_VL) || (defined STM32F10X_MD_VL)

/* #define SYSCLK_FREQ_HSE HSE_Value */
```

#define SYSCLK_FREQ_24MHz 24000000

#else

/* #define SYSCLK_FREQ_HSE HSE_Value */

/* #define SYSCLK_FREQ_24MHz 24000000 */

/* #define SYSCLK_FREQ_36MHz 36000000 */

/* #define SYSCLK_FREQ_48MHz 48000000 */

/* #define SYSCLK_FREQ_56MHz 56000000 */

#define SYSCLK_FREQ_72MHz 72000000

#endif

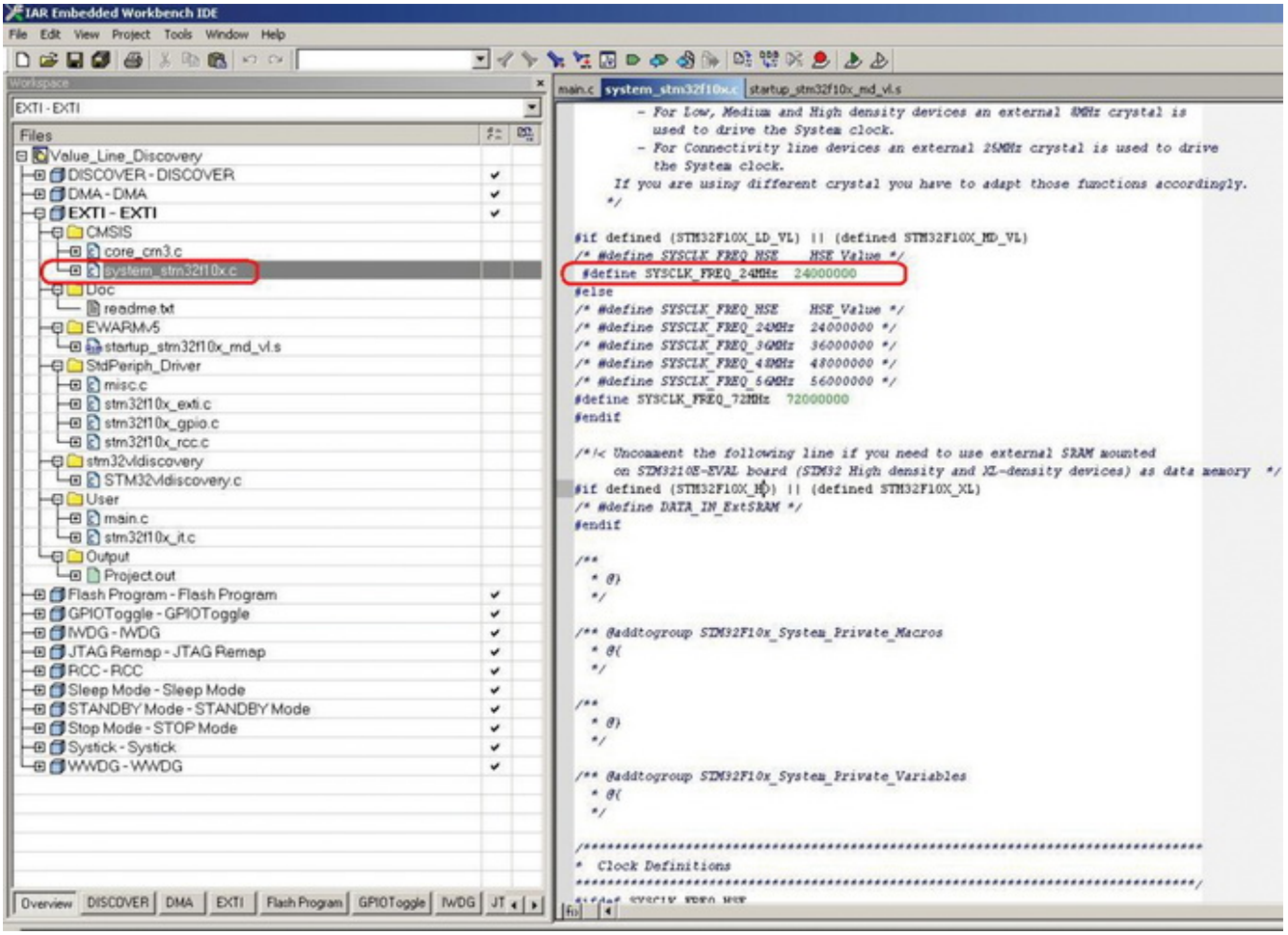

Рис. 11. Первоначальная инициализация в файле system_stm32f10x.c

Как видно из примера, после условной компиляции раскомментирована строчка #define SYSCLK_FREQ_24MHz 24000000 – соответственно, тактовая частота равна 24 МГц.

Теперь перейдем к основному файлу проекта main.c (рис. 12)

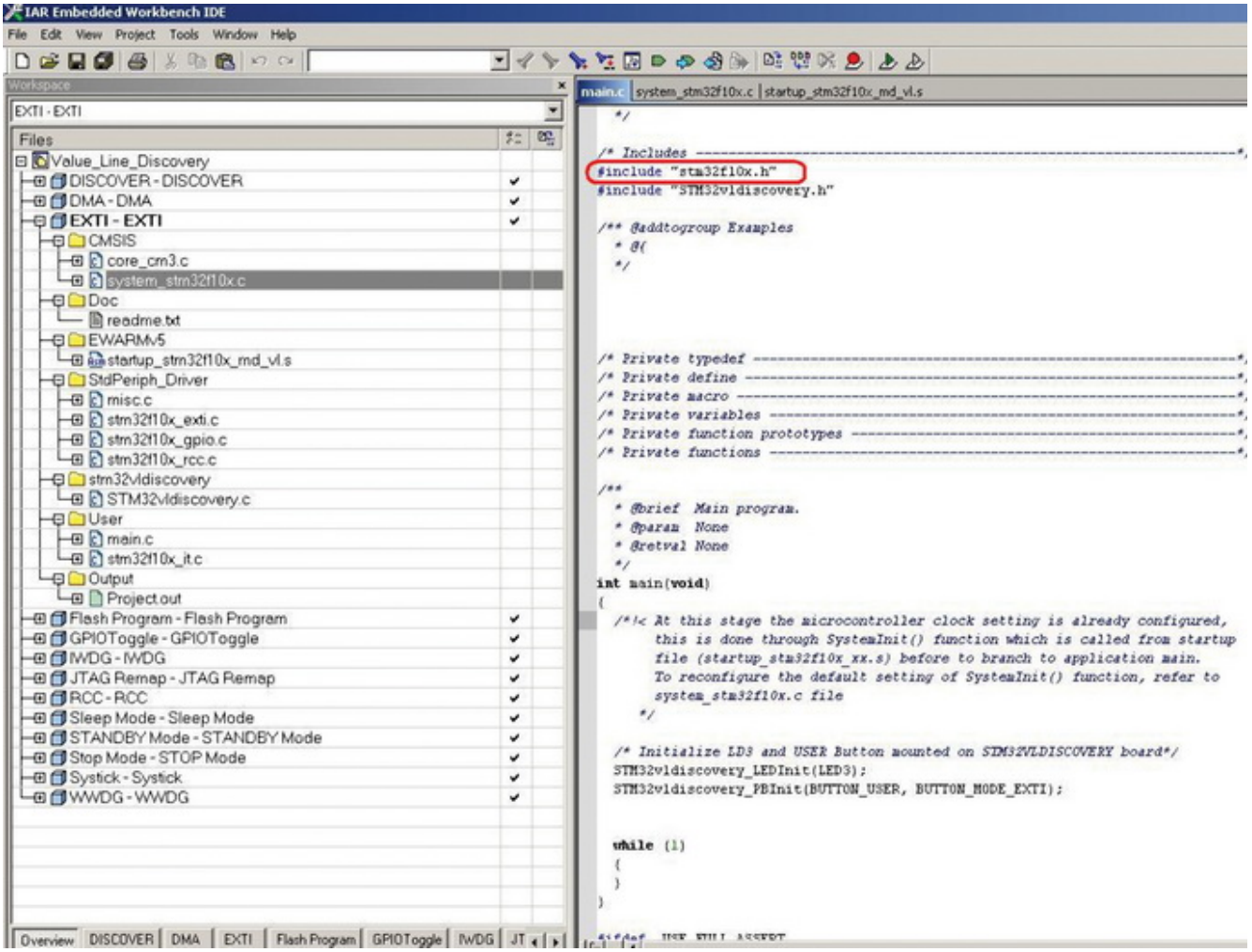

Рис. 12. Основной файл проекта main.c

Для использования стандартной библиотеки периферии необходимо прописать в основном файле проекта строку stm32f10x.h, раскомментировать используемые периферийные модули в файле stm32f10x_conf.h и включить их исполняемые файлы в проект в папку *StdPeriph_Driver.* В нашем проекте это файлы – misc.c – NVIC функции, stm32f10x_exti.c – функции внешних прерываний портов ввода-вывода, stm32f10x_gpio.c – функции настройки портов ввода-вывода и stm32f10x_rcc.c – функции настройки тактирования модулей периферии (рис. 13)

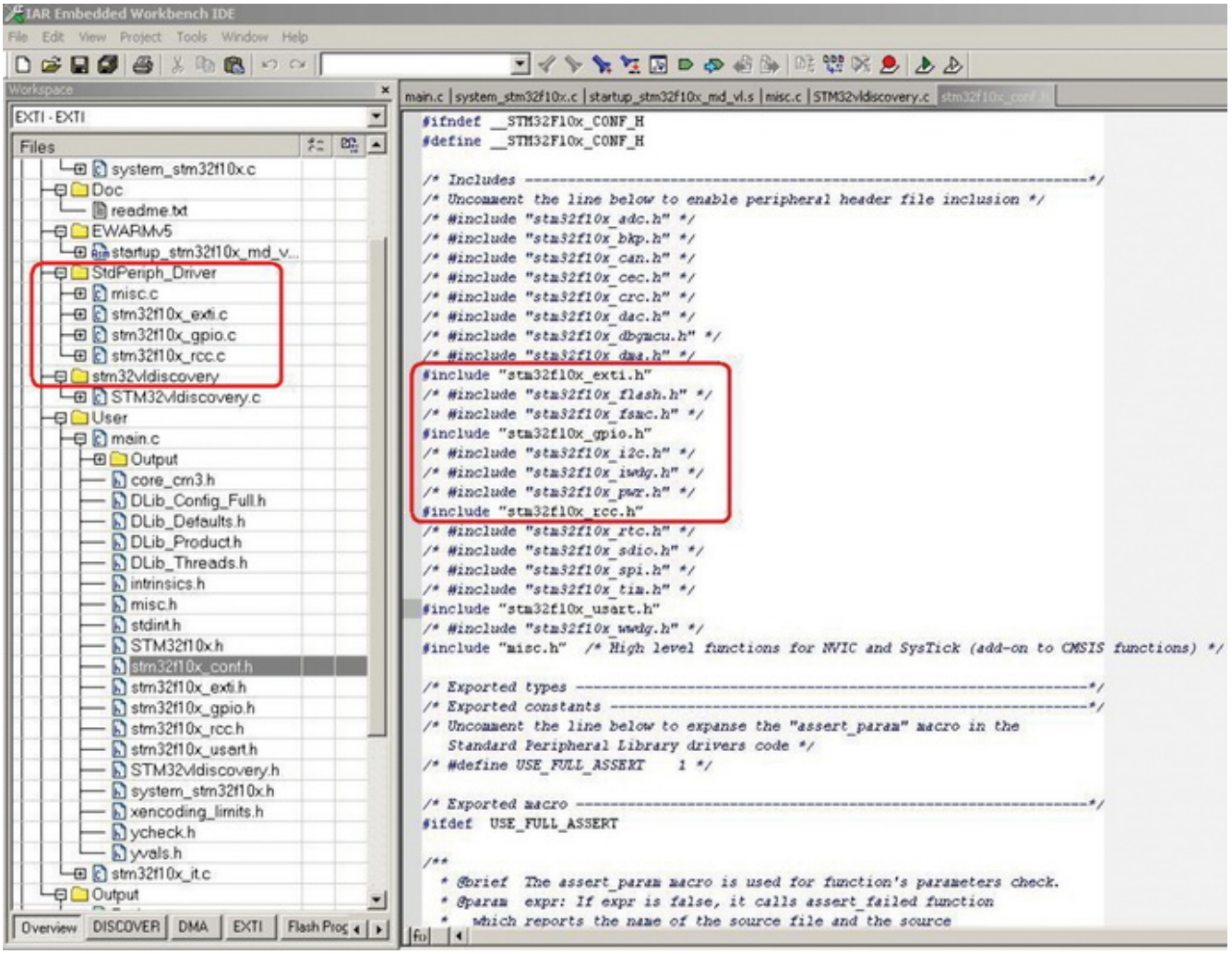

Рис. 13. Конфигурирование проекта

При просмотре содержимого функции main.c в проекте используются всего две функции – STM32vldiscovery_LEDInit(LED3) и STM32vldiscovery_PBInit(BUTTON_USER, BUTTON_MODE_EXTI), далее программа переходит на бесконечный цикл while(1) и ожидает событий источников прерывания. Из названия функций видно, что первая настраивает вывод микроконтроллера для управления светодиодом, а вторая настраивает вывод микроконтроллера для управления кнопкой. Рассмотрим более подробно данные функции, находящиеся в файле stm32vldiscovery.c.

Для быстрого перехода к реализации функции STM32vldiscovery_LEDInit(LED3) можно кликнуть правой кнопкой компьютерной мыши на имени функции и в выпадающем меню также кликнуть *Go to definition of имя_функции:*

```
void STM32vldiscovery_LEDInit(Led_TypeDef Led)

{

// Объявление структуры

GPIO_InitTypeDef GPIO_InitStructure;

// Разрешение тактирования модуля(RCC_APB2Periph_GPIOC)

RCC_APB2PeriphClockCmd(GPIO_CLK[Led], ENABLE);
```

```
// Установка вывода для дальнейшего использования(GPIO_Pin_9)

GPIO_InitStructure.GPIO_Pin = GPIO_PIN[Led];

// Настройка вывода на выход push-pull

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;

// Настройка частоты переключения вывода

GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz;

// Передача структуры – применение параметров

GPIO_Init(GPIO_PORT[Led], &GPIO_InitStructure);

}
```

Итак, для настройки ножки ввода-вывода сначала объявляется структура данных типа GPIO_InitTypeDef, далее заполняются ее поля, и эта структура с помощью функции GPIO_Init() «накладывается» на регистры периферийного модуля *GPIO* и, соответственно, настраивает модуль.

Функция STM32vldiscovery_PBInit(BUTTON_USER, BUTTON_MODE_EXTI) имеет более сложную реализацию, так как в ней настраиваются ножка микроконтроллера и внешнее прерывание на этой же ножке:

```
void STM32vldiscovery_PBInit(Button_TypeDef Button, ButtonMode_TypeDef Button_Mode)

{

// Объявление структуры модуля GPIO

GPIO_InitTypeDef GPIO_InitStructure;

// Объявление структуры модуля EXTI

EXTI_InitTypeDef EXTI_InitStructure;

// Объявление структуры модуля EXTI

NVIC_InitTypeDef NVIC_InitStructure;

// Разрешение тактирования модуля GPIOA(RCC_APB2Periph_GPIOA)

RCC_APB2PeriphClockCmd(BUTTON_CLK[Button] | RCC_APB2Periph_AFIO, ENABLE);

// Настройка вывода на вход

GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN_FLOATING;

// Установка вывода для дальнейшего использования (GPIO_Pin_0)

GPIO_InitStructure.GPIO_Pin = BUTTON_PIN[Button];
```

```
// Передача структуры – применение параметров
GPIO_Init(BUTTON_PORT[Button], &GPIO_InitStructure);
if (Button_Mode == BUTTON_MODE_EXTI)
{
// Конфигурирование кнопки для внешнего прерывания
GPIO_EXTILineConfig(BUTTON_PORT_SOURCE[Button],
BUTTON_PIN_SOURCE[Button]);
// Установка линии прерывания(EXTI_Line0)
EXTI_InitStructure.EXTI_Line = BUTTON_EXTI_LINE[Button];
// Конфигурация прерывания
EXTI_InitStructure.EXTI_Mode = EXTI_Mode_Interrupt;
// Конфигурирование внешнего прерывания по переднему фронту
EXTI_InitStructure.EXTI_Trigger = EXTI_Trigger_Rising;
// Разрешение прерывания
EXTI_InitStructure.EXTI_LineCmd = ENABLE;
// Применение параметров
EXTI_Init(&EXTI_InitStructure);
// Прерывание EXTI0_IRQn
NVIC_InitStructure.NVIC_IRQChannel = BUTTON_IRQn[Button];
// Установка приоритета группы
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0x0F;
// Установка приоритета подгруппы
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0x0F;
// Разрешение прерывания
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
// Применение параметров
NVIC_Init(&NVIC_InitStructure);
```

}}

Как уже говорилось выше, все вектора прерываний определены в файле startup_stm32f10x_md_vl.s, а обработчики прерываний реализуются в файле stm32f10x_it.c. Наш обработчик прерывания довольно прост: при нажатии на пользовательскую кнопку синего цвета происходит срабатывание внешнего прерывания по возрастающему фронту. В самом обработчике прерывания происходит переключение состояния светодиода и очищается флаг прерывания:

```
void EXTI0_IRQHandler(void)

{

if (EXTI_GetITStatus(USER_BUTTON_EXTI_LINE) != RESET)

{

// Переключение состояния светодиода

STM32vldiscovery_LEDToggle(LED3);

// Сброс флага прерывания

EXTI_ClearITPendingBit(USER_BUTTON_EXTI_LINE);

}}
```

Можно скомпилировать данный пример – *Project ® Rebuilt all,* загрузить ПО в память микроконтроллера Project ® *Download and Debug* (Ctrl+D) и проанализировать исполнение кода в режиме отладки. При нажатии на пользовательскую кнопку светодиод должен загораться или гаснуть. Но он будет вести себя не совсем предсказуемо, так как в данном примере не реализована функция устранения дребезга контактов кнопки.

Заключение

В статье на основе оценочной платы STM32VLDiscovery мы рассмотрели аспекты работы с семейством микроконтроллеров STM32, работу стандартной библиотеки периферийных устройств, а также разобрали небольшой пример. К сожалению, объем статьи ограничен форматом журнала и расписать все до мелочей нет возможностей. За дополнительной информацией следует обращаться на официальный сайт STMicroelectronics [1], где вы найдете большое количество разнообразных примеров, описаний и многое другое [8]. Автор искренне надеется, что данная статья поможет вам в первых шагах изучения семейства микроконтроллеров STM32.

Ссылки:

1. *Ссылка 1*
2. *Ссылка 2*
3. *Ссылка 3*
4. *Ссылка 4*
5. *Ссылка 5*
6. *Ссылка 6*
7. *Ссылка 7*
8. *Ссылка 8*

Нет механическим кнопкам!

STM8T141 - одноканальный, полностью интегрированный емкостной датчик, разработанный лидером европейской электронной индустрии – **STMicroelectronics** для замены электромеханических переключателей в чувствительных к цене приложениях. Во многих устройствах электромеханические переключатели уже полностью или почти вытеснены touch-sence-технологиями, и на это есть ряд объективных причин, прежде всего – невысокая надежность механических переключателей.

Микросхема STM8T141 производится в 8-выводном корпусе с минимальным количеством внешних компонентов и идеально подходит для приложений, требующих одного переключателя. Он может быть сконфигурирован на срабатывание либо по прикосновению к датчику, либо на некотором расстоянии от него. Ультранизкое энергопотребление делает датчик идеальным решением для систем с батарейным питанием. STM8T141 обладает высокой чувствительностью и детектирует прикосновения почти через любой диэлектрик, что позволяет его использовать в различных приложениях. Например, он может быть замаскирован или загерметизирован в различных материалах, что позволяет многократно увеличить срок службы устройства в целом, а также использовать его в системах безопасности.

Основные характеристики:

- детектирование по прикосновению или на расстоянии (несколько см)
- встроенная функция управления защитой:
  - улучшенное детектирование на расстоянии
  - защита встроенного электрода от шумовых помех
- ультранизкое энергопотребление (11 мкА)
- встроенный регулятор напряжения
- фильтр компенсации влияния окружающей среды
- конфигурируемые опции:
  - четыре порога детектирования
  - четыре выходных режима
  - четыре режима низкого энергопотребления
  - конфигурация времени детектирования
- минимальное количество внешних компонентов.

Рубрика: Процессоры Метки: ARM, STM, Микроконтроллеры, Средства разработки и отладки

STM32: эпоха 32-битных микроконтроллеров наступила

Прыжок производительности: микроконтроллеры серии STM32F2