

В. И. Бугаев, М. П. Мусиенко, Я. М. Крайнык

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

**по изучению микроконтроллеров архитектуры ARM Cortex-M4
на базе отладочного модуля STM32F4 Discovery**



**Работа выполнена при инвестиционной
финансовой поддержке в рамках реализации
международного студенческого проекта
«Нанопорошковый 3D-принтер»**

Лабораторный практикум для изучения микроконтроллеров архитектуры ARM Cortex-M4 на базе отладочного модуля STM32F4 Discovery / Бугаев В.И., Мусиенко М.П., Крайнык Я.М. – Москва-Николаев: МФТИ-ЧГУ, 2013. – 71 с.

Бугаев Виктор Иванович, Московский физико-технический институт,
г. Москва, Россия

Мусиенко Максим Павлович, д.т.н., профессор, Черноморский государственный
университет им. П. Могилы, г. Николаев, Украина

Крайнык Ярослав Михайлович, аспирант, Черноморский государственный
университет им. П. Могилы, г. Николаев, Украина

Лабораторный практикум содержит материалы, которые будут полезны для изучения микроконтроллеров архитектуры ARM Cortex-M4. В материалах приведено общее описание архитектуры ARM и 32-разрядных микроконтроллеров STM, а также дана общая информация, которая необходима для начала работы с отладочной платой STM32F4Discovery. Приведено восемь лабораторных работ для изучения основных возможностей, устройств и характеристик платы: ШИМ, АЦП, USART, SPI, DMA, таймеры и др.

Материалы могут быть полезны студентам электронных, компьютерных и информационных специальностей, а также аспирантам, преподавателям и всем желающим, которые занимаются изучением микроконтроллеров архитектуры ARM.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	4
I. ОБЩЕЕ ОПИСАНИЕ АРХИТЕКТУРЫ ARM И 32-РАЗРЯДНЫХ МИКРОКОНТРОЛЛЕРОВ STM	5
32-разрядная архитектура ARM	5
Семейство микроконтроллеров STM32	7
Короткое описание платы STM32F4 Discovery	7
II. НАЧАЛО РАБОТЫ С ОТЛАДОЧНОЙ ПЛАТОЙ STM32F4 DISCOVERY	10
Подключение платы и установка среды разработки	10
Дополнительное программное обеспечение	14
Графические средства настройки микроконтроллеров STM32	14
Среда разработки CoIDE	17
Средства настройки частоты работы микроконтроллера	18
Программные средства для прошивки платы	19
III. ЛАБОРАТОРНЫЕ РАБОТЫ	21
Лабораторная работа № 1. Создание проекта в среде разработки. Использование портов ввода/вывода ..	21
Лабораторная работа № 2. Прерывания и их использование. Использование таймеров	24
Лабораторная работа № 3. Генерация сигнала ШИМ	26
Лабораторная работа № 4. Использование АЦП	29
Лабораторная работа № 5. Использование USART	32
Лабораторная работа № 6. Работа с SPI	35
Лабораторная работа № 7. Работа с DMA	35
Лабораторная работа № 8. Генерация сигналов и управление их характеристиками	56
ПРИЛОЖЕНИЕ А. Работа с дисплеем	66
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ	70

ВВЕДЕНИЕ

Материал лабораторного практикума служит для помощи в изучении возможностей 32-разрядных микроконтроллеров и их использования в построении информационных систем, организации взаимодействия нескольких устройств между собой, использование передачи и отображения информации в компьютерных системах, обучении самостоятельной разработке программного обеспечения с использованием специализированных программ, проведении тестирования и отладки на реальных устройствах.

В ходе выполнения лабораторных работ изучаются принципы работы микроконтроллеров, их основной периферии, организации передачи данных с их использованием, управление другими устройствами для измерения внешних показателей, настройка отображения информации, полученной от внешних устройств. Обучаемые получают возможность на практике самостоятельно настроить работу демонстрационных примеров и разработать собственные в соответствии с индивидуальным заданием.

Материал лабораторного практикума состоит из трех основных разделов:

- общее описание архитектуры ARM и 32-разрядных микроконтроллеров STM;
- общая информация, которая необходима для начала работы с отладочной платой STM32F4Discovery;
- восемь лабораторных работ для изучения основных возможностей, устройств и характеристик платы:

ШИМ, АЦП, USART, SPI, DMA, таймеры и прочее.

Кроме того, в приложении приводится информация относительно возможности подключения к плате дисплея.

Для выполнения лабораторных работ предпочтительно знание основ теории цифровой схемотехники, разработки программного обеспечения и алгоритмизации, а также знание языка программирования C++.

I. ОБЩЕЕ ОПИСАНИЕ АРХИТЕКТУРЫ ARM И 32-РАЗРЯДНЫХ МИКРОКОНТРОЛЛЕРОВ STM

32-разрядная архитектура ARM

Процессоры ARM являются ключевым компонентом для большого количества успешных 32-битных встраиваемых систем. Процессоры ARM широко используются в мобильных телефонах, планшетах и других портативных устройствах. ARM основаны на RISC-архитектуре, что позволяет уменьшить потребление энергии процессором и, таким образом, делает их идеальным выбором для встраиваемых систем.

Хотя ARM основаны на RISC-архитектуре, они не полностью повторяют принципы построения таких систем. Для того, чтобы сделать ARM более приспособленными к использованию во встраиваемых системах, пришлось пойти на следующие отклонения от принципов RISC:

1. Переменное количество циклов выполнения для простых инструкций. Простые инструкции ARM могут потребовать на выполнение более одного цикла. Например, выполнение инструкций Load и Save зависит от количества регистров, которые им переданы.
2. Возможность соединять команды сдвига и вращения с командами обработки информации.
3. Условное выполнение – инструкция выполняется только в том случае, если выполняется конкретное условие. Это увеличивает производительность и позволяет избавиться от операторов ветвления.
4. Улучшенные инструкции – процессоры ARM поддерживают улучшенные DSP-инструкции для операций с цифровыми сигналами.

Программист может рассматривать ядро ARM как набор функциональных блоков – ALU, MMU и др., – соединенных шиной данных. Данные поступают в процессор через шину данных. Декодер инструкций обрабатывает инструкции перед их выполнением. ARM могут работать только с данными, которые записаны в регистрах, поэтому перед выполнением инструкций в регистры записываются данные для их выполнения. ALU считывает данные из регистров, выполняет необходимые операции и записывает результат обратно в регистр, откуда его можно записать во внешнюю память.

Процессоры ARM содержат до 18 регистров: 16 регистров данных и 2 регистра процессоров. Все регистры содержат 32 бита и именуются от R0 до R15. Регистры R13, R14, R15 используются для выполнения определенных специфических задач:

- R13 используется в качестве указателя стека;
- R14 используется как связывающий регистр;
- R15 играет роль счетчика.

В зависимости от контекста эти регистры могут использоваться как регистры общего назначения. Также имеется два программных регистра, которые называются CPSR (Current Program Status Register) и SPSR (Saved Program Status Register), которые используются для сохранения состояния процессора и программы.

Одними из последних процессоров для встраиваемых систем, являются процессоры, основанные на архитектуре ARM Cortex-M4. Эти процессоры предназначены для использования в цифровой обработке сигналов (Digital Signal Processing, DSP). В общем виде микроконтроллеры, основанные на базе ARM Cortex-M4 имеют следующие внутренние модули (рис. 1.1): Микроконтроллер, установленный на рассматриваемой плате, STM32F407VG, в качестве основы использует именно решение ARM Cortex-M4.

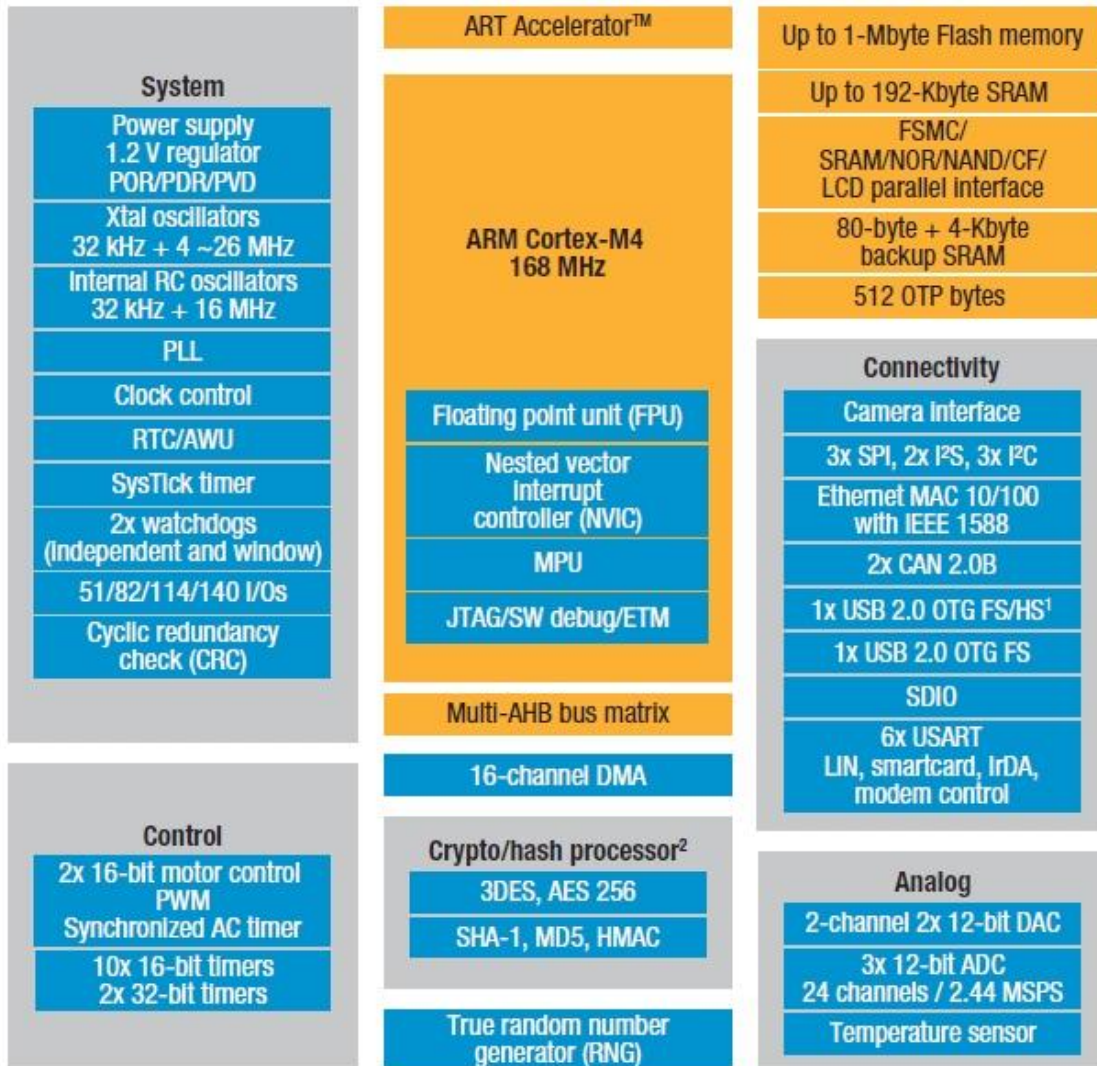


Рис. 1.1. Встроенные модули ARM Cortex-M4

На рис. 1.1 видно, что рассматриваемый процессор обладает богатой периферией и может использоваться для решения многих практических задач различной направленности.

Семейство микроконтроллеров STM32

Семейство микроконтроллеров STM32 построено с использованием 32-разрядного ядра Cortex различных версий (в микроконтроллере, установленном на плате используется ядро Cortex-M4). Некоторые основные характеристики ядра микроконтроллеров STM32 представлены в табл. 1.1.

Таблица 1.1. Основные характеристики ядра микроконтроллеров STM32

Характеристика	Значение
Ширина слов для данных, разряд	32
Архитектура	Гарвард
Конвейер	3-ступенчатый
Набор инструкций	RISC
Организация памяти программ, разряд	32
Буфер предвыборки, разряд	2x64
Средний размер инструкции, байт	2
Тип прерываний	Векторизированные
Задержка реагирования на прерывания	12 циклов
Режимы управления энергопотреблением	Сон, сон по выходу, глубокий сон
Отладочный интерфейс	ST-LINK, JTAG

Микроконтроллеры данного типа построены на гарвардской архитектуре и имеют 3-ступенчатый конвейер, который минимизирует время выполнения команд. Они разработаны для построения систем с максимальной энергоэффективностью и имеют несколько режимов управления энергопотреблением. В них используются внутренние интерфейсы памяти шириной больше, чем средняя длина инструкции. Это минимизирует число доступов к шине памяти, а следовательно, и потребление электроэнергии, связанное с операциями по шине и чтением энергонезависимой памяти. Технология непрерывной обработки прерываний с исключением внутренних операций над стеком (tail chaining) сокращает время реакции на прерывания и исключает лишние операции со стеком.

На рис. 1.2 представлено упрощенное представление цифрового периферийного устройства.

Периферийный узел может быть разделен на два главных блока. Первый блок – это ядро, которое содержит конечные автоматы, счетчики и любой вид комбинаторной или последовательной логики. Оно предназначено для выполнения задач, не требующих участия процессора, таких как простые задачи передачи данных, управления аналоговыми входами или выполнения функций, привязанных к синхросигналам. Ядро периферийного узла связывается с внешним миром через порты ввода/вывода МК. Внешние соединения могут состоять из нескольких сигналов или сложных шин. Второй блок – настройка и управление периферией, которые осуществляются приложением через регистры, соединенные с внутренней шиной, разделяемой с другими ресурсами МК.

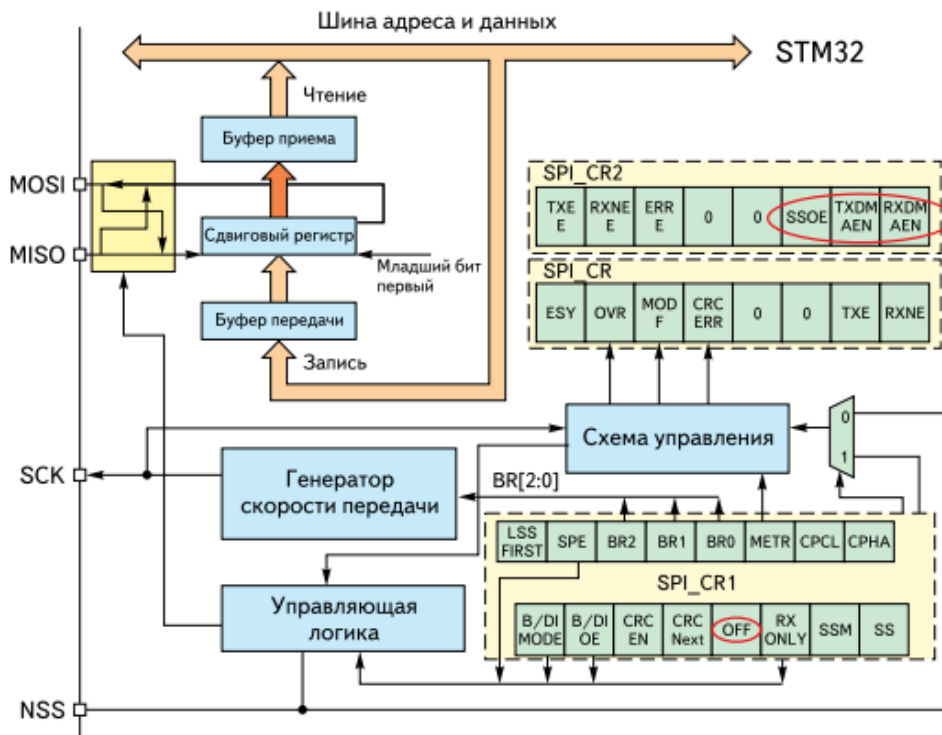


Рис. 1.2. Представление цифрового периферийного устройства

Короткое описание платы STM32F4 Discovery

Плата STM32F4 Discovery (рис. 1.3) предназначена для ознакомления с возможностями 32-битного МК на основе ARM-архитектуры, а также для реализации собственных устройств и приложений с использованием аппаратного обеспечения платы.

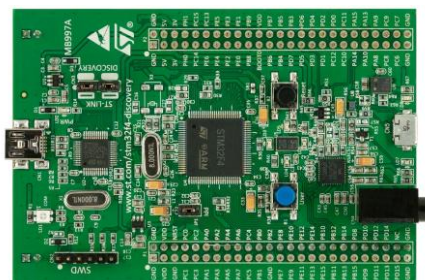


Рис. 1.3. Внешний вид платы STM32F4 Discovery

Плата STM32F4 Discovery оснащена:

- микроконтроллером STM32F407VGT6 с ядром Cortex-M4F тактовой частотой 168 МГц, 1 Мб Flash-памяти, 192 кб RAM в корпусе LQFP100;
- отладчиком ST-Link/V2 для отладки и программирования МК;
- питанием платы через USB или от внешнего источника питания 5 В;
- датчиком движения ST MEMS LIS302DL и выходами цифрового акселерометра по трем осям;
- датчиком звука ST MEMS MP45DT02;
- звуковым ЦАП CS43L22;
- восемью светодиодами: LD1 (красный/зеленый) для USB-подключения, LD2 (красный) для питания 3.3 В, четыре пользовательские светодиода: LD3 (оранжевый), LD4 (зеленый), LD5 (красный), LD6 (синий) и два светодиода для USB On-The-Go – LD7 (зеленый) и LD8 (красный);
- двумя кнопками (для программирования пользователем и для перезапуска).

Таким образом, отладочная плата оснащена большим количеством периферии, что позволяет сразу же реализовывать на ней примеры различной сложности.

II. НАЧАЛО РАБОТЫ С ОТЛАДОЧНОЙ ПЛАТОЙ STM32F4 DISCOVERY

Подключение платы и установка среды разработки

Данный раздел предназначен для тех, кто до этого момента не имел ни практических, ни теоретических знаний по работе с микроконтроллерами, а также для тех, кому имеющихся знаний недостаточно для начала работы с платой. Несмотря на внешнюю сложность платы, ее устройство хорошо продумано, поэтому даже абсолютным новичкам в этом деле не составит труда освоить азы работы с STM32F4 Discovery.

Какое аппаратное обеспечение нужно для подключения?

Единственное, что потребуется для подключения платы к ПК, – это кабель USB с разъемами USB типа А и mini-USB типа В. В комплекте поставки такой шнур отсутствует, поэтому его придется либо поискать у себя дома, либо купить в ближайшем компьютерном магазине. Скорее всего, Вам такие кабели (рис. 2.1) знакомы по телефонам, планшетам или фотоаппаратам, которые часто подключаются к компьютеру именно с их использованием.



Рис. 2.1. Кабель для подключения платы к компьютеру

При подключении будьте внимательны: на плате есть два разъема USB, один из которых предназначен для подключения, а второй – для реализации работы с USB.

Предполагается, что на компьютере установлена операционная система семейства Windows (XP, Vista, Windows 7 и др.). Существует возможность подключения к ПК с Linux, но это потребует намного больше усилий по установке программного обеспечения.

Какое программное обеспечение нужно для подключения?

Для начала работы следует установить среду разработки. Далее рассмотрен процесс установки и начала работы для одной из самых популярных средств разработки для ARM – Keil. Рассмотрим детали установки и использования среды разработки Keil uVision.

Скачать программу можно с сайта производителя – <https://www.keil.com/download/product/>. Выбрав для загрузки пункт MDK-ARM последней версии (на момент написания материала таковой была версия 4.71a), мы попадаем на страницу, на которой необходимо ввести свои персональные данные (рис 2.2).

ARM Software

Microcontroller Development Kit
Version 4.71a

Complete the following form to download the Keil software development tools.

Enter Your Contact Information Below

First Name:

Last Name:

E-mail:

Company:

Address:

City:

State/Province:

Zip/Postal Code:

Country:

Phone:

Send me e-mail when there is a new update.

NOTICE:
If you select this check box, you **will** receive an e-mail message from Keil whenever a new update is available. If you don't wish to receive an e-mail notification, don't check this box.

I am using devices from: Analog Devices Holtek SiLabs
(Select all that apply) Atmel Infineon ST
 Cypress Nuvoton TI
 Energy Micro NXP Toshiba
 Freescale Other Other
 Fujitsu Samsung

Which ARM architectures are you using? Cortex-M0 Cortex-M4
(Select all that apply) Cortex-M1 Other
 Cortex-M3

Рис. 2.2. Форма регистрации для загрузки Keil

После заполнения формы мы получаем возможность скачать программный продукт.

Установка потребует достаточно много свободного дискового пространства (около 2.5 Гб), поэтому следует заранее подготовить необходимые ресурсы. После запуска установки откроется следующее окно (рис. 2.3).

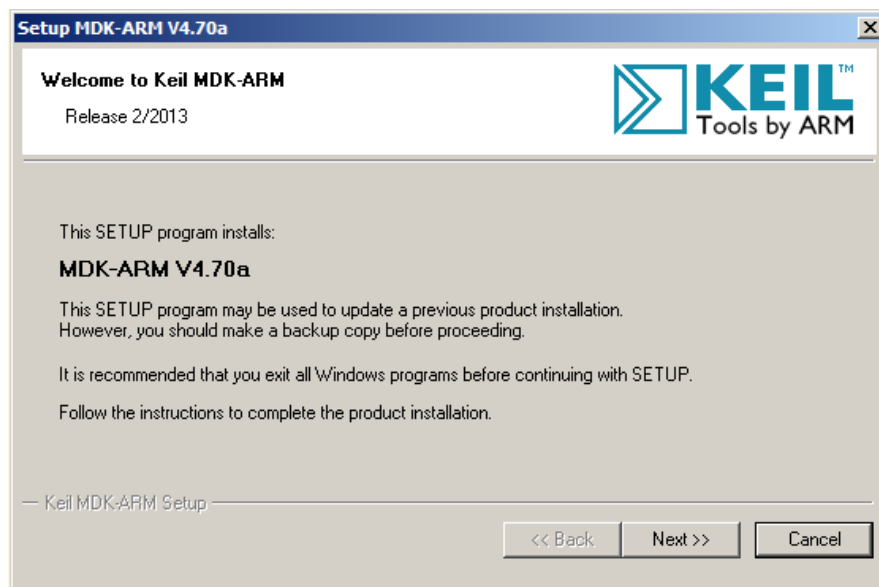


Рис. 2.3. Окно установщика Keil

Далее необходимо принять условия лицензионного соглашения, а также выбрать директорию для инсталляции (рис. 2.4), а также указать информацию о пользователе.

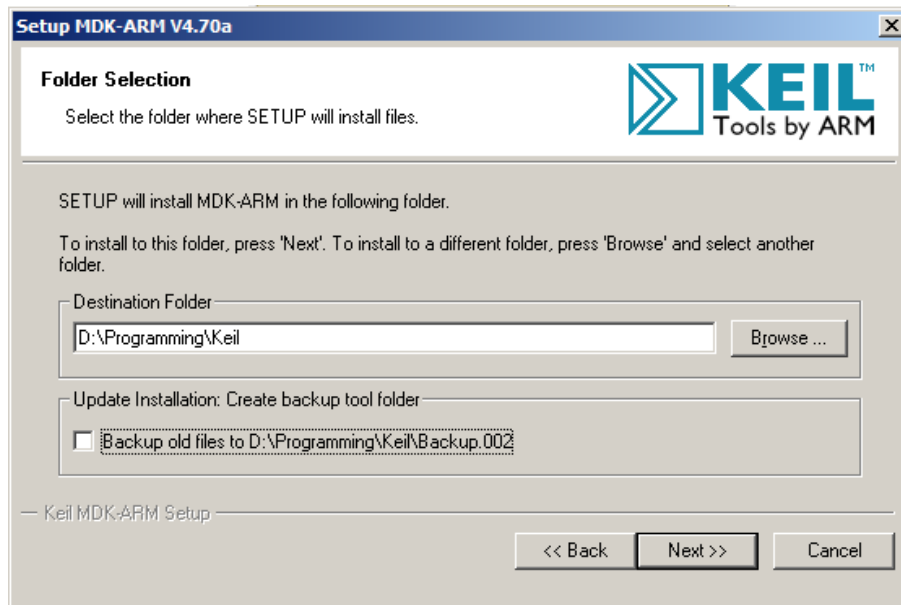


Рис. 2.4. Выбор директории для инсталляции

После этого начнется непосредственно процесс установки.

Кроме самой среды разработки для написания программ необходимы также дополнительные библиотеки Cortex Microcontroller Software Interface Standard (CMSIS) и Standard Peripheral Library (SPL). Лучше всего, если Вы загрузите данные библиотеки с сайта производителя st.com.

Библиотека SPL служит для управления всеми основными устройствами, входящими в состав микроконтроллера: USART, SPI, DMA, ADC, DAC и другие. Основным достоинством библиотеки является то, что она делает более понятным управление микроконтроллером для разработчика, в том числе и начинающего, который еще не знает всех тонкостей настроек. Кроме того, вместе с самой библиотекой распространяются и демонстрационные проекты для работы с основными устройствами, на которые можно опираться при написании собственных проектов.

После открытия окно среды выглядит следующим образом (рис. 2.5):

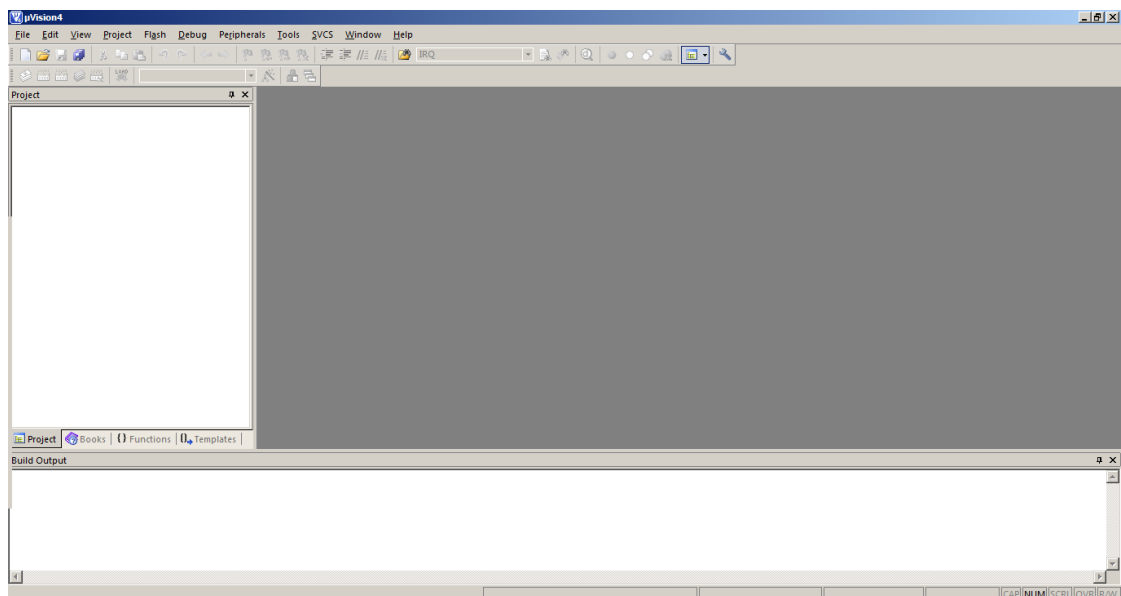


Рис. 2.5. Окно среды разработки

Теперь мы можем создать проект для нашего устройства. Для этого выполняем команду меню Project\New uVision Project. В результате появится диалоговое окно для выбора папки расположения проекта. Для каждого проекта желательно создавать отдельную папку, поскольку проект может разрастаться и содержать большое количество файлов, поэтому держать 2 проекта в одной папке будет неудобно. После выбора папки следует выбрать микроконтроллер, для которого планируется написание программы. Выбираем производителя (STMicroelectronics) и конкретную модель контроллера STM32F407VG (рис. 2.6).

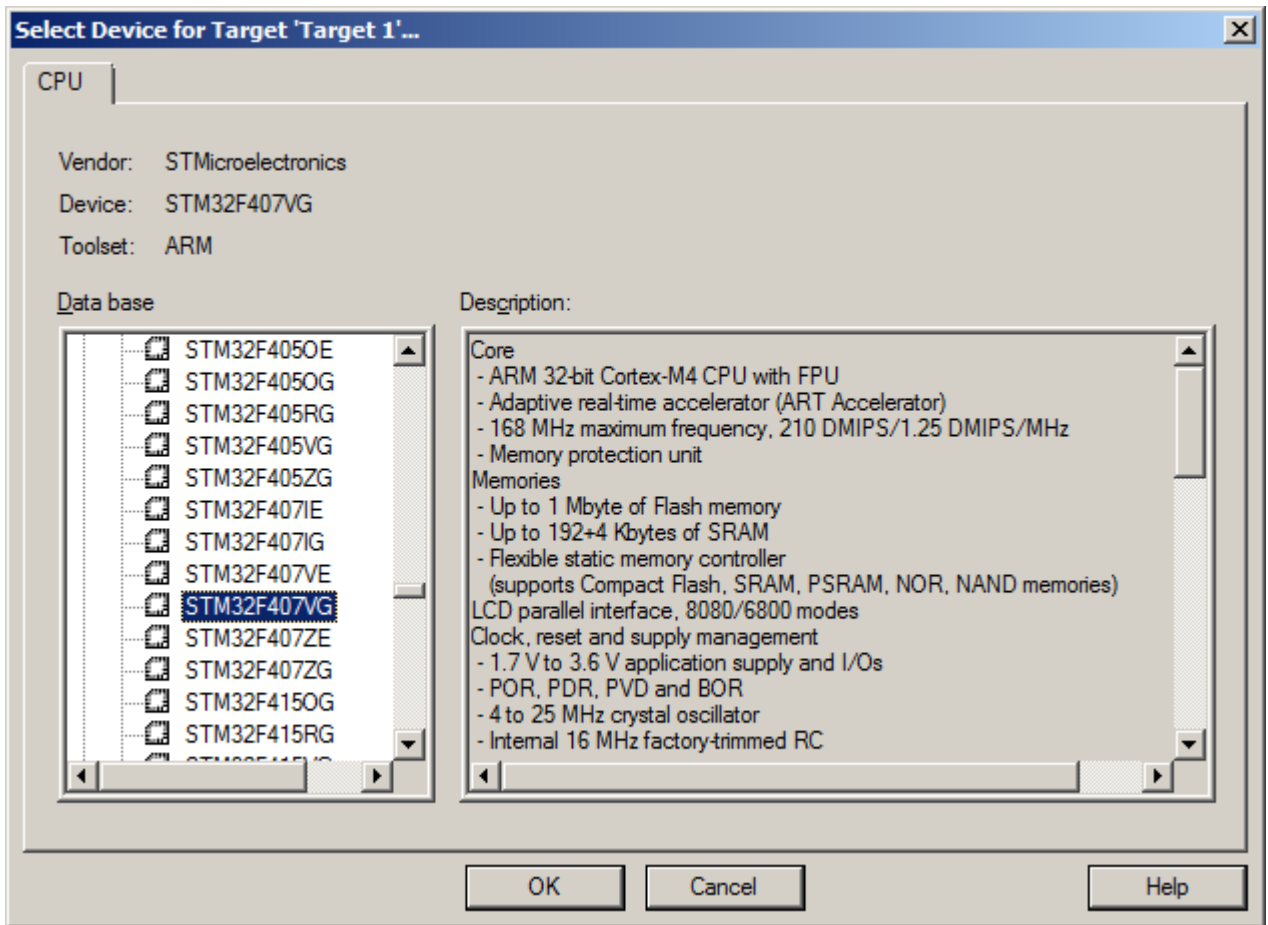


Рис. 2.6. Выбор производителя (STMicroelectronics) и модели микроконтроллера STM32F407VG

Далее откроется диалоговое окно с предложением добавить к проекту файл запуска startup_stm32f4xx.s, которое нужно подтвердить. Среда разработки создаст проект, а также создаст в нем группу для исходных файлов, в которой и будет расположен упомянутый выше файл. Группы файлов служат для удобной организации представления файлов в проекте. В данном примере мы будем использовать 4 группы файлов: Startup, User, CMSIS и SPL. Для начала переименуем созданную по умолчанию группу и назовем ее Startup и добавим еще 3 группы с помощью контекстного меню проекта в панели Project, выполнив команду Add Group.

Теперь добавим необходимые нам файлы в проект. Перед добавлением файлов желательно скопировать их в директорию проекта. Из библиотеки CMSIS нам понадобятся следующие файлы:

- stm32f4xx.h;
- system_stm32f4xx.h;
- system_stm32f4xx.c;
- stm32f4xx_conf.h;
- core_cm4.h.

Все перечисленные выше файлы находятся в директории, где распакован архив с библиотекой.

В группу SPL Вам нужно добавить файлы из библиотеки SPL для работы с устройствами. Необходимо добавлять пару исходного и заголовочных файлов. Например, для работы с портами ввода/вывода следует добавить файлы stm32f4xx_gpio.c и stm32f4xx_gpio.h. Обязательно следует подключить файлы для управления устройством сброса и тактирования, а именно, stm32f4xx_rcc.c и stm32f4xx_rcc.h. Они содержат функции для включения тактирования периферийных устройств.

После этого остается добавить еще один файл, в котором содержится функция main. Вначале создадим и сохраним его с помощью команд меню File, а затем добавим его в группу Users. В результате получим проект со следующей структурой (рис. 2.7):

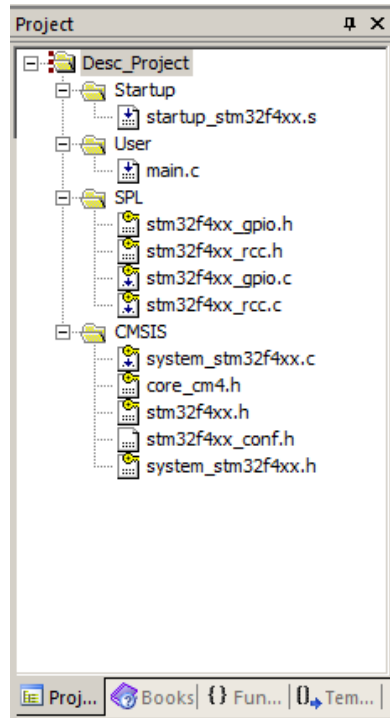


Рис. 2.7. Созданный проект

Осталось сделать еще некоторые настройки, которые нужны для компиляции кода и выполнения отладки. Настройки производятся в окне настроек, которое открывается через команду Project/Options for target ... или кнопкой на панели инструментов, или, используя комбинацию клавишей Alt-F7. В открывшемся окне на вкладке C/C++ зададим определения USE_STDPERIPH_DRIVER, STM32F4XX в текстовом поле Define. Здесь же можно задать пути к директориям с заголовочными файлами.

Настройки отладки по умолчанию позволяют проводить отладку в режиме симуляции, однако, если устройство подключено к компьютеру, то лучше производить отладку на самом устройстве. Для этого необходимо на вкладке Debug для пункта Use указать ST-Link Debugger. После этого следует нажать кнопку Settings. На вкладке Debug значение Port установить как SW. Затем перейти к вкладке Trace Download и добавить в список Programming Algorithm устройство STM32F4xx как показано на рис. 2.8.

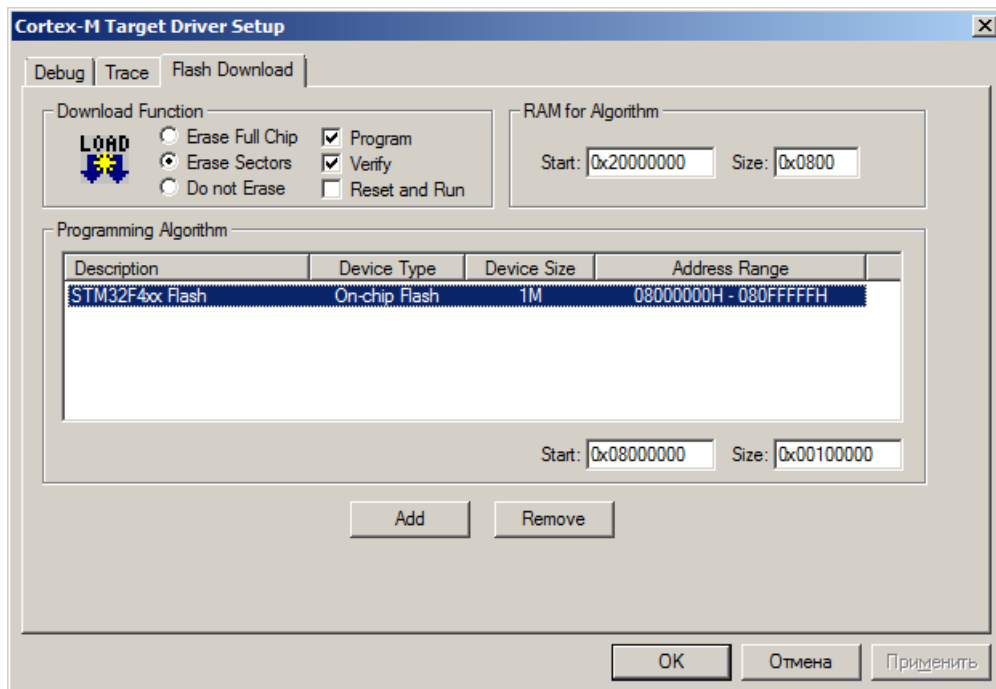


Рис. 2.8. Добавление в список Programming Algorithm устройства STM32F4xx

На вкладке Utilities выбрать вариант Use Target Driver for Flash Programming и также установить значение ST-Link Debugger. Проверить, что в окне после нажатия кнопки Settings значения совпадают с заданными для отладки.

После проведения данных действий можно приступать к написанию программы и проведению отладки на демонстрационной плате.

Однако перед этим следует подключить плату к ПК.

Для работы с платой необходимо, чтобы прошла установка драйверов. Обычно, при установке средств разработки, например, Keil, происходит также установка сопутствующего программного обеспечения (если этого не произошло, то следует найти в директории установки необходимые программы и установить их самостоятельно), поэтому рекомендуется производить указанные ниже действия уже после установки одной из средств разработки. Если же вариант с установкой среды разработки не подходит, то следует загрузить и установить программу STM32 ST-Link Utility, которая также рассматривается в методических указаниях.

Процесс установки драйверов при подключении устройства рассмотрим на примере операционной системы Windows 7.

При подключении устройства сразу же начинается установка драйверов для нового устройства. В области системного трее появляется уведомление следующего вида (рис. 2.8):

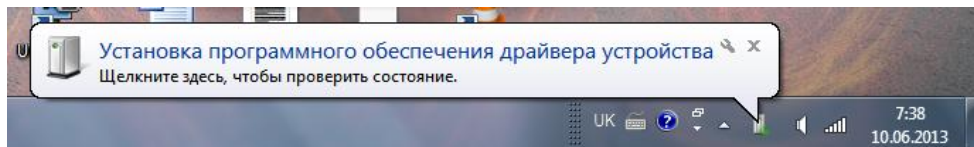


Рис. 2.8. Уведомление при подключении устройства

Нажав на уведомлении левой кнопкой мышки, откроем окно, которое отображает динамику установки драйверов (рис. 2.9).

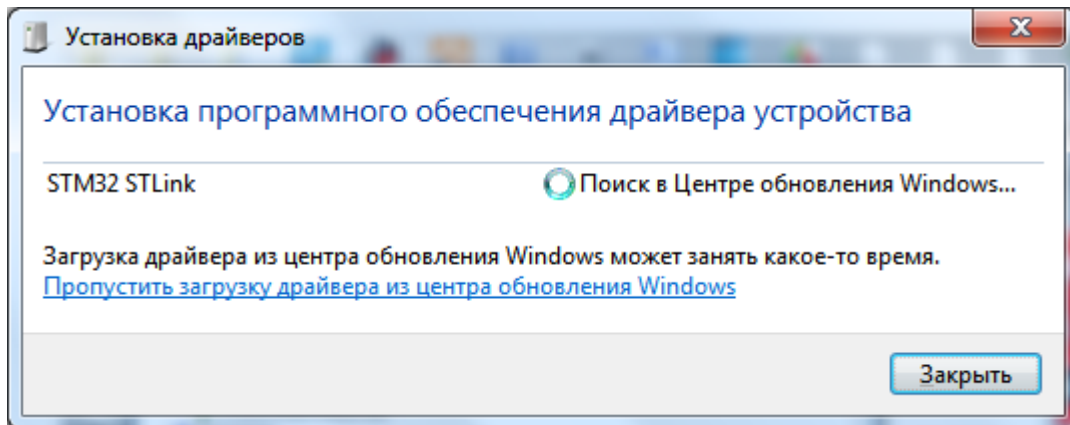


Рис. 2.9. Динамика установки драйвера

В результате успешной установки наблюдаем сообщение следующего вида, которое говорит нам о том, что имя подключенного устройства – STMicroelectronics STLink Dongle (рис. 2.10).

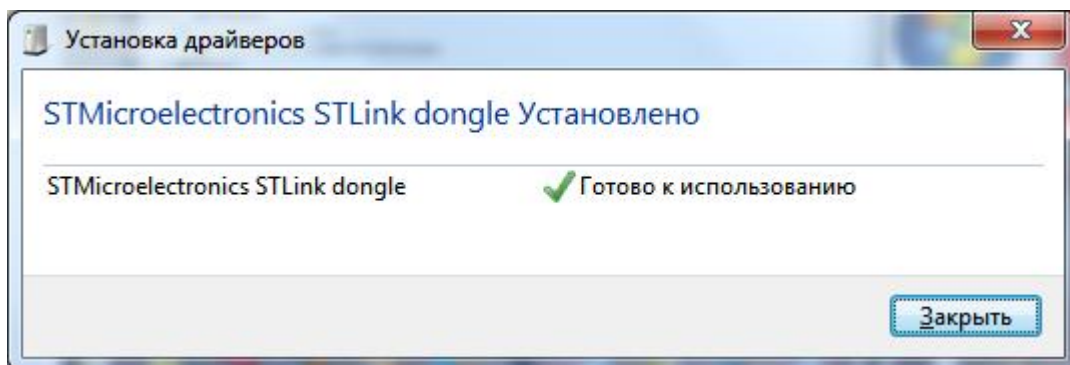


Рис. 2.10. Сообщение при успешной установке

В Диспетчере устройств (Пуск\Компьютер\Свойства\Диспетчер устройств) можем найти подключенное устройство (рис. 2.11).

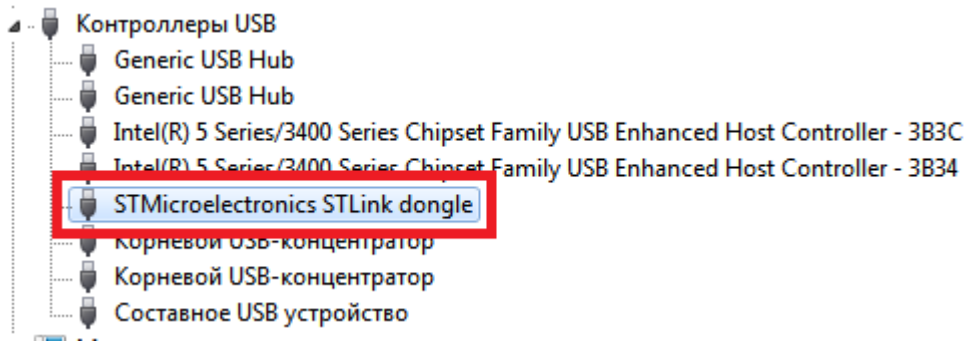


Рис. 2.11. Поиск подключенного устройства в диспетчере устройств

Теперь устройство готово для взаимодействия с программным обеспечением, установленным на ПК.

Дополнительное программное обеспечение

Графические средства настройки микроконтроллеров STM32

Производитель микроконтроллеров STMicroelectronics предлагает программу с возможностью графической настройки микроконтроллеров серии STM32 – MicroXplorer. Она позволяет получить исходный код инициализации периферийных устройств и соответствующих выводов и отображает результат настройки (занятые выводы, режим работы и др.). Тем не менее, с ее помощью нельзя настроить работу с прерываниями для конкретного устройства, провести инициализацию устройств, входящих в состав контроллера, поэтому данный инструмент больше подходит для наглядности при разработке, а не как полноценная альтернатива написания кода инициализации.

Программа представляет собой плагин к среде разработки Eclipse. Соответственно, предварительно необходимо установить Eclipse, а затем добавить к ней плагин.

Eclipse можно скачать с сайта <http://eclipse.org>. При этом загружается архив, который необходимо распаковать и создать нужные ярлыки для запуска. Для использования плагина подходят последние версии программного продукта.

Сам плагин можно доступен для загрузки на сайте производителя <http://www.st.com/web/en/catalog/tools/PF257931> (поскольку адрес может измениться, то, возможно, следует воспользоваться поиском по сайту). В результате также загружается архив.

Установка расширения производится из среды разработки. Для этого выбираем команду меню *Help\Install New Software*. В открывшемся окне нажимаем кнопку *Add* и в диалоговом окне, нажав кнопку *Archive*, указываем путь к расширению. Называем полученное расположение и переходим к процессу установки. Отмечаем пункт *MicroXplorer*, который появился в списке ниже. Далее соглашаемся с условиями лицензионного соглашения и во время установки подтверждаем запросы, которые появляются от диалоговых окон. После окончания процесса среда предложит перезагрузить среду разработки для того, чтобы применить изменения и загрузить расширение.

После перезагрузки в главном меню можно увидеть новый пункт – *MicroXplorer* (рис. 2.12):

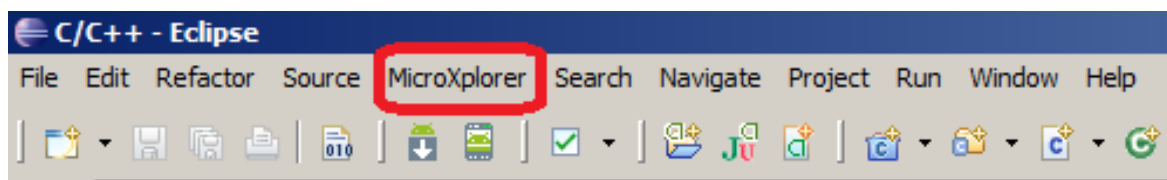


Рис. 2.12. Новый пункт меню

С помощью него можно на выбор открыть расширение в отдельном представлении Eclipse или установить функциональность *MicroXplorer* доступной во всем расположении (perspective) Eclipse.

Рассмотрим работу в режиме отдельного представления. Выполним команду *MicroXplorer\Open MicroXplorer As New View*. В результате среди прочих откроется еще одно окно, которое можно развернуть на всю ширину рабочей области дважды щелкнув по заголовку. В результате окно приложения выглядит следующим образом (рис. 2.13).

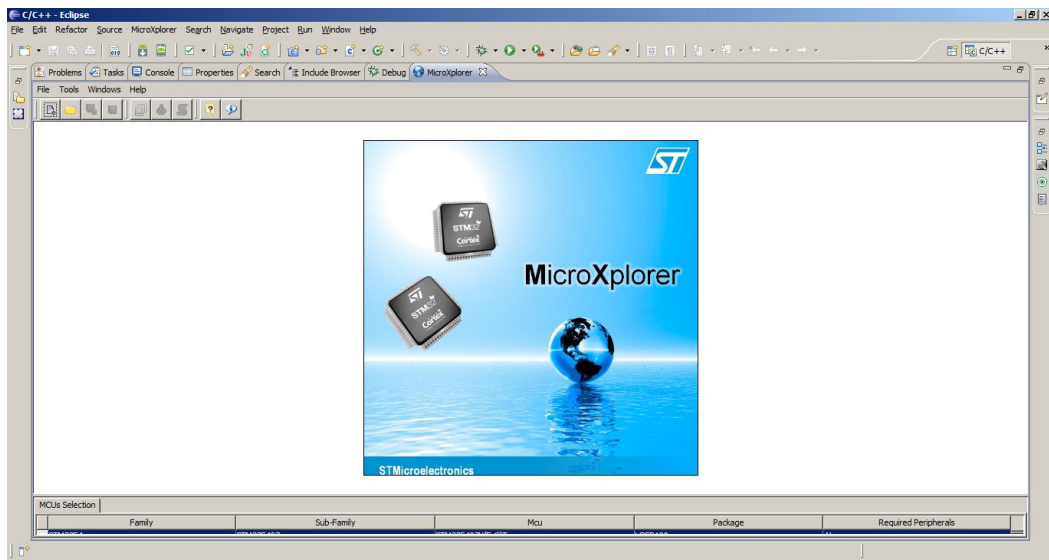


Рис. 2.13. Окно приложения в Eclipse

Для начала необходимо задать, какое именно устройство используется, выполнив команду *Tools\MCUs Selector*. После этого откроется диалоговое окно (рис. 2.14), в котором в списке указываем интересующий нас микроконтроллер и нажимаем кнопку ОК.

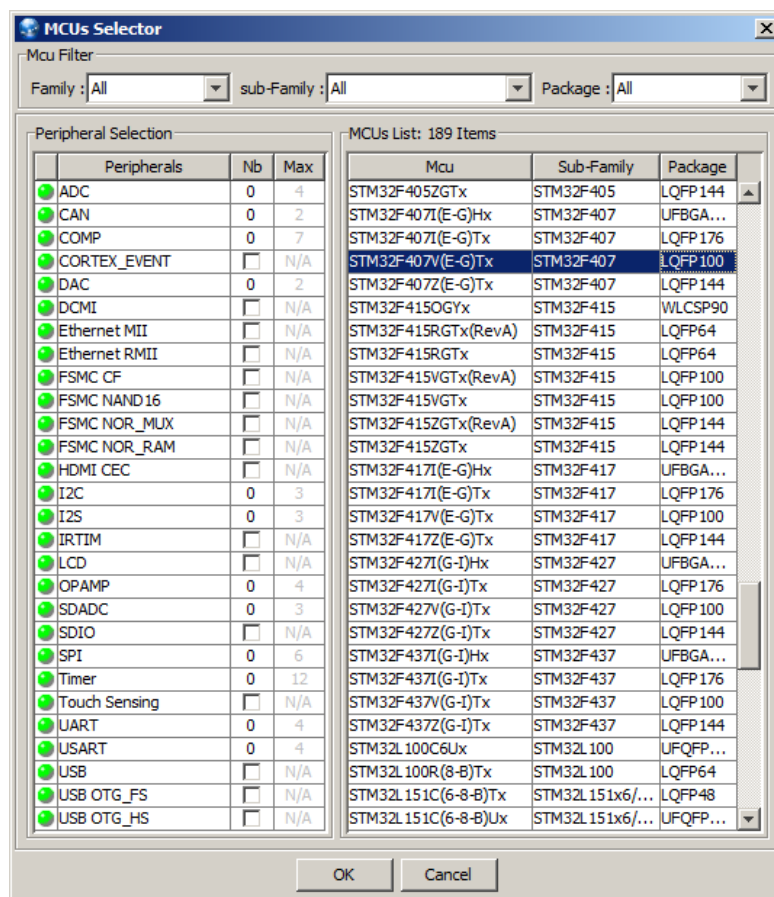


Рис. 2.14. Окно выбора микроконтроллера

После этого в рабочей области появится панель с устройствами, входящими в состав контроллера, а также его условное изображение, на котором обозначены все основные выводы (рис. 2.15). Сама рабочая область представлена тремя вкладками: Pinout, Configuration, Power Consumption Calculator. По мере того, как происходит настройка и включение соответствующих устройств, выводы, которые будут задействованы, выделяются зеленым цветом. В то же время, некоторые из устройств в панели помечаются желтыми или

красными иконками. Это происходит из-за того, что устройства используют одинаковые выводы в своей работе, поэтому может происходить частичный (использование устройства все еще возможно) либо полный конфликт (задействовать устройство не удастся), о чем и сообщают соответствующие иконки. Таким образом, можно выявить и избежать подобных конфликтов уже на стадии проектирования, что значительно упрощает дальнейшую разработку.

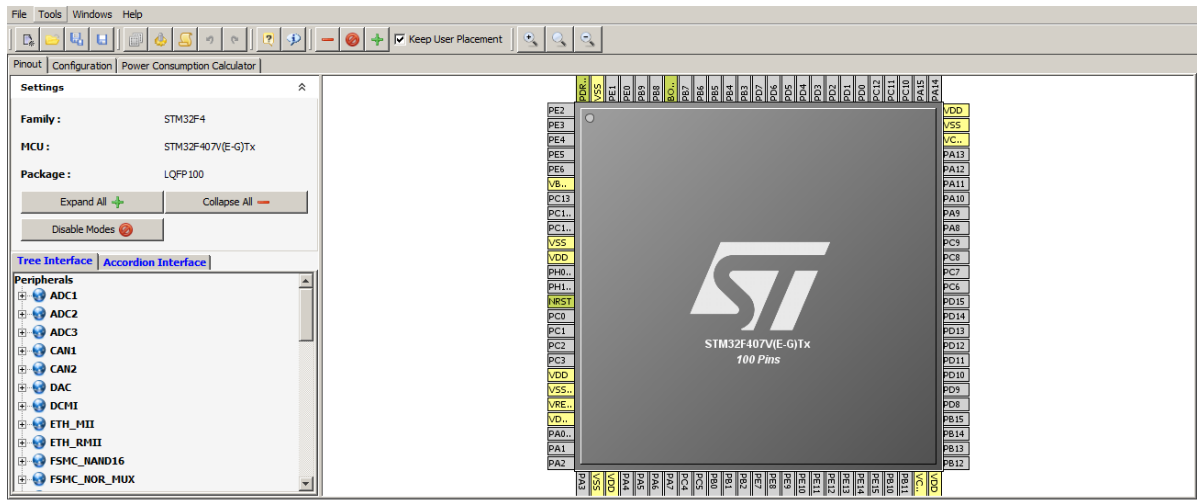


Рис. 2.15. Условное графическое обозначение контроллера

Рассмотрим пример настройки первого АЦП (ADC1) в визуальном редакторе. Предположим, что необходимо принимать сигнал с нулевого входа. Эту настройку задаем, развернув узел ADC1 и установив поле выбора IN0 (рис. 2.16). Вывод PA0 теперь должен быть выделен зеленым, а возле него появится подпись ADC1_IN0. Кроме того, переместившись в панели устройств к другим АЦП (ADC2 и ADC3), можно увидеть, что они обозначены желтыми иконками. Это связано с тем, что нулевой вход всех АЦП одинаков, поэтому одновременно несколькими устройствами он использоваться не может. Тем не менее, остальные АЦП все еще могут использоваться, но измерения проводить не с нулевого входа.

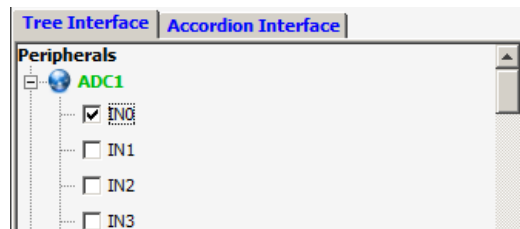


Рис. 2.16. Установка входа АЦП

После этого перейдем на вкладку Configuration (рис. 2.17). Данная вкладка предназначена для настроек портов ввода/вывода в соответствии с выбранной периферией контроллера.

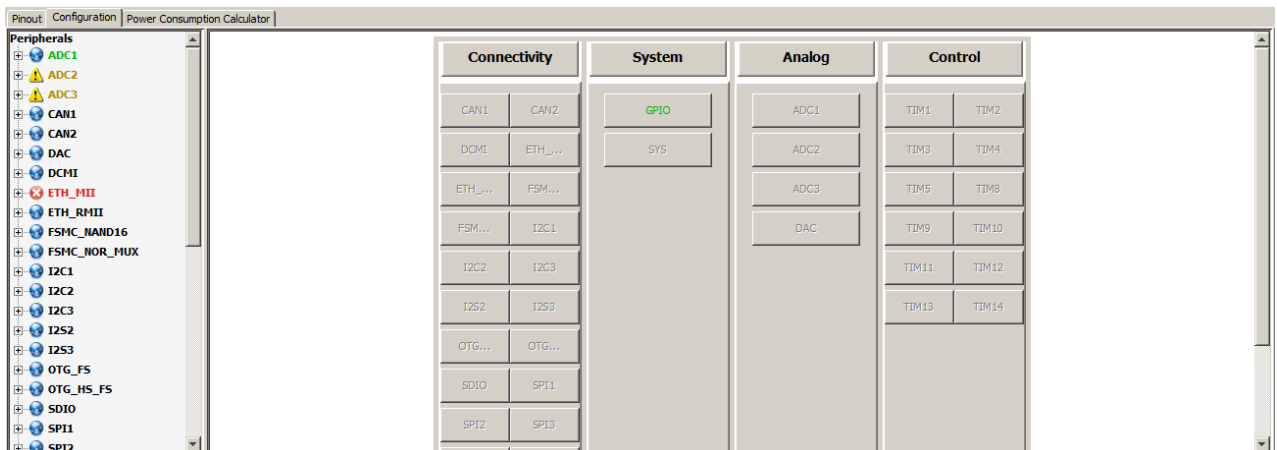


Рис. 2.17. Вкладка Configuration

После нажатия на кнопку GPIO, откроется окно Pin Configuration (рис. 2.18). Именно в нем и производится настройка работы выводов.

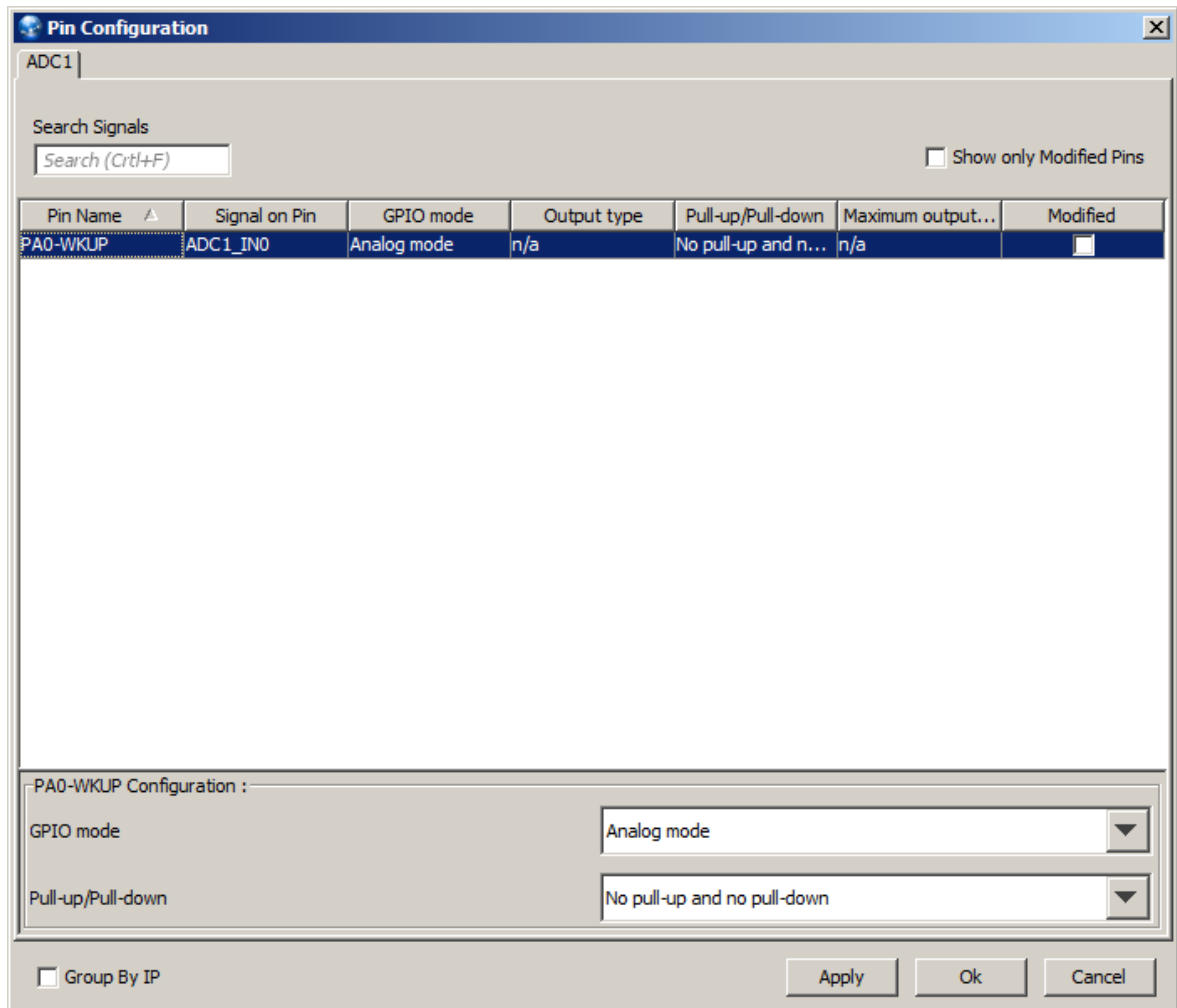


Рис. 2.18. Окно настройки выводов

В нашем случае сгенерированная настройка не требует изменений, однако, при использовании других устройств, параметры по умолчанию следует изменять для их корректной работы. При этом на каждое выбранное устройство появляется своя вкладка в окне, что позволяет сразу же произвести полную настройку.

Завершающим этапом работы с дополнением является генерация программного кода инициализации. Она производится командой меню *Tools\Generate Code...*. В результате создаются папки с файлами исходных кодов и заголовочных файлов. Как говорилось ранее, код, генерируемый программой нужно дорабатывать, так как многие важные настройки не генерируются программой. Лучше всего скопировать код, который выполняет инициализацию портов в свою программу, а остальное настраивать самостоятельно

Среда разработки CoIDE

В качестве альтернативы для начинающих опишем также среду разработки CoIDE, которая намного проще в использовании по сравнению с другими программными продуктами, поэтому в ее сторону следует обратить внимание абсолютным новичкам.

Для разработки программного обеспечения под 32-разрядные процессоры ARM (в том числе и для рассматриваемой платы) рекомендуются следующие программные продукты:

- Atollic TrueSTUDIO;
- IAR Embedded Workbench;
- Keil μ Vision с инструментами MDK-ARM;
- Altium TASKING VX-Toolset.

Основной отличительной чертой всех перечисленных сред есть то, что они являются коммерческими. Бесплатное их использование возможно лишь с ограничениями по размеру бинарного кода программы или по срокам использования.

Наиболее известные бесплатные альтернативы – CoCo IDE и использование Eclipse вместе с плагином для разработки для 32-разрядных процессоров. Второй вариант требует множество дополнительных настроек и достаточно сложен для тех, кто только начинает разработку под МК.

Среда разработки CoCo IDE 1.7 – бесплатный инструмент, который ориентирован на разработку для 32-разрядных процессоров ARM разных производителей: Atmel, Energy Micro, Freescale, Holtek, NXP, Nuvoton, TI. Она построена на основе Eclipse, однако не имеет таких же широких возможностей для расширения. Несмотря на это, данная среда требует минимальных настроек для начала программирования и отладки, поэтому представляется идеальным вариантом для начала работы с платой STM32F4 Discovery.

Для начала разработки, кроме самой CoIDE, потребуется установка средств для построения проекта GNU Toolchain for ARM Embedded Processors.

Разработка с использованием CoIDE связана с концепцией репозитория: все доступные компоненты и библиотеки устанавливаются с помощью мастера, исходя из того, для какого именно процессора создан проект. Для загрузки библиотек необходимо подключение к сети Internet.

Работа с проектом в среде CoIDE рассмотрена далее в лабораторных работах.

Средства настройки частоты работы микроконтроллера

По умолчанию рабочая частота микроконтроллера на отладочной плате составляет 16 МГц. Данную частоту обеспечивает внутренний высокочастотный генератор тактовых импульсов (High Speed Internal Oscillator – HSI). Однако максимальная рабочая частота составляет 168 МГц, что значительно больше значения по умолчанию. Для того, чтобы настроить работу устройства на нужной частоте производитель предлагает воспользоваться специально разработанным программным обеспечением – Clock Configuration Tool. Оно представляет собой xls-файл с макросами, поэтому для его использования следует задать разрешение выполнения макросов в Excel. Интерфейс программы (рис. 2.19) организован на одном листе с использованием полей ввода и выбора значений и наглядно отображает схему тактирования, которая используется в устройстве.

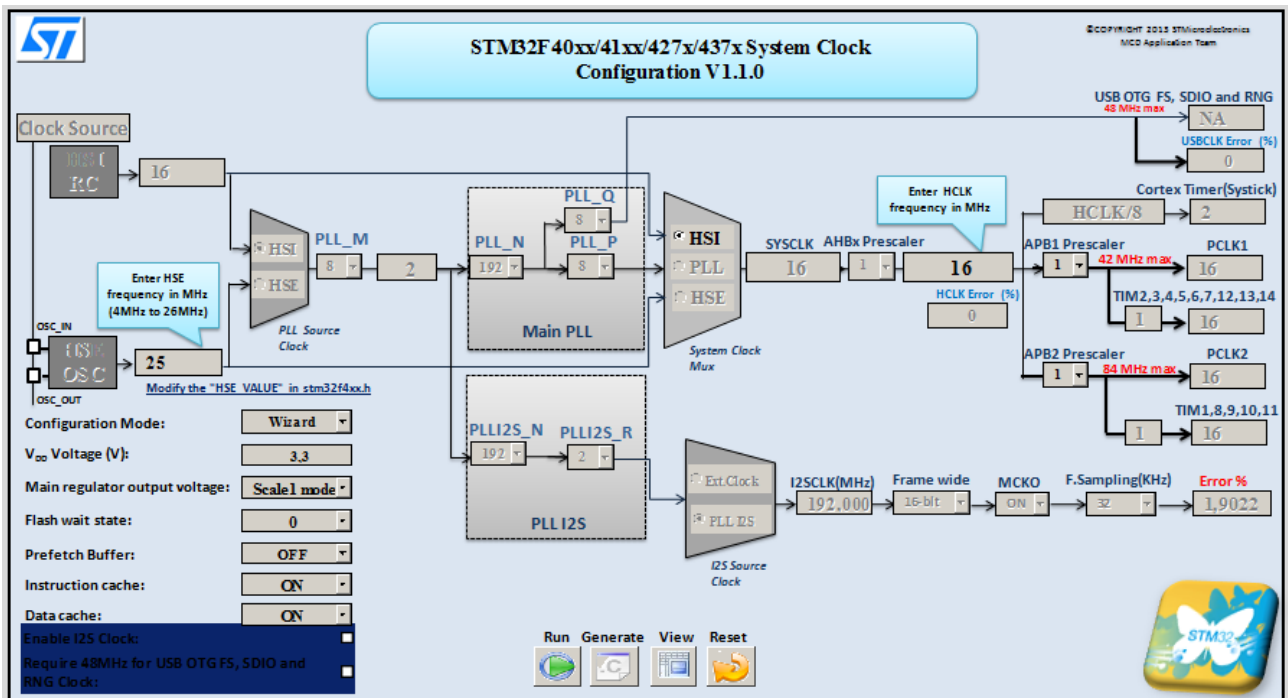


Рис. 2.19. Интерфейс программы для настройки системы тактирования

В нижней части схемы находится кнопки, с помощью которых запускаются макросы, выполняющие основную работу. При нажатии на кнопку *Generate* в папке с файлом генерируется новый файл конфигурации с именем *system_stm32f4xx.c*. Его можно подключить к проекту в среде разработки, заменив существующий файл. Для того, чтобы применить настройки следует отследить, вызывается ли функция *SystemInit()*, поскольку без ее вызова устройство будет работать на частоте по умолчанию. В случае отсутствия вызова функции ее следует вызвать самостоятельно, например, в основной функции *main*. После этого ваше устройство будет работать на той частоте, которая задана, поэтому, возможно, следует изменить значения параметров работы устройств для поддержания корректной работы в дальнейшем.

Программные средства для прошивки платы

Несмотря на то, что среды разработки интегрируют в себе функциональность по прошивке микроконтроллеров, полезно будет узнать, что существуют и отдельные программы, которые предназначены специально для работы с памятью (программирование, очистка, проверка) устройств. Кроме того, при работе с режимами пониженного энергопотребления именно эти программы позволяют выполнить программирование независимо от текущего режима работы, с чем могут возникнуть проблемы при выполнении данных действий из среды разработки.

В данном разделе рассмотрим два программных продукта, которые реализуют функциональность для выполнения прошивки без использования среды разработки: STM32 ST-LINK Utility и ST Visual Programmer.

Программа STM32 ST-LINK Utility предназначена для работы с 32-разрядными контроллерами через интерфейс ST-LINK (в том числе и с его второй версией). Ее интерфейс (рис. 2.20) организован максимально просто и понятно. Основная рабочая область организована в виде двух вкладок, в которых отображается память устройства, а также представлен двоичный файл для записи. Наиболее важные команды сосредоточены в меню Target. Если в момент открытия программы устройство не было подключено к ПК, то выполнение подключения следует командой *Target\Connect*. После этого в панели сообщений должно появиться сообщение об успешном подключении и информация о подключенном устройстве. Выполнить программирование устройства можно командой *Target\Program....* При этом следует заранее открыть файл с программой. Стирание памяти программ устройства производится командой *Target\Erase Chip*.

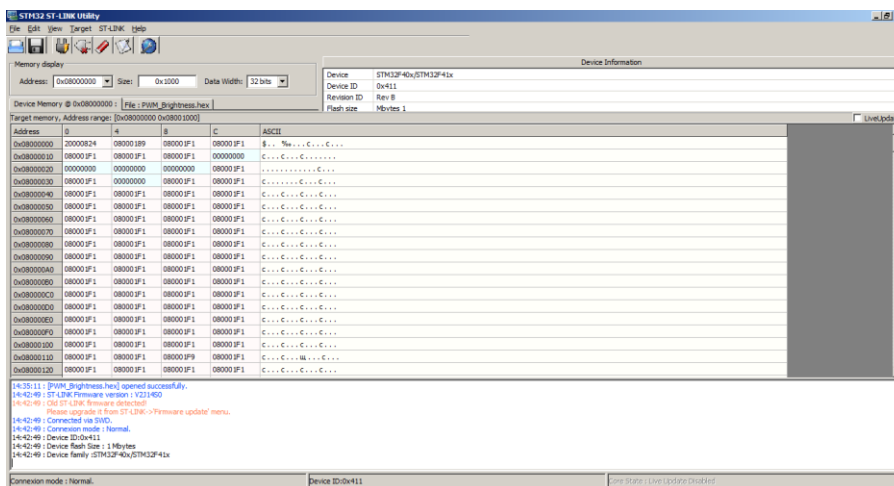


Рис. 2.20. Окно программы STM32 ST-LINK Utility

Программа ST Visual Programmer (STVP) является универсальным средством для прошивки микроконтроллеров производства STMicroelectronics. Она может работать с устройствами семейств STM32, STM8, а также STM7 и с большим количеством интерфейсов, что также является преимуществом. В остальном же STVP и STM32 ST-LINK Utility очень похожи между собой как по интерфейсу (рис. 2.21), так и по функциональности.

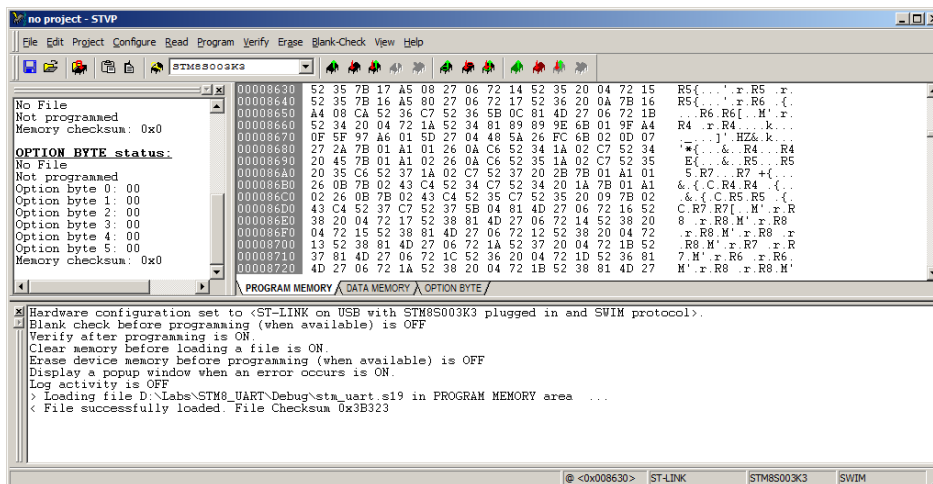


Рис. 2.21. Окно программы STVP

Использование STVP во многом аналогично использованию, кроме того, что необходимо самостоятельно задать настройки подключения. Эти действия выполняются в диалоговом окне (рис. 2.22), которое открывается командой. Configure\Configure ST Visual Programmer. В нем указывается аппаратное обеспечение, которое используется для подключения и устройство.

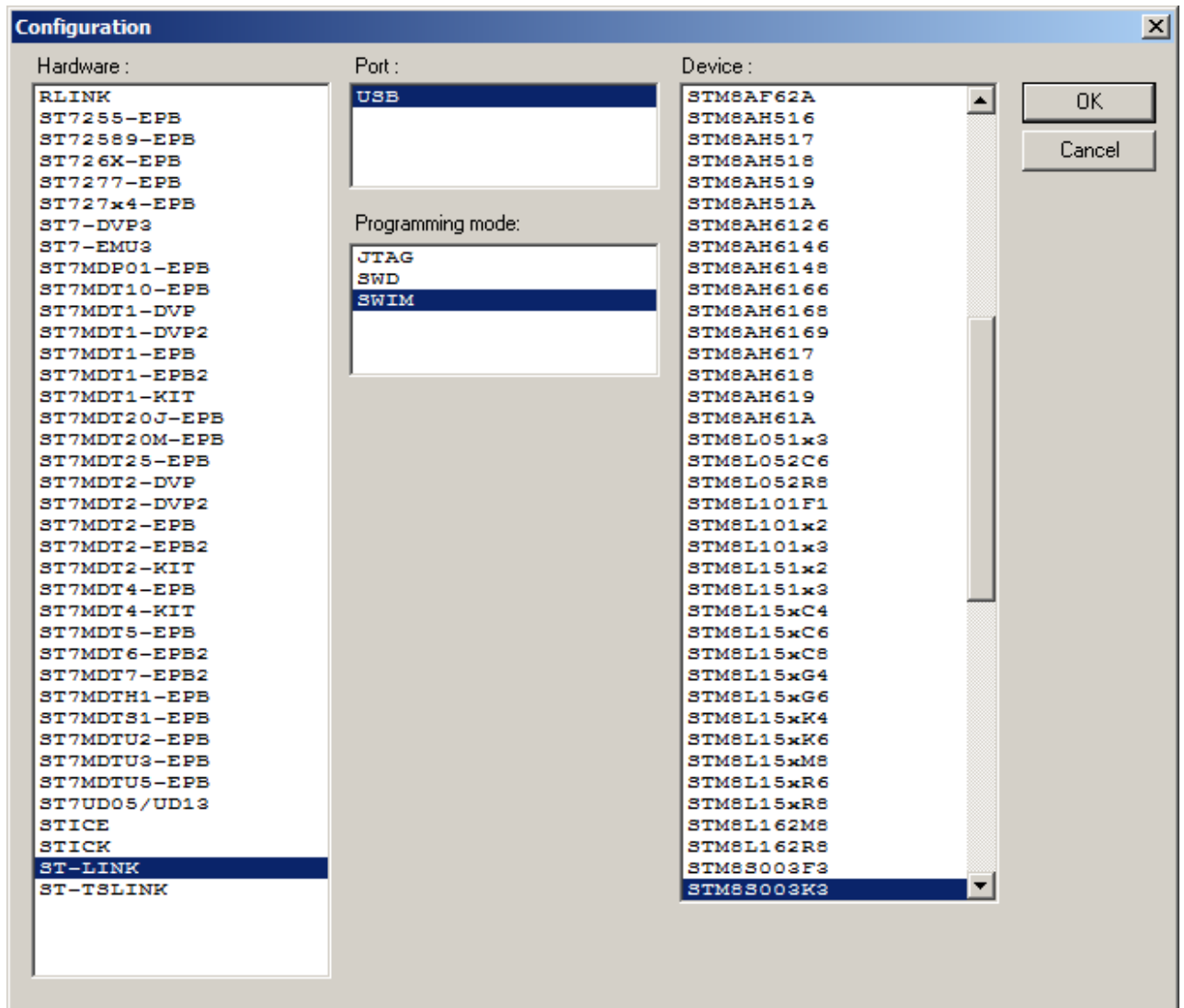


Рис. 2.22. Окно настройки программирования устройства

Работа с отладочной платой STM32F4 Discovery с использованием описанных инструментов происходит по схожему сценарию, поэтому предпочтение в их использовании зависит от пользователя. Автоматическое подключение при запуске облегчит работу новичков, для тех, кто работает с различными устройствами можно рекомендовать STVP.

III. ЛАБОРАТОРНЫЕ РАБОТЫ

Лабораторная работа № 1: Создание проекта в среде разработки. Использование портов ввода/вывода

Цель работы: ознакомиться с созданием проектов для платы, рассмотреть их структуру; изучить принципы работы с портами ввода/вывода и организации их взаимодействия

Оборудование и программное обеспечение: плата STM32F4 Discovery, среда разработки CooCox CoIDE 1.7, инструменты построения проекта GNU Toolchain for ARM Embedded Processors, библиотека CMSIS при необходимости самостоятельного подключения файлов к проекту.

Теоретический материал

Контроллер STM32F407VG содержит пять 16-разрядных портов ввода/вывода общего назначения, которые обозначены как GPIOx, где x может иметь значения A, B, C, D, E. Каждый порт GPIO имеет четыре 32-битных регистра конфигурации (GPIOx_MODER, GPIOx_TYPER, GPIOx_SPEEDR, GPIOx_ORD), два 32-битных регистра данных (GPIOx_ODR, GPIOx_IDR) и два 32-битных регистра выбора дополнительных функций (GPIOx_AFRH и GPIOx_AFRL).

Светодиоды, предназначенные для программирования, на плате подключены к порту D (рис. 3.1).

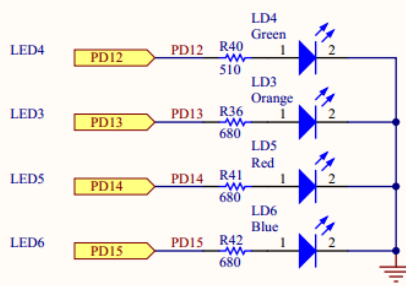


Рис. 3.1. Схема подключения светодиодов к порту на плате

Остальные четыре светодиода выполняют служебные функции индикации и в программировании определенных действий не используются.

Создание проекта и его структура

Для выполнения лабораторных работ рекомендуется установить среду разработки CooCox CoIDE 1.7. Создание проекта выполняется с помощью мастера, который открывается при выполнении команды меню *Project/New Project*. В мастере следует пошагово указать имя проекта и его расположение, производителя и МК, для которого предназначена программа. Далее работа с проектом осуществляется с помощью окна *Repository*, которая позволяет добавить необходимые библиотеки управления периферийными частями МК, а также окна *Project*. Окно *Repository* также представляет собой мастер, содержащий несколько вкладок, основной из которых является вкладка *Peripherals* (рис. 3.2).

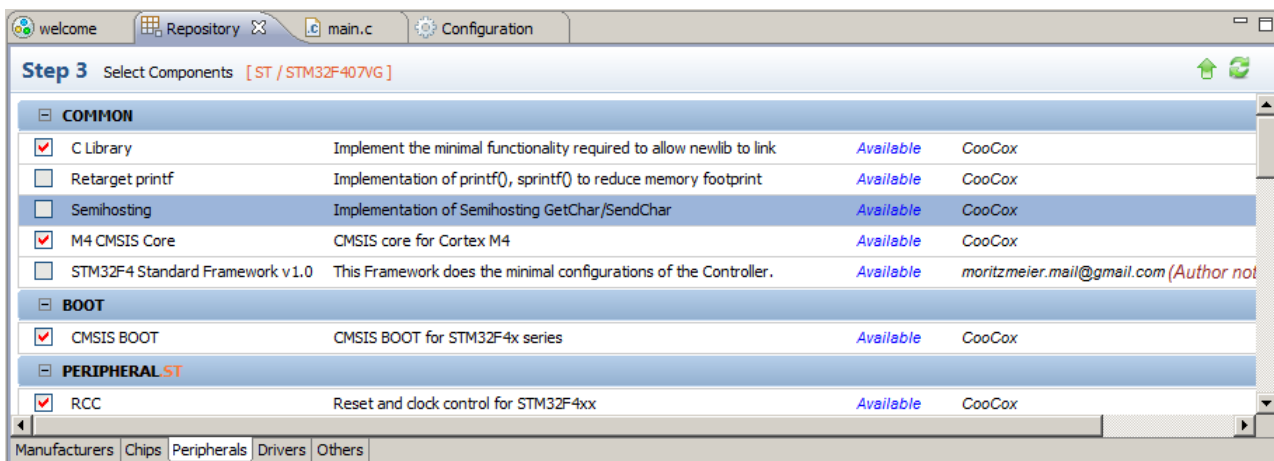


Рис. 3.2. Окно *Repository*

Проект программы для платы STM32F4 Discovery имеет схожую структуру для всех средств разработки. Обычно он включает в себя два внутренних каталога: один – для файлов из библиотеки CMSIS, а другой – для файлов, предназначенных для работы с периферией. Для примера представлены типичные структуры проектов в средах разработки CoIDE и Keil (рис. 3.3).

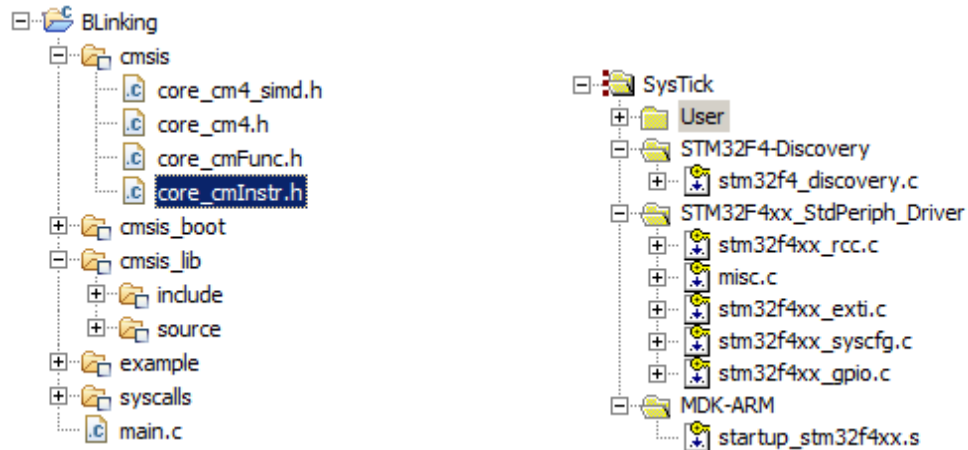


Рис. 3.3. Структуры проектов в разных средах разработки

Пример программы

Рассмотрим пример программы, которая демонстрирует работу с портами ввода/вывода. Она позволяет переключать состояния светодиода при нажатии пользовательской кнопки, которая находится на плате.

```
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_gpio.h"

void Delay(uint32_t nCount)
{
    while(nCount--)
    {
    }
}

int main(void)
{
    GPIO_InitTypeDef gpioConf;
    // инициализация входа, подключенного к кнопке
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
    gpioConf.GPIO_Pin = GPIO_Pin_0;
    gpioConf.GPIO_Mode = GPIO_Mode_IN;
    GPIO_Init(GPIOA, &gpioConf);

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
    // инициализация входа, подключенного к светодиоду
    gpioConf.GPIO_Pin = GPIO_Pin_13;
    gpioConf.GPIO_Mode = GPIO_Mode_OUT;
    gpioConf.GPIO_Speed = GPIO_Speed_100MHz;
    gpioConf.GPIO_OType = GPIO_OType_PP;
    gpioConf.GPIO_PuPd = GPIO_PuPd_NOPULL;

    GPIO_Init(GPIOD, &gpioConf);

    while(1) {
        if(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0) == 0) {
            if (GPIO_ReadOutputDataBit(GPIOD, GPIO_Pin_13))
                GPIO_ResetBits(GPIOD, GPIO_Pin_13);
        }
    }
}
```

```

else
    GPIO_SetBits(GPIOD, GPIO_Pin_13);
Delay(5000);
while(GPIO_ReadInputDataBit(GPIOA, GPIO_Pin_0)==0);
Delay(5000);
}
}
}
}
}

```

Ход работы

1. На основе кода примера приложения создать свой проект в среде разработки и проверить его работоспособность.
2. Ознакомиться с работой используемых функций, просмотрев исходный код, а также комментарии в исходных файлах библиотеки и в режиме отладки.
3. Создать новый проект в среде разработки для выполнения индивидуального задания, полученного от преподавателя.
4. Реализовать необходимую функциональность.
5. Запрограммировать плату и продемонстрировать работу программы.

Индивидуальные задания

В соответствии с вариантом по табл. 3.1 реализовать схему включения/выключения светодиодов, расположенных на плате:

Таблица 3.1. Варианты включения светодиодов

Вариант	1-ый светодиод	2-ой светодиод	3-ий светодиод	4-ый светодиод
1	1	2	3	4
2	2	1	3	4
3	2	3	1	4
4	2	3	4	1
5	2	4	3	1
6	4	2	1	3
7	3	4	2	1
8	4	3	1	2
9	4	1	3	2
10	1	4	2	3

Номера включения светодиодов присваиваются по их номерам в составе порта D (можно найти в документации).

Лабораторная работа № 2: Прерывания и их использование. Использование таймеров

Цель работы: ознакомиться с понятием прерывания, возможностями, которые они предоставляют; научиться использовать таймеры для выполнения действий в определенные временные интервалы.

Оборудование и программное обеспечение: плата STM32F4 Discovery, среда разработки Coocox CoIDE 1.7, инструменты построения проекта GNU Toolchain for ARM Embedded Processors, библиотека CMSIS при необходимости самостоятельного подключения файлов к проекту.

Теоретический материал

Прерывания – механизм, который позволяет аппаратному обеспечению сообщать о наступлении важных событий в своей работе. В момент, когда происходит прерывание, процессор переключается с выполнения основной программы на выполнение соответствующего обработчика прерываний. Как только выполнение обработчика завершено, продолжается выполнение основной программы с места, в котором она была прервана.

Для использования прерываний необходимо вначале настроить регистр, который называется Nested Vector Interrupt Controller (NVIC), вложенный контроллер вектора прерываний. Данный регистр является стандартной частью архитектуры ARM и встречается на всех процессорах, независимо от производителя. NVIC разработан таким образом, что задержка прерывания минимальна. NVIC поддерживает вложенные прерывания с 16-ю уровнями приоритета.

Микроконтроллер STM32F407VG содержит 14 таймеров. В общем виде схема управления подсчетом импульсов может быть представлена следующим образом (рис. 3.4):

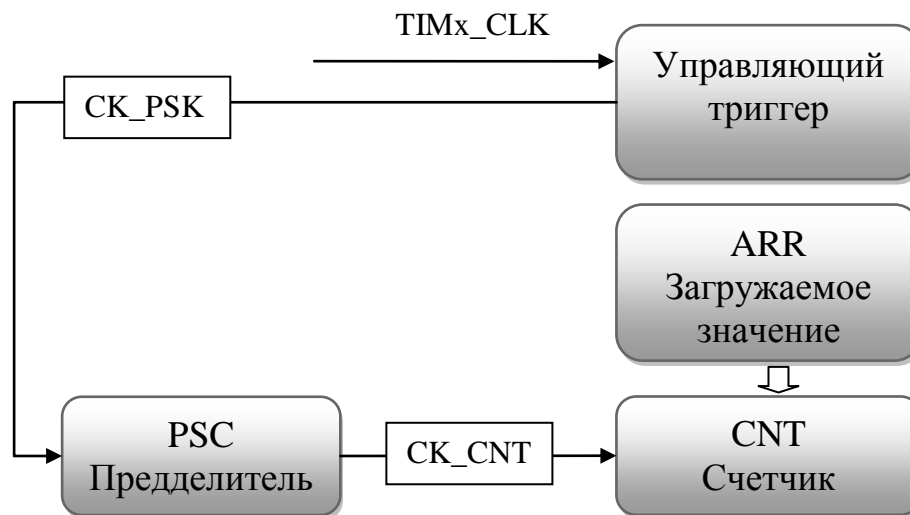


Рис. 3.4. Схема управления подсчетом импульсов

Производитель разделяет все таймеры на три типа:

- 1) с расширенными возможностями;
- 2) общего назначения;
- 3) базовые.

Каждый таймер может иметь до 4 линий захвата/сравнения (именно они используются в режиме генерации ШИМ).

Пример программы

Данная программа демонстрирует работу с прерываниями и с таймерами. В ней реализовано переключение светодиода, который подключен к порту ввода/вывода, через определенные интервалы времени.

```

#include «stm32f4xx.h»
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_tim.h"
#include "misc.h"
    
```



```

void INTTIM_Config(void);
void GPIO_Config(void);

int main(void)
{
    GPIO_Config();
    INTTIM_Config();

    while(1)
    {
    }
}

void TIM2_IRQHandler(void) {
    if (TIM_GetITStatus(TIM2, TIM_IT_Update) != RESET) {
        TIM_ClearITPendingBit(TIM2, TIM_IT_Update);
        GPIOD->ODR ^= GPIO_Pin_13;
    }
}

void GPIO_Config(void) {
    GPIO_InitTypeDef gpio_struct;
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
    gpio_struct.GPIO_Pin = GPIO_Pin_13;
    gpio_struct.GPIO_Mode = GPIO_Mode_OUT;
    gpio_struct.GPIO_OType = GPIO_OType_PP;
    gpio_struct.GPIO_PuPd = GPIO_PuPd_NOPULL;
    gpio_struct.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_Init(GPIOD, &gpio_struct);
}

void INTTIM_Config(void)
{
    NVIC_InitTypeDef nvic_struct;
    nvic_struct.NVIC_IRQChannel = TIM2_IRQn;
    nvic_struct.NVIC_IRQChannelPreemptionPriority = 0;
    nvic_struct.NVIC_IRQChannelSubPriority = 1;
    nvic_struct.NVIC_IRQChannelCmd = ENABLE;

    NVIC_Init(&nvic_struct);

    TIM_TimeBaseInitTypeDef tim_struct;
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
    tim_struct.TIM_Period = 10000 - 1;
    tim_struct.TIM_Prescaler = 168 - 1;
    tim_struct.TIM_ClockDivision = 0;
    tim_struct.TIM_CounterMode = TIM_CounterMode_Up;
    TIM_TimeBaseInit(TIM2, &tim_struct);
    TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE);
    TIM_Cmd(TIM2, ENABLE);
}

```

Обращает на себя внимание включение дополнительного заголовочного файла – misc.h. Именно в нем описаны функции, перечисления, структуры для работы с прерываниями.

Ход работы

1. На основе кода примера приложения создать свой проект в среде разработки и проверить его работоспособность.
2. Ознакомиться с работой используемых функций, просмотрев исходный код, а также комментарии в исходных файлах библиотеки и в режиме отладки.

3. Создать новый проект в среде разработки для выполнения индивидуального задания, полученного от преподавателя.
4. Реализовать необходимую функциональность.
5. Запрограммировать плату и продемонстрировать работу программы.

Индивидуальные задания

1. С помощью одного из таймеров общего назначения сгенерировать задержку длительностью 1 с (на основе данных о рабочей частоте). Задержку генерировать на основе прерываний таймера. Продемонстрировать расчеты, подтверждающие правильность задания задержки.
2. Реализовать с помощью прерывания по переполнению таймера временную задержку с поочередным включением светодиодов на плате по кругу.

Лабораторная работа № 3: Генерация сигнала ШИМ

Цель работы: ознакомиться с возможностями генерации ШИМ с помощью демонстрационной платы; научиться генерировать сигнал с заданными параметрами и использовать его для управления внешними устройствами.

Оборудование и программное обеспечение: плата STM32F4 Discovery, среда разработки Coocox CoIDE 1.7, инструменты построения проекта GNU Toolchain for ARM Embedded Processors, библиотека CMSIS при необходимости самостоятельного подключения файлов к проекту.

Теоретический материал

Широтно-импульсная модуляция (ШИМ) – способ управления средним значением напряжения на нагрузке путем изменения скважности импульсов, управляемых ключом. В основном, микроконтроллеры позволяют генерировать цифровой ШИМ различной частоты.

Поскольку вывод сигнала ШИМ не является основной функцией пинов, которые подключены к светодиодам, то необходимо выполнить соответствующую их конфигурацию. Для этого следует задать выполнение пинами альтернативных функций. Генерация ШИМ в них связана с использованием дополнительных режимов таймера.

Перед подключением устройства известны такие параметры ШИМ, как частота и коэффициент заполнения. Для их расчета в контроллерах STM32 необходимо определить значение предделителя и автоматически загружаемое значение в регистре ARR (Auto-Reload Register). Расчет значения, которое следует записать в предделитель выполняется следующим образом:

$$PSC = \frac{TIMxCLK}{TIMxCNT} - 1, \quad (1)$$

где PSC – значение предделителя,

$TIMxCLK$ – входная частота работы таймера,

$TIMxCNT$ – частота счетчика.

Для получения необходимой выходной частоты следует записать значений в регистр ARR, которое получается из следующего соотношения:

$$ARR_VAL = \frac{TIMxCNT}{TIMx_out_freq} - 1, \quad (2)$$

где ARR_VAL – значение для записи в регистр ARR,

$TIMxCNT$ – частота счетчика,

$TIMx_out_freq$ – нужная выходная частота ШИМ.

Последним этапом является задание нужного коэффициента заполнения, что обеспечит нужное значение напряжения на выходе. Данная настройка производится с помощью регистра захвата/сравнения (capture/compare register, CCRx), исходя из следующего соотношения:

$$D = \frac{CCRx_VAL}{ARR_VAL} * 100\%, \quad (3)$$

где D – коэффициент заполнения,

$CCRx_VAL$ – значение в регистре CCRx (x – номер регистра для конкретной линии),

ARR_VAL – значение для записи в регистр ARR.

Особенностью данных микроконтроллеров является то, что в предделитель и другие регистры можно записать любое значение, которое можно описать с помощью отведенного количества разрядов.

Выдача сигнала ШИМ на выход не является основным режимом работы выводов порта, а относится к дополнительным (альтернативным) режимам. Поэтому предварительно нужно задать нужный режим в настройках порта.

Для подключения альтернативных функций к портам ввода/вывода необходимо:

1. Подключить пин к альтернативной функции соответствующего периферийного устройства с помощью функции GPIO_PinAFConfig.
2. Сконфигурировать пин в режим выполнения альтернативной функции – GPIO_Mode_AF.
3. Выполнить остальные настройки.
4. Вызвать GPIO_Init для применения указанных настроек.

Пример программы

Для ознакомления с возможностью генерации сигнала ШИМ и использования его для управления яркостью светодиодов представлена следующая программа.

В листинге представлено содержание файла с главной функцией.

```
#include "stm32f4xx.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
#include "misc.h"
#include "stm32f4xx_tim.h"
#include "init.h"

int main(void)
{
    init();

    int brightness = 0;
    int i;
    while(1)
    {
        brightness++;

        TIM4->CCR3 = 333 - (brightness + 0) % 333;
        TIM4->CCR4 = 333 - (brightness + 166 / 2) % 333;
        TIM4->CCR1 = 333 - (brightness + 333 / 2) % 333;
        TIM4->CCR2 = 333 - (brightness + 499 / 2) % 333;

        for(i=0;i < 10000; ++i);
            for(i=0;i < 10000; ++i);
                for(i=0;i < 10000; ++i);
    }
}
```

В основной функции происходит изменение значений регистра захвата/сравнения с определенной задержкой.

Функции с выполнением настроек перенесены в отдельный файл init.c. Соответственно, объявления функций находятся в файле init.h. В следующем листинге представлено содержимое файла init.c.

```
#include "init.h"
#include "stm32f4xx.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
#include "misc.h"
#include "stm32f4xx_tim.h"
```

```

void init() {
    GPIOInit();
    TimerInit();
}

void TimerInit() {
    TIM_TimeBaseInitTypeDef time_init;
    TIM_OCInitTypeDef oc_init;
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4, ENABLE);

    uint16_t PrescalerValue = (uint16_t)((SystemCoreClock / 2) / 21000000);

    time_init.TIM_Period = 665;
    time_init.TIM_Prescaler = PrescalerValue;
    time_init.TIM_ClockDivision = 0;
    time_init.TIM_CounterMode = TIM_CounterMode_Up;

    TIM_TimeBaseInit(TIM4, &time_init);

    oc_init.TIM_OCMode = TIM_OCMode_PWM1;
    oc_init.TIM_OutputState = TIM_OutputState_Enable;
    oc_init.TIM_Pulse = 0;
    oc_init.TIM_OCPolarity = TIM_OCPolarity_High;

    TIM_OC1Init(TIM4, &oc_init);
    TIM_OC1PreloadConfig(TIM4, TIM_OCPreload_Enable);

    oc_init.TIM_OutputState = TIM_OutputState_Enable;
    oc_init.TIM_Pulse = 0;

    TIM_OC2Init(TIM4, &oc_init);
    TIM_OC2PreloadConfig(TIM4, TIM_OCPreload_Enable);

    oc_init.TIM_OutputState = TIM_OutputState_Enable;
    oc_init.TIM_Pulse = 0;

    TIM_OC3Init(TIM4, &oc_init);
    TIM_OC3PreloadConfig(TIM4, TIM_OCPreload_Enable);

    oc_init.TIM_OutputState = TIM_OutputState_Enable;
    oc_init.TIM_Pulse = 0;

    TIM_OC4Init(TIM4, &oc_init);
    TIM_OC4PreloadConfig(TIM4, TIM_OCPreload_Enable);

    TIM_ARRPreloadConfig(TIM4, ENABLE);
    TIM_Cmd(TIM4, ENABLE);
}

void GPIOInit() {
    GPIO_InitTypeDef gpio_init;
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);

    gpio_init.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13 | GPIO_Pin_14 | GPIO_Pin_15;
    gpio_init.GPIO_Mode = GPIO_Mode_AF;
    gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
    gpio_init.GPIO_OType = GPIO_OType_PP;
    gpio_init.GPIO_PuPd = GPIO_PuPd_UP;
    GPIO_Init(GPIOD, &gpio_init);

    GPIO_PinAFConfig(GPIOD, GPIO_PinSource12, GPIO_AF_TIM4);
}

```

```
GPIO_PinAFConfig(GPIOD, GPIO_PinSource13, GPIO_AF_TIM4);
GPIO_PinAFConfig(GPIOD, GPIO_PinSource14, GPIO_AF_TIM4);
GPIO_PinAFConfig(GPIOD, GPIO_PinSource15, GPIO_AF_TIM4);
}
```

Как видно из примера, для инициализации таймера необходимо сразу две структуры: одна – для инициализации непосредственно таймера, а другая – для задания режима сравнения выхода. Приведенная программа позволяет последовательно уменьшать яркость свечения пользовательских светодиодов на плате.

Ход работы

1. На основе кода примера приложения создать свой проект в среде разработки и проверить его работоспособность.
2. Ознакомиться с работой используемых функций, просмотрев исходный код, а также комментарии в исходных файлах библиотеки и в режиме отладки.
3. Создать новый проект в среде разработки для выполнения индивидуального задания, полученного от преподавателя.
4. Реализовать необходимую функциональность.
5. Запрограммировать плату и продемонстрировать работу программы.

Лабораторная работа № 4: Использование АЦП

Цель работы: исследовать возможности использования АЦП

Оборудование и программное обеспечение: переключки для подключения источника напряжения (батарейки) ко входам АЦП

Теоретический материал

Микроконтроллер STM32F407VG включает в себя три АЦП. Разрядность всех АЦП составляет 12 бит. Каждый преобразователь способен принимать сигнал из шестнадцати внешних каналов.

Кроме того, в состав контроллера (не платы) входит датчик температуры. Диапазон входных напряжений составляет 1.8...3.6 В. Датчик температуры подключен к входному каналу ADC_IN16, который используется для того, чтобы преобразовать выходное напряжение сенсора в цифровое значение.

Внутренний датчик температуры предназначен для отслеживания изменения температуры, а не для ее измерения, поскольку смещение показателей датчика может меняться в ходе изменений параметров процесса. Поэтому, если необходимо точное измерение абсолютных значений температуры, то для этого лучше использовать внешний датчик.

Пример программы

В данном примере приведена программа, которая позволяет измерять напряжение, которое подается на вход АЦП. Просмотр значения напряжения производится в режиме отладки (рис. 3.5).

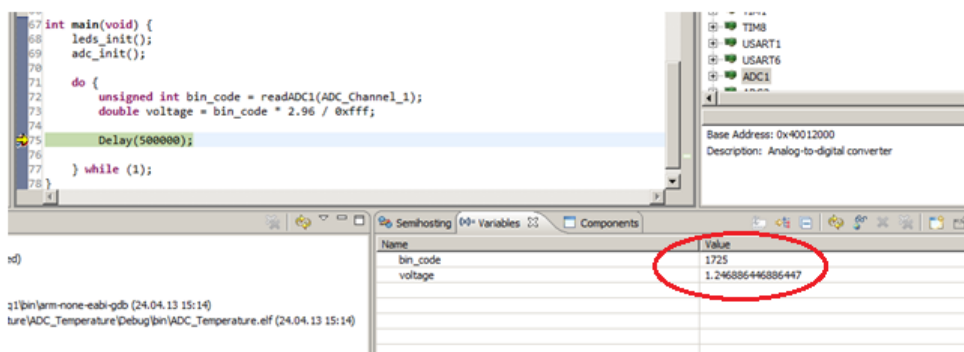


Рис. 3.5. Просмотр значения напряжения в режиме отладки

Для того, чтобы правильно определить поданное напряжение, необходимо провести дополнительные измерения. Для этого с помощью мультиметра или вольтметра определяем напряжение, которое соответствует 3 В, подключив их к соответствующим выводам на плате. Мультиметр показал, что в данном случае

напряжение равно 2.96 В. Поэтому записываем данный коэффициент непосредственно в код программы. Расчет напряжения производится по формуле:

$$U_{res} = \frac{U_{ref} * BIN_{res}}{BIN_{max}}, \quad (1)$$

где U_{res} – значение поданного напряжения,

U_{ref} – напряжение, относительно которого производится сравнение,

BIN_{max} – двоичный код максимально возможного значения, которое может храниться в регистре данных АЦП, зависит от его разрядности (в нашем случае разрядность АЦП 12, поэтому максимальному напряжению соответствует число 0xFFF – число, у которого в младших 12 разрядах все единицы),

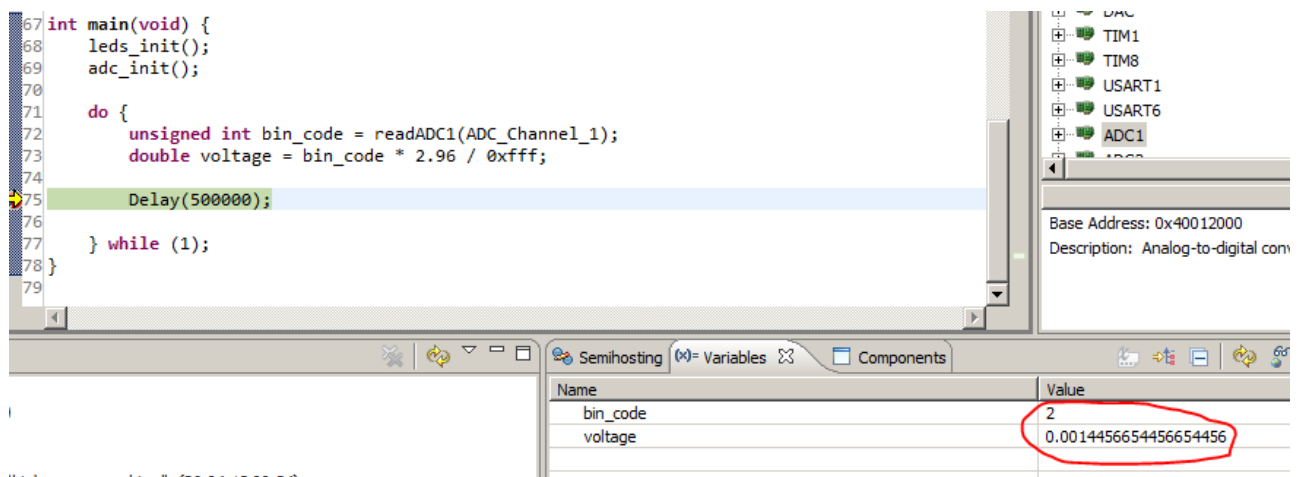
BIN_{res} – значение двоичного кода из регистра данных АЦП, которое записано после проведения измерений.

Соответственно, исходя из формулы 1, при подаче напряжения в 0 В, мы получим минимальное значение напряжения. Сделаем это соединив выводы земли GND и первого вывода порта А перемычкой. В результате этого значения в регистре данных АЦП изменится следующим образом (рис. 3.6).

```

67 int main(void) {
68     leds_init();
69     adc_init();
70
71     do {
72         unsigned int bin_code = readADC1(ADC_Channel_1);
73         double voltage = bin_code * 2.96 / 0xffff;
74
75         Delay(500000);
76
77     } while (1);
78 }
79

```



Name	Value
bin_code	2
voltage	0.0014456654456654456

Рис. 3.6. Просмотр значения напряжения в режиме отладки при подключении земли к входу

Как видим значение достаточно близко к нулю, что говорит о правильности работы программы.

Для проведения проверки данной программы подключение источника питания выполнено следующим образом (рис. 3.7):

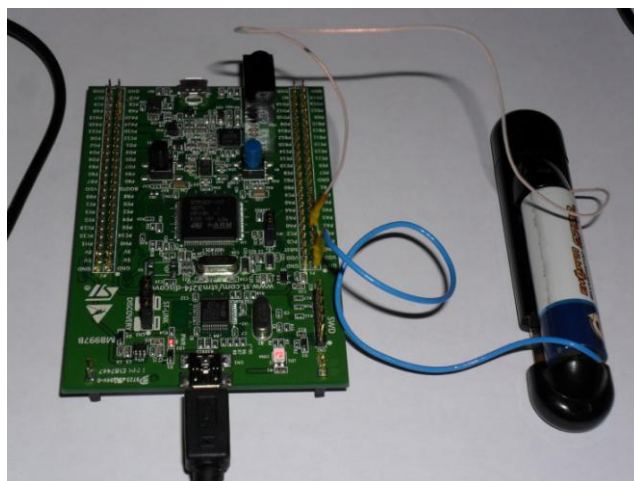


Рис. 3.7. Подключение источника питания для проведения измерений АЦП

Рассмотрим более детально программу, которая выполняет необходимые действия. Содержимое основного файла main.c представлено ниже.

```

#include <stm32f4xx_rcc.h>
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_adc.h>

void leds_init() {
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);

    GPIO_StructInit(&gpio);
    gpio.GPIO_Mode = GPIO_Mode_AN;
    gpio.GPIO_Pin = GPIO_Pin_1;
    GPIO_Init(GPIOA, &gpio);
}

void adc_init() {
    ADC_InitTypeDef ADC_InitStructure;
    ADC_CommonInitTypeDef adc_init;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1, ENABLE);
    ADC_DeInit();

    ADC_StructInit(&ADC_InitStructure);
    adc_init.ADC_Mode = ADC_Mode_Independent;
    adc_init.ADC_Prescaler = ADC_Prescaler_Div2;

    ADC_InitStructure.ADC_ScanConvMode = DISABLE;
    ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
    ADC_InitStructure.ADC_ExternalTrigConv = ADC_ExternalTrigConvEdge_None;
    ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
    ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b;

    ADC_CommonInit(&adc_init);
    ADC_Init(ADC1, &ADC_InitStructure);
    ADC_Cmd(ADC1, ENABLE);
}

u16 readADC1(u8 channel) {
    ADC_RegularChannelConfig(ADC1, channel, 1, ADC_SampleTime_3Cycles);
    ADC_SoftwareStartConv(ADC1);
    while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
    return ADC_GetConversionValue(ADC1);
}

void Delay(unsigned int Val) {
    for (; Val != 0; Val--);
}

int main(void) {
    leds_init();
    adc_init();

    do {
        unsigned int bin_code = readADC1(ADC_Channel_1);
        double voltage = bin_code * 2.96 / 0xffff;

        Delay(500000);
    } while (1);
}

```

В ней есть несколько моментов, на которые нужно обратить особое внимание. Во-первых, при инициализации порта мы задаем значение, которое нужно для его работы в аналоговом режиме с помощью значения `GPIO_Mode_AN`. Во-вторых, инициализация и считывание значений из АЦП. Мы настраиваем, АЦП таким образом, что преобразование производится программно и выполняем его с помощью вызова функции `readADC1`.

Ход работы

1. На основе кода примера приложения создать свой проект в среде разработки и проверить его работоспособность.
2. Ознакомиться с работой используемых функций, просмотрев исходный код, а также комментарии в исходных файлах библиотеки и в режиме отладки.
3. Создать новый проект в среде разработки для выполнения индивидуального задания, полученного от преподавателя.
4. Реализовать необходимую функциональность.
5. Запрограммировать плату и продемонстрировать работу программы.

Индивидуальные задания

1. Разработать программу, которая при подаче напряжения, чем половина максимальной выключает светодиод, а при подаче большего напряжения включает светодиод. Подтвердить корректность работы программы с помощью мультиметра.
2. Продемонстрировать прием данных с нескольких каналов АЦП.
3. Подключить к АЦП для измерения встроенный датчик изменения температуры. При изменении в большую сторону включать красный светодиод, при уменьшении – синий.
4. Подключить к АЦП встроенный источник напряжения. При изменении в большую сторону включать красный светодиод, при уменьшении – синий.

Лабораторная работа № 5: Использование USART

Цель работы: научиться использовать универсальный синхронный/асинхронный приемопередатчик, организовывать передачу данных другим устройствам.

Оборудование и программное обеспечение: плата STM32F4 Discovery, среда разработки Coocox CoIDE 1.7.

Теоретический материал

Микроконтроллер на демонстрационной плате содержит в общей сложности 6 приемо-передатчиков. При этом 4 из них являются синхронно-асинхронными (USART1, USART2, USART3, USART6) и 2 только асинхронными (UART4, UART5).

Рассмотрим процесс настройки USART для приема данных. Для этого необходимо выполнить следующие действия:

1. Включить тактирование USART и порта выходы, которого используются для работы USART.
2. Выполнить настройку соответствующих выходов, указав при этом режим работы выполнение альтернативной функции.
3. Подключить выполнение альтернативной функции к указанным выходам.
4. Задать настройки работы USART (частота, размер передачи, количество стоп-бит, проверка на парность).
5. Включить прерывание по приему данных для USART.
6. Включить USART.

Пример программы

В данном примере рассмотрим прием данных с помощью USART1. Для передачи и приема используются выходы порта A под номерами 9 (Transmit) и 10 (Receive). Для каждого приемо-передатчика выделены свои пары выводов для приема и передачи, которые можно найти в документации к микроконтроллеру. Прежде всего, включаем тактирование нужных устройств.

```
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);  
RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
```


Далее проводим инициализацию выводов порта и назначаем выполнение альтернативной функции:

```
GPIO_InitStruct.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_10;
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
GPIO_InitStruct.GPIO_OType = GPIO_OType_PP;
GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
GPIO_Init(GPIOA, &GPIO_InitStruct);
```

Далее выбираем альтернативную функцию, выполняемую выводами порта. Делаем это с помощью функции `GPIO_PinAFConfig()`.

```
GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_USART1);
GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_USART1);
```

Следует обратить внимание, что настройке подлежат только определенные выводы, другие выводы, например, PA2 и PA3, использовать в качестве выводов USART не удастся.

Далее необходимо инициализировать USART. Код инициализации очень похож на инициализацию порта.

```
USART_InitStruct.USART_BaudRate = baudrate;
USART_InitStruct.USART_WordLength = USART_WordLength_8b;
USART_InitStruct.USART_StopBits = USART_StopBits_1;
USART_InitStruct.USART_Parity = USART_Parity_No;
USART_InitStruct.USART_HardwareFlowControl =
USART_HardwareFlowControl_None;
USART_InitStruct.USART_Mode = USART_Mode_Rx;
USART_Init(USART1, &USART_InitStruct);
```

На последнем этапе инициализации необходимо настроить прерывание и включить USART.

```
USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
NVIC_InitStructure.NVIC_IRQChannel = USART1_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
USART_Cmd(USART1, ENABLE);
```

После того как мы включили прерывание необходимо определить обработчик этого прерывания. Обработчик прерывания имеет имя `USART1_IRQHandler` и он не возвращает и не принимает никаких данных. Кроме того, USART поддерживает несколько прерываний и для всех прерываний используется один обработчик, поэтому необходимо проверять флаг (RXNE) в регистре статуса, чтобы организовать обработку разных прерываний в разных блоках кода обработчика. После проверки состояния флага есть возможность сбросить флаг RXNE. Далее считываем данные из регистра данных и выводим их на дисплей.

```
void USART1_IRQHandler(void) {
    if( USART_GetITStatus(USART1, USART_IT_RXNE) ){
        USART_ClearITPendingBit(USART1, USART_IT_RXNE);
        static uint8_t cnt = 0;
        uint8_t t = USART_ReceiveData(USART1);
        write_char(t);
    }
}
```

В данном примере рассмотрим прием данных с другого устройства (использована плата STM8S Value Line, однако возможно использовать любое другое устройство с UART или USART) и отображение принятой информации на символьном LCD-дисплее. Более того, для проверки работы USART даже не нужно использовать другое устройство, так как можно просто замкнуть между собой выходы передатчика и

приемника. Важным моментом является то, что необходимо задать **одинаковые параметры сообщения** как на устройстве-передатчике, так и на приемнике.

Результат представлен ниже (рис. 3.8):



Рис. 3.8. Результат работы программы

Ход работы

1. На основе кода примера приложения создать свой проект в среде разработки и проверить его работоспособность.
2. Ознакомиться с работой используемых функций, просмотрев исходный код, а также комментарии в исходных файлах библиотеки и в режиме отладки.
3. Создать новый проект в среде разработки для выполнения индивидуального задания, полученного от преподавателя.
4. Реализовать необходимую функциональность.
5. Запрограммировать плату и продемонстрировать работу программы.

Индивидуальные задания

1. Задействовать два модуля USART на плате и передать данные из одного в другой в асинхронном режиме. Для демонстрации приема данных изменять состояние одного из пользовательских светодиодов на противоположный.
2. Продемонстрировать прием/передачу в синхронном режиме. Для демонстрации приема данных изменять состояние одного из пользовательских светодиодов на противоположный.

Лабораторная работа № 6: Работа с SPI

Цель работы: научиться использовать интерфейс SPI для организации передачи данных между устройствами, исследование режима ведущего и ведомого.

Оборудование и программное обеспечение: плата STM32F4 Discovery, среда разработки CoCoX CoIDE 1.7, Keil 4.xx, инструменты построения проекта GNU Toolchain for ARM Embedded Processors, библиотека CMSIS, набор перемычек (3 шт.).

Теоретический материал

Serial Peripheral Interface (SPI) – популярный интерфейс для последовательного обмена данными между микросхемами. Шина SPI организована по принципу "ведущий-подчиненный". В качестве ведущего шины обычно выступает микроконтроллер, но им также может быть программируемая логика, DSP-контроллер или специализированная ИС. Подключенные к ведущему шины внешние устройства образуют подчиненных шины. В их роли выступают различного рода микросхемы, в т.ч. запоминающие устройства (EEPROM, Flash-память, SRAM), часы реального времени (RTC), АЦП/ЦАП, цифровые потенциометры, специализированные контроллеры и др.

Главным составным блоком интерфейса SPI является обычный сдвиговый регистр, сигналы синхронизации и ввода/вывода битового потока которого и образуют интерфейсные сигналы. Таким образом, протокол SPI правильнее назвать не протоколом передачи данных, а протоколом обмена данными между двумя сдвиговыми регистрами, каждый из которых одновременно выполняет и функцию приемника, и функцию передатчика. Непременным условием передачи данных по шине SPI является генерация сигнала синхронизации шины. Этот сигнал имеет право генерировать только ведущий шины и от этого сигнала полностью зависит работа подчиненного шины.

Пример наиболее простого подключения по шине SPI показан на рисунке 3.9. Одноименные выводы соединяются между собой. Присутствуют два канала передачи данных (MOSI и MISO) один канал для подачи тактовых импульсов (SCLK), а также линия включения ведомого устройства (SS). Ведомый становится активным при подаче низкого уровня по линии SS.

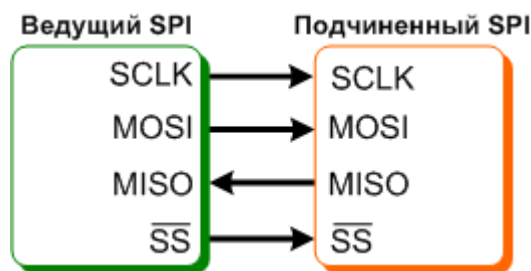


Рис. 3.9. Схема простого подключения с использованием SPI

SPI – чрезвычайно простой и распространенный последовательный интерфейс передачи данных, который основывается на сдвиговых регистрах. Его преимуществом по сравнению с USART является возможность подключения нескольких ведомых устройств. Однако, по сравнению с USART, он поддерживает только синхронную передачу. Наличие нескольких модулей SPI в составе микроконтроллера позволяет обойтись без подключения дополнительных устройств с таким интерфейсом, а использовать несколько устройств на плате, соединив их входы между собой перемычками. Этот вариант является оптимальным для учебных целей, поскольку требует минимум дополнительного оборудования для начала работы.

```

#include <stm32f4xx.h>
#include <stm32f4xx_rcc.h>
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_spi.h>
#include <misc.h>

#define SPI_PINS GPIO_Pin_5 | \
                GPIO_Pin_6 | \
                GPIO_Pin_7

void init(void);
void delay(int count);
  
```

```

void SPI1_IRQHandler(void) {
    int res;
    if (SPI_I2S_GetITStatus(SPI1, SPI_I2S_IT_RXNE) != RESET) {
        SPI_I2S_ClearITPendingBit(SPI1, SPI_I2S_IT_RXNE);
        res = SPI_I2S_ReceiveData(SPI1);
    }
}

int main(void)
{
    init();
    int sendData = 0;
    while(1)
    {
        delay(100);
        SPI_I2S_SendData(SPI2, sendData);
        while (SPI_I2S_GetFlagStatus(SPI2, SPI_I2S_FLAG_TXE) == RESET);
        sendData++;
        if (sendData == 0xff) sendData = 0;
    }
}

void init(void) {
    GPIO_InitTypeDef gpio_init;
    SPI_InitTypeDef spi_init;
    NVIC_InitTypeDef nvic_init;

    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA |
        RCC_AHB1Periph_GPIOB |
        RCC_AHB1Periph_GPIOC, ENABLE);

    /* Configure slave pins */
    gpio_init.GPIO_Mode = GPIO_Mode_AF;
    gpio_init.GPIO_Pin = SPI_PINS;
    gpio_init.GPIO_Speed = GPIO_Speed_50MHz;
    gpio_init.GPIO_OType = GPIO_OType_PP;
    gpio_init.GPIO_PuPd = GPIO_PuPd_DOWN;
    GPIO_Init(GPIOA, &gpio_init);

    GPIO_PinAFConfig(GPIOA, GPIO_PinSource5, GPIO_AF_SPI1);
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource6, GPIO_AF_SPI1);
    GPIO_PinAFConfig(GPIOA, GPIO_PinSource7, GPIO_AF_SPI1);

    gpio_init.GPIO_Pin = GPIO_Pin_2 | GPIO_Pin_3;
    GPIO_Init(GPIOC, &gpio_init);

    GPIO_PinAFConfig(GPIOC, GPIO_PinSource2, GPIO_AF_SPI2);
    GPIO_PinAFConfig(GPIOC, GPIO_PinSource3, GPIO_AF_SPI2);

    gpio_init.GPIO_Pin = GPIO_Pin_10;
    GPIO_Init(GPIOB, &gpio_init);

    GPIO_PinAFConfig(GPIOB, GPIO_PinSource10, GPIO_AF_SPI2);

    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SPI1, ENABLE);
    SPI_I2S_DeInit(SPI1);
    spi_init.SPI_Mode = SPI_Mode_Slave;
    spi_init.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
    spi_init.SPI_DataSize = SPI_DataSize_8b;
    spi_init.SPI_CPOL = SPI_CPOL_Low;
    spi_init.SPI_CPHA = SPI_CPHA_1Edge;
    spi_init.SPI_FirstBit = SPI_FirstBit_MSB;
}

```

```

spi_init.SPI_NSS = SPI_NSS_Soft;
spi_init.SPI_BaudRatePrescaler = SPI_BaudRatePrescaler_2;
SPI_Init(SPI1, &spi_init);
SPI_I2S_ITConfig(SPI1, SPI_I2S_IT_RXNE, ENABLE);

RCC_APB1PeriphClockCmd(RCC_APB1Periph_SPI2, ENABLE);
SPI_StructInit(&spi_init);
SPI_I2S_DeInit(SPI2);
spi_init.SPI_Mode = SPI_Mode_Master;
spi_init.SPI_Direction = SPI_Direction_2Lines_FullDuplex;
spi_init.SPI_DataSize = SPI_DataSize_8b;
spi_init.SPI_CPOL = SPI_CPOL_Low;
spi_init.SPI_CPHA = SPI_CPHA_1Edge;
spi_init.SPI_FirstBit = SPI_FirstBit_MSB;
spi_init.SPI_NSS = SPI_NSS_Soft;
SPI_Init(SPI2, &spi_init);

nvic_init.NVIC_IRQChannel = SPI1_IRQn;
nvic_init.NVIC_IRQChannelCmd = ENABLE;
nvic_init.NVIC_IRQChannelPreemptionPriority = 0;
nvic_init.NVIC_IRQChannelSubPriority = 0;
NVIC_Init(&nvic_init);

SPI_Cmd(SPI1, ENABLE);
SPI_Cmd(SPI2, ENABLE);
}

void delay(int count) {
    while(--count);
}

```

Ход работы

1. На основе кода примера приложения создать свой проект в среде разработки и проверить его работоспособность.
2. Ознакомиться с работой используемых функций, просмотрев исходный код, а также комментарии в исходных файлах библиотеки и в режиме отладки.
3. Создать новый проект в среде разработки для выполнения индивидуального задания, полученного от преподавателя.
4. Реализовать необходимую функциональность.
5. Запрограммировать плату и продемонстрировать работу программы.

Индивидуальные задания

1. Настроить на отладочной плате один из модулей в режим приема данных и принять данные от другого устройства с отображением на семисегментном индикаторе.
2. Продемонстрировать работу в режиме ведущего с подключением нескольких устройств: первое устройство – другой модуль SPI на плате, а второе – любое другое устройство, которое может выводить полученные данные на подключенный семисегментный индикатор.
3. Организовать передачу данных нескольким ведомым устройствам через определенный интервал времени. Подключить два устройства к одному модулю SPI на отладочной плате и передавать данные попеременно с интервалом приблизительно в 1 секунду с отображением принятых данных.
4. Передача данных другому модулю SPI на отладочной плате (значения от 0x00 до 0x0F) с отображением двоичного значения на светодиодах на плате.
5. Одновременная передача данных нескольким ведомым устройствам, которые отображают полученное значение на семисегментном индикаторе и в двоичном коде с использованием светодиодов.

Лабораторная работа № 7: Работа с DMA

Цель работы: исследовать возможности использования DMA, основные характеристики DMA, взаимодействие с другими устройствами в составе микроконтроллера.

Оборудование и программное обеспечение: отладочная плата STM32F4 Discovery, мультиметр, редактор разработки Coocox CoIDE 1.7, Keil 4.xx, инструменты построения проекта GNU Toolchain for ARM Embedded Processors, библиотека CMSIS.

Теоретические сведения

Direct Memory Access (DMA, рос. “Прямой доступ к памяти”) – механизм, используемый в контроллерах ARM для перемещения данных между памятью и периферией без участия процессора. Ключевым моментом является, то что при использовании DMA на перемещение данных не используются ресурсы процессора, что может быть особенно критичным при создании приложений, работающих с большим количеством данных и активно использующих периферию. Работа DMA обеспечивается отдельным контроллером, который выполняет определенные действия по команде процессора.

Рассматриваемый контроллер имеет в своем распоряжении сразу два контроллера DMA, которые обеспечивают в общей сложности 16 потоков (по 8 потоков на каждый контроллер), каждый из которых предназначен для управления запросами к памяти от одного или нескольких устройств. Каждый поток может обеспечивать до 8 каналов (запросов). Каждый контроллер DMA имеет устройство разрешения конфликтов для обработки запросов в соответствии с их приоритетом.

Контроллер DMA позволяет производить запись данных в трех направлениях:

- от периферии в память;
- из памяти к периферии;
- из памяти в память.

Передача ведется либо в режиме непосредственной передачи, либо в режиме очереди. Поддерживается различный размер данных для передачи, причем размер данных для приемника и источника может быть неодинаковым. В таком случае DMA определяет данную ситуацию и выполняет необходимые действия для оптимизации передачи, однако эта возможность поддерживается только в режиме очереди.

Кроме того, очень полезной может оказаться функция циклической передачи, при использовании которой передача данных начинается с начального адреса снова после передачи последней единицы данных источника.

Программист может задавать приоритеты для запроса каждого конкретного потока или приоритет будет определяться на основе значений по умолчанию.

Пример программы

Рассмотрим пример программы, которая использует DMA для передачи данных из памяти в ЦАП. Изменение уровня сигнала на выходе ЦАП происходит циклически с частотой изменения в 1 с. Изменения происходят на основе значения, которое считывается из памяти по сигналу переполнения таймера. В результате этого ЦАП отправляет запрос к DMA и получает данные, которые записываются в регистр данных, что приводит к изменению уровня сигнала. Схема взаимодействия выглядит следующим образом (рис. 3.10):

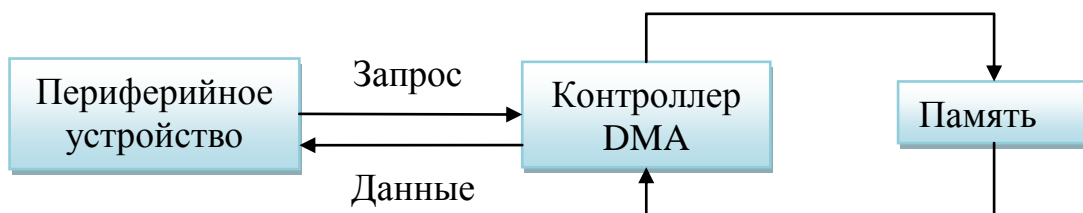


Рис. 3.10. Схема использования DMA

В самой программе следует обратить внимание на большое количество параметров, необходимых для настройки DMA по сравнению с другими устройствами. С этим связана и сложность использования DMA, поскольку необходимо учитывать большое количество возможных настроек. Тем не менее, многие из них достаточно просты (направление передачи, значения адресов), поэтому изучение DMA можно сопоставить по сложности с другими устройствами, в чем очень помогает библиотека Standard Peripheral Library.

```

#include <stm32f4xx.h>
#include <stm32f4xx_rcc.h>
#include <stm32f4xx_dma.h>
  
```

```

#include <stm32f4xx_gpio.h>
#include <stm32f4xx_tim.h>
#include <stm32f4xx_dac.h>

uint8_t levels[] = {0x00, 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77};

void init(void);
void init_gpio(void);
void init_timer(void);
void init_dac(void);
void init_dma(void);

int main(void)
{
    init();
    while(1)
    {
    }
}

void init(void) {
    init_gpio();
    init_timer();
    init_dac();
    init_dma();
}

void init_gpio(void) {
    GPIO_InitTypeDef gpio_init;
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
    gpio_init.GPIO_Mode = GPIO_Mode_AN;
    gpio_init.GPIO_Pin = GPIO_Pin_4;
    gpio_init.GPIO_OType = GPIO_OType_PP;
    gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
    gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_Init(GPIOA, &gpio_init);
}

void init_timer(void) {
    TIM_TimeBaseInitTypeDef tim_init;
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6, ENABLE);
    tim_init.TIM_CounterMode = TIM_CounterMode_Up;
    tim_init.TIM_Period = 16000 - 1;
    tim_init.TIM_Prescaler = 1000 - 1;
    TIM_TimeBaseInit(TIM6, &tim_init);

    TIM_SelectOutputTrigger(TIM6, TIM_TRGOSource_Update);

    TIM_Cmd(TIM6, ENABLE);
}

void init_dac(void) {
    DAC_InitTypeDef dac_init;
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
    DAC_StructInit(&dac_init);
    dac_init.DAC_Trigger = DAC_Trigger_T6_TRGO;
    dac_init.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
    dac_init.DAC_WaveGeneration = DAC_WaveGeneration_None;
    DAC_Init(DAC_Channel_1, &dac_init);
}

void init_dma(void) {

```

```

DMA_InitTypeDef dma_init;
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA1, ENABLE);
DMA_DeInit(DMA1_Stream5);
dma_init.DMA_Channel = DMA_Channel_7;
dma_init.DMA_PeripheralBaseAddr = (uint32_t)(DAC_BASE + 0x10);
dma_init.DMA_Memory0BaseAddr = (uint32_t)&levels;
dma_init.DMA_DIR = DMA_DIR_MemoryToPeripheral;
dma_init.DMA_BufferSize = 8;
dma_init.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
dma_init.DMA_MemoryInc = DMA_MemoryInc_Enable;
dma_init.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Byte;
dma_init.DMA_MemoryDataSize = DMA_PeripheralDataSize_Byte;
dma_init.DMA_Mode = DMA_Mode_Circular;
dma_init.DMA_Priority = DMA_Priority_High;
dma_init.DMA_FIFOMode = DMA_FIFOMode_Disable;
dma_init.DMA_FIFOThreshold = DMA_FIFOThreshold_HalfFull;
dma_init.DMA_MemoryBurst = DMA_MemoryBurst_Single;
dma_init.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;
DMA_Init(DMA1_Stream5, &dma_init);
DMA_Cmd(DMA1_Stream5, ENABLE);

DAC_Cmd(DAC_Channel_1, ENABLE);
DAC_DMAMCmd(DAC_Channel_1, ENABLE);
}

```

После компиляции программы-примера и прошивки отладочной платы следует наблюдать за напряжением на выходе PA4 с помощью мультиметра. Прибор должен показать ступенчатое изменение напряжение от 0 В до определенного значения, после чего цикл повторяется. Шаг увеличения одинаков для всего цикла, что можно увидеть по массиву значений для записи в регистр данных.

Ход работы

1. Проверить работу программы-примера, скомпилировав ее в одной из сред разработки и выполнив прошивку.
2. Ознакомиться с документацией по DMA для микроконтроллера на отладочной плате.
3. Попробовать изменить определенные параметры в программе-примере.
4. Организовать взаимодействие DMA и другого периферийного устройства в соответствии с индивидуальным заданием.

Индивидуальные задания

1. Реализовать асинхронную передачу данных через USART, считывая данные через определенные промежутки времени посредством DMA.
2. Организовать прием данных через USART с записью в память через DMA. Количество передаваемых данных известно заранее и равно размеру массива.
3. Продемонстрировать передачу данных через SPI из памяти в циклическом режиме.
4. Продемонстрировать прием данных через SPI с записью в память. Режим записи – циклический.
5. Реализовать запись в память значений, полученных с помощью работы АЦП через определенные промежутки времени. После записи первых 50 значений начинается новый цикл записи.

Лабораторная работа № 8: Генерация сигналов и управление их характеристиками

Цель работы: изучить возможности генерации сигналов и способы управления их характеристиками с использованием отладочной платы и отображением информации.

Оборудование и программное обеспечение: отладочная плата STM32F4 Discovery, символьный LCD-дисплей, 3 тактовые кнопки и резисторы для их подключения, среда разработки для STM32 (CoIDE, Keil или др.), библиотеки CMSIS и SPL.

Пример программы

В качестве примера рассмотрим программу, которая позволяет на выходе получить сигнал прямоугольной формы в заданном диапазоне частот и с заданным коэффициентом заполнения. Данная программа является своего рода закреплением пройденного до этого материала и охватывает многие уже пройденные моменты. Также ее можно рассматривать как продолжение работы с таймерами в режиме ШИМ. Соответственно пользователю необходимо предоставить возможность для регулирования указанных характеристик. Для примера взяты следующие значения:

- выходная частота – 3-4 кГц;
- коэффициент заполнения $\pm 50\%$ от начального значения, которым считаем значение в 0,5.

Для генерации сигналов прямоугольной как нельзя лучше подходит включение таймера в режиме ШИМ для одного из выходных каналов. Необходимо лишь рассчитать значения для настройки самого таймера при указанной тактовой частоте устройства (они будут разными для 16 МГц и 168 МГц) и включить тактирование и нужный режим работы для остальных устройств в составе контроллера.

В соответствии с указанными значениями проведем расчет для настройки таймера (рассматривается случай с внутренней частотой 16 МГц). Поскольку более детально этот процесс описан в одной из предыдущих работ, то далее приведены лишь результаты вычислений. Для того, чтобы получить на выходе частоту в диапазоне 3-4кГц, устанавливаем значение предделителя равным 10, тогда диапазон изменения значений в регистре автозагрузки будет меняться от 533 до 400. Именно эти значения используются как граничные.

Коэффициент заполнения не требует специальных расчетов, поэтому лишь скажем, что это значение хранится в программе в виде действительного числа, которое пользователь может изменять и указанные изменения записываются в регистр сравнения для выбранного выходного канала.

В качестве решения для предоставления пользователю возможности регулирования параметров выходного сигнала предлагается также подключить символьный LCD-дисплей и три кнопки для выполнения регулировки. Функциональное назначение кнопок может меняться в соответствии с действиями пользователя. В то же время на экране следует отображать информацию о текущих изменениях (смена пунктов меню, значений характеристик), которые выполняет пользователь.

Перейдем к ознакомлению с кодом проекта. Функциональная часть объявлена и реализована в файлах *init.h* и *init.c*. Далее приводится листинг файла *init.c*

```
#include "init.h"
#include "hd44780lib.h"
#include <stm32f4xx_gpio.h>
#include <stm32f4xx_rcc.h>
#include <stm32f4xx_exti.h>
#include <stm32f4xx_syscfg.h>
#include <stm32f4xx_tim.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <misc.h>

// инициализация портов, к которым подключен дисплей
void init_gpio_lcd(void) {
    GPIO_InitTypeDef gpio_init;
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA | RCC_AHB1Periph_GPIOE, ENABLE);
    gpio_init.GPIO_Pin = DATA_PINS;
    gpio_init.GPIO_Mode = GPIO_Mode_OUT;
    gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
    gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_Init(DATA_PORT, &gpio_init);
}
```

```

    gpio_init.GPIO_Pin = CONTROL_PINS;
    gpio_init.GPIO_Mode = GPIO_Mode_OUT;
    gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
    gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
    GPIO_Init(CONTROL_PORT, &gpio_init);
}

// инициализация порта, к которому подключены кнопки
void init_gpio_buttons(void) {
    GPIO_InitTypeDef gpio_init;
    RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOB, ENABLE);
    gpio_init.GPIO_Mode = GPIO_Mode_IN;
    gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
    gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
    gpio_init.GPIO_Pin = BUTTON_PINS;
    GPIO_Init(GPIOB, &gpio_init);
}

// configures buttons inputs on board as interrupt source
void configure_buttons(void) {
    EXTI_InitTypeDef exti_init;
    RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource0);
    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource1);
    SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOB, EXTI_PinSource2);
    exti_init.EXTI_LineCmd = ENABLE;
    exti_init.EXTI_Line = EXTI_Line0 | EXTI_Line1 | EXTI_Line2;
    exti_init.EXTI_Mode = EXTI_Mode_Interrupt;
    exti_init.EXTI_Trigger = EXTI_Trigger_Rising;
    EXTI_Init(&exti_init);
}

// функция разрешения указанного прерывания
void enable_interrupt(IRQn_Type irq) {
    NVIC_InitTypeDef nvic_init;
    nvic_init.NVIC_IRQChannel = irq;
    nvic_init.NVIC_IRQChannelCmd = ENABLE;
    nvic_init.NVIC_IRQChannelSubPriority = 1;
    nvic_init.NVIC_IRQChannelPreemptionPriority = 1;
    NVIC_Init(&nvic_init);
}

// разрешаем прерывания по нажатию кнопки
void configure_interrupts(void) {
    enable_interrupt(EXTI0_IRQn);
    enable_interrupt(EXTI1_IRQn);
    enable_interrupt(EXTI2_IRQn);
}

void configure_delay_timer(void) {
    TIM_TimeBaseInitTypeDef tim_init;
    tim_init.TIM_CounterMode = TIM_CounterMode_Up;
    tim_init.TIM_Prescaler = 1600 - 1; //настраиваем предделитель таймера для задержек
    RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM7, ENABLE);
    TIM_TimeBaseInit(DELAY_TIMER, &tim_init);
}

// настройка таймер для генерации ШИМ
void configure_pwm_timer(void) {
    GPIO_InitTypeDef gpio_init;
    TIM_TimeBaseInitTypeDef tim_init;
    TIM_OCInitTypeDef oc_init;

```

```

RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA, ENABLE);
gpio_init.GPIO_Mode = GPIO_Mode_AF;
gpio_init.GPIO_Pin = GPIO_Pin_0;
gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
gpio_init.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init(GPIOA, &gpio_init);

GPIO_PinAFConfig(GPIOD, GPIO_PinSource0, GPIO_AF_TIM2);

RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
tim_init.TIM_Prescaler = 10 - 1;
tim_init.TIM_Period = frequency_value - 1;
tim_init.TIM_CounterMode = TIM_CounterMode_Up;
tim_init.TIM_ClockDivision = 0;
tim_init.TIM_RepetitionCounter = 0;
TIM_TimeBaseInit(PWM_TIMER, &tim_init);

oc_init.TIM_OCMode = TIM_OCMode_PWM1;
oc_init.TIM_Pulse = frequency_value / 2;
oc_init.TIM_OutputState = TIM_OutputState_Enable;
TIM_OC1Init(PWM_TIMER, &oc_init);

TIM_Cmd(PWM_TIMER, ENABLE);
}

// заполняем меню
void fill_menu(void) {
    main_option_index = 0;
    main_options[0] = create_menu_item("1.Change frequency", set_active_freq_menu);
    main_options[1] = create_menu_item("2. Change fill", set_active_fill_menu);

    main_menu.items[0] = create_menu_item("", down_menu);
    main_menu.items[1] = main_options[main_option_index];
    main_menu.items[2] = create_menu_item("", up_menu);

    freq_menu.items[0] = create_menu_item("", decrement_freq);
    freq_menu.items[1] = create_menu_item("", set_active_main_menu);
    freq_menu.items[2] = create_menu_item("", increment_freq);

    filling_menu.items[0] = create_menu_item("", increment_fill);
    filling_menu.items[1] = create_menu_item("", set_active_main_menu);
    filling_menu.items[2] = create_menu_item("", decrement_fill);
    set_active_menu(&main_menu);
}

void set_active_menu(struct menu * menu) {
    active_menu = menu;
}

void set_active_freq_menu(void) {
    active_menu = &freq_menu;
    show_freq_on_screen();
}

void set_active_fill_menu(void) {
    active_menu = &filling_menu;
    show_fill_on_screen();
}

void set_active_main_menu(void) {
    active_menu = &main_menu;
}

```

```

        show_text_on_screen(&main_menu.items[1]);
    }

    // реализация функции задержки
    void delay(int times) {
        DELAY_TIMER->CNT = 0;
        DELAY_TIMER->ARR = times;
        TIM_Cmd(DELAY_TIMER, ENABLE); // запуск таймера
        while(TIM_GetFlagStatus(DELAY_TIMER, TIM_FLAG_Update) == RESET); // ожидание
        TIM_ClearFlag(DELAY_TIMER, TIM_FLAG_Update);
        TIM_Cmd(DELAY_TIMER, DISABLE); // остановка таймера
    }

    struct menu_item create_menu_item(char * text, action act) {
        struct menu_item item;
        strcpy(item.menu_text, text);
        item.action = act;
        return item;
    }

    void show_text_on_screen(struct menu_item * item) {
        lcd_clear();
        write_string(item->menu_text);
    }

    // увеличение значения переменной для изменения частоты
    void increment_freq(void) {
        if (frequency_value == MAX_FREQ_VALUE) return;
        frequency_value++;
        TIM_SetAutoreload(PWM_TIMER, frequency_value - 1);
        TIM_SetCompare1(PWM_TIMER, frequency_value * fill_value);
        show_freq_on_screen();
    }

    // уменьшение значения переменной для изменения частоты
    void decrement_freq(void) {
        if (frequency_value == MIN_FREQ_VALUE) return;
        frequency_value--;
        TIM_SetAutoreload(PWM_TIMER, frequency_value - 1);
        TIM_SetCompare1(PWM_TIMER, frequency_value * fill_value);
        show_freq_on_screen();
    }

    void increment_fill(void) {
        if (MAX_FILL_VALUE - fill_value < 0.01) return;
        fill_value += 0.01;
        TIM_SetCompare1(PWM_TIMER, frequency_value * fill_value);
        show_fill_on_screen();
    }

    void decrement_fill(void) {
        if (fill_value - MIN_FILL_VALUE < 0.01) return;
        fill_value -= 0.01;
        TIM_SetCompare1(PWM_TIMER, frequency_value * fill_value);
        show_fill_on_screen();
    }

    void change_menu_item(struct menu_item * item, char * text, action act) {
        strcpy(item->menu_text, text);
        item->action = act;
    }

```

```

void change_menu_item(struct menu_item * cur, struct menu_item * next) {
    cur = next;
}

void up_menu(void) {
    if (main_option_index == 1) {
    } else {
        main_option_index++;
        main_menu.items[1] = main_options[main_option_index];
        show_text_on_screen(&main_menu.items[1]);
    }
}

void down_menu(void) {
    if (main_option_index == 0) {
    } else {
        main_option_index--;
        main_menu.items[1] = main_options[main_option_index];
        show_text_on_screen(&main_menu.items[1]);
    }
}

void show_freq_on_screen(void) {
    char text[20];
    double freq = (double)16000000 / (frequency_value * 10) / 1000;
    gcvt(freq, 4, text);
    strcat(text, " kHz");
    lcd_clear();
    write_string(text);
}

void show_fill_on_screen(void) {
    char text[20];
    gcvt(fill_value, 4, text);
    strcat(text, "%");
    lcd_clear();
    write_string(text);
}

// инициализация всех компонентов
void init_all(void) {
    frequency_value = 533;
    fill_value = 0.5;
    init_gpio_buttons();
    init_gpio_lcd();
    configure_buttons();
    configure_interrupts();
    configure_delay_timer();
    configure_pwm_timer();
    fill_menu();
    lcd_init();
    show_text_on_screen(&main_menu.items[1]);
}

```

В заголовочном файле находятся объявления функций, а также используемых переменных и констант.

```

#ifndef INIT_H
#define INIT_H

#include <stm32f4xx.h>
#include <stm32f4xx_gpio.h>

```

```

#include <stm32f4xx_tim.h>

#define BUTTON_PINS GPIO_Pin_0 | \
                    GPIO_Pin_1 | \
                    GPIO_Pin_2

#define DELAY_TIMER TIM7
#define PWM_TIMER TIM2
#define MAX_FREQ_VALUE 533
#define MIN_FREQ_VALUE 400
#define MAX_FILL_VALUE 0.75
#define MIN_FILL_VALUE 0.25

typedef void (*action)(void);

/*
 * Структура для представления пункта меню
 */
struct menu_item {
    action action; // действие
    char menu_text[20]; // текст меню
};

struct menu {
    struct menu_item items[3]; // пункты меню
};

int frequency_value; // переменная для задания частоты, имеет значения 400...533
double fill_value; // переменная для задания коэффициента заполнения
struct menu * active_menu; // текущее меню
struct menu_item main_options[3]; // пункты главного меню
struct menu main_menu;
struct menu freq_menu;
struct menu filling_menu;
int main_option_index; // индекс текущего пункта в главном меню

void init_gpio_lcd(void);
void init_gpio_buttons(void);

void configure_buttons(void);
void configure_interrupts(void);
void configure_delay_timer(void);
void configure_pwm_timer(void);

void init_all(void);

struct menu_item create_menu_item(char * text, action act);
void show_text_on_screen(struct menu_item * item);
void show_freq_on_screen(void);
void show_fill_on_screen(void);
void fill_menu(void);
void set_active_menu(struct menu * menu);
void set_active_freq_menu(void);
void set_active_fill_menu(void);
void set_active_main_menu(void);
void empty_action(void);
void change_menu_item(struct menu_item * item, char * text, action act);
void change_menu_item(struct menu_item * cur, struct menu_item * next);
void up_menu(void);
void down_menu(void);

void increment_freq(void);

```

```
void decrement_freq(void);
void increment_fill(void);
void decrement_fill(void);

#endif // INIT_H
```

В файле с основной функцией вызывается функция инициализации всех используемых устройств и описаны обработчики прерываний для нажатий кнопок.

```
#include <stm32f4xx_exti.h>
#include <string.h>
#include "hd44780lib.h"
#include "init.h"

void EXTI0_IRQHandler(void) {
    EXTI_ClearITPendingBit(EXTI_Line0);
    if (active_menu != NULL)
        delay(100);
    active_menu->items[0].action();
}

void EXTI1_IRQHandler(void) {
    EXTI_ClearITPendingBit(EXTI_Line1);
    if (active_menu != NULL)
        delay(100);
    active_menu->items[1].action();
}

void EXTI2_IRQHandler(void) {
    EXTI_ClearITPendingBit(EXTI_Line2);
    if (active_menu != NULL)
        delay(100);
    active_menu->items[2].action();
}

int main(void)
{
    init_all();
    while(1)
    {
    }
}
```

Описание установки

В собранном виде с запущенной программой установка имеет вид, показанный на рис. 3.11.

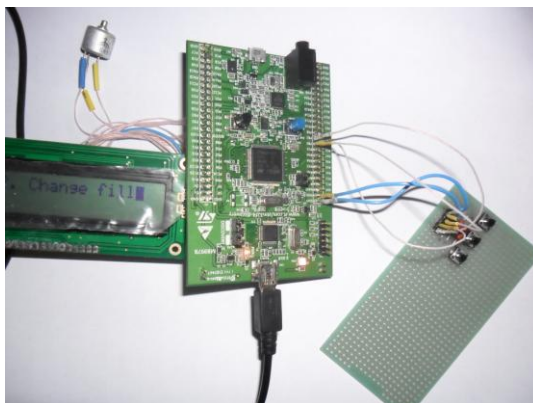


Рис. 3.11. Внешний вид установки

В подключении участвуют 3 порта ввода/вывода (к выводу порта А, на котором генерируется сигнал, подключается измерительное оборудование, которое на рисунке не показано). Кнопки подключаются к выводам 0-2 порта В. Выходной сигнал можно наблюдать на выходе 0 порта А.

Схема подключения компонентов показана на рис. 3.12.

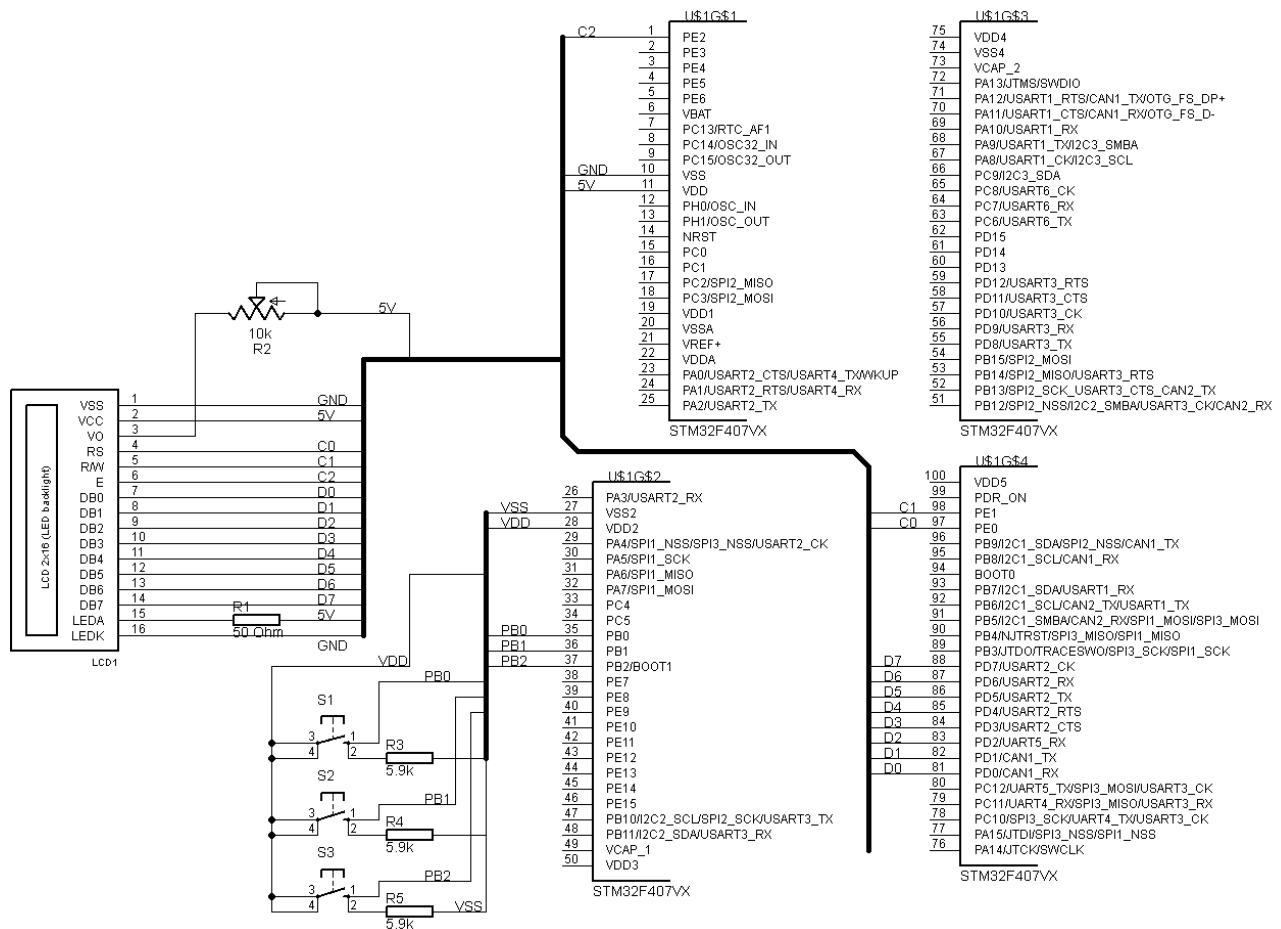


Рис. 3.12. Схема подключения компонентов

Описание подключения дисплея к плате можно найти в приложении, однако следует отметить необходимость для этого дополнительных резисторов, один из которых обладает переменным сопротивлением для регулирования яркости отображаемых символов.

Для управления вариантами меню, а также изменением характеристик используются подключенные к выводам 0-2 порта В тактовые кнопки. В данном примере использованы 4-контактные тактовые кнопки TS-06V (рис. 3.13), однако можно использовать и другие кнопки, которые удастся найти.



Рис. 3.13. Внешний вид тактовой кнопки

Перед тем, как выполнить подключение, следует изучить с помощью мультиметра в режиме измерения сопротивления, какие именно контакты замыкаются при нажатии, а какие остаются соединенными всегда. Кроме того, для подтяжки выхода к земле используются резисторы номиналом 5,9 кОм.

Для размещения описанного выше соединения используется макетная плата для пайки. Все выводы на плате (3 – для кнопок и 2 – для питания и земли) для удобства подключены к штыревому разъему. Подключение с отладочной платой выполняется с помощью перемычек. Возможно, более удобным вариантом для начинающих будет использование безопасной макетной платы.

Индивидуальные задания

1. Максимальная частота работы контроллера на плате составляет 168 МГц. Используя инструкцию по заданию частот, выполнить расчет и запустить рассмотренные пример с соблюдением значений параметров частоты и коэффициента заполнения.
2. По аналогии с приведенным примером напишите программу, которая позволяет изменять частоту прямоугольного сигнала в диапазоне от 1 до 100 Гц и регулировать коэффициент заполнения во всем диапазоне. Работу продемонстрировать на частоте 1 Гц, изменяя значение коэффициента.

ПРИЛОЖЕНИЕ А

Работа с дисплеем

Вместе с непосредственной обработкой сигналов важной частью для работы схемы является отображение информации, результатов обработки для пользователя схемы. В том случае, когда схема достаточно проста и направлена на отслеживание небольшого количества показателей, достаточно, например, использовать семисегментный индикатор или символьный LCD-дисплей. Такие дисплеи зачастую могут отображать больше информации, чем индикаторы, составленные из нескольких семисегментных, а также способны отображать намного больше различных символов.

Для ознакомления с работой таких дисплеев рассмотрим дисплей RC1602A (рис. А.1), а также его использование под управлением STM32F4Discovery. Данный дисплей отображает 2 строки по 16 символов в каждой и работает под управлением контроллера KS0066U. Он обеспечивает следующую функциональность по работе с дисплеем:

- выполнение команд управления (управление разрядностью, курсором и другие операции);
- выполнение записи и считывания памяти дисплея.



Рис. А.1. Внешний вид дисплея RC1602A

Использованная схема подключения дисплея представлена на рис. А.2.

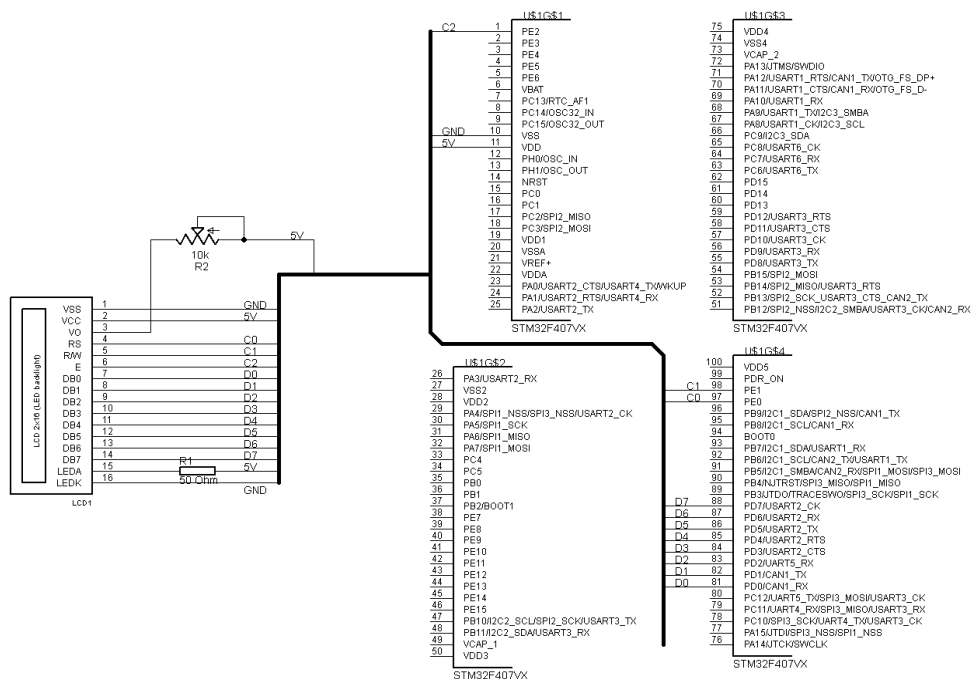


Рис. А.2. Схема подключения дисплея

Для управления дисплеем с помощью платы STM32F4Discovery написана библиотека (реализована не вся функциональность, а лишь наиболее необходимое: выполнение команд и запись данных в память для отображения). Сама память разделена на две части: память отображения (DDRAM) и память отведенная под символы пользователя (CGRAM). Информация, записанная в память для отображения появляется на экране. Поскольку размер памяти больше размера экрана, то отображается лишь ее часть, а отображение остальной части можно обеспечить с помощью сдвига отображаемой на экране памяти. Более детальное описание возможностей дисплеев такого типа можно найти в документации к ним в Интренете.

Далее приведен код библиотеки: заголовочный файл и файл с реализацией.

Ниже представлен заголовочный файл библиотеки.

```
#ifndef KS006ULIB_H
#define KS006ULIB_H

#include <stm32f4xx_gpio.h>
#define DATA_PORT GPIOD
#define CONTROL_PORT GPIOE
#define DATA_PINS GPIO_Pin_0 | \
    GPIO_Pin_1 | \
    GPIO_Pin_2 | \
    GPIO_Pin_3 | \
    GPIO_Pin_4 | \
    GPIO_Pin_5 | \
    GPIO_Pin_6 | \
    GPIO_Pin_7
#define CONTROL_PINS GPIO_Pin_0 | \
    GPIO_Pin_1 | \
    GPIO_Pin_2
#define RS_PIN GPIO_Pin_0
#define RW_PIN GPIO_Pin_1
#define E_PIN GPIO_Pin_2

void lcd_init();
void lcd_command(unsigned char command_data);
void write_data(unsigned char data);
void write_char(unsigned char data);
void write_string(char * string);
void write_string_at(char * string, unsigned char address);
void write_at(unsigned char symbol, unsigned char address);
void set_rs();
void unset_rs();
void set_rw();
void unset_rw();
void set_e();
void unset_e();

#define lcd_clear() lcd_command(0x01)
#define lcd_set_address(address) lcd_command(0b10000000 & address)

#endif
```

Далее представлен файл реализации библиотеки.

```
#include "ks0066uilib.h"

extern void delay(int times);

void lcd_init() {
    delay(20);
    lcd_command(0b00110000);
    lcd_command(0b00110000);
    lcd_command(0b00110000);
    lcd_command(0b00111000);
}
```

```

    lcd_command(0b00001111);
    lcd_command(0b00000001);
    lcd_command(0b00000110);
}

void lcd_command(unsigned char command_data) {
    unset_rs(); unset_rw();
    write_data(command_data);
    set_e();
    delay(40);
    unset_e();
}

void write_char(unsigned char data) {
    set_rs(); unset_rw();
    write_data(data);
    set_e();
    delay(40);
    unset_e();
}

void write_string(char * string) {
    uint32_t i;
    for(i = 0; string[i] != '\0'; ++i)
        write_char(string[i]);
}

void write_string_at(char * string, unsigned char address) {
    lcd_set_address(address);
    write_string(string);
}

void write_data(unsigned char data) {
    GPIO_Write(DATA_PORT, data);
}

void write_at(unsigned char symbol, unsigned char address) {
    lcd_set_address(address);
    write_char(symbol);
}

void set_rs() {
    GPIO_SetBits(CONTROL_PORT, RS_PIN);
}

void unset_rs(){
    GPIO_ResetBits(CONTROL_PORT, RS_PIN);
}

void set_rw(){
    GPIO_SetBits(CONTROL_PORT, RW_PIN);
}

void unset_rw(){
    GPIO_ResetBits(CONTROL_PORT, RW_PIN);
}

void set_e(){
    GPIO_SetBits(CONTROL_PORT, E_PIN);
}

void unset_e(){

```

```
GPIO_ResetBits(CONTROL_PORT, E_PIN);
```

Остановимся на особенностях использования данной библиотеки. Предполагается, что она будет использоваться как часть проекта в среде разработки. Соответственно следует настроить для проекта пути поиска заголовочных файлов, чтобы они включали путь к `stm32f4xx_gpio.h` или отредактировать файл соответствующим образом. Кроме того, необходимо задать значения `CONTROL_PORT` и `DATA_PORT` и номера пинов для Вашего варианта настройки. Также Вам необходимо определить функцию `delay(int)`, которая используется для генерации задержек между сигналами от платы к дисплею. При тестировании библиотеки использовался вариант функции, которая в цикле декрементирует значение переменной. Кроме того, Вам необходимо самостоятельно настроить соответствующие порты для вывода перед использованием, после чего вызвать функцию `lcd_init()`.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

Комментарий:

В некоторых позициях в конце в скобках дается описание того, что можно полезного взять из данного источника.

1. ARM (архитектура) [Электронный ресурс] – Режим доступа : URL: [http://ru.wikipedia.org/wiki/ARM_\(архитектура\)](http://ru.wikipedia.org/wiki/ARM_(архитектура)). – Загол. с экрана.
2. STM32F4DISCOVERY, Отладочный комплект на базе STM32F407VGT6 ARM CortexM4-F [Электронный ресурс] – Режим доступа : URL: <http://www.chipdip.ru/product/stm32f4discovery/>. – Загол. с экрана. (*короткое описание платы + документация PDF*)
3. GNU Tools for ARM Embedded Processors [Электронный ресурс] – Режим доступа : URL: <https://launchpad.net/gcc-arm-embedded/+download>. – Загол. с экрана. (*средства для компиляции*)
4. Бородулин А. STM8 и STM32 – объединенное пространство 8- и 32-разрядных микроконтроллеров // Компоненты и технологии №10. – 2009. – с. 55-59. (*описание общих характеристик 2 архитектур, делается акцент на общих чертах*)
5. STM32F4 GPIO tutorial [Электронный ресурс] – Режим доступа : URL: <http://eliaselectronics.com/stm32f4-tutorials/stm32f4-gpio-tutorial/>. – Загол. с экрана.
6. STM32F4: PWM [Электронный ресурс] – Режим доступа : URL: <http://amarkham.com/?p=37>. – Загол. с экрана.
7. STM32F4: INTERRUPT TIMER [Электронный ресурс] – Режим доступа : URL: <http://amarkham.com/?p=29>. – Загол. с экрана. (*описание работы с прерываниями таймера*)
8. Программирование STM32F4. USART. Пример программы. [Электронный ресурс] – Режим доступа : URL: <http://microtechnics.ru/programmirovanie-stm32f4-usart-primer-programmy/>. – Загол. с экрана.
9. Микроконтроллеры AVR. UART. Использование прерываний. [Электронный ресурс] – Режим доступа : URL: <http://microtechnics.ru/mikrokontrollery-avr-uart-ispolzovanie-preryvanij/>. – Загол. с экрана.
10. STM32 ADC Примеры использования. Шаг 1 [Электронный ресурс] – Режим доступа : URL: <http://mycontroller.ru/stm32-adc-primeryi-ispolzovaniya-shag-1/>. – Подзагол. с экрана.
11. Подключаем HD44780 дисплей к STM32. [Электронный ресурс] – Режим доступа : URL: <http://easystem32.ru/indication/22-hd44780-and-stm32>. – Загол. с экрана.
12. MicroXplorer Eclipse plugin, graphical tool to configure STM32 microcontrollers. [Электронный ресурс] – Режим доступа : URL: <http://www.st.com/web/en/catalog/tools/PF257931>. – Подзагол. с экрана.
13. Clock configuration tool for STM32F40x/41x microcontrollers [Электронный ресурс] – Режим доступа : URL: <http://www.st.com/web/catalog/tools/FM147/CL1794/SC961/SS1533/PF257927>. – Подзагол. с экрана.
14. ST Visual Programmer for programming ST7, STM8 and STM32 [Электронный ресурс] – Режим доступа : URL: <http://www.st.com/web/catalog/tools/FM147/CL1794/SC961/SS1533/PF210568>. – Подзагол. с экрана.
15. Brown J. Discovering the STM32 Microcontroller. January 8, 2013. (*книга полностью посвященная микроконтроллерам данного типа. Правда, код для данной платы не подходит, но многие моменты очень хорошо расписаны и можно освоить самостоятельно. Практически единственная книга по данной тематике*)
16. Reference manual. STM32F40xxx, STM32F41xxx, STM32F42xxx, STM32F43xxx advanced ARM-based 32-bit MCUs. [Электронный ресурс] – Режим доступа : URL: http://www.st.com/web/en/resource/technical/document/reference_manual/DM00031020.pdf. – Загол. с экрана. (*главный мануал по МК*)
17. STM32F4DISCOVERY STM32F4 high-performance discovery board [Электронный ресурс] – Режим доступа : URL: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00039084.pdf. – Загол. с экрана. (*описание платы*)