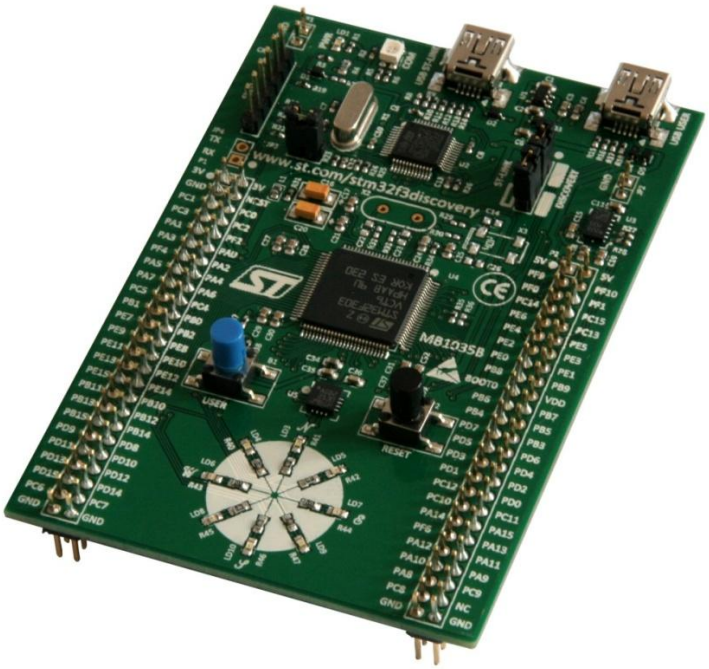
В. И. Бугаев, М. П. Мусиенко, Я. М. Крайнык

ЛАБОРАТОРНЫЙ ПРАКТИКУМ

по изучению микроконтроллеров STM32
на базе отладочного модуля STM32F3 Discovery



Москва-Николаев – 2014

Лабораторный практикум по изучению микроконтроллеров STM32 на базе отладочного модуля STM32F3 Discovery / Бугаев В.И., Мусиенко М.П., Крайнык Я.М. – Москва-Николаев: МФТИ-ЧГУ, 2014. – 33 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

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В соответствии с возрастающими потребностями в производительности встраиваемых систем возрастают возможности основных компонентов таких систем – микроконтроллеров. Большим шагом в этом процессе стало появление 32-разрядных микроконтроллеров, построенных на базе архитектуры ARM. Увеличение разрядности привнесло большие преимущества, а также предоставило возможность увеличить потенциальную продуктивность систем и обеспечить больший запас ресурсов при реализации встраиваемых систем.

Типичными представителями ARM-микроконтроллеров являются микроконтроллеры семейства Cortex-M4, которые также являются одними из наиболее технологичных с аппаратной точки зрения. Широкий набор периферийных интерфейсов, высокая тактовая частота, использование DMA – далеко не полный список возможностей, использование которых предоставляют микроконтроллеры данного семейства.

Данный лабораторный практикум предполагает использование отладочного модуля STM32F3 Discovery для изучения возможностей микроконтроллеров, построенных на базе архитектуры ARM и ориентирован на развитие практических навыков работы с аппаратным обеспечением и программными средствами для реализации практических задач.

Начало работы с отладочным модулем

Для работы с отладочным модулем STM32F3 Discovery необходимо следующее программное обеспечение:

- среда разработки Keil uVision версии не ниже 4.7 с поддержкой возможности создания проекта для микроконтроллера, который используется на плате (STM32F303VC);

- утилита STM32 ST-Link Utility для выполнения прошивки, работы с памятью микроконтроллера; при установке утилиты также устанавливаются драйвера для подключения устройства.

Первым заданием для выполнения является проверка корректности подключения платы и ее распознавание как устройства программными средствами. Подключите плату к ПК с помощью кабеля USB и откройте STM32 ST-LINK Utility. Выполните команду меню *Target/Connect*. В результате в списке Device Information (рис. 1) должна появиться информация об устройство. Сохраните программу, которая находится в микроконтроллере как hex-файл.

Данная программа может быть особенно полезной, когда возникаю трудности с программированием устройства в результате программных и аппаратных проблем, которые сложно отслеживать. В таком случае следует выполнить команду *Target/Erase Chip*, после чего выполнить программирование устройства.

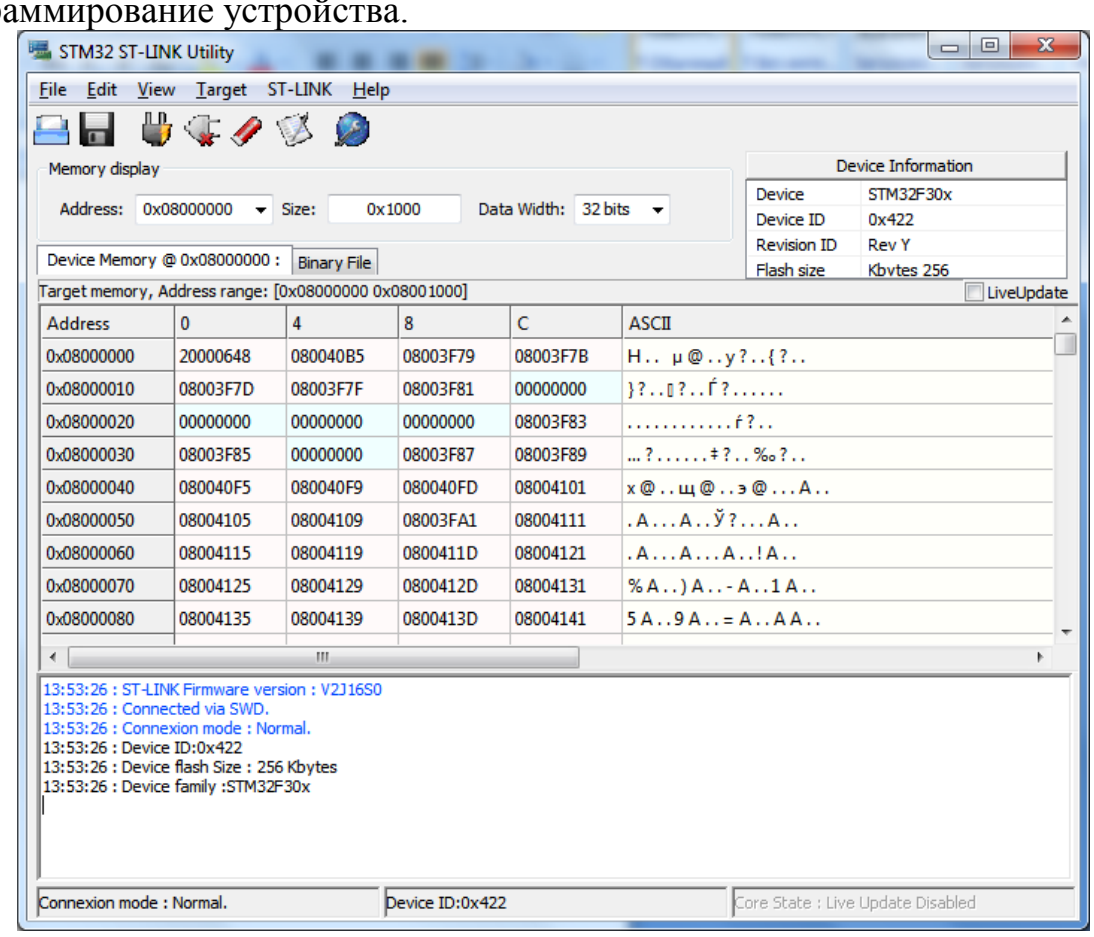


Рис. 1. Окно утилиты после выполнения команды

Основная работа с модулем основана на использовании среды разработки Keil uVision. Процесс установки в данном материале не рассматривается, представлена лишь настройка проекта для работы с отладочной платой.

Создание проекта начинается командой меню Project/New uVision Project. Необходимо указать расположение проекта, для чего желательно предварительно создать отдельную директорию. Далее следует выбрать фирму-производитель микроконтроллера и собственно сам микроконтроллер (в нашем случае – ST Microelectronics и STM32F303VC). На последнем шаге будет предложено добавить к проекту файл начальной инициализации на языке ассемблера, что необходимо подтвердить. После этого сам проект будет содержать лишь один упомянутый файл, который в дереве проекта будет отображаться в файловой группе Startup. С помощью файловых групп обеспечивается структурное и логическое разделение функциональности проекта. Файловые группы не отображаются в реальные директории операционной системы. Поэтому при добавлении файлов в проект не обязательно создавать соответствующие директории. Новую файловую группу можно создать с помощью контекстного меню в окне структуры проекта командой *Add Group*.

Следующим этапом настройки проекта является добавление необходимых файлов. Для создания программного обеспечения необходимы файлы 2 библиотек – CMSIS и Standard Peripheral Library. Первая из них описывает базовые функции для работы и описания регистров, а другая обеспечивает работу с периферийными устройствами. Загрузить библиотеку можно на сайте производителя со страницы, посвященной отладочному модулю. Обычно Standard Peripheral Library уже включает в себя файлы библиотеки CMSIS (поддиректория CMSIS), поэтому достаточно загрузить только одну библиотеку.

Из библиотеки CMSIS необходимы следующие файлы:

- core_cm4.h;
- core_cmFunc.h;
- core_cmInstr.h;
- stm32f30x.h;
- stm32f30x_conf.h;
- system_stm32f30x.h
- system_stm32f30x.c.

Для начального подключения их следует скопировать в директорию проекта.

Подключение файлов стандартной библиотеки зависит от того, какие устройства будут использованы в работе. Однако, при использовании функций библиотеки рекомендуется обязательно добавить файлы для настройки тактирования устройства (Reset Clock Control, RCC), поскольку без включения тактирования работа устройств невозможна. Для работы с конкретным устройством к проекту следует добавить 2 файла – файл заголовков и файл исходных кодов (отличаются расширениями *.h* и *.c*), в зависимости от

устройства следует добавить файлы с сокращенным обозначением в конце имени файла, например, для портов ввода/вывода – окончание *gpio*.

Следует также отметить, что файл с главной функцией *main* не создается автоматически, поэтому его следует создать самостоятельно. Для этого следует выполнить команду меню *File/New* и сохранить новый файл как *main.c* и добавить в него код стандартной функции *main,* которая не принимает параметров и возвращает целочисленное значение и содержит бесконечный цикл. После сохранения файл автоматически не добавляется к проекту, поэтому это действие следует выполнить самостоятельно. Рекомендуется создать отдельную группу файлов в проекте, в которую добавлять файлы(команда контекстного меню Add Files to Group …), созданные программистом (например, группу файлов можно назвать User), чтобы отделить их от библиотечных файлов. Таким же образом следует создать отдельные группы для файлов библиотек.

В результате выполнения указанных действий получаем структуру проекта, пример которой представлен на рисунке 2. Данного стиля формирования структуры проекта рекомендуется придерживаться для того, чтобы другие пользователи могли быстро ориентироваться в нем.

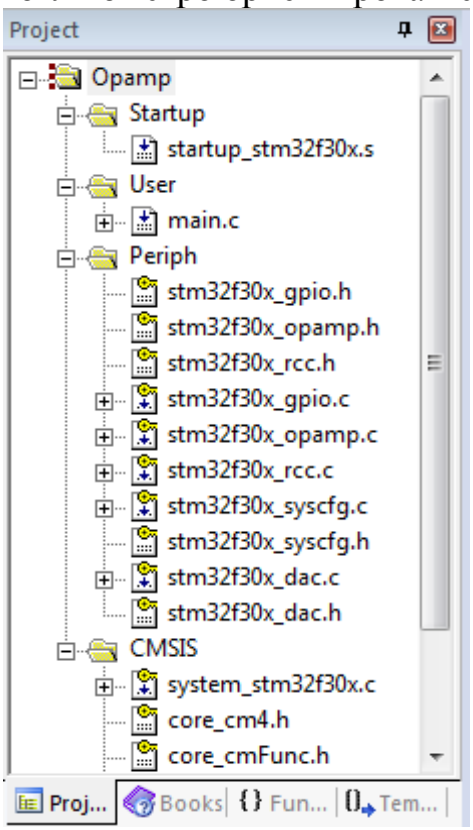


Рис. 2. Пример структуры рабочего проекта

Также следует отметить, что добавление новых файлов заголовков к проекту не является обязательным – обязательно добавление только файлов исходных кодов. Компилятор сам сможет найти необходимые файлы, если они будут находиться в местах поиска файлов заголовков.

После создания и настройки структуры проекта следующей задачей является настройка компилятора, а также аппаратных средств для проведения

программирования и отладки. Окно основных настроек проекта можно открыть командой *Project/Options for Target*…. Настройки компилятора выполняются с помощью вкладки C/C++. В ней указываем, что необходимо во время компиляции определить такие символы: USE_STDPERIPH_DRIVER, STM32F30X; указываем это, введя указанные значения в текстовое поле *Define* (рис. 3). Поскольку при создании проекта будем придерживаться правила копирования файлов в директорию проекта и подключения их из указанной директории, то настройку директорий файлов заголовков можно не выполнять, как показано на рисунке 3 (поле Include Paths пустое).

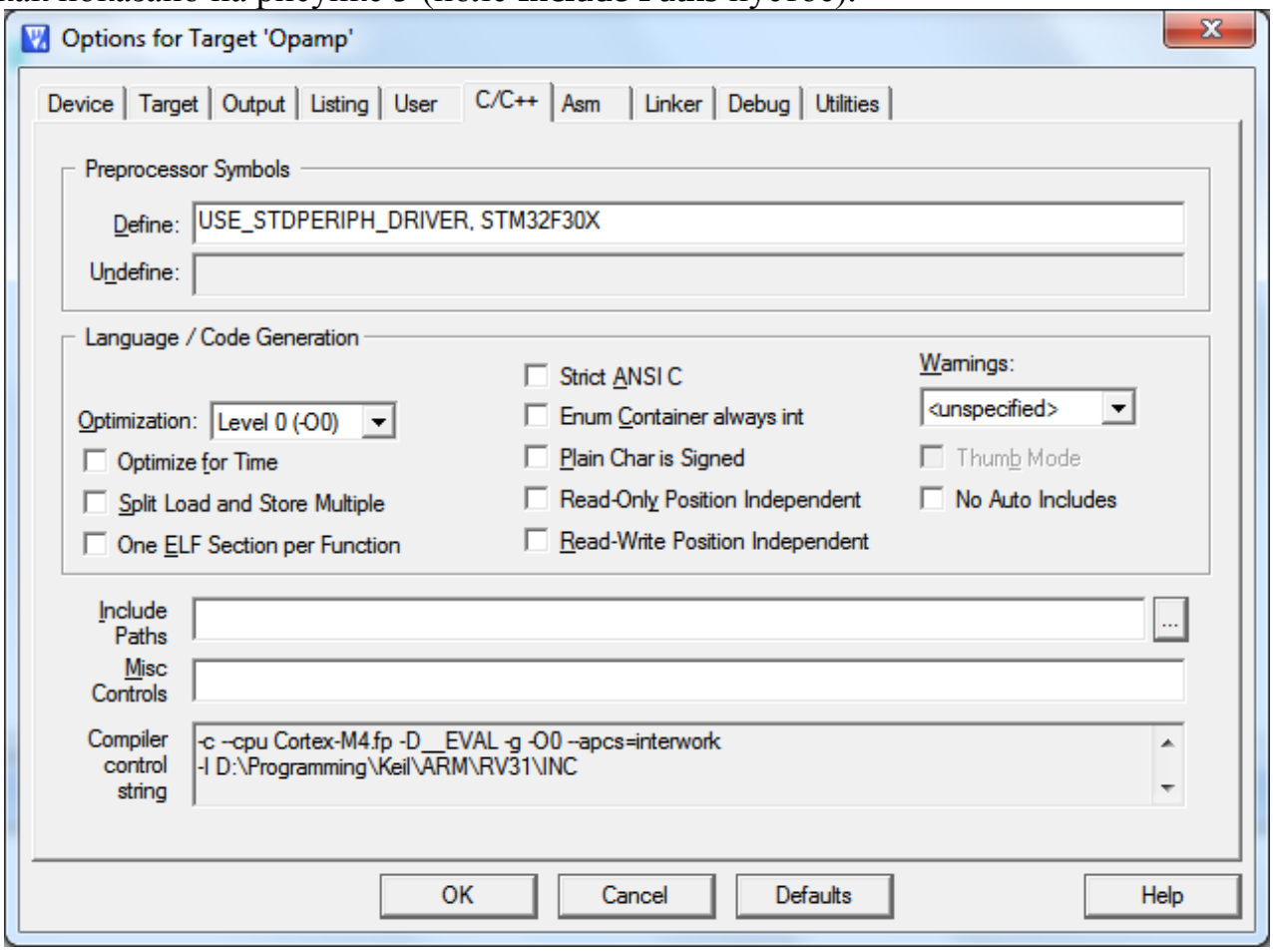


Рис. 3. Настройка опций компилятора

Далее выполним настройку отладки. Эти действия проводятся во вкладке *Debug*. Для отладки используется интерфейс ST-Link, который уже встроен в плату. Поэтому устанавливаем соответствующее значение кнопкой выбора и выбираем указанный интерфейс из выпадающего списка (рис. 4).

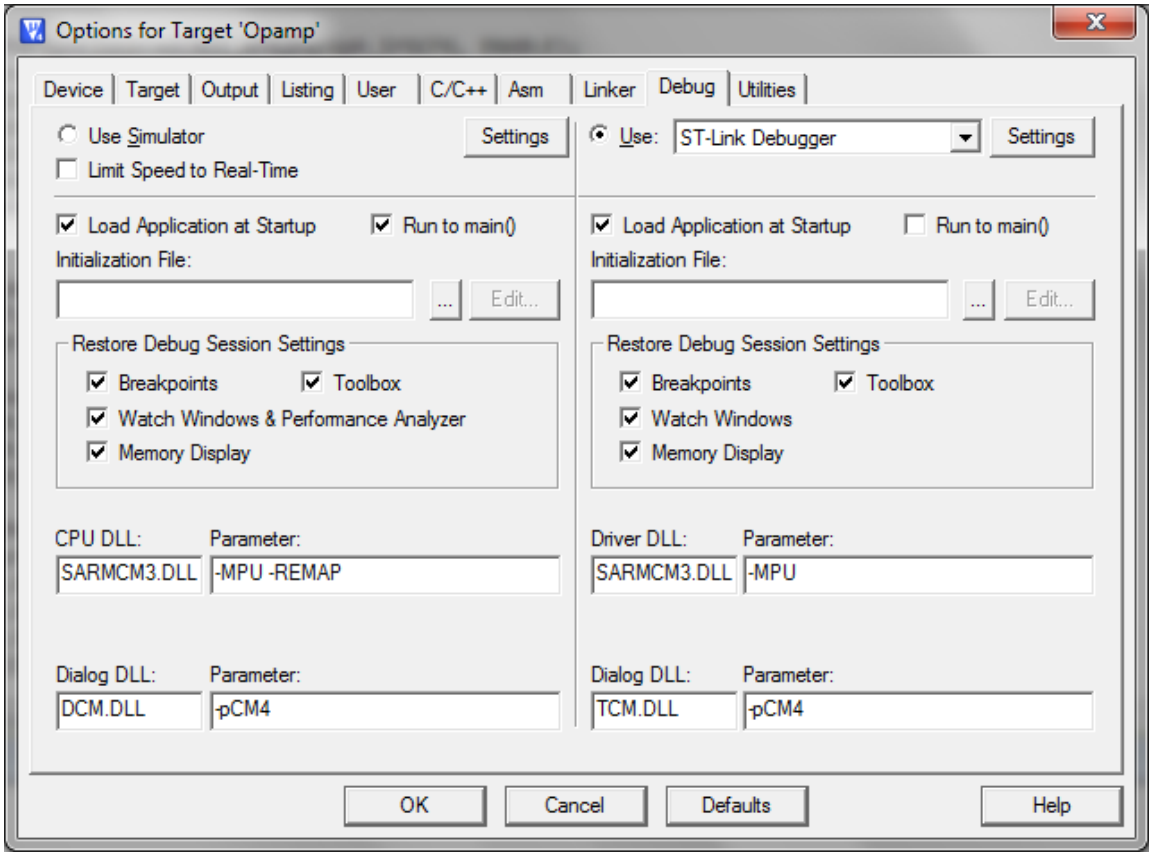


Рис. 4. Вкладка настройки отладки

Кроме того, необходимо также выполнить настройки самого устройства отладки, нажав кнопку *Settings*. В результате открывается дополнительное диалоговое окно с 3 вкладками. Во вкладке Debug изменяем значение с JTAG на SW (рис. 5).

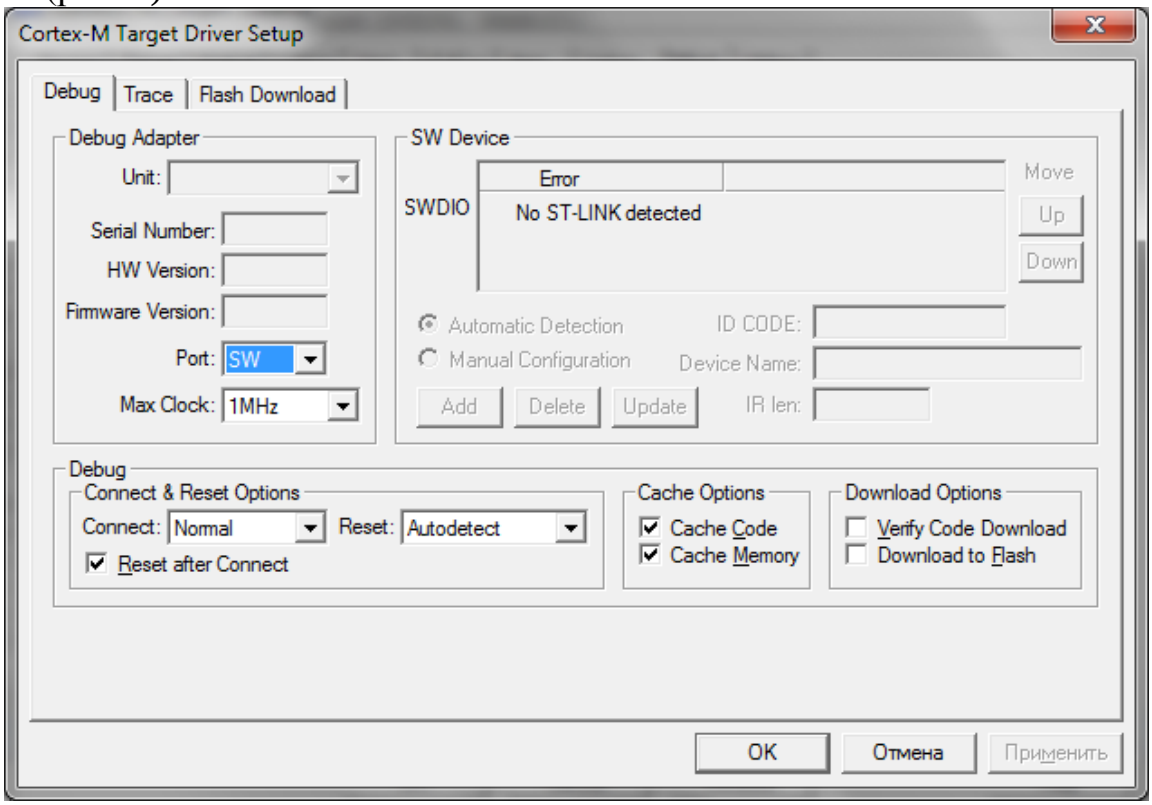


Рис. 5. Настройка интерфейса подключения

На вкладке Flash Download выполняются настройки памяти устройства для программирования и действий, которые выполняются после выполнения программирования. Рекомендуется установить флаг Reset and Run, чтобы сразу после загрузки начать отладку, иначе необходимо нажать кнопку Reset на плате. Также следует добавить тип памяти устройства, как это показано на рис. 6.

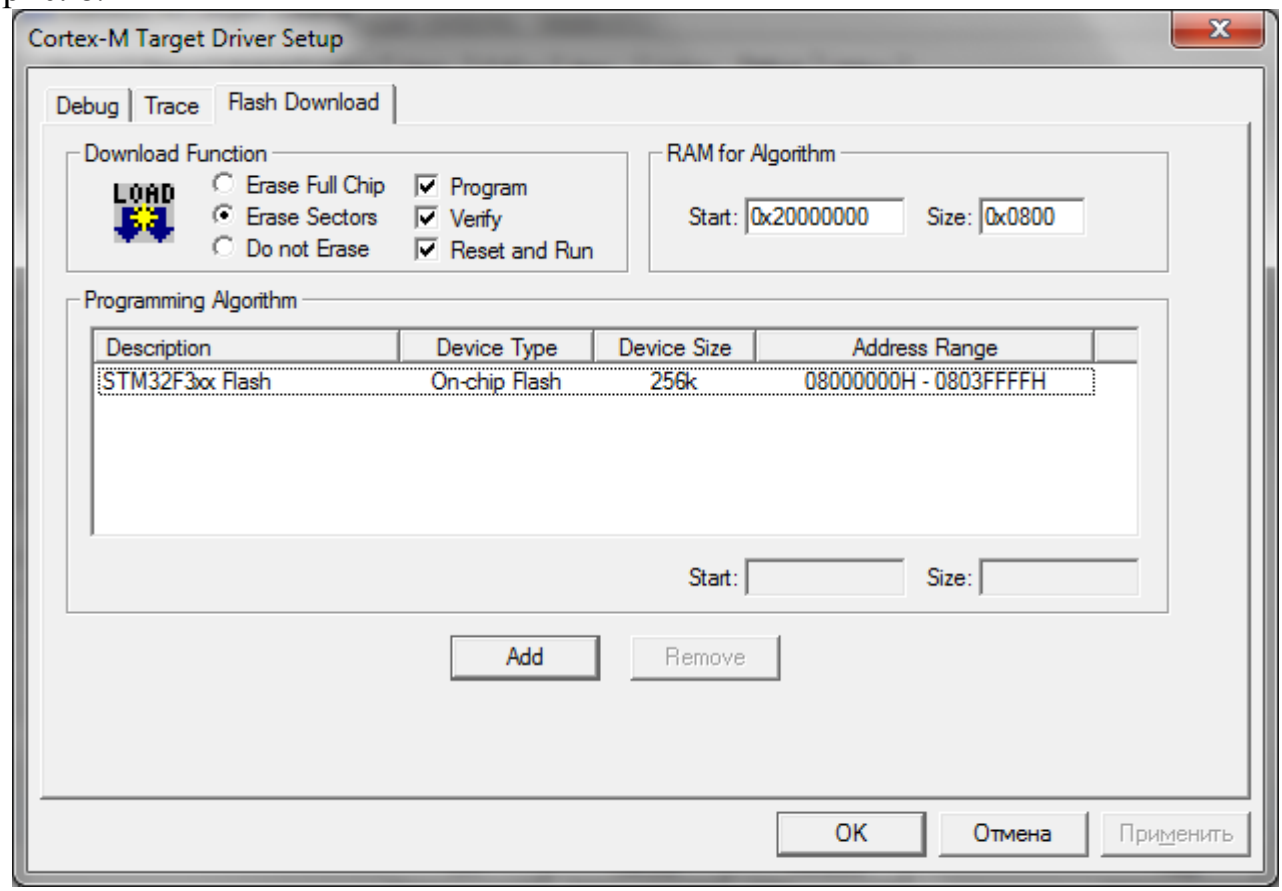


Рис. 6. Настройка памяти устройства для отладки

После выполнения указанных действий подтвердить изменения, нажав OK.

Таким же образом следует выполнить настройки интерфейса для программирования (вкладка *Utilities*). Выбрать из списка упомянутый выше интерфейс (рис. 7). Нажав кнопку *Settings*, проверить, что настройки соответствуют представленным на рис. 6.

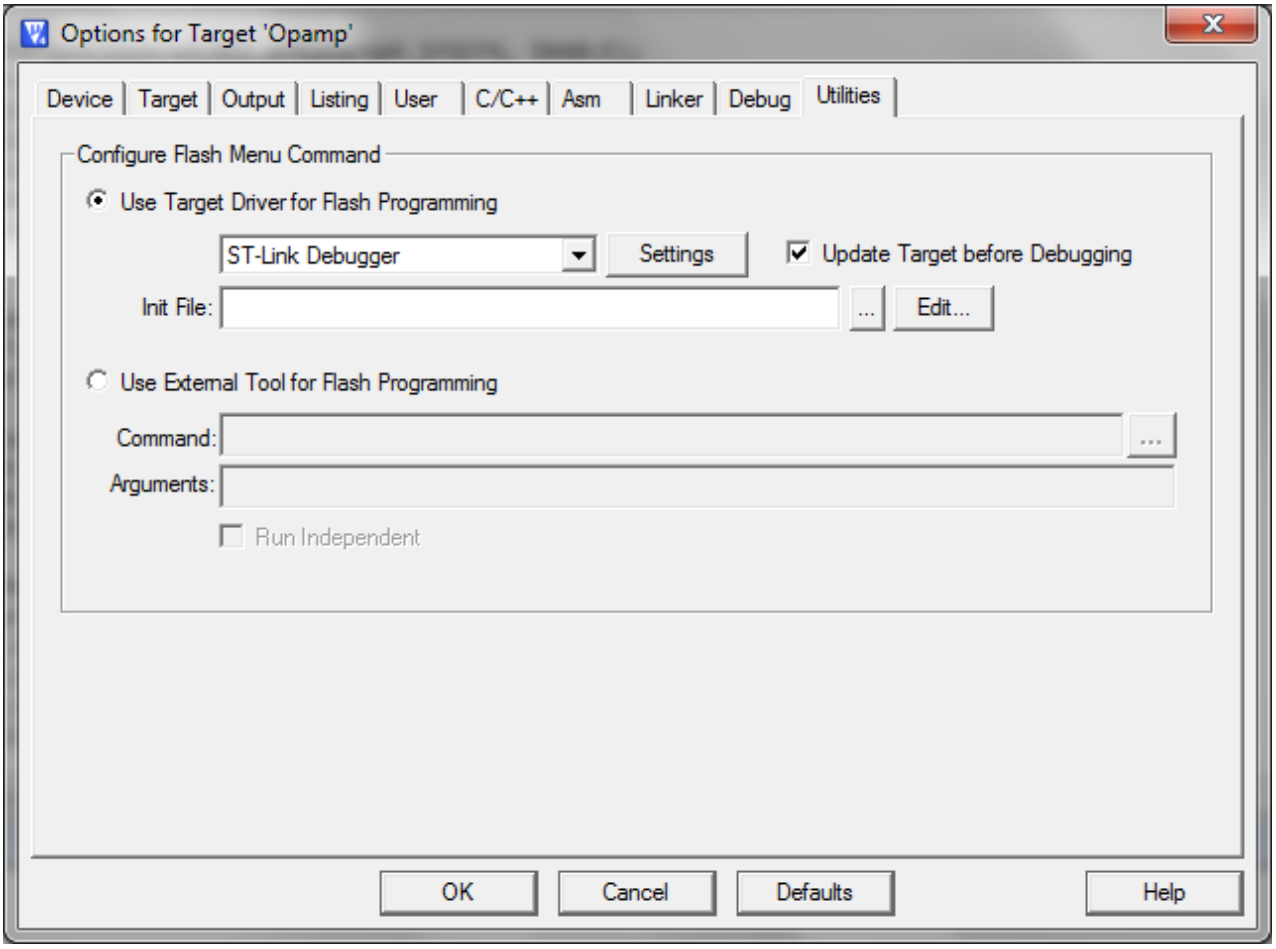


Рис. 7. Выбор средств для программирования

После проведения указанных действий следует выполнить построение проекта для проверки правильности настроек. Запустить построение проекта можно командой меню *Project/Build target* или клавишей *F7*. В результате сборка проекта должна пройти без ошибок. Если возникли ошибки, то проверьте еще раз выполнение указанных выше шагов.

Лабораторная работа №1
Работа с портами ввода/вывода

Цель работы: научиться работать с портами ввода/вывода, организовывать код программы в виде функций, организовывать базовую структуру программы.

Оборудование: отладочная плата, модуль со светодиодным кубом размером 3x3x3 (для включения отдельного уровня и светодиода используется сигнал логической 1), перемычки для подключения между платами при необходимости.

Теоретическая часть

На плате, которая используется в ходе выполнения работ, присутствуют сразу 6 портов ввода/вывода. Большинство из них являются 16-разрядными. Для управления портами используется 10 регистров.

Есть следующие режимы работы выводов порта:

- работа в качестве выхода;
- работа в качестве входа;
- работа в качестве аналогового вывода (используется при работе с АЦП и ЦАП);
- использование в качестве вывода выполнения альтернативной функции устройств, которые входят в состав микроконтроллера (например, UART, SPI и др.).

При работе в режиме выхода возможно использование 2 режимов: двухтактного выхода (push-pull) и открытого стока (open-drain). Режим двухтактного выхода используется в большинстве случаев, а работа в режиме открытого стока необходимо, например, при использовании интерфейса передачи данных I2C. К каждому выводу есть возможность подключения подтягивающего резистора. Есть возможность подтянуть сигнал на выходе к лог. 0 или лог. 1. По умолчанию подтягивающие резисторы не используются.

Если необходимо принять данные в виде логических состояний, то можно перевести вывод в режим входа. При такой конфигурации появляется возможность считывать данные с выводов.

Тем не менее, в большинстве случаев для считывания данных используются периферийные устройства. Они используют выводы порта в качестве собственных выводов. Такой режим работы получил название альтернативной функции. При этом происходит отображение выводов порта на выводы устройства. Такой режим рассмотрен в следующих работах.

Пример программы

В примере рассматривается режим работы выводов в качестве двухтактного выхода для простого включения/выключения устройств. Перейдем к более детальному рассмотрению практического примера.

В качестве примера используем работу со светодиодным 3D-кубом и созданием визуальных эффектов на его основе.

Принцип, который используется для создания эффектов с использованием светодиодного 3D-куба, базируется на использовании динамической индикации. Используется поочередное включение/выключение элементов схемы, которое незаметно для человеческого глаза из-за того, что происходит с большой частотой. Конкретная реализация зависит от того, как организовано подключение светодиодов в структуре куба.

Чаще всего используется схема подключения, при которой выделяются выводы для управления включением уровней, а также светодиодов на включенном уровне. Таким образом, для управления кубом размерностью 3x3x3 светодиода необходимо 3 выхода для управления уровнями и 9 выходов для управления светодиодами на уровне. Будем считать, что используется схема с общим катодом для организации структуры. В таком случае для включения одного светодиода следует подать сигнал логической единицы на выход, который отвечает за включение одного из уровней, а также подать лог. 1 на один из выходов, который отвечает за подключенные светодиоды.

Перейдем к рассмотрению схемы подключения компонентов в структуре куба (рис. 8).

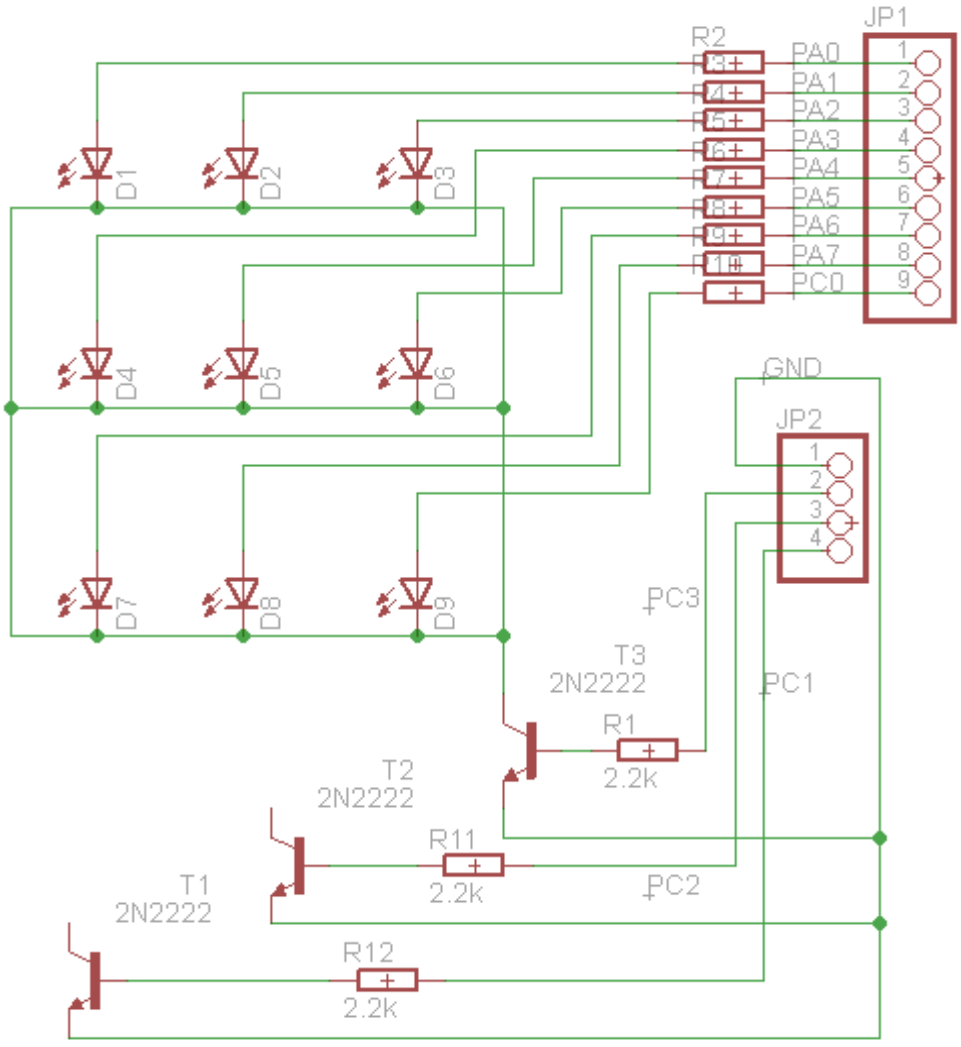


Рис. 8. Схема подключения компонентов при создании куба

Все катоды светодиодов на одном уровне объединяются и через транзистор подключаются к земле. В данном случае транзистор работает в режиме ключа: при подаче лог. 1 на базу транзистор открывается и появляется возможность включить светодиоды на уровне. Использование нескольких уровней приводит к тому, что светодиоды удобно группировать в столбец по 3 элемента. Но следует подчеркнуть, что при этом следует использовать резистор с большим сопротивлением, чем при подключении одного светодиода. В данном случае используются резисторы с сопротивлением 220 Ом.

Рассмотрим, как необходимо выполнить подключение между отладочным модулем и платой со светодиодным кубом. Подключения указаны в таблице 1. Также следует проверить подключения с помощью рис. 8.

Таблица 1. Подключения между модулями

Номер виводу на плате з кубом	**Вивод на модуле**
1	PC1
2	PC2
3	PC3
4	GND
5	PC0
6	PA7
7	PA6
8	PA5
9	PA4
10	PA3
11	PA2
12	PA1
13	PA0

Программная реализация эффекта вращающейся плоскости представлена далее. Основным моментом, на который следует обратить внимание, является инициализация портов ввода/вывода. Для этого используется экземпляр структуры типа *GPIO_InitTypeDef*. Вначале выполняется инициализация структуры, после чего устанавливаются параметры инициализации порта: режим работы, скорость, выводы, использование подтягивающих резисторов. После установки необходимых значений вызывается функция инициализации на основе значений, которые содержит структура – *GPIO_Init*. Также важным моментом является вызов функции включения тактирования устройства. Это важно, поскольку без включения тактирования устройства ни один периферийный модуль не будет выполнять заданных функций. Подключение к тактированию выполняется вызовом функции типа *RCC_AxxxPeriphClockCmd* (вместо *xxx* подставить обозначение шины, к которой подключается данное устройство, – оно может иметь значения AHB, APB1, APB2; значение можно найти по документации или с помощью программного кода, найдя определение

RCC_AxxxPeriph_yyy, где *yyy* – устройство, которое необходимо подключить к тактированию, например, *GPIOA*)

Для повышению понятности программного кода используются определения с помощью директив препроцессора *#define* для обозначения выходов, предназначенных для управления светодиодами, значений задержек. Задержки в данном случае реализованы на основе таймера, но, поскольку таймеры не являются основной темой при выполнении данной работы, то достаточно просто использовать готовую функцию задержки.

```
#include "stm32f30x_rcc.h"
#include "stm32f30x_gpio.h"
#include "stm32f30x_tim.h"

#define PORTA_LED 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 PORTC_LED GPIO_Pin_0

#define PORTC_LEVELS GPIO_Pin_1 | \
                     GPIO_Pin_2 | \
                     GPIO_Pin_3

#define LAYER1 GPIO_Pin_1
#define LAYER2 GPIO_Pin_2
#define LAYER3 GPIO_Pin_3

#define SMALL_DELAY 10
#define BIG_DELAY 1000

void init(void);
void delay_ms(int ms);
void set_layer_data(int layer, int data);
void turn_needed_layer(int layer);
void set_pict_by_layer(int data1, int data2, int data3);
void set_pict_by_layer_for(int time, int data1, int data2, int data3);

int main() {
	init();
	delay_ms(2);
	while(1) {
		set_pict_by_layer_for(BIG_DELAY, 0x7, 0x38, 0x1C0);
		set_pict_by_layer_for(BIG_DELAY, 0x38, 0x38, 0x38);
		set_pict_by_layer_for(BIG_DELAY, 0x1C0, 0x38, 0x7);
		set_pict_by_layer_for(BIG_DELAY, 0x0, 0x1ff, 0x0);
		set_pict_by_layer_for(BIG_DELAY, 0x7, 0x38, 0x1C0);
		set_pict_by_layer_for(BIG_DELAY, 0x38, 0x38, 0x38);
		set_pict_by_layer_for(BIG_DELAY, 0x1C0, 0x38, 0x7);
		set_pict_by_layer_for(BIG_DELAY, 0x0, 0x1ff, 0x0);
	}
}

void set_pict_by_layer_for(int time, int data1, int data2, int data3) {
	int times;
	int i;
	times = time / 30;
```

```
	for ( i = 0; i < times; ++i ) {
		set_pict_by_layer(data1, data2, data3);
	}
}

void set_pict_by_layer(int data1, int data2, int data3) {
	set_layer_data(LAYER1, data1);
	delay_ms(SMALL_DELAY);
	set_layer_data(LAYER2, data2);
	delay_ms(SMALL_DELAY);
	set_layer_data(LAYER3, data3);
	delay_ms(SMALL_DELAY);
}

void set_layer_data(int layer, int data) {
	turn_needed_layer(layer);
	GPIO_Write(GPIOA, data & 255);
	GPIO_WriteBit(GPIOC, GPIO_Pin_0, ((data & 256) > 0) ? Bit_SET : Bit_RESET);
}

void turn_needed_layer(int layer) {
	int i;
	for ( i = 1; i <= 3; i++ ) {
		GPIO_WriteBit(GPIOC, 1 << i, ((1 << i) == layer) ? Bit_SET : Bit_RESET);
	}
}

void delay_ms(int ms) {
	TIM2->CNT = 0;
	TIM2->PSC = 16000;
	TIM2->ARR = ms;
	TIM_Cmd(TIM2, ENABLE);
	while (TIM_GetFlagStatus(TIM2, TIM_FLAG_Update) == RESET);
	TIM_ClearFlag(TIM2, TIM_FLAG_Update);
	TIM_Cmd(TIM2, DISABLE);
}

void init(void) {
	GPIO_InitTypeDef gpio_init;
	TIM_TimeBaseInitTypeDef time_init;

	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);
	GPIO_StructInit(&gpio_init);
	gpio_init.GPIO_Pin = PORTA_LED;
	gpio_init.GPIO_Mode = GPIO_Mode_OUT;
	gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
	gpio_init.GPIO_OType = GPIO_OType_PP;
	GPIO_Init(GPIOA, &gpio_init);

	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOC, ENABLE);
	gpio_init.GPIO_Pin = PORTC_LED | PORTC_LEVELS;
	GPIO_Init(GPIOC, &gpio_init);

	RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
	TIM_TimeBaseStructInit(&time_init);
	time_init.TIM_CounterMode = TIM_CounterMode_Up;
	time_init.TIM_RepetitionCounter = 0;
	TIM_TimeBaseInit(TIM2, &time_init);
}
```

Для упрощения процесса создания эффектов реализовано несколько функций:

1. Включение необходимого уровня.

2. Включение светодиодов на выбранном уровне.

3. Переключение между уровнями с установкой необходимых уровней сигналов для изменения изображения.

4. Задержка для контроля временных промежутков при отображении.

Реализация этих функций позволяет создать визуальный эффект (в данном примере – вращение плоскости) при помощи нескольких вызовов функций в цикле главной функции. Для этого необходимо определить, какие числовые значения соответствуют включению светодиодов на уровне и установить правильную последовательность их передачи на уровни.

Задание

Реализовать эффект вращения плоскости вокруг оси, размещенной диагонально.

Лабораторная работа №2
Работа с внешними прерываниями

Цель работы: научится использовать внешние прерывания для выполнения программных функций по внешнему сигналу.

Оборудование: отладочный модуль STM32F3 Discovery, модуль с 3 тактовыми кнопками для подключения к плате.

Теоретическая часть

Прерывания – важное событие в работе микроконтроллера, которое может приводить к приостановке выполнения основной программы и обработке данного прерывания. Работа с прерываниями – важная и основная составляющая при написании программ для микроконтроллеров.

В данной работе рассмотрим использование внешних прерываний. Внешние прерывания – прерывания, которые происходят вследствие изменения логического уровня сигнала на одном из выводов порта ввода/вывода. Такой тип внешних прерываний является наиболее распространенным и используется в STM32F303VC.

Поскольку до этого момента прерывания не использовались, то следует отметить особенности использования прерываний для данного аппаратного обеспечения. Контроль работы с прерываниями выполняется с помощью встроенного контроллера вектора прерываний (Nested Vector Interrupt Controller, NVIC). С его помощью выполняется разрешение и запрет прерываний, установка приоритетов прерываний.

Управление внешними прерываниями как отдельным видом прерываний выполняется с помощью модуля SYSCFG, а также модулями EXTI.

Пример программы

В данном примере рассмотрим простой пример реакции на нажатие кнопки с помощью внешних прерываний. При нажатии кнопки,

предназначенной для программирования пользователем, происходит поочередное переключение двух светодиодов (синего и красного на плате).

Для программной реализации полезно знать, что имена всех обработчиков прерываний можно найти в ассемблерном файле *startup_stm32f30x.s*. обработчик прерываний представляет собой функцию, которая не принимает параметров и не возвращает значений, поэтому в обоих случаях следует указать *void*. Имя прерывания можно найти в файле *stm32f30x.c*, например, EXTI0_IRQn; общей чертой является то, что все имена заканчиваются на IRQn.

По сравнению с предыдущим примером добавляется использование программных модулей для управления внешними прерываниями, для управления вектором прерываний и модулем SYSCFG.

Программная реализация примера приведена ниже. В ней следует обратить внимание на использование уже знакомой последовательности выполнения инициализации периферийных устройств в составе микроконтроллера, а также на то, как именно выполняется реализация обработчика прерываний (он представляет собой функцию).

```
#include "stm32f30x_rcc.h"
#include "stm32f30x_gpio.h"
#include "stm32f30x_exti.h"
#include "stm32f30x_misc.h"
#include "stm32f30x_syscfg.h"

typedef void (*action)(void);
action button_action;
void turn_on_red(void);

void turn_on_blue(void) {
	GPIO_SetBits(GPIOE, GPIO_Pin_8);
	GPIO_ResetBits(GPIOE, GPIO_Pin_9);
	button_action = turn_on_red;
}

void turn_on_red(void) {
	GPIO_SetBits(GPIOE, GPIO_Pin_9);
	GPIO_ResetBits(GPIOE, GPIO_Pin_8);
	button_action = turn_on_blue;
}

void EXTI0_IRQHandler(void) {
	EXTI_ClearITPendingBit(EXTI_Line0);
	button_action();
}

void init(void);

int main() {
	init();
	button_action = turn_on_blue;
	while(1) {
	}
}

void init(void) {
	GPIO_InitTypeDef gpio_init;
	EXTI_InitTypeDef exti_init;
```

```
	NVIC_InitTypeDef nvic_init;

	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOE, ENABLE);
	GPIO_StructInit(&gpio_init);
	gpio_init.GPIO_Mode = GPIO_Mode_OUT;
	gpio_init.GPIO_Pin = GPIO_Pin_8 | GPIO_Pin_9;
	gpio_init.GPIO_PuPd = GPIO_PuPd_NOPULL;
	gpio_init.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init(GPIOE, &gpio_init);

	RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
	exti_init.EXTI_Line = EXTI_Line0;
	exti_init.EXTI_LineCmd = ENABLE;
	exti_init.EXTI_Mode = EXTI_Mode_Interrupt;
	exti_init.EXTI_Trigger = EXTI_Trigger_Rising;
	EXTI_Init(&exti_init);

	SYSCFG_EXTILineConfig(EXTI_PortSourceGPIOA, EXTI_PinSource0);

	nvic_init.NVIC_IRQChannel = EXTI0_IRQn;
	nvic_init.NVIC_IRQChannelCmd = ENABLE;
	nvic_init.NVIC_IRQChannelSubPriority = 1;
	nvic_init.NVIC_IRQChannelPreemptionPriority = 1;
	NVIC_Init(&nvic_init);
}
```

Особенностью предложенной программной реализации является использование указателя на функцию для выполнения действий в обработчике внешнего прерывания. Для этого описан новый тип указателя, *action*, и описана переменная данного типа. Упомянутая переменная инициализирована таким образом, чтобы ссылаться на одну из функций переключения светодиодов. В этих функциях происходит присваивание указателю значения адреса другой функции. Таким образом, обеспечивается изменение функциональности при происхождении внешнего прерывания.

Следует отметить, что, в соответствии с 10 правилами Хольцмана, которые описывают рекомендации к разработке программного обеспечения для встаиваемых систем, использование указателей на функции является практикой, от которой необходимо отказываться при построении сложных систем, которые требуют высокой надежности, однако данный пример не является сложным, а использование указателей на функции упрощает реализацию изменения функциональности и повышает понятность программного кода. По аналогии с объектно-ориентированным программированием, можно сказать, что таким образом обеспечивается полиморфное поведение системы.

Задание

Подключить к отладочной плате модуль с 3 кнопками и реализовать переключение светодиодов по нажатию кнопок на модуле. Светодиоды выбрать самостоятельно.

Лабораторная работа №3
Работа с АЦП. Устройства индикации

Цель работы: исследовать принцип работы с АЦП в составе микроконтроллера, продемонстрировать использование динамической индикации при отображении результатов измерений.

Оборудование: отладочный модуль, модуль 2 двухразрядным семисегментным индикатором, потенциометр, вольтметр.

Теоретическая часть

Аналогово-цифровой преобразователь (АЦП) – устройство, предназначенное для преобразования величины входного напряжения в числовое значение. В микроконтроллере, который используется на отладочном модуле, доступны 4 АЦП с большим количеством каналов для подключения сигналов. Есть возможность настройки разрядности АЦП, установив значения 6, 8, 10 или 12 бит. Скорость преобразования для быстрых и медленных каналов входного напряжения по документации оценивается следующими значениями:

- для быстрых – 0,19 мкс для 12-битного преобразования (5,1 Мвыборки/с);
- для медленных – 0,21 мкс для 12 битного преобразования (4,9 Мвыборки/с).

Еще большей скорости преобразования можно добиться, уменьшив значение разрядности для преобразования.

Кроме того, присутствуют внутренние подключения между другими периферийными устройствами (внутренний датчик температуры, операционные усилители) и АЦП, что упрощает их практическое использование при разработке и проверке схем проектов.

Пример программы

В качестве примера работы с АЦП рассмотрим реализацию измерения напряжения на входе АЦП и вывода полученной информации на двухразрядный семисегментный индикатор (в данном примере используется индикатор с общим анодом, который не может отображать десятичную точку).

В качестве источника входного напряжения воспользуемся подключением потенциометра. Таким образом, подключенные модули будут выглядеть так, как показано на рис. 9.

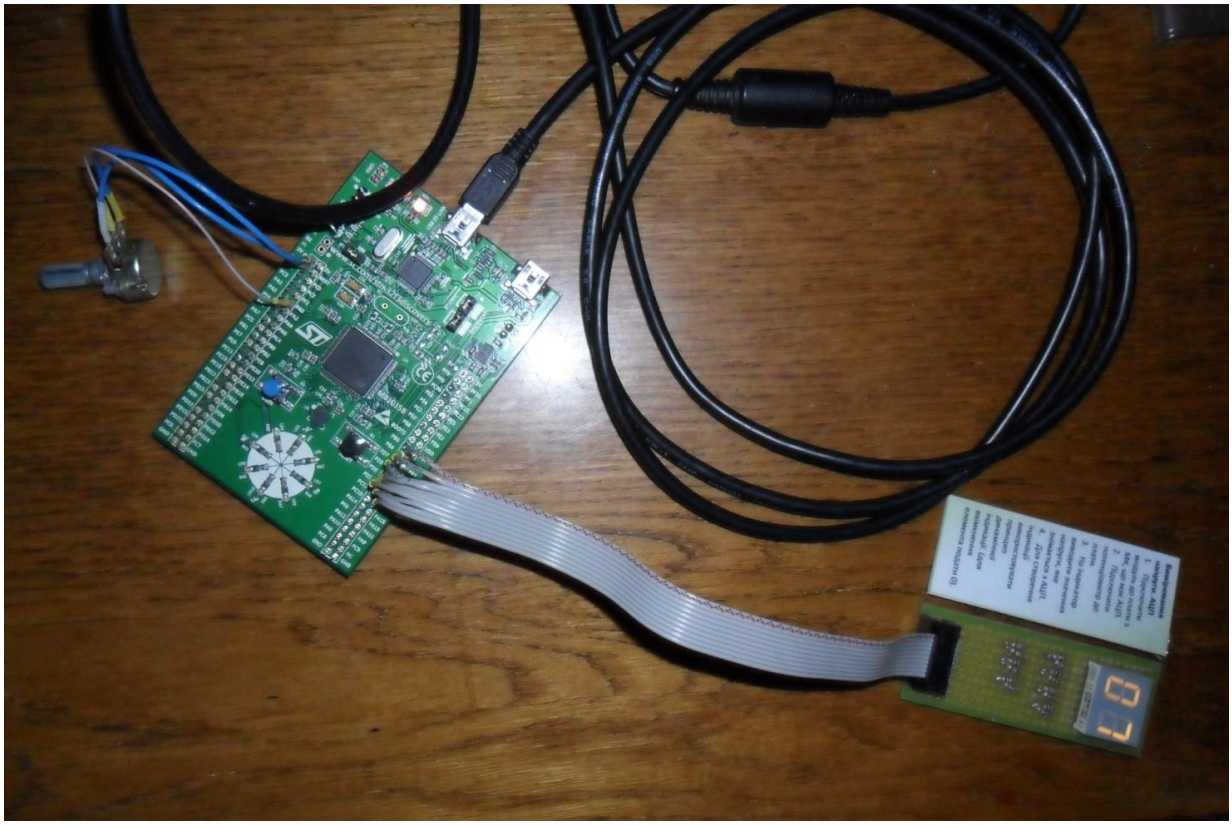


Рис. 9. Подключение модулей при выполнении задания

Для уменьшения количества используемых контактов реализован принцип динамической индикации. Выполняется попеременное включение и выключение конкретного разряда индикатора с выводом информации и задержкой. Для реализации динамической индикации используется 9 линий ввода/вывода:

- 2 для выбора индикатора, на который выводится значение;
- 7 остальных для вывода информации.

Для программной реализации удобно, чтобы эти 2 группы принадлежали разным портам ввода/вывода, а также при подключении располагались по возрастанию. Тем не менее, это не является обязательным требованием, поскольку лишь немного усложняет процесс программирования из-за необходимости составления таблиц соответствия для выводов портов и значений, которые необходимо вывести.

Программная реализация примера приведена далее.

```
#include "stm32f30x_rcc.h"
#include "stm32f30x_gpio.h"
#include "stm32f30x_adc.h"

#define DIGIT_PORT GPIOD
#define DIGIT_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 DIGIT_SELECTION_PORT GPIOB
#define DIGIT_SELECTION_PINS GPIO_Pin_4 | GPIO_Pin_5
#define FIRST_DIGIT_PIN GPIO_Pin_4
#define SECOND_DIGIT_PIN GPIO_Pin_5

int dig_code[] = { 0x01, 0x3d, 0x12, 0x18, 0x2c, 0x88, 0x80, 0x39, 0x00, 0x08 };

void init(void);
void delay(int count);
void show_converted_value(uint16_t convValue);
void turn_on_first_digit(void);
void turn_on_second_digit(void);
void turn_off_first_digit(void);
void turn_off_second_digit(void);
void set_digit(int);

int main() {
	static uint16_t convertedValue;
	init();
	while (!ADC_GetFlagStatus(ADC1, ADC_FLAG_RDY));
	ADC_StartConversion(ADC1);
	while(1) {
		delay(10000);
		while (ADC_GetFlagStatus(ADC1, ADC_FLAG_EOC) == RESET);
		convertedValue = ADC_GetConversionValue(ADC1);
		show_converted_value(convertedValue);
	}
}

void init(void) {
	GPIO_InitTypeDef gpio_init;
	ADC_InitTypeDef adc_init;
	ADC_CommonInitTypeDef adc_common;

	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);
	GPIO_StructInit(&gpio_init);
	gpio_init.GPIO_Pin = GPIO_Pin_0;
	gpio_init.GPIO_Mode = GPIO_Mode_AN;
	GPIO_Init(GPIOA, &gpio_init);
	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOD | RCC_AHBPeriph_GPIOB, ENABLE);
	gpio_init.GPIO_Mode = GPIO_Mode_OUT;
	gpio_init.GPIO_OType = GPIO_OType_PP;
	gpio_init.GPIO_Pin = DIGIT_PINS;
	gpio_init.GPIO_PuPd = GPIO_PuPd_DOWN;
	gpio_init.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init(DIGIT_PORT, &gpio_init);

	gpio_init.GPIO_Pin = DIGIT_SELECTION_PINS;
	gpio_init.GPIO_PuPd = GPIO_PuPd_UP;
	GPIO_Init(DIGIT_SELECTION_PORT, &gpio_init);

	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_ADC12, ENABLE);
	RCC_ADCCLKConfig(RCC_ADC12PLLCLK_Div2);

	ADC_VoltageRegulatorCmd(ADC1, ENABLE);
	delay(10000);
	ADC_SelectCalibrationMode(ADC1, ADC_CalibrationMode_Single);
	ADC_StartCalibration(ADC1);
	while (ADC_GetCalibrationStatus(ADC1) == RESET);
	ADC_GetCalibrationValue(ADC1);

	ADC_StructInit(&adc_init);
	adc_init.ADC_DataAlign = ADC_DataAlign_Right;
	adc_init.ADC_Resolution = ADC_Resolution_12b;
```

```
	adc_init.ADC_ContinuousConvMode = ADC_ContinuousConvMode_Enable;
	adc_init.ADC_NbrOfRegChannel = 1;
	ADC_Init(ADC1, &adc_init);

	ADC_CommonStructInit(&adc_common);
	adc_common.ADC_Mode = ADC_Mode_Independent;
	ADC_CommonInit(ADC1, &adc_common);

	ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1, ADC_SampleTime_7Cycles5);

	ADC_Cmd(ADC1, ENABLE);
}

void delay(int count) {
	for ( ; count > 0; count--);
}

void show_converted_value(uint16_t convValue) {
	char first;
	char second;
	int i;
	double voltageValue;
	voltageValue = (double) convValue * 3.0 / 4095;
	first = (int)voltageValue;
	second = ((int)(voltageValue * 10) % 10);
	for (i = 0; i < 500; i++) {
		turn_off_first_digit();
		turn_on_second_digit();
		set_digit(first);
		delay(10000);
		turn_on_first_digit();
		turn_off_second_digit();
		set_digit(second);
		delay(10000);
	}
}

void turn_on_first_digit(void) {
	GPIO_SetBits(DIGIT_SELECTION_PORT, FIRST_DIGIT_PIN);
}

void turn_on_second_digit(void) {
	GPIO_SetBits(DIGIT_SELECTION_PORT, SECOND_DIGIT_PIN);
}

void turn_off_first_digit(void) {
	GPIO_ResetBits(DIGIT_SELECTION_PORT, FIRST_DIGIT_PIN);
}

void turn_off_second_digit(void) {
	GPIO_ResetBits(DIGIT_SELECTION_PORT, SECOND_DIGIT_PIN);
}

void set_digit(int index) {
	GPIO_Write(DIGIT_PORT, dig_code[index]);
}
```

Также, как для аналогового компаратора и операционного усилителя, вход АЦП –это вывод, который работает в аналоговом режиме. Далее наибольшее внимание следует уделить тому, как выполняется настройка АЦП. Это требует сравнительно большого количества шагов, по сравнению с рассмотренными ранее устройствами. Следует отметить то, что выполняется

калибровка АЦП, а также необходимо использовать две структуры для инициализации разных типов: первая для инициализации конкретных свойств АЦП, а другая – для инициализации общих характеристик АЦП в составе контроллера. Проверка готовности выполняется перед входом в главный цикл программы. Там же начинается выполнение преобразования (вызов функции *ADC_StartConversion*). Поскольку при инициализации был задан параметр выполнения постоянного преобразования, то выполнение дальнейших преобразований не требует повторного вызова данной функции.

Задание

Детально ознакомьтесь с предложенным примером и продемонстрируйте его работу. Для выбранной схемы подключения составить таблицу соответствия выводов и сегментов на индикаторе. При условии изменения последовательности подключения (собственный вариант), продемонстрировать новые значения в массиве для вывода. Переместить реализацию динамической индикации из функции в главный цикл программы.

Лабораторная работа №4
Организация последовательной передачи информации. Интерфейс UART

Цель работы: исследовать возможности передачи данных между устройствами с помощью последовательного интерфейса UART, организовать связь между ПК и микроконтроллером.

Оборудование: отладочный модуль, модуль с микросхемой преобразователя уровней MAX3232, переходник USB/COM.

Теоретическая часть

Универсальный асинхронный приемо-передатчки (англ. Universal Asynchronous Receiver-Transmitter, UART) – один из наиболее распространенных интерфейсов, который встречается в микроконтроллерах. Распространенность объясняется простотой и нетребовательностью к ресурсам: для организации передачи данных используются всего 2 вывода (обычно, они называются Rx и Tx). При подключении двух устройств UART линии передачи и приема соединяются разноименно. Свое развитие получил в том, что некоторые реализации интерфейса в периферии поддерживают синхронный режим передачи – в таком случае модуль называю USART. Он использует третью линию для подачи тактового сигнала. Кроме того, могут подключаться другие выводы для выполнения служебных функций.

Пример программы

В практическом примере рассмотрим работу с UART для приема данных.

Для выполнения задания понадобится кабель-переходник USB/COM, а также модуль с микросхемой преобразователя уровней MAX3232, которая обеспечивает преобразование из уровней сигнала COM-порта к уровням

TTL-логики, и может работать от источника питания 3 В (такой вывод присутствует на отладочной плате). Схема подключения между переходником, модуле и платой представлена на рис. 10.

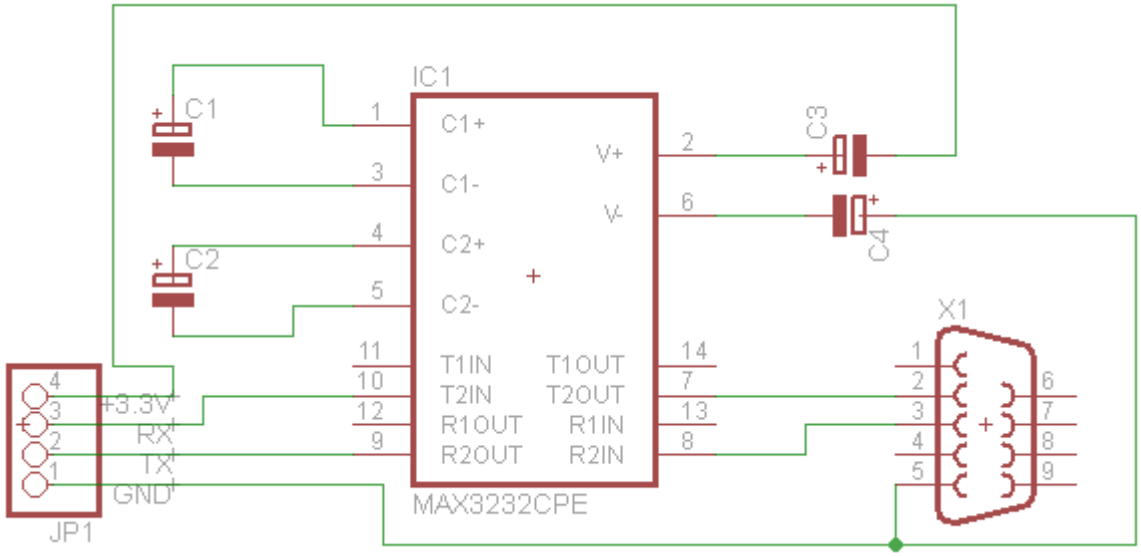


Рис. 10. Схема подключения преобразователя уровней

Обратите внимание на то, что схема на рисунке является упрощенной, поскольку не отображает выводы 15 и 16.

Для проверки программной реализации воспользуемся терминальной программой *putty*. Перейти в режим отладки (комбинация клавиш Ctrl+F5 или команда меню *Debug\Start/Stop Debug Session*). При отправке символа продемонстрировать прохождение точки останова и полученный символ.

Программная реализация, которая используется при выполнения вышеописанных действий, представлена ниже.

```
#include "stm32f30x_rcc.h"
#include "stm32f30x_gpio.h"
#include "stm32f30x_usart.h"
#include "stm32f30x_misc.h"

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

void USART1_IRQHandler(void) {
	char recData;
	if (USART_GetITStatus(USART1, USART_IT_RXNE) == SET) {
		USART_ClearITPendingBit(USART1, USART_IT_RXNE);
		recData = USART_ReceiveData(USART1);
		recData++;
	}
}

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

void init(void) {
	GPIO_InitTypeDef gpio_init;
```

```
	USART_InitTypeDef usart_init;
	NVIC_InitTypeDef nvic_init;
	EXTI_InitTypeDef exti_init;

	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);
	GPIO_StructInit(&gpio_init);
	gpio_init.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_10; // 9 - Tx, 10 - Rx
	gpio_init.GPIO_Mode = GPIO_Mode_AF;
	gpio_init.GPIO_Speed = GPIO_Speed_50MHz;
	GPIO_Init(GPIOA, &gpio_init);

	GPIO_PinAFConfig(GPIOA, GPIO_PinSource9, GPIO_AF_7);
	GPIO_PinAFConfig(GPIOA, GPIO_PinSource10, GPIO_AF_7);

	nvic_init.NVIC_IRQChannel = USART1_IRQn;
	nvic_init.NVIC_IRQChannelCmd = ENABLE;
	nvic_init.NVIC_IRQChannelSubPriority = 1;
	nvic_init.NVIC_IRQChannelPreemptionPriority = 1;
	NVIC_Init(&nvic_init);

	RCC_APB2PeriphClockCmd(RCC_APB2Periph_USART1, ENABLE);
	USART_StructInit(&usart_init);
	usart_init.USART_BaudRate = 19200;
	usart_init.USART_WordLength = USART_WordLength_8b;
	usart_init.USART_StopBits = USART_StopBits_1;
	usart_init.USART_Parity = USART_Parity_No;
	usart_init.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;
	usart_init.USART_HardwareFlowControl = USART_HardwareFlowControl_None;
	USART_Init(USART1, &usart_init);
	USART_ITConfig(USART1, USART_IT_RXNE, ENABLE);
	USART_Cmd(USART1, ENABLE);
}

void delay(int count) {
	for ( ; count > 0; count--);
}
```

Прием символов выполняется с помощью прерывания. Следует отметить, что кроме настройки прерывания в контроллере прерываний, следует также разрешить прерывания для самого устройства. Это делается с помощью бита в регистре управления или функцией *USART_ITConfig()* из стандартной библиотеки. Следует это запомнить, поскольку на этом примере можно видеть, какие имена имеют функции для настройки прерываний отдельных устройств: *Periph_ITConfig()*, где *Periph* – тип устройства. В качестве параметров чаще всего выступают название периферийного устройства, сокращенное название прерывания и новое состояние бита, который отвечает за разрешение прерывания, соответственно, разрешено или запрещено.

Задание

Ознакомится с примером программы. На основе примера реализовать функциональность включения светодиодов при получении определенного управляющего сигнала (символа). Значения символов и светодиодов выбрать самостоятельно (8 светодиодов на плате подключены в выводам PE8-PE15).

Лабораторная работа №5
Использование аналогового компаратора

Цель работы: исследовать возможность использования аналогового компаратора в составе микроконтроллера на отладочной плате.

Оборудование: отладочная плата, мультиметр или вольтметр.

Теоретическая часть

Компаратор – аналоговое электронное устройство, предназначенное для сравнения величин сигналов. Обычно он имеет два входа, на которые подаются сигналы, которые сравниваются. Сигнал на выходе зависит от результатов сравнения: если первый сигнал больше – на выходе лог. 1, если наоборот – лог. 0.

STM32F303VC содержит 7 компараторов, которые могут использоваться как самостоятельные устройства или совместно с таймерами. Они могут использоваться для разных функций в составе самого устройства:

- пробуждение из режима низкого потребления энергии по аналоговому сигналу;
- сравнение сигналов;

Таким образом, данное устройство используется и как самостоятельный модуль, так и в составе аппаратного комплекса.

Пример программы

В качестве примера рассмотрим реализацию переключения уровня сигнала из 0 в 1 при превышении заданного уровня напряжения. Это фактически и будет использование простого компаратора. Благодаря тому, что есть возможность выбора сигнала для подачи напряжения, необходимо подавать сигнал только на один из входов компаратора. В данном случае в качестве такого источника выбрано значение половины напряжения питания микроконтроллера.

Программная реализация примера представлена далее.

```
#include "stm32f30x_rcc.h"
#include "stm32f30x_gpio.h"
#include "stm32f30x_syscfg.h"
#include "stm32f30x_comp.h"
#include "stm32f30x_misc.h"
#include "stm32f30x_exti.h"

void init(void);

void COMP1_2_3_IRQHandler(void) {
	int foo;
	uint32_t level;
	if (EXTI_GetITStatus(EXTI_Line21) == SET) {
		EXTI_ClearITPendingBit(EXTI_Line21);
		foo = 0;
		level = COMP_GetOutputLevel(COMP_Selection_COMP1);
		if (level == COMP_OutputLevel_High)
			foo = 1;
	}
```

```
}

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

void init(void) {
	GPIO_InitTypeDef gpio_init;
	COMP_InitTypeDef comp_init;
	NVIC_InitTypeDef nvic_init;
	EXTI_InitTypeDef exti_init;

	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);
	GPIO_StructInit(&gpio_init);
	gpio_init.GPIO_Mode = GPIO_Mode_AN;
	gpio_init.GPIO_PuPd = GPIO_PuPd_DOWN;
	gpio_init.GPIO_Pin = GPIO_Pin_1;
	GPIO_Init(GPIOA, &gpio_init);

	gpio_init.GPIO_Mode = GPIO_Mode_AF;
	gpio_init.GPIO_Pin = GPIO_Pin_0;
	GPIO_Init(GPIOA, &gpio_init);
	GPIO_PinAFConfig(GPIOA, GPIO_PinSource0, GPIO_AF_8);

	RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
	COMP_StructInit(&comp_init);
	comp_init.COMP_Mode = COMP_Mode_HighSpeed;
	comp_init.COMP_OutputPol = COMP_OutputPol_NonInverted;
	comp_init.COMP_NonInvertingInput = COMP_NonInvertingInput_IO1;
	comp_init.COMP_InvertingInput = COMP_InvertingInput_1_2VREFINT;
	COMP_Init(COMP_Selection_COMP1, &comp_init);
	COMP_Cmd(COMP_Selection_COMP1, ENABLE);

	exti_init.EXTI_Line = EXTI_Line21;
	exti_init.EXTI_LineCmd = ENABLE;
	exti_init.EXTI_Mode = EXTI_Mode_Interrupt;
	exti_init.EXTI_Trigger = EXTI_Trigger_Rising;
	EXTI_Init(&exti_init);

	nvic_init.NVIC_IRQChannel = COMP1_2_3_IRQn;
	nvic_init.NVIC_IRQChannelCmd = ENABLE;
	nvic_init.NVIC_IRQChannelSubPriority = 1;
	nvic_init.NVIC_IRQChannelPreemptionPriority = 1;
	NVIC_Init(&nvic_init);
}
```

При работе с компаратором видно, что есть возможность подключить к входам различные значения сигналов. По аналогии с операционным усилителем входы компаратора называются инвертирующий и неинвертирующий.

Прерывания от аналогового компаратора в данном микроконтроллере реализованы как внешние прерывания (линия 21). Поэтому для их использования необходимо подключить те же модули, что и для обычного внешнего прерывания. Кроме того, за работу компараторов отвечает уже упоминавшийся модуль SYSCFG, поэтому при использовании компараторов подача тактирования на данный модуль является обязательным.

Задание

На основе примера, который предлагается для использования, а также документации для микроконтроллера, разработать программу, которая будет выполнять функцию контроля за уровнем напряжения с индикацией: пока напряжение ниже заданного уровня, должен быть включен один из светодиодов, при переходе через условную допустимую границу, включать другой светодиод, сообщая об этом.

Лабораторная работа №6
Операционный усилитель

Цель работы: исследовать возможности платы при работе аналоговыми сигналами, особенности использования операционного усилителя.

Оборудование: отладочная плата, набор перемычек для соединения выводов, мультиметр или вольтметр.

Теоретическая часть

В состав микроконтроллера STM32F303VC также входят четыре операционных усилителя.

Предусматривается использование усилителя в двух режимах в зависимости от использования резисторов в схеме:

- самостоятельный режим (необходимо подключать внешние резисторы, которые определяют уровень усиления);
- программный режим (выполняется подключение к внутренним резисторам, которые определяют уровень усиления);

С помощью установленного операционного усилителя (ОУ) можно построить такие известные схемы, как дифференциальный усилитель или неинвертирующий усилитель. В реальном использовании это позволит избежать установки дополнительного устройства при условии достаточности параметров, которые обеспечивает ОУ в составе контроллера. Тем не менее, производитель, предостерегает, что существуют трудности использования ОУ , поэтому не рекомендует использовать эти модули в сложных системах.

Режим, при котором управление резисторами, подключенными к ОУ, выполняется программно, по документации обозначается как Programmable Gain Amplifier (PGA). Фактически в данном режиме имеем дело с инвертирующим усилителем. Схема подключения для неинвертирующего усилителя представлена на рис. 11.

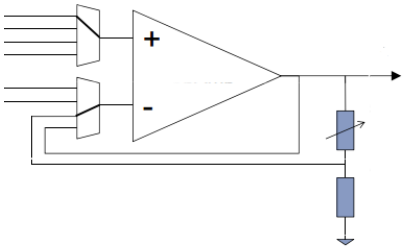


Рис. 11. Схема внутреннего подключения при работе в режиме неинвертирующего усилителя

Значение выходного напряжения при такой схеме подключения определяется как

$$V_{out} = V_{in} \cdot \left(1 + \frac{R_2}{R_1}\right),$$

где V_{in} – входное напряжение;

R_1, R_2 – значения сопротивления резисторов.

Как понятно из рисунка, значение сопротивления второго резистора можно программно указать из списка доступных значений. Для данного режима доступны значения коэффициента усиления (единица уже учтена): 2, 4, 8,16.

Пример программы

Ознакомление с работой ОУ в составе микроконтроллера проведем на примере, который также использует ЦАП для генерации напряжения, которое с помощью ОУ в режиме неинвертирующего усилителя увеличивается в 2 раза.

Поскольку использование выводов портов ввода/вывода не является дополнительной функцией их работы, то найти, какие именного выводы будут использоваться при работе ОУ можно воспользоваться плагином MicroXplorer для среды разработки Eclipse. В нем можно выбрать устройство и узнать, как именно используется вывод микроконтроллера (рис. 12).

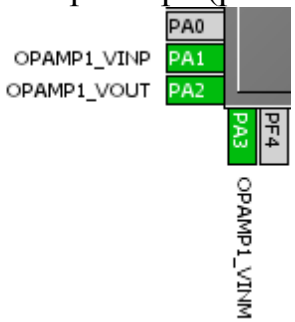


Рис. 12. Функциональное назначение выводов при использовании ОУ

Больше всего следует обратить внимание на настройки ОУ. Для этого следует указать, какие линии используются в качестве входов. В нашем случае указываем, что следует использовать в качестве неинвертирующего входа 4 линию, которой соответствует 1 вывод порта А. На эту линию подается

напряжение, которое генерируется с помощью ЦАП. Значение напряжение на выходе 4 порта А – приблизительно 0,77 В. Соединив с помощью перемычки оба пина, следует проверить напряжение на выходе АЦП – пине 2 порта А. Показатели вольтметра составят приблизительно 1,55 В. Следует обратить внимание на то, что все перечисленные выше пины настроены на работу в аналоговом режиме.

Программная реализация указанной функциональности приведена далее.

```
#include "stm32f30x_rcc.h"
#include "stm32f30x_gpio.h"
#include "stm32f30x_opamp.h"
#include "stm32f30x_syscfg.h"
#include "stm32f30x_dac.h"

void init_opamp(void);
void init_dac(void);

int main() {
	init_dac();
	init_opamp();
	while(1);
}

void init_opamp(void) {
	GPIO_InitTypeDef gpio_init;
	OPAMP_InitTypeDef opamp_init;

	RCC_AHBPeriphClockCmd(RCC_AHBPeriph_GPIOA, ENABLE);
	GPIO_StructInit(&gpio_init);
	gpio_init.GPIO_Pin = GPIO_Pin_1 | GPIO_Pin_2 | GPIO_Pin_3;
	gpio_init.GPIO_Mode = GPIO_Mode_AN;
	GPIO_Init(GPIOA, &gpio_init);

	RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
	OPAMP_StructInit(&opamp_init);
	opamp_init.OPAMP_InvertingInput = OPAMP_InvertingInput_PGA;
	opamp_init.OPAMP_NonInvertingInput = OPAMP_NonInvertingInput_IO4;
	OPAMP_Init(OPAMP_Selection_OPAMP1, &opamp_init);
	OPAMP_PGAConfig(OPAMP_Selection_OPAMP1, OPAMP_OPAMP_PGAGain_2, OPAMP_PGAConnect_No);
	OPAMP_Cmd(OPAMP_Selection_OPAMP1, ENABLE);
}

void init_dac(void) {
	DAC_InitTypeDef dac_init;
	GPIO_InitTypeDef gpio_init;

	GPIO_StructInit(&gpio_init);
	gpio_init.GPIO_Mode = GPIO_Mode_AN;
	gpio_init.GPIO_Pin = GPIO_Pin_4;
	GPIO_Init(GPIOA, &gpio_init);

	RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
	DAC_StructInit(&dac_init);
	dac_init.DAC_WaveGeneration = DAC_WaveGeneration_None;
	dac_init.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
	dac_init.DAC_Trigger = DAC_Trigger_None;
	DAC_Init(DAC_Channel_1, &dac_init);

	DAC_Cmd(DAC_Channel_1, ENABLE);
	DAC_SetChannel1Data(DAC_Align_8b_R, 0x44);
}
```

Следует обратить внимание на то, что указанная функциональность реализована с помощью простой инициализации устройств в составе микроконтроллера. Среди этих устройств присутствует также ЦАП. Важно помнить, как именно реализуется подача необходимого расчетного напряжения ЦАП, устанавливаются значения в регистре данных, которое переводится в нужный уровень напряжения. Значения выходного напряжения на выходе ЦАП определяется на основе напряжения питания.

Задание

Ознакомится с предложенным примером и документацией относительно данного раздела. Задать напряжение, которое можно усилит при КУ = 16. Продемонстрировать, что при дальнейшем увеличении входного напряжения усиления становится невозможным. Для получения дополнительных баллов реализовать схему инвертирующего усилителя согласно документации.

ЛИТЕРАТУРА

1. STM32F3Discovery Discovery kit for STM32F303xx microcontrollers
2. RM0316 Reference manual
3. STM32F302xx, STM32F303xx
4. Гонин М. Компас на отладочной плате от STMicroelectronics // Новости электроники, №2. – 2013. – 30-38 с.
5. STM32F3DISCOVERY Discovery kit for STM32F303xx microcontrollers [Электронный ресурс]. Режим доступа: URL : http://www.st.com/web/catalog/tools/FM116/SC959/SS1532/PF254044. – Загол. с экрана.
6. STM32F3 Discovery kit firmware package [Электронный ресурс]. Режим доступа: URL : http://www.st.com/web/en/catalog/tools/PF258154. – Загол. с экрана.
7. Keil Tools by ARM [Электронный ресурс]. Режим доступа: URL : https://www.keil.com/download/product/. – Загол. с экрана.