МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное
образовательное учреждение высшего образования
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ РАДИОЭЛЕКТРОНИКИ И ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

Кафедра «Радиоэлектроника и телекоммуникации»

ПРОГРАММИРОВАНИЕ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ.
Лабораторный практикум по дисциплине

Учебно-методическое пособие
для студентов очной и заочной форм обучения направления
11.03.01 - Радиотехника
и специальности
11.05.01 - Радиоэлектронные системы и комплексы

Севастополь
СевГУ
2021

УДК 004.42(076.5)
ББК 32.973.202-018я73
П784

Р е ц е н з е н т:
И. Л. Афонин – д-р техн. наук, профессор

Ответственный за выпуск:
И. Л. Афонин – д-р техн. наук, профессор,
заведующий кафедрой «Радиоэлектроника и телекоммуникации»

Составители : А. А. Щекатурин, Е. П. Руднев

П784 Программирование вычислительных систем. Лабораторный практикум по дисциплине : учебно-методическое пособие для студентов очной и заочной форм обучения направления 11.03.01 - Радиотехника и специальности 11.05.01 - Радиоэлектронные системы и комплексы / Севастопольский государственный университет, Институт радиоэлектроники и информационной безопасности, кафедра «Радиоэлектроника и телекоммуникации»; составители: А. А. Щекатурин, Е. П. Руднев. – Севастополь : Изд-во СевГУ, 2021. – 63 с. – Текст : электронный.

Цель пособия: оказание помощи студентам в выполнении лабораторных работ по дисциплине «Программирование вычислительных систем» при изучении структуры ARM микроконтроллеров и получении навыков их программирования.

УДК 004.42(076.5)
ББК 32.973.202-018я73

Учебно-методическое пособие рассмотрено и рекомендовано к изданию на заседании методической комиссии Института радиоэлектроники и информационной безопасности, протокол № 9 от 25 марта 2021 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Дисциплина «Программирование вычислительных систем» изучается студентами направления 11.03.01 — «Радиотехника» и специальности 11.05.01 «Радиоэлектронные системы и комплексы» очной и заочной форм обучения в 6-м семестре.

Методические рекомендации по выполнению каждой лабораторной работы содержат следующие разделы: цель работы, теоретическую часть, порядок выполнения работы, рекомендации по оформлению отчета.

Порядок выполнения лабораторных работ

Лабораторные работы выполняются фронтальным методом, т. е. студенческая группа, разбитая на подгруппы, выполняет одну и ту же лабораторную работу, что позволяет синхронизировать изучение материала по дисциплине, а также дает возможность преподавателю работать со всей группой одновременно.

Перед выполнением лабораторной работы студенты обязаны:

— до начала занятия самостоятельно изучить методические указания к предстоящей лабораторной работе;

— в начале занятия представить и защитить отчет по предыдущей лабораторной работе;

— ответить на контрольные вопросы, связанные с выполняемой следующей работы.

Каждый студент выполняет лабораторную работу в соответствии с индивидуальным заданием на учебном отладочном стенде *Pinboard II*. Отладочный стенд построен на базе микроконтроллера *ARM Cortex-M3 STM32F103x8* и программируется с помощью персонального компьютера. Используемый программно–аппаратный комплекс позволяет осуществлять набор программы на языке высокого уровня С, выполнять ее отладку и компилирование, записывать программу в память отладочного стенда, после чего программа начинает автоматически выполняться. Правильность работы программы может быть проконтролирована по показаниям программируемых индикаторов отладочного стенда.

Защита отчета по лабораторной работе производится каждым студентом индивидуально на очередном лабораторном занятии. В процессе защиты проверяются: правильность оформления отчета, умение студентов пояснять выполненные расчеты, знание ими методики моделирования, умение пояснить полученные зависимости и обосновать выводы по работе.

Требования к отчету по лабораторной работе

Отчет по работе оформляется в соответствии с методическими указаниями по оформлению текстовых работ на кафедре РТ [1] и должен содержать:

— титульный лист;

— цель работы;

— задание;

— блок-схему алгоритма работы программы;

— программу на ассемблере;

— результаты работы программы;

— выводы по работе.

1. ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

1.1. Основные сведения о микроконтроллере *STM*32*F*103*C*8

Микроконтроллеры (МК) *STM*32 являются 32-битными микроконтроллерами, построенными по архитектуре *ARM* на основе процессора *ARM Cortex-M*3.

Структурная схема МК семейства *STM*32*F*103x8 приведена на рис. 1.1.

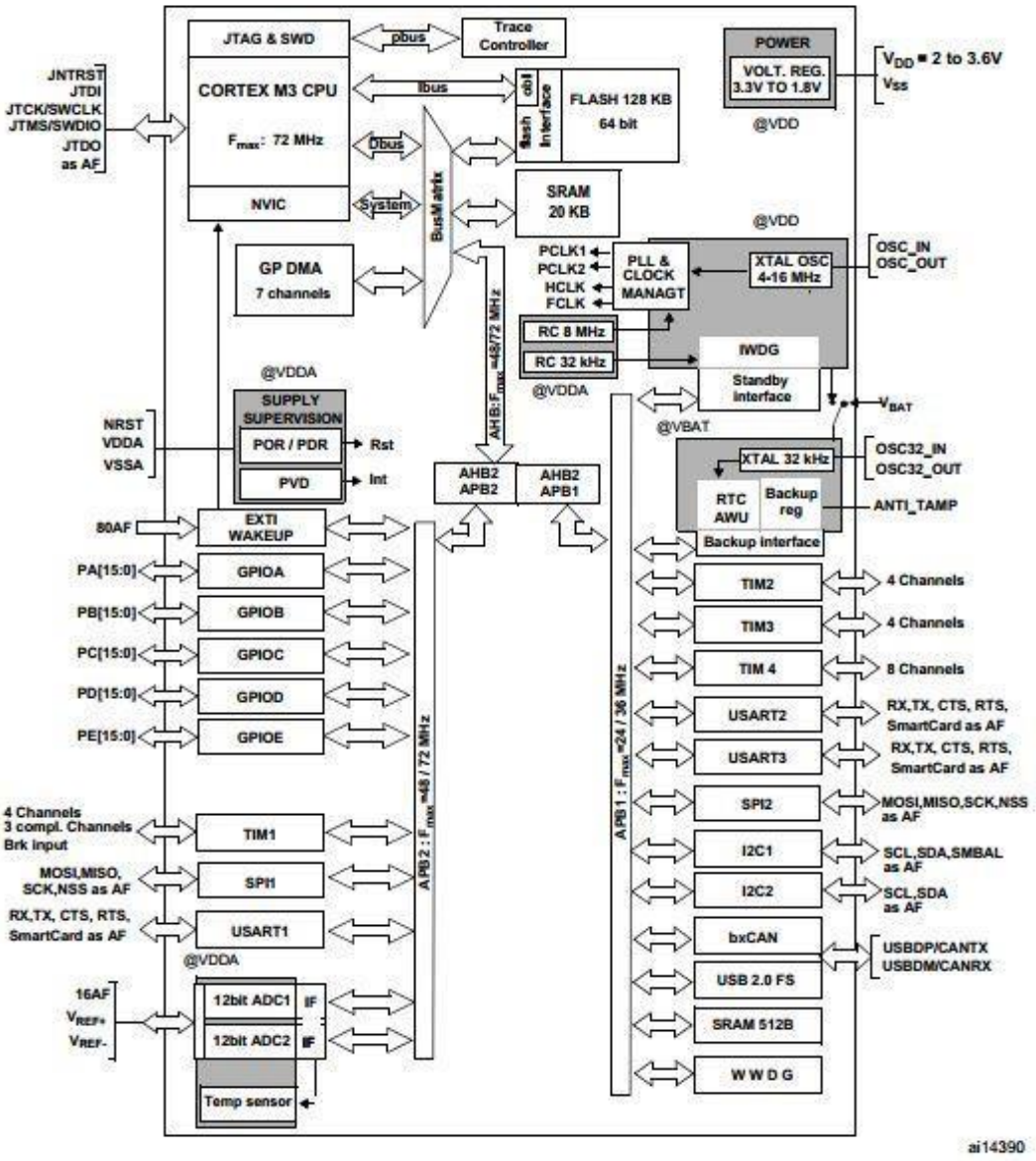


Рис. 1.1— Структурная схема МК семейства *STM*32*F*103x8

МК содержит следующие периферийные устройства:

— четыре 16-битных таймер/счетчика;

— 37 портов ввода/вывода (*GPIO*);

— 16 каналов ШИМ;

— 10 12-битных каналов АЦП;

— интерфейсы для связи с периферией: *UART, SPI, USB, I²C, CAN, DMA*;

— и т.д.

Процессор может работать с тактовой частотой до 72 МГц.

МК имеет *FLASH*-память объемом 64 КБайт и ОЗУ объемом 20 КБайт. Питается МК напряжением 2-3,6 В, однако часть портов ввода/вывода могут работать с сигналами до 5 В.

Более подробную информацию о данном МК можно найти в приложениях. **1.2. Основные сведения об отладочном стенде**

Лабораторные работы выполняются на отладочной плате *PinBoard* II (внешний вид показан на рис.1.2.), особенность которой заключается в отсутствии жестких связей между элементами, что позволяет создавать различные варианты конфигурации.

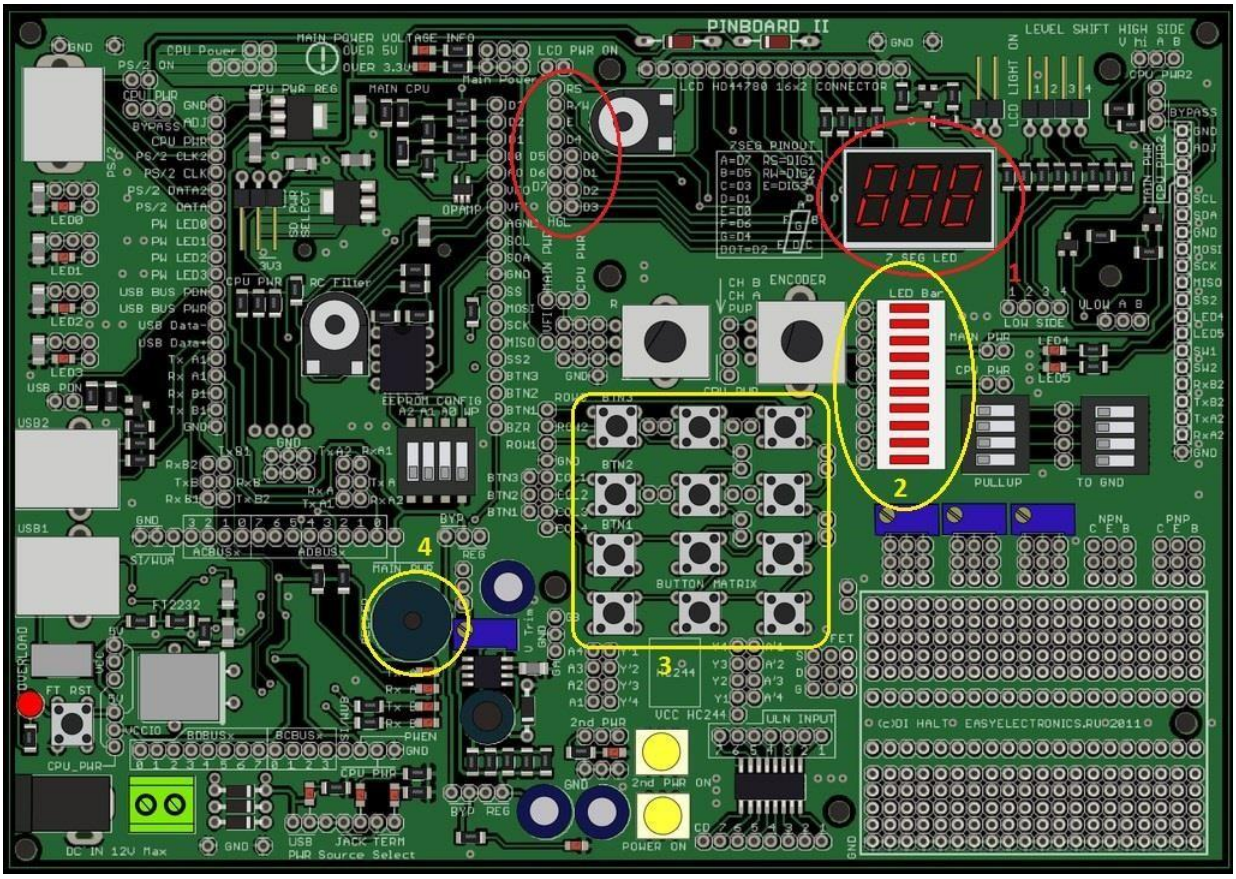


Рис. 1.2— Внешний отладочной платы

На плате имеются следующие основные модули:

— Процессорный модуль (МК с необходимой обвязкой);

— Линейка из светодиодов (2);

— Трехразрядный семисегментный индикатор (1);

— Матричная клавиатура 3x4 (3);

— Пьезоизлучатель (4).

Рассмотрим перечисленные элементы более подробно.

Внешний вид процессорного модуля приведен на рис. 1.3.

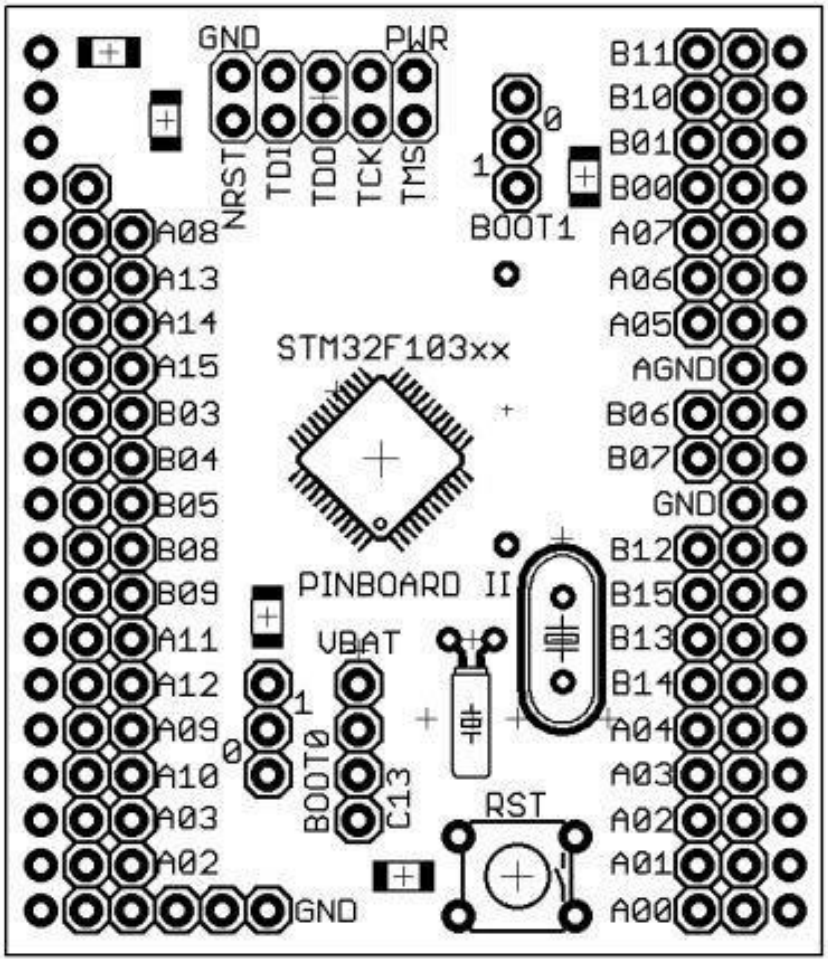


Рис. 1.3 — Внешний вид процессорного модуля

Порты ввода/вывода МК выведены на штыревые разъемы и обозначены как *A*00-*A*15 и *B*00-*B*15, что соответствует *PORTA* и *PORTB*. Подключив сюда шлейфы можно подключать различные периферийные устройства имеющиеся на стенде, например светодиодную линейку.

Подключение конкретных модулей показано в соответствующих лабораторных работах. При осуществлении соединений следует руководствоваться надписями, нанесенными на плате. Более детальные сведения об отладочной плате *PinBoard* II приведены в учебных приложениях.

2. РАБОТА С ОТЛАДОЧНЫМ СТЕНДОМ

В соответствии с индивидуальным заданием по лабораторной работе необходимо написать программу на языке *C*, откомпилировать и ввести в память МК, после чего программа начнет выполняться автоматически.

Программа набирается, компилируется и вводится в память МК с помощью бесплатной интегрированной среды разработки *CoIDE*, которую разработала фирма *CooCox*. Скачать *CoIDE* и узнать более подробную информацию о ней можно по адресу http://www.coocox.org/CooCox_CoIDE.htm (http://www.coocox.org/software.html). После установки необходимо отдельно скачать компилятор *GCC ARM Embedded* с сайта http://www.coocox.org/CoIDE/Compiler_Settings.html и установить его.

Для начала работы необходимо запустить программу *CoIDE* путем двойного щелчка на соответствующем ярлыке на рабочем столе. После этого откроется рабочее окно программы (рис.2.1), где во вкладке ***Project*** следует выбрать пункт ***New Project***.

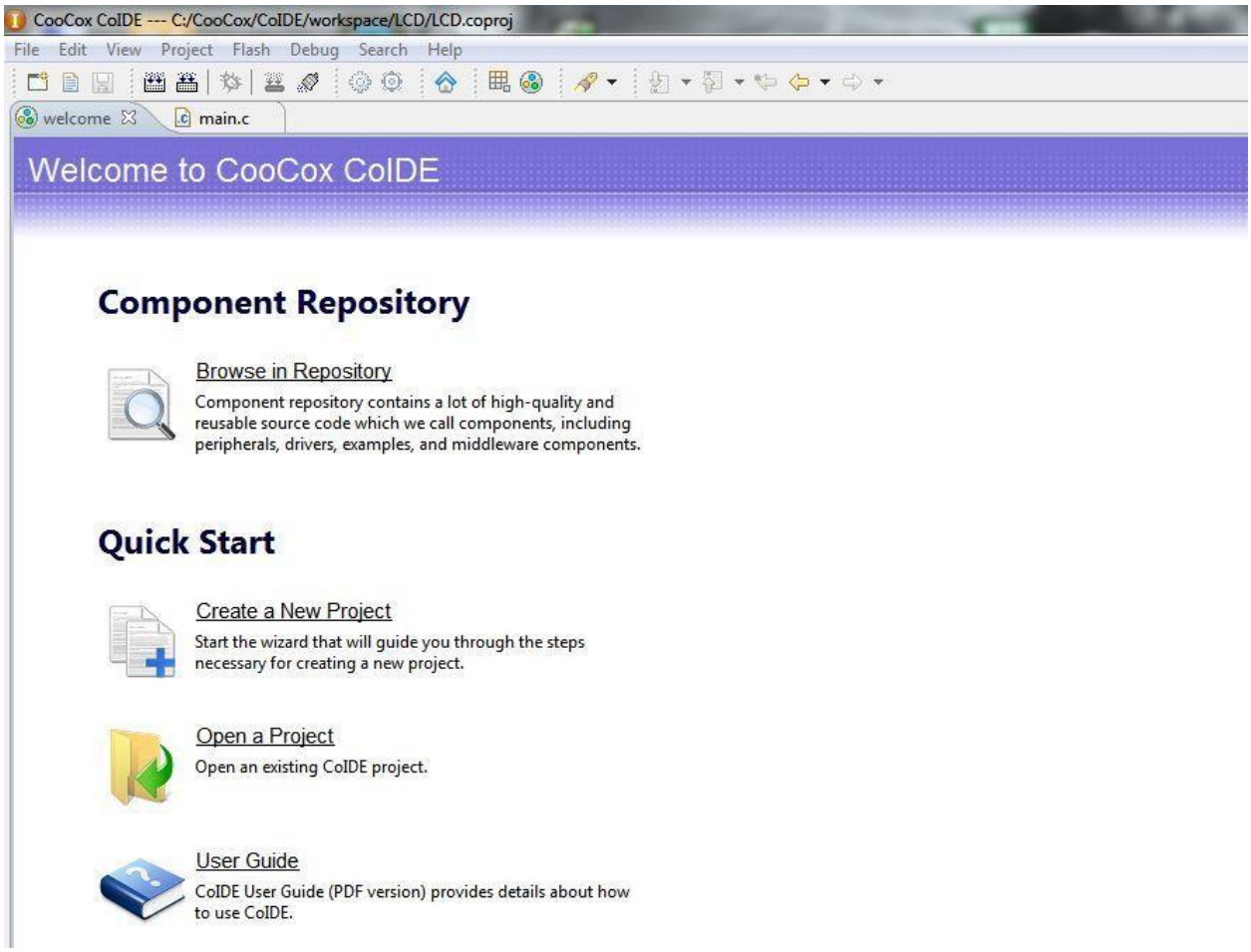


Рис. 2.1 — Главное окно программы *CoIDE*

Далее вам будет предложено ввести название проекта, для удобства не следует изменять стандартный путь к папке проекта, также не рекомендуется использовать в названиях кириллицу. В следующем окне, которое появится при нажатии кнопки ***Next***, следует выбрать ***Chip***. После чего вам будет предложено выбрать конкретный МК. В макете используется МК *STM32F103C8*, поэтому выбираем: *ST*> *STM32F103x*> *STM32F103C8*. Осуществив выбор МК нажимаем ***Finish***.

Появится окно для добавления доступных библиотек в проект (*Repository*), внешний вид которого приведен на рис. 2.2.

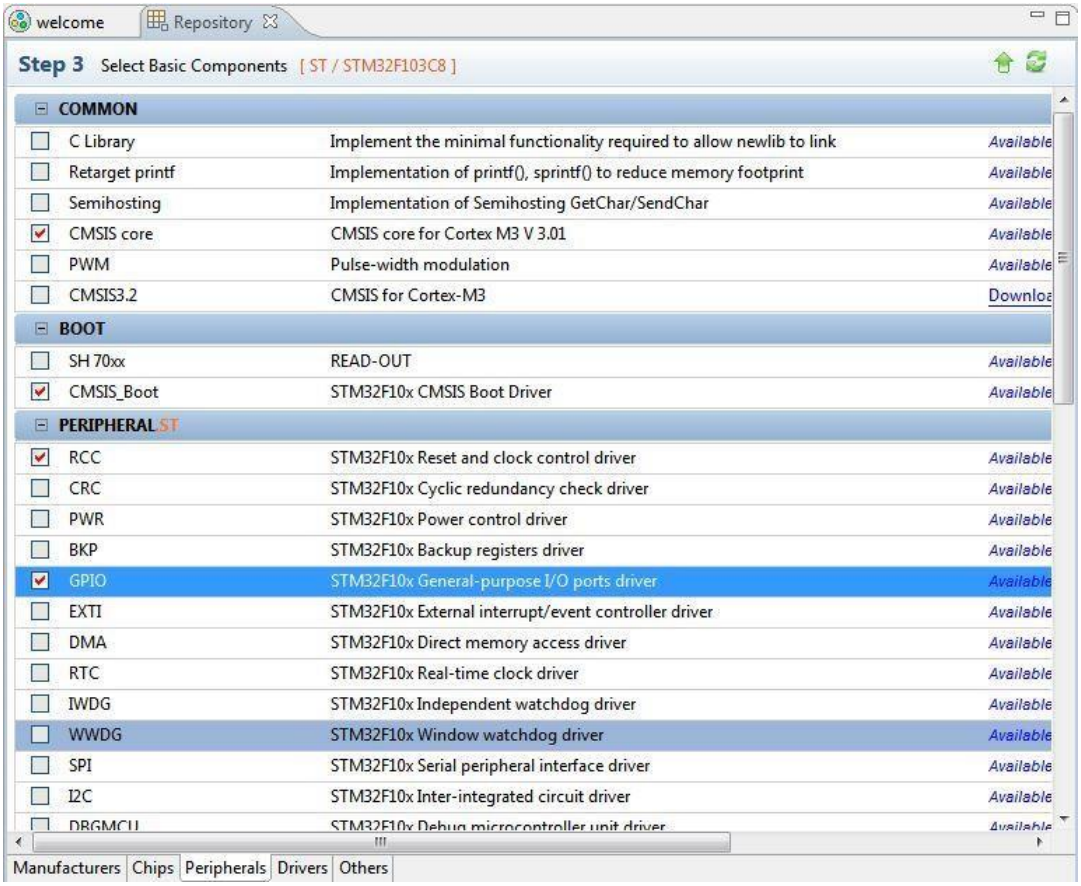


Рис. 2.2— Окно выбора библиотек

При выполнении каждой из лабораторных работ будет указано какие компоненты следует выбрать. Далее необходимо развернуть окно «дерева проекта» и в нем двойным щелчком открыть файл исходного кода main.c. После выполнения этих действий появится рабочее поле (рис. 2.3), где следует писать код программы.

Когда написание кода программы завершено необходимо выполнить сборку проекта путем нажатия клавиши *F7* или же щелчком мыши на соответствующей пиктограмме  .

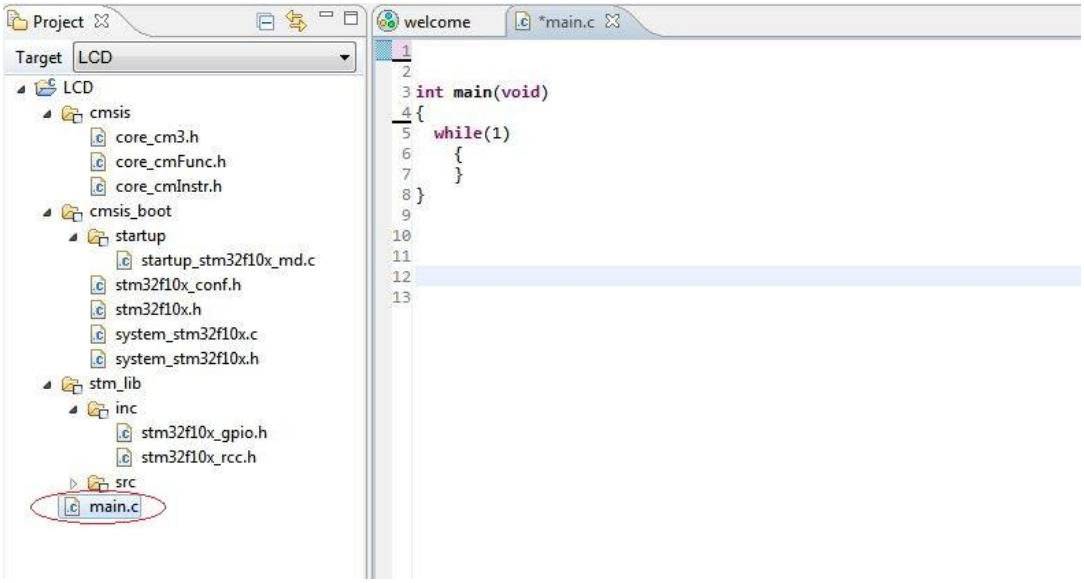


Рис. 2.3— Рабочее поле программы *CoIDE*

В результате появится окно (рис. 2.4) в котором отображается процесс сборки и компиляции и, если код написан без ошибок, сообщение об успешном завершении «*Build successful*». Если же появилась надпись «*Build failed*» следует проверить код на наличие ошибок (в окне компилятора выделены красным шрифтом) и исправить их.

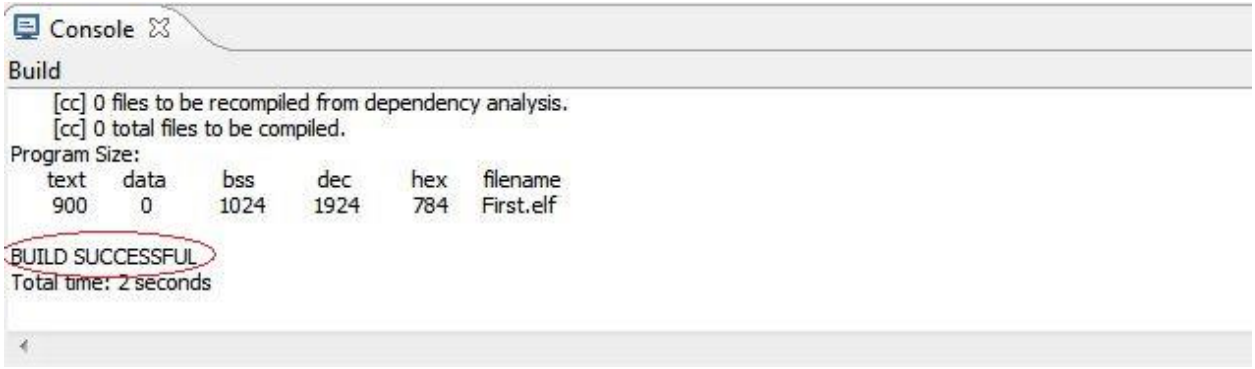


Рис. 2.4— Окно компилятора с сообщением об успешном завершении

Теперь можно загрузить прошивку в память МК, но перед этим следует выбрать модель программатора, для этого нажмите пиктограмму . Появится окно конфигурации проекта (рис. 2.5), где во вкладке ***Debugger*** нужно выбрать программатор *Colink*, используемый в отладочном стенде.

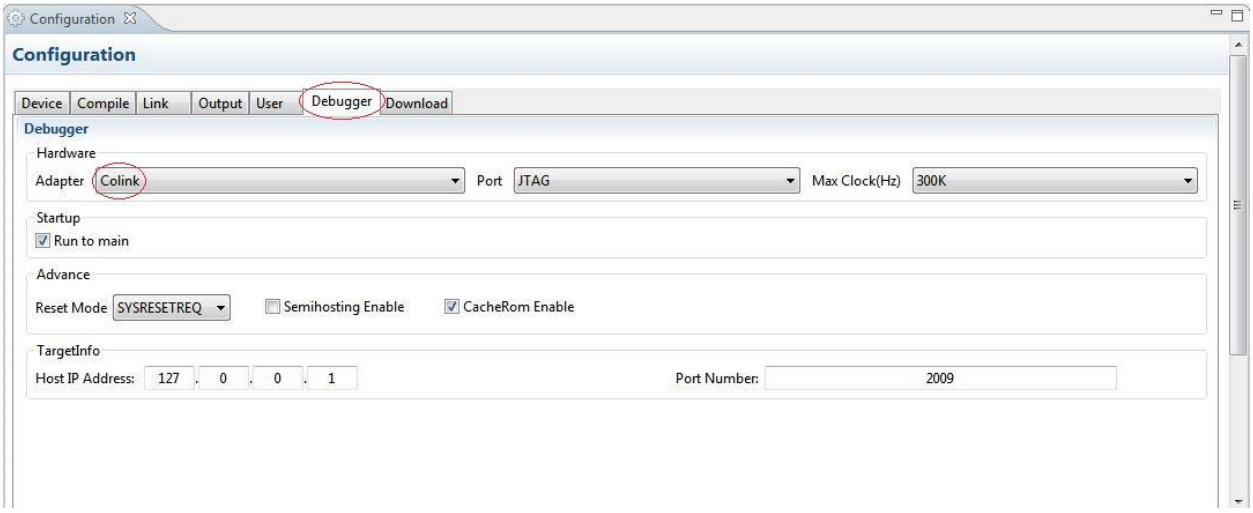
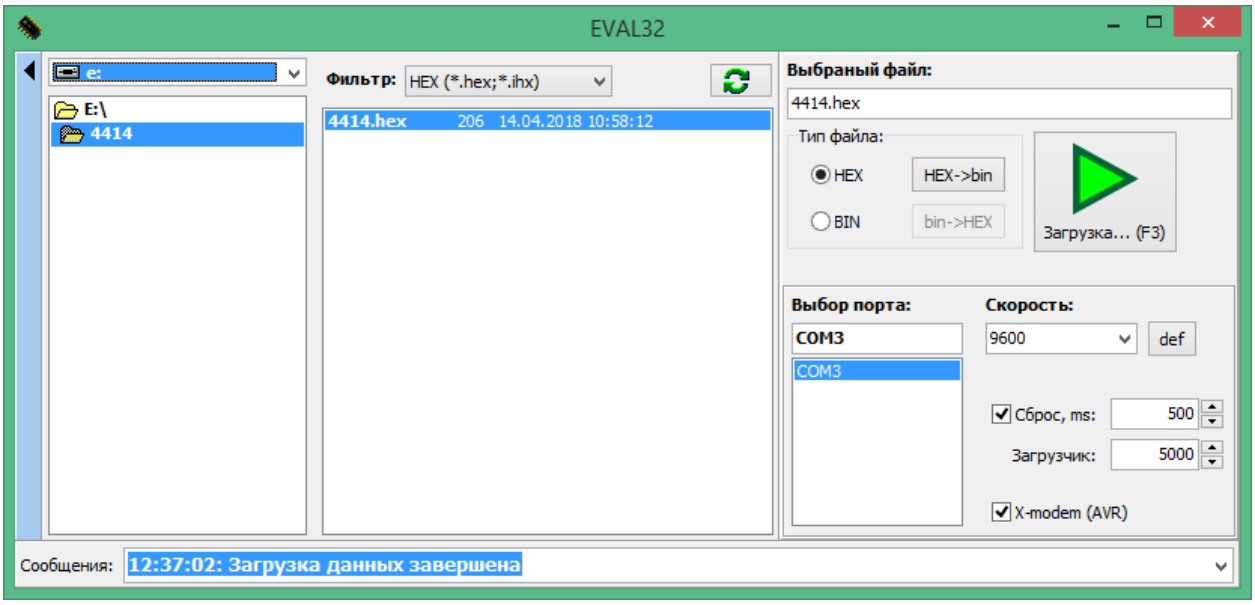


Рис. 2.5— Окно конфигурации проекта

Для загрузки прошивки в память МК нажмите на пиктограмму  , расположенную на панели инструментов.



В случае успешного выполнения появится окно с тремя сообщениями: *Erase: Done; Program: Done;Verify: Done.* Это означает что прошивка успешно загружена. Теперь МК должен автоматически начать выполнение программы.

Также в программе есть раздел «*Help*», где можно найти разнообразную информацию по вопросам программирования МК.

3. ЛАБОРАТОРНАЯ РАБОТА № 1

ПРОГРАММИРОВАНИЕ СВЕТОДИОДНЫХ ИНДИКАТОРОВ

3.1. Цель работы

Целью работы является изучение способов программирования светодиодных индикаторов в микроконтроллерных системах.

3.2. Теоретические сведения

Если к порту МК подключить линейку светодиодов и вывести в порт некоторое двоичное число, то светодиоды, на которые подана логическая единица, засветятся. Таким образом можно управлять различными устройствами: светодиодами, транзисторами и т.д.

Например, если вывести в порт число 0b10101010, то зажгутся светодиоды **x*x*x*x* (*— светодиод светится, *x* — светодиод не светится). Причем, неважно в каком виде записывать число в порт — в двоичном, десятичном или шестнадцатеричном, МК все равно преобразует его в двоичный вид.

Чтобы вывести число в порт, следует записать его в регистр *ODR* соответствующего порта.

Светодиодная линейка подключена таким образом, как изображено на рис.3.1.

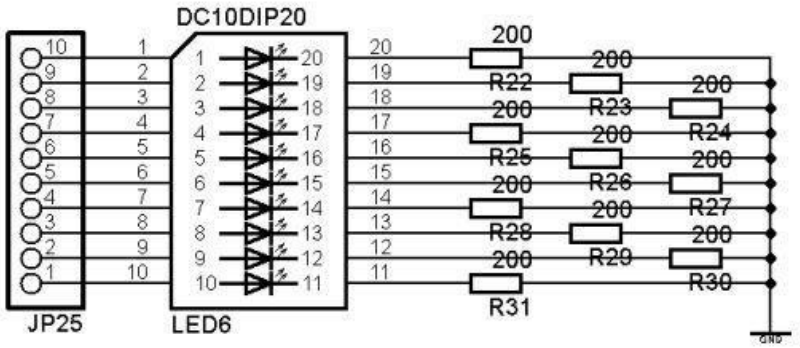


Рис. 3.1— Схема подключения светодиодной линейки

Разъем *JP*25 через шлейф следует подключить к тем выводам МК которые планируется использовать для управления светодиодами. Программа, приведенная в примере требует такого подключения светодиодов к процессорному модулю:

*JP*25(1)<–>*A*00; *JP*25(2)<–>*A*01; *JP*25(3)<–>*A*02; *JP*25(4)<–>*A*03;

*JP*25(5)<–>*A*04; *JP*25(6)<–>*A*05; *JP*25(7)<–>*A*06; *JP*25(8)<–>*A*07;

При создании проекта в разделе *Repository* следует поставить отметку на пункте *GPIO* (автоматически будет выделено еще несколько категорий), чтобы в проект были добавлены необходимые компоненты.

Программа 3.1 — Программа, выводящая на линейку светодиодов “шкалу заполнения”:

```
//подключение библиотек необходимых для работы МК
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
//-----------------------------------------------------------
void delay(void)// объявление функции временной задержки
{
        int i;
        for(i=0;i<0x100000;i++);
}
//-----------------------------------------------------------
int main(void)// объявление основной функции {
RCC->APB2ENR|=RCC_APB2Periph_GPIOA;// включение тактирования порта A
// по умолчанию тактирование выключено для экономии энергии
GPIOA->CRL = 0b11001100110011001100110011;// конфигурация режима работы младшей группы порта A, данная комбинация означает, что порт работает на выход, изменять не следует   while(1) // бесконечный цикл (здесь пишем основную программу)
  {
        GPIOA->ODR=0b00000001; //установка состояния выводов порта A
 delay(); // задержка        GPIOA->ODR=0b00000011;        delay();
        GPIOA->ODR=0b00000111;
 delay();
        GPIOA->ODR=0b00001111;
 delay();
        GPIOA->ODR=0b00011111;
 delay();          GPIOA-
>ODR=0b00111111;        delay();
        GPIOA->ODR=0b01111111;
 delay();
```

```
            GPIOA->ODR=0b11111111;
  delay();
                    GPIOA->ODR=0b00000000; // Гасим все светодиоды
    }
}
```

3.3. Порядок выполнения работы

Пользуясь программой 3.1 в качестве образца, выполнить индивидуальное задание в соответствии с таблицей 3.1.

Таблица 3.1 — Индивидуальное задание по лабораторной работе № 1

№ варианта	**Задание**
1	Перемещать комбинацию **x*x*x*x* циклически слева направо со скоростью одно положение в секунду.
2	Перемещать комбинацию ***xx**xx* циклически справа налево со скоростью 2 с на одно положение.
3	Сделать так, чтобы с интервалом в 1 с сменялись две ситуации: *xx****xx* и ***xxxx***.
4	Повторять циклически с интервалом в 1 с следующие три ситуации: *xxx**xxx, xx*xx*xx, x*xxxx*x*.
5	На линейке светодиодов четыре подряд следующие “огонька” должны следовать слева направо со скоростью 1 с на позицию.
6	Правая и левая половины линейки светодиодов должны зажигаться попеременно с интервалом 2 с.
7	Сделать так, чтобы в течение 1 с светилась комбинация светодиодов **xx**x*x*, потом в течение 1 с инверсная ей, а потом все повторялось с начала.
8	Попеременно с интервалом в 1 с должны зажигаться одновременно оба крайних светодиода и одновременно два центральных светодиода.

№ варианта	Задание
9	Сделать так, чтобы “бегущий огонек”, занимающий одну позицию на линейке светодиодов, продвигался слева направо, каждый раз перескакивая через одну позицию. Например, сначала **xxxxxxx*, потом *xx*xxxxx*, потом *xxxx*xxx* и т. д. По достижении правого края индикатора процесс должен циклически повторяться.
10	Последовательно должны светиться три следующие комбинации *****xxxx, xx****xx, xxxx*****. Затем процесс должен циклически повторяться.
11	Комбинация *xx**xx*** должна циклически перемещаться слева направо со скоростью 3 с на позицию.
12	На линейке светодиодных индикаторов два крайних левых светодиода должны зажигаться попеременно с двумя крайними правыми светодиодами с интервалом в 3 с.
13	Комбинация **xx*xx*** должна оставаться неподвижной в течение 1 с, затем сдвигаться вправо, находиться в новом положении в течение 1 с, а затем вернуться в исходное положение. После этого описанное должно повторяться циклически.
14	Комбинация ***xx**xx* должна циклически сдвигаться вправо на две позиции, затем на одну позицию влево и снова вправо на две позиции и т. д.
15	Комбинация *xxxxx**** должна циклически сдвигаться влево на три позиции, затем вправо на две позиции, потом снова влево на три позиции и т. д.
16	Комбинация ***x*xxxx* должна светиться в течение 1 с, затем должна быть инвертирована и снова светиться в течение 1 с, затем должна быть сдвинута вправо на 1 позицию и светиться в течение 1 с, затем снова проинвертирована и т. д.

3.4. Содержание отчета

Отчет по работе оформляется в соответствии с требованиями, приведенными во введении.

3.5. Вопросы для самоконтроля

3.5.1. Поясните, каким образом можно организовать циклическое выполнение программы.

3.5.2. Каким образом можно увеличить или уменьшить скорость движения огонька на линейке светодиодов?

3.5.3. Как зажечь светодиоды *x*x*x*x*x?

3.5.4. Как вывести на линейку светодиодов движущийся огонек?

3.5.5. Какие изменения нужно внести в программу 3.1, чтобы на линейку светодиодов выводился медленно движущийся пульсирующий огонек?

3.5.6. Какие изменения нужно внести в программу 3.1, чтобы время задержки увеличилось в 2 раза?

3.5.7. Какие изменения нужно внести в программу 3.1, чтобы время задержки уменьшилось в 20 раз?

4. ЛАБОРАТОРНАЯ РАБОТА № 2

ПРОГРАММИРОВАНИЕ СЕМИСЕГМЕНТНЫХ ИНДИКАТОРОВ

4.1. Цель работы

Целью работы является изучение способов программирования семисегментных индикаторов (ССИ) с использованием динамической индикации в микроконтроллерных системах.

4.2 Теоретические сведения

Как известно, ССИ содержит 7(8) отдельных сегментов, с помощью которых формируется отображаемое число, таким образом для управления сегментами требуется 7(8) выводов МК. Однако при использовании ССИ с несколькими разрядами становится затруднительным использование отдельной линии для каждого сегмента, например для подключения четырехразрядного ССИ потребуется 4x8=32 линии управления.

В таком случае целесообразно использовать ССИ с динамической индикацией. Устройство одного из них приведено на рис. 4.1.

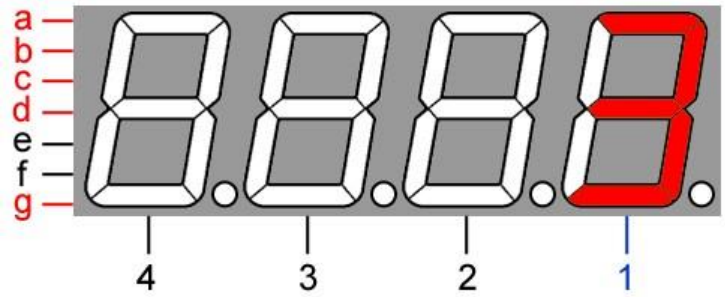


Рис. 4.1— Четырехразрядный ССИ

Одноименные сегменты из каждого разряда соединены параллельно, а общие выводы (катоды или аноды светодиодов) выведены отдельно. Подключая к шине питания общие выводы можно зажигать соответствующие разряды. Включая поочередно каждый разряд с задержкой порядка 20 мс можно выводить разные цифры на каждый разряд. При этом в каждый момент времени горит только одна цифра.

На отладочной плате *PinBoard* II установлен трехразрядный ССИ с общим катодом. Схема подключения изображена на рис. 4.2.

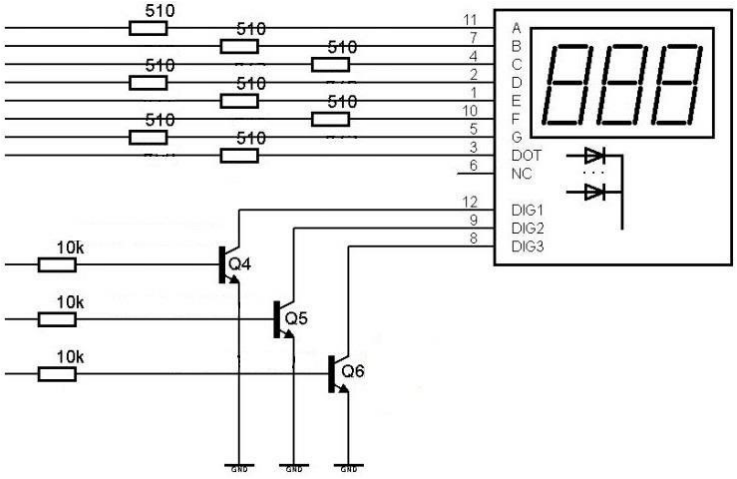


Рис. 4.2— Схема подключения ССИ на *PinBoard* II

Подавая лог. 1 на базы транзисторов можно включать соответствующие разряды. Все линии управления выведены на штыревой разъем (1), а соответствия приведены на отладочной плате в виде таблицы (2) (рис. 4.3).

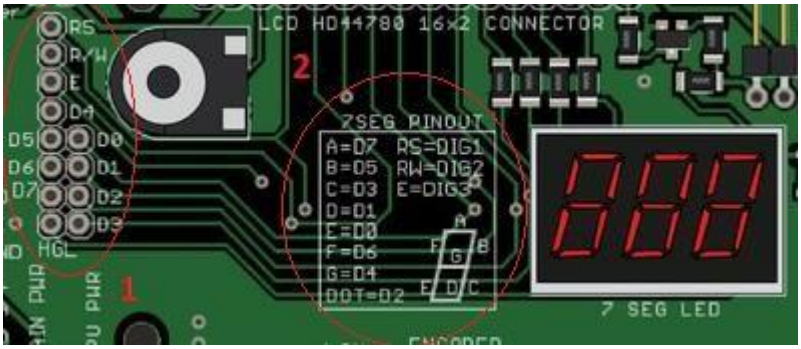


Рис.4.3 — Элементы необходимые для подключения ССИ

В отличии от программы из предыдущей работы здесь используется директива *define*, ее смысл в том, чтобы упростить написание кода, например благодаря строке #define D0 GPIO_Pin_7 вместо GPIO_Pin_7 можно писать D0, а компилятор все равно будет понимать это как GPIO_Pin_7.

Также стоит отметить, что в Л.Р. 1 при конфигурации портов мы работали с конфигурационными регистрами напрямую, а здесь используются инструменты библиотеки *CMSIS*, призванные максимально упростить процесс программирования. Программа, приведенная в примере, требует такого подключения ССИ к процессорному модулю:

*D*7<–>*A*00; *D*5<–>*A*01; *D*3<–>*A*02; *D*1 (4) <–>*A*03; *D*0 <–>*A*04;

*D*6<–>*A*05; *D*4<–>*A*06; *RS*<–>*A*08; *RW*<–>*A*09; *E*<–>*A*10;

Программа 4.1 — Программа, осуществляющая обратный отсчет от 100 до 0:

```
// Подключаем необходимые библиотеки
```

```
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
// Ниже идут директивы препроцессора типа define
// Порт к которому подключен индикатор #define
IND_PORT GPIOA
// Общие выводы индикатора
#define D0 GPIO_Pin_7
#define D1 GPIO_Pin_8
#define D2 GPIO_Pin_9
#define D3 GPIO_Pin_10
// К какой ноге какой сегмент подключен
#define SEG_A GPIO_Pin_0
#define SEG_B GPIO_Pin_1
#define SEG_C GPIO_Pin_2
#define SEG_D GPIO_Pin_3
#define SEG_E GPIO_Pin_4
#define SEG_F GPIO_Pin_5
#define SEG_G GPIO_Pin_6
//Собираем цифры из сегментов
#define DIG0 ( SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F )
#define DIG1 ( SEG_B | SEG_C )
#define DIG2 ( SEG_A | SEG_B | SEG_G | SEG_E | SEG_D )
#define DIG3 ( SEG_A | SEG_B | SEG_G | SEG_C | SEG_D )
#define DIG4 ( SEG_F | SEG_G | SEG_B | SEG_C)
#define DIG5 ( SEG_A | SEG_F | SEG_G | SEG_C | SEG_D )
#define DIG6 ( SEG_A | SEG_C | SEG_D | SEG_E | SEG_F | SEG_G )
#define DIG7 ( SEG_A | SEG_B | SEG_C )
#define DIG8 ( SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F | SEG_G)
#define DIG9 ( SEG_A | SEG_B | SEG_C | SEG_D | SEG_F | SEG_G)
#define ALL_PINS (DIG8 | D0 | D1 | D2 | D3 )
```

```
//Функция формирующая небольшую задержку void
delay(void)
{ int
i=0;
  for (i=0; i != 0x1000; i++); }
// Функция, осуществялющая динамическую индикацию
void indication(int chislo) {
// объявляем массив с элементами соответствующими цифрам 0-9   unsigned int
digit[]={DIG0,DIG1,DIG2,DIG3,DIG4,DIG5,DIG6,DIG7,DIG8,DIG9};
// далее определим значение цифр в каждом из разрядов
unsigned char desyatki=0,sotni=0;
  while(chislo>=100) // первый разряд — «сотни»
            {
     chislo-=100;
sotni++;
            }
  while(chislo>=10) //второй разряд — «десятки»
            {
    chislo-=10;     desyatki++;
            }
// то что осталось — «единицы»
  IND_PORT->ODR|=D1;// Включаем первый разряд
  IND_PORT->ODR|=digit[sotni];//Выводим число в разряд "сотни"
delay(); // Небольшое время задержки в течении которого горит разряд
  IND_PORT->ODR=0;// Выключаем первый разряд
  IND_PORT->ODR|=D2;// Включаем второй разряд
  IND_PORT->ODR|=digit[desyatki];//Выводим число в разряд "десятки"
  delay();
  IND_PORT->ODR=0;// Выключаем второй разряд
```

```
  IND_PORT->ODR|=D3;// Включаем третий разряд
  IND_PORT->ODR|=digit[chislo];//Выводим число в разряд "единицы"
delay();
  IND_PORT->ODR=0;// Выключаем третий разряд
}
// Основная программа
int main(void) {
  //Настраиваем на выход все выводы подключенные к индикатору
  RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOA , ENABLE);
  GPIO_InitTypeDef PORT; //Структура содержащая настройки порта
  PORT.GPIO_Pin = ALL_PINS; //Указываем какие выводы нужно настроить
  PORT.GPIO_Mode = GPIO_Mode_Out_PP; // Настраиваем как выход Push-pull
  PORT.GPIO_Speed = GPIO_Speed_2MHz; // Частота 2 МГц
  GPIO_Init(IND_PORT, &PORT); //Вызываем функцию настройки порта
int n=0;   int a=0;   int b=0; while(1)
  {
  n=100;
  while(n>0)
          {
           for (a=0; a != 0x200; a++)
           {
  indication(n);// Вызываем функцию индикации и передаем ей то, что нужно отобразить   for
(b=0; b != 0x100; b++);
           }
           n--;
          }
  }
}
```

4.3. Порядок выполнения работы

Пользуясь программой 4.1 в качестве образца, выполнить индивидуальное задание в соответствии с таблицей 4.1.

Таблица 4.1 — Индивидуальное задание по лабораторной работе № 2

№ варианта	**Задание**
1	Число 123 на семисегментных индикаторах должно светиться в течение 1 с, потом гаснуть на 1 с. Затем процесс должен повторяться.
2	В 2-м и 3-м разрядах индикатора с задержкой на 1 с число должно увеличиваться от 0 до *FFh*. После того, как число станет равным *FFh*, процесс должен повторяться сначала.
3	Цифра 5 должна перемещаться по индикатору слева направо (остальные индикаторы должны быть погашены). Процесс должен циклически повторяться.
4	На индикатор выводить последовательно (с интервалом 1 с) члены геометрической прогрессии с начальным значением 1 и знаменателем 2.
5	Сделать бегущую точку слева направо на семисегментном индикаторе. По достижении точкой крайней правой позиции процесс должен повторяться.
6	На семисегментном индикаторе число 15*x* (*x* — индикатор погашен) меняться на число x20, потом снова на 15*x* и т. д. Каждое из двух чисел до того, как будет изменено, должно светиться в течение 1 с.
7	Выводить по очереди на семисегментный индикатор числа 111, 222, 333, потом процесс должен повторяться. Каждое число должно индицироваться в течение 1 с.
8	Выводить убывающие числа от *FFh* до 0 в двух разрядах семисегментного индикатора (левом и среднем), оставшийся разряд должен быть погашен. Каждое число до изменения должно индицироваться в течение 1 с.

№ варианта	Задание
9	Выводить в двух левых разрядах семисегментного индикатор число, начиная с 1, увеличивающееся каждый раз в два раза. Каждое состояние должно индицироваться в течение 1 с.
10	Выводить перемещающееся слева направо увеличивающееся одноразрядное число от 1 до 3 по разрядам семисегментных индикаторов.
11	Поверх разрядов числа *BBBh* слева направо должна перемещаться цифра *Ah*.
12	На левой части индикатора число *AAh* должно мигать с частотой 1 Гц, на правом индикаторе число *Bh* должно мигать с частотой 2 Гц.
13	Выводить перемещающееся циклически справа налево уменьшающееся от 8 до 6 одноразрядное число по разрядам семисегментного индикатора.
14	Число *FFh* на левой части индикатора должно мигнуть два раза. После этого индикаторы должны быть выключены на 2 с. Этот процесс должен циклически повторяться.
15	В двух правых разрядах семисегментного индикатора должно поочередно шестнадцатеричное число, каждую секунду увеличивающееся на единицу, начиная со значения 1.
16	Выводить в среднем разряде семисегментного индикатора число, увеличивающиеся каждую секунду на 1, начиная с 1*h* и до *Fh*. Этот процесс должен циклически повторяться.

4.4. Содержание отчета

Отчет по работе оформляется в соответствии с требованиями, приведенными во введении.

4.5. Вопросы для самоконтроля

4.5.1. Как зажечь точку в одном разряде статического семисегментного индикатора?

4.5.2. Как погасить все разряды статического семисегментного индикатора?

4.5.3. Чем отличается статическая индикация от динамической?

4.5.4. Как осуществить увеличение числа, выводимого на индикатор, на единицу?

4.5.5. Можно ли выводить произвольную шестнадцатеричную цифру в один разряд статического семисегментного индикатора (при погашенных остальных разрядах)?

5. ЛАБОРАТОРНАЯ РАБОТА № 3

ПРОГРАММИРОВАНИЕ ВНЕШНИХ ПРЕРЫВАНИЙ

5.1. Цель работы

Целью работы является изучение принципов работы с внешними прерываниями в МК семейства *STM*.

5.2. Теоретические сведения

Прерывание — это событие, при котором происходит приостановка основной программы и переход на выполнение подпрограммы прерывания. Прерывания в МК бывают внешние и внутренние. Примером внешнего прерывания может служить прерывание от изменения логического состояния какого-либо из входов МК.

В МК *STM*32*F*103*C*8 доступно 16 выводов для внешних прерываний – *EXTI*0*EXTI*15, причем можно назначать каждое из них на любой из портов *GPIO*. Назначение осуществляется с помощью регистра *AFIO_EXTICR*. Регистр *EXTI_IMR* разрешает или запрещает прерывания от отдельных каналов.

Для осуществления внешнего прерывания используется кнопка *BTN*3 (выделена красным цветом на рис. 5.1.).

Программа, приведенная в примере требует такого подключения ССИ, светодиодной шкалы и кнопки к процессорному модулю:

*D*7<–>*A*00; *D*5<–>*A*01; *D*3<–>*A*02; *D*1 (4) <–>*A*03; *D*0 <–>*A*04; *D*6<–>*A*05; *D*4<–>*A*06; *RS*<–>*A*08; *RW*<–>*A*09; *E*<–>*A*10;

*JP*25 (1) <–>*B*08; *JP*25 (2) <–> *B*09; *JP*25 (3) <–> *B*10; *JP*25 (4) <–> *B*11; *JP*25 (5) <–> *B*12; *JP*25 (6) <–> *B*13; *JP*25 (7) <–> *B*14; *JP*25 (8) <–> *B*15; *COL*1<–> *B*00;

Также следует убедиться, что установлена перемычка так, как показано на рис. 5.1.

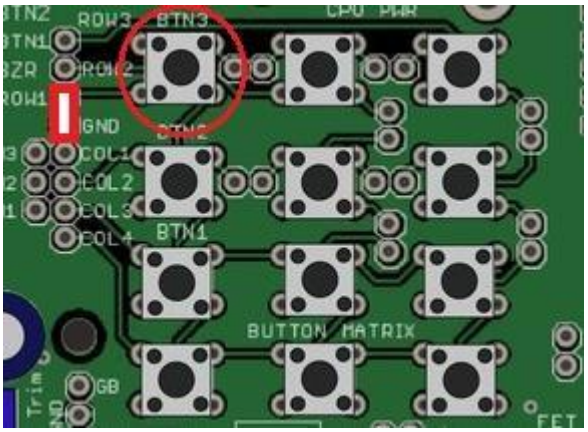


Рис. 5.1 — Необходимые соединения

Программа 5.1 — Программа осуществляет обратный отсчет от 100 до 0, а при нажатии кнопки выполняет подпрограмму обработки прерывания в виде «заполняющейся шкалы» на светодиодной линейке, после чего возвращается к выполнению основной программы.

```
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
#define LB_PORT GPIOB // В качестве порта для управления светодиодной шкалой
// (Led Bar) выберем PORTB
#define IND_PORT GPIOA // Порт к которому подключен индикатор
// Общие выводы индикатора
#define D0 GPIO_Pin_7
#define D1 GPIO_Pin_8
#define D2 GPIO_Pin_9
#define D3 GPIO_Pin_10
// К какому выводу какой сегмент подключен
#define SEG_A GPIO_Pin_0
#define SEG_B GPIO_Pin_1
#define SEG_C GPIO_Pin_2
#define SEG_D GPIO_Pin_3
#define SEG_E GPIO_Pin_4
#define SEG_F GPIO_Pin_5
#define SEG_G GPIO_Pin_6
//Собираем цифры из сегментов
#define DIG0 ( SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F )
#define DIG1 ( SEG_B | SEG_C )
#define DIG2 ( SEG_A | SEG_B | SEG_G | SEG_E | SEG_D )
#define DIG3 ( SEG_A | SEG_B | SEG_G | SEG_C | SEG_D )
#define DIG4 ( SEG_F | SEG_G | SEG_B | SEG_C )
#define DIG5 ( SEG_A | SEG_F | SEG_G | SEG_C | SEG_D )
```

```
#define DIG6 ( SEG_A | SEG_C | SEG_D | SEG_E | SEG_F | SEG_G )
#define DIG7 ( SEG_A | SEG_B | SEG_C )
#define DIG8 ( SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F | SEG_G)
#define DIG9 ( SEG_A | SEG_B | SEG_C | SEG_D | SEG_F | SEG_G)
#define ALL_PINS (DIG8 | D0 | D1 | D2 | D3 ) void delay(int k) // Функция, осуществялющая временную задержку заданной длительности
{ int
i=0;
  for (i=0; i != k*4000; i++);
}

// Функция, осуществялющая динамическую индикацию void
indication(int chislo)
{
  unsigned int digit[]={DIG0,DIG1,DIG2,DIG3,DIG4,DIG5,DIG6,DIG7,DIG8,DIG9};
unsigned char desyatki=0,sotni=0;   while(chislo>=100)
            {
     chislo-=100;
sotni++;
            }
  while(chislo>=10)
            {
    chislo-=10;      desyatki++;
            }
  IND_PORT->ODR|=D1;// Включаем первый разряд
  IND_PORT->ODR|=digit[sotni];//Выводим число в разряд "сотни"
delay(1);// Небольшое время задержки в течении которого горит разряд
  IND_PORT->ODR=0;// Выключаем первый разряд
  IND_PORT->ODR|=D2;// Включаем второй разряд
```

```
  IND_PORT->ODR|=digit[desyatki];//Выводим число в разряд "десятки"
delay(1);

  IND_PORT->ODR=0;// Выключаем второй разряд
  IND_PORT->ODR|=D3;// Включаем третий разряд
  IND_PORT->ODR|=digit[chislo];//Выводим число в разряд "единицы"
delay(1);
  IND_PORT->ODR=0;// Выключаем третий разряд
} int
main()
  {
   //Включим тактирование PORTA и PORTB
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA , ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB , ENABLE);
  RCC_APB2PeriphClockCmd(RCC_APB2ENR_AFIOEN , ENABLE);
  LB_PORT->CRH = 0b11001100110011001100110011;// конфигурация режима работы старшей группы порта B (работает как выход)
  LB_PORT->CRL = 0b1000;// конфигурация режима работы младшей группы порта B(работает как вход с подтяжкой)
  IND_PORT->CRL = 0b11001100110011001100110011;// конфигурация режима работы младшей группы порта A (работает как выход)
  IND_PORT->CRH = 0b11001100110011001100110011;// конфигурация режима работы младшей группы порта A (работает как выход)
  AFIO->EXTICR[0]|=AFIO_EXTICR1_EXTI0_PB; // выбираем в качесте вывода для внешнего прерывания EXTI0 вывод GPIOB_0
  EXTI->IMR|=EXTI_IMR_MR0 ; // разрешаем прерывание EXTI0
  EXTI->FTSR|=EXTI_FTSR_TR0; //Прерывания по спадающему фронту
  NVIC_EnableIRQ (EXTI0_IRQn);//Разрешаем  прерывание в контроллере прерываний
  LB_PORT->BSRR=GPIO_BSRR_BS0; // включаем подтяжку к лог. "1" на выводе GPIOB_0
  // служебные переменные для организации обратного отсчета
int n=0;   int a=0;   int b=0;   while(1)
    {
```

```
            n=100;
  while(n>0)
          {
            for (a=0; a != 0x200; a++)
    {
// Вызываем функцию индикации и передаем ей то, что нужно отобразить
   indication(n);            for (b=0; b != 0x100; b++);
              }
     n--;
          }
   }
}
// Обработчик прерывания EXTI0 void
EXTI0_IRQHandler(void)
{
        IND_PORT->ODR=0; // отключаем ССИ
  LB_PORT->BSRR=GPIO_BSRR_BS8;//установка состояния выводов PORTB
  delay(1000);
  LB_PORT->BSRR=GPIO_BSRR_BS9;     delay(1000);
  LB_PORT->BSRR=GPIO_BSRR_BS10;    delay(1000);
  LB_PORT->BSRR=GPIO_BSRR_BS11;    delay(1000);
  LB_PORT->BSRR=GPIO_BSRR_BS12;    delay(1000);
  LB_PORT->BSRR=GPIO_BSRR_BS13;    delay(1000);
  LB_PORT->BSRR=GPIO_BSRR_BS14;    delay(1000);
  LB_PORT->BSRR=GPIO_BSRR_BS15;    delay(1000);
        LB_PORT->ODR=0b00000000;
   LB_PORT->BSRR=GPIO_BSRR_BS0;
        EXTI->PR|=0x02; //Сбрасываем флаг прерывания
}
```

5.3. Порядок выполнения работы

Используя программу 5.1 в качестве образца, выполнить индивидуальное задание в соответствии с таблицей 5.1.

Таблица 5.1 — Индивидуальное задание по лабораторной работе № 3

№ варианта	**Задание**	
	Основная программа	Подпрограмма обработки прерывания
1	Перемещать комбинацию ***xx****xx* циклически справа налево со скоростью 2 с на одно положение.	Выводить в среднем разряде семисегментного индикатора число, увеличивающиеся каждую секунду на 1, начиная с 1*h* и до *Fh*. Этот процесс должен циклически повторяться.
2	Сделать так, чтобы с интервалом в 1 с сменялись две ситуации: *xx******xx* и ***xxxx***.	Число 123 на семисегментных индикаторах должно светиться в течение 1 с, потом гаснуть на 1 с. Затем процесс должен повторяться.
3	Повторять циклически с интервалом в 1 с следующие три ситуации: *xxx****xxx, xx***xx***xx, x***xxxx***x*.	В 2-м и 3-м разрядах индикатора с задержкой на 1 с число должно увеличиваться от 0 до *FFh*. После того, как число станет равным *FFh*, процесс должен повторяться сначала.
4	На линейке светодиодов четыре подряд следующие “огонька” должны следовать слева направо со скоростью 1 с на позицию.	Цифра 5 должна перемещаться по индикатору слева направо (остальные индикаторы должны быть погашены). Процесс должен циклически повторяться.

№ варианта	**Задание**	
	Основная программа	Подпрограмма обработки прерывания
5	Правая и левая половины линейки светодиодов должны зажигаться попеременно с интервалом 2 с.	На индикатор выводить последовательно (с интервалом 1 с) члены геометрической прогрессии с начальным значением 1 и знаменателем 2.
6	Сделать так, чтобы в течение 1 с светилась комбинация светодиодов **xx**x*x*, потом в течение 1 с инверсная ей, а потом все повторялось с начала.	Сделать бегущую точку слева направо на семисегментном индикаторе. По достижении точкой крайней правой позиции процесс должен повторяться.
7	Попеременно с интервалом в 1 с должны зажигаться одновременно оба крайних светодиода и одновременно два центральных светодиода.	На семисегментном индикаторе число 15*x* (*x* — индикатор погашен) меняться на число x20, потом снова на 15*x* и т. д. Каждое из двух чисел до того, как будет изменено, должно светиться в течение 1 с.
8	Сделать так, чтобы “бегущий огонек”, занимающий одну позицию на линейке светодиодов, продвигался слева направо, каждый раз перескакивая через одну позицию. Например, сначала **xxxxxxx*, потом *xx*xxxxx*, потом *xxxx*xxx* и т. д. По достижении правого края индикатора процесс должен циклически повторяться.	Выводить по очереди на семисегментный индикатор числа 111, 222, 333, потом процесс должен повторяться. Каждое число должно индицироваться в течение 1 с.

№ варианта	**Задание**	
	Основная программа	Подпрограмма обработки прерывания
9	Последовательно должны светиться три следующие комбинации ****xxxx, xx****xx, xxxx****. Затем процесс должен циклически повторяться.	Выводить убывающие числа от *FFh* до 0 в двух разрядах семисегментного индикатора (левом и среднем), оставшийся разряд должен быть погашен. Каждое число до изменения должно индицироваться в течение 1 с.
10	Комбинация *xx**xx** должна циклически перемещаться слева направо со скоростью 3 с на позицию.	Выводить в двух левых разрядах семисегментного индикатор число, начиная с 1, увеличивающееся каждый раз в два раза. Каждое состояние должно индицироваться в течение 1 с.
11	На линейке светодиодных индикаторов два крайних левых светодиода должны зажигаться попеременно с двумя крайними правыми светодиодами с интервалом в 3 с.	Выводить перемещающееся слева направо увеличивающееся одноразрядное число от 1 до 3 по разрядам семисегментных индикаторов.
12	Перемещать комбинацию *x*x*x*x циклически слева направо со скоростью одно положение в секунду.	Поверх разрядов числа *BBBh* слева направо должна перемещаться цифра *Ah*.
13	Комбинация *xx*xx** должна оставаться неподвижной в течение 1 с, затем сдвигаться вправо, находиться в новом положении в течение 1 с, а затем вернуться в исходное положение. После этого описанное должно повторяться циклически.	На левой части индикатора число *AAh* должно мигать с частотой 1 Гц, на правом индикаторе число *Bh* должно мигать с частотой 2 Гц.

№ варианта	Задание	
	Основная программа	Подпрограмма обработки прерывания
14	Комбинация ***xx**xx* должна циклически сдвигаться вправо на две позиции, затем на одну позицию влево и снова вправо на две позиции и т. д.	Выводить перемещающееся циклически справа налево уменьшающееся от 8 до 6 одноразрядное число по разрядам семисегментного индикатора.
15	Комбинация *xxxxx*** должна циклически сдвигаться влево на три позиции, затем вправо на две позиции, потом снова влево на три позиции и т. д.	Число *FFh* на левой части индикатора должно мигнуть два раза. После этого индикаторы должны быть выключены на 2 с. Этот процесс должен циклически повторяться.
16	Комбинация ***x*xxxx* должна светиться в течение 1 с, затем должна быть инвертирована и снова светиться в течение 1 с, затем должна быть сдвинута вправо на 1 позицию и светиться в течение 1 с, затем снова проинвертирована и т. д.	В двух правых разрядах семисегментного индикатора должно поочередно шестнадцатеричное число, каждую секунду увеличивающееся на единицу, начиная со значения 1.

5.4. Содержание отчета

Отчет по работе оформляется в соответствии с требованиями, приведенными во введении.

5.5. Вопросы для самоконтроля

5.5.1. Сколько внешних прерываний может быть в микроконтроллере *STM32F103x8*?

5.5.2. Дайте определение вектору прерывания.

5.5.3. Что такое исключения и как они связаны с прерываниями?

5.5.4. Что такое прерывание?

5.5.5. Чем отличаются внешние прерывания от внутренних прерываний?

5.5.6. Как программно разрешить или запретить выполнение прерываний?

5.5.7. Где хранятся векторы прерываний?

5.5.8. Почему для обработки прерываний должен быть инициализирован стек?

6. ЛАБОРАТОРНАЯ РАБОТА № 4

ИСПОЛЬЗОВАНИЕ АЦП

6.1. Цель работы

Целью работы является изучение принципов работы с АЦП в МК семейства *STM*32.

6.2 Теоретические сведения

Для работы с аналоговыми сигналами МК нуждается в АЦП. В *STM*32*F*103x8 может быть до 10 каналов АЦП (в зависимости от модели) с разрядностью 12 бит. Для работы АЦП необходимо обеспечить стабильное опорное напряжение на выводе *VDDA*, поскольку именно относительно этого напряжения будут проводиться измерения. Чтобы определить абсолютное значение напряжения следует воспользоваться формулой

$$V = \frac{V_{Ref} \cdot ADC\_RES}{2^{12}}$$

где V_{Ref} — величина опорного напряжения, ADC_RES — результат измерения АЦП.

В качестве измеряемого используем напряжение с потенциометра (выделен красным цветом), подключенного между шиной питания и общим проводом таким образом напряжение может изменяться от 0 до V_{cc}.

Программа, приведенная в примере, требует такого подключения ССИ к процессорному модулю:

*D*7<–>*B*00; *D*5<–>*B*01; *D*3<–>*B*11; *D*1 (4) <–>*B*12; *D*0 <–>*B*05; *D*6<–>*B*06;
*D*4<–>*B*07; *RS*<–>*B*08; *RW*<–>B09; *D*2<–>*B*14.

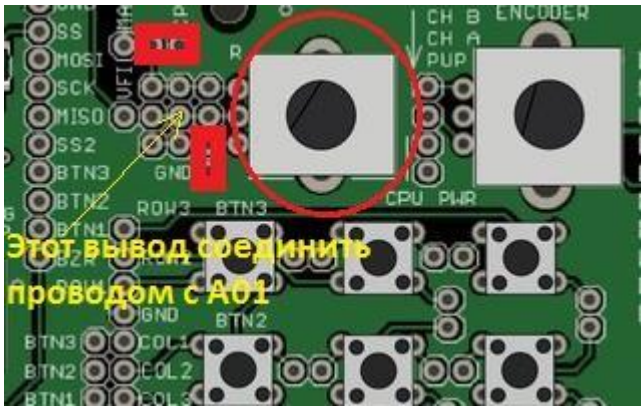


Рис. 6.1 — Необходимые соединения

При создании проекта в разделе *Repository* следует поставить отметку на пунктах *GPIO* и *ADC*, чтобы в проект были добавлены необходимые компоненты.

Программа 6.1 — Программа отображает измеряемое напряжение (с потенциометра) на ССИ с точностью 0,1 В.

```
#include<stm32f10x_rcc.h>
#include<stm32f10x_gpio.h>
#include "stm32f10x.h"
#define IND_PORT GPIOB // Порт к которому подключен индикатор
// Общие выводы индикатора
#define D1 GPIO_Pin_8
#define D2 GPIO_Pin_9
#define D3 GPIO_Pin_10
// К какому выводу какой сегмент подключен
#define SEG_A GPIO_Pin_0
#define SEG_B GPIO_Pin_1
#define SEG_C GPIO_Pin_11
#define SEG_D GPIO_Pin_12
#define SEG_E GPIO_Pin_5
#define SEG_F GPIO_Pin_6
#define SEG_G GPIO_Pin_7
#define DOT   GPIO_PIN_14
//Собираем цифры из сегментов
#define DIG0 ( SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F )
#define DIG1 ( SEG_B | SEG_C )
#define DIG2 ( SEG_A | SEG_B | SEG_G | SEG_E | SEG_D )
#define DIG3 ( SEG_A | SEG_B | SEG_G | SEG_C | SEG_D )
#define DIG4 ( SEG_F | SEG_G | SEG_B | SEG_C )
#define DIG5 ( SEG_A | SEG_F | SEG_G | SEG_C | SEG_D )
#define DIG6 ( SEG_A | SEG_C | SEG_D | SEG_E | SEG_F | SEG_G )
#define DIG7 ( SEG_A | SEG_B | SEG_C )
#define DIG8 ( SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F | SEG_G)
```

```
#define DIG9 ( SEG_A | SEG_B | SEG_C | SEG_D | SEG_F | SEG_G)
#define ALL_PINS (DIG8 | D0 | D1 | D2 | D3 )
void sdelay(void)
  {   int
k=0;
  for (k=0; k != 0x1000; k++);
  }
void delay(uint32_t i) {
volatile uint32_t j;  for
(j=0; j!= i * 1000; j++)
  ;
}
// Функция, осуществялющая динамическую индикацию
void indication(float chislo)
{
  unsigned int digit[]={DIG0,DIG1,DIG2,DIG3,DIG4,DIG5,DIG6,DIG7,DIG8,DIG9};
unsigned char edinici=0,desyatie=0;   while(chislo>=1)
            {
    chislo-=1;       edinici++;
            }
  while(chislo>=0.1)
            {
    chislo-=0.1;
desyatie++;
            }
  IND_PORT->ODR|=D1;// Включаем первый разряд
  IND_PORT->ODR|=digit[edinici];//Выводим число в разряд "единицы"
IND_PORT->BSRR=GPIO_BSRR_BS14;   sdelay();
  IND_PORT->ODR=0;// Выключаем первый разряд
  IND_PORT->ODR|=D2;// Включаем второй разряд   IND_PORT-
>ODR|=digit[desyatie];//Выводим число в разряд "десятые доли"   sdelay();
```

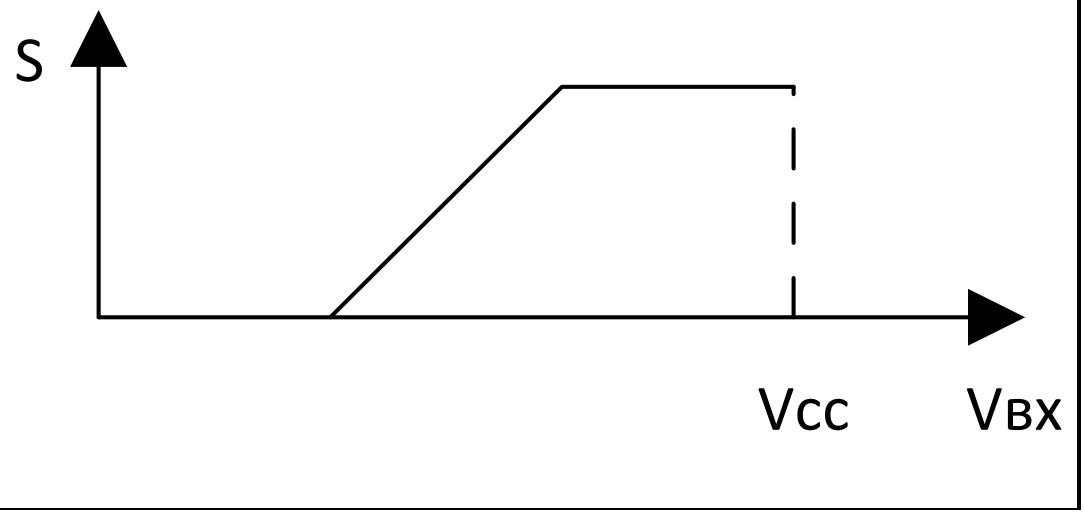
```
  IND_PORT->ODR=0;// Выключаем второй разряд
} int
main(void)
{
  GPIO_InitTypeDef PORT;
  //Включаем порты A и B
  RCC_APB2PeriphClockCmd((RCC_APB2Periph_GPIOB | RCC_APB2Periph_GPIOA) ,
ENABLE);
  //Настраиваем все вывода порта B на выход
  PORT.GPIO_Pin = GPIO_Pin_All;
  PORT.GPIO_Mode = GPIO_Mode_Out_PP;
  PORT.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_Init( GPIOB , &PORT);
  //Порт A настраивать не нужно,так как все его выводы умолчанию входы
  RCC_APB2PeriphClockCmd(RCC_APB2ENR_ADC1EN, ENABLE); //Включаем
тактирование АЦП
  ADC1->CR2 |= ADC_CR2_CAL; //Запуск калибровки АЦП
  while (!(ADC1->CR2 & ADC_CR2_CAL)); //Ожидаем окончания калибровки
  ADC1->SMPR2 |= (ADC_SMPR2_SMP1_2 | ADC_SMPR2_SMP1_1 |
ADC_SMPR2_SMP1_0); //Задаем  длительность выборки
  ADC1->CR2 |= ADC_CR2_JEXTSEL; //Преобразование инжектированной группы
                                  //запустится установкой бита JSWSTART
  ADC1->CR2 |= ADC_CR2_JEXTTRIG; //Разрешаем внешний запуск инжектированной
группы
  ADC1->CR2 |= ADC_CR2_CONT; //Преобразования запускаются одно за другим
  ADC1->CR1 |= ADC_CR1_JAUTO; //Разрешить преобразование инжектированной
группы
                        //после регулярной.
  ADC1->JSQR |= (1<<15); //Задаем номер канала (выбран ADC1)
  ADC1->CR2 |= ADC_CR2_ADON;//Теперь включаем АЦП
  ADC1->CR2 |= ADC_CR2_JSWSTART; //Запуск преобразований
  while (!(ADC1->SR & ADC_SR_JEOC)); //ждем пока первое преобразование завершится
```

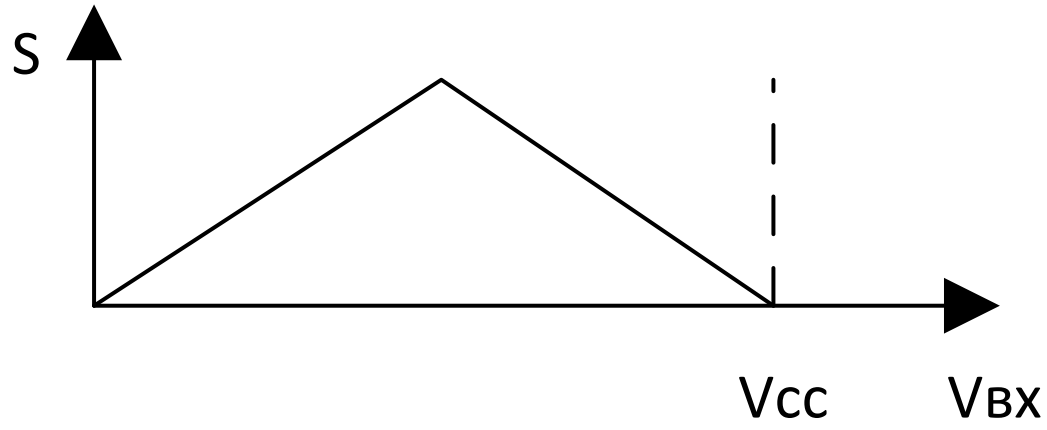
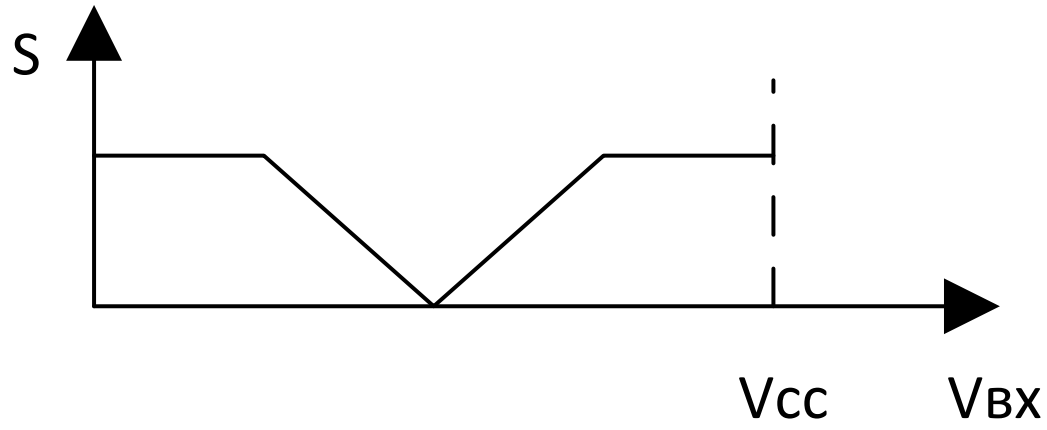
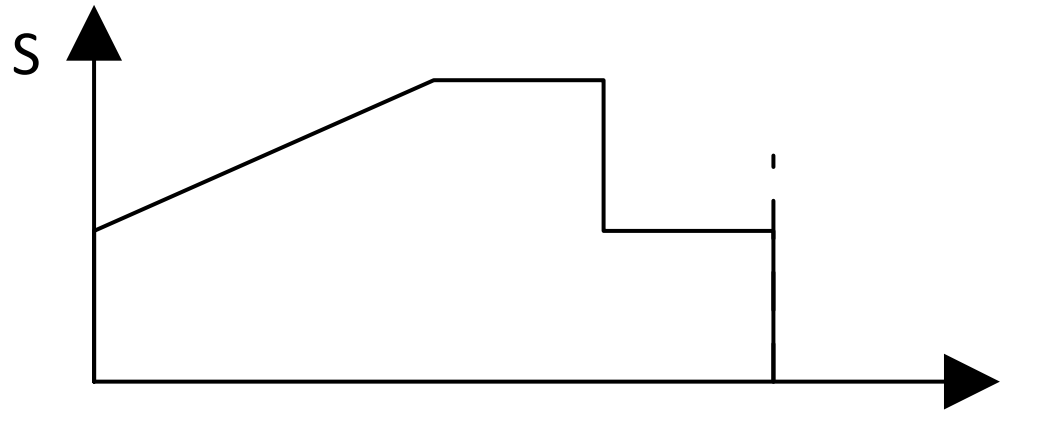
```
 //Теперь можно читать результат из JDR1
uint32_t adc_res; //   float voltage=0;//
значение напряжения   while(1)
 {
   adc_res=ADC1->JDR1;
voltage=2.56*2*adc_res/4096;     indication(voltage);
 } }
```

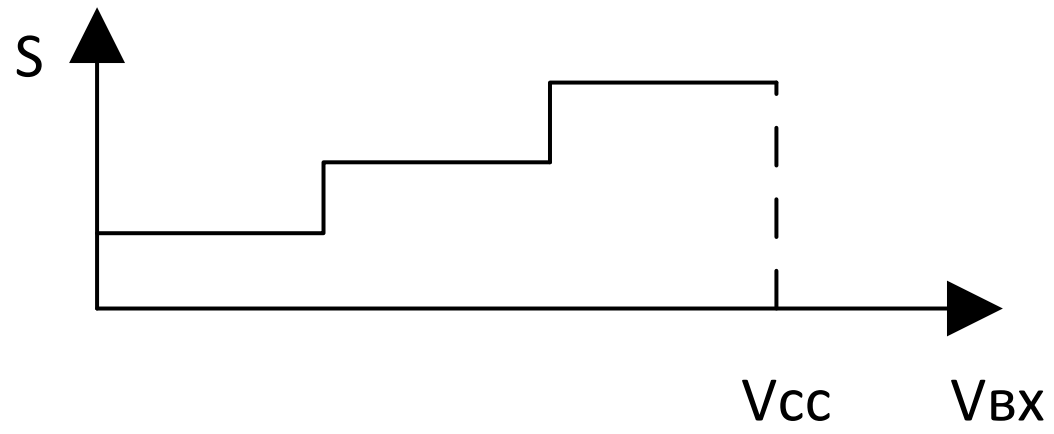
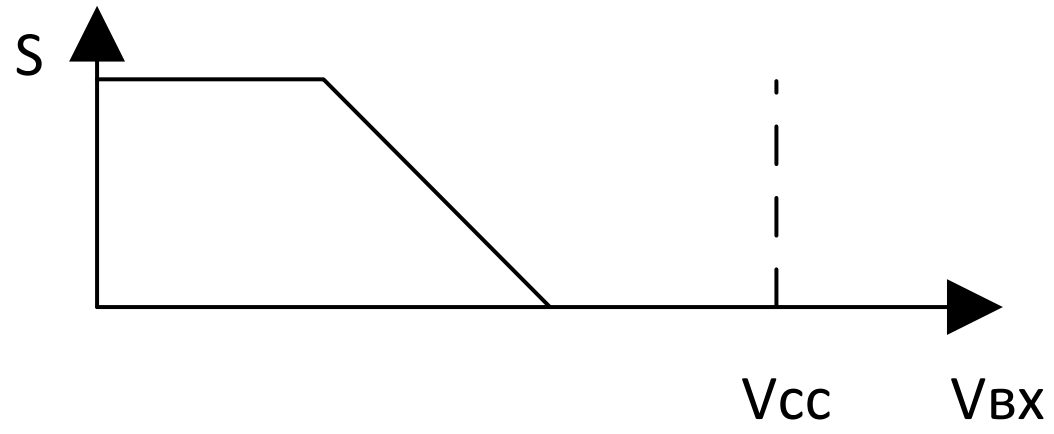
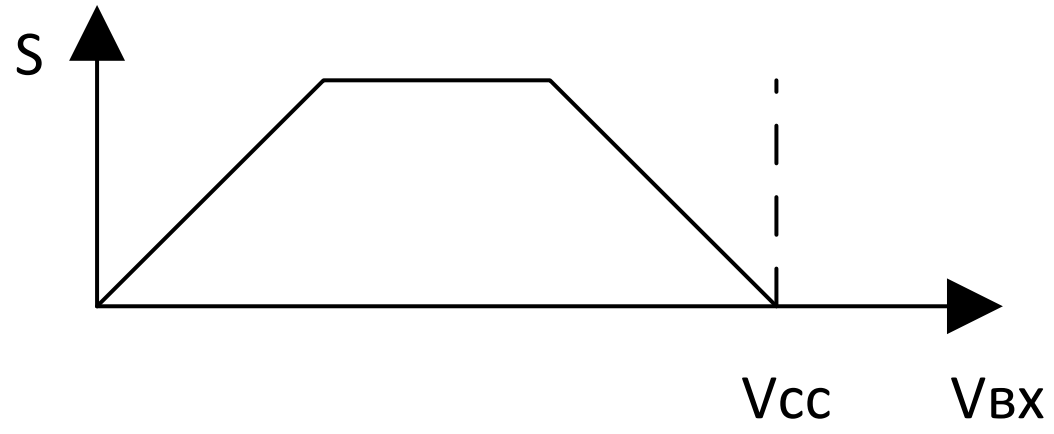
6.3. Порядок выполнения работы

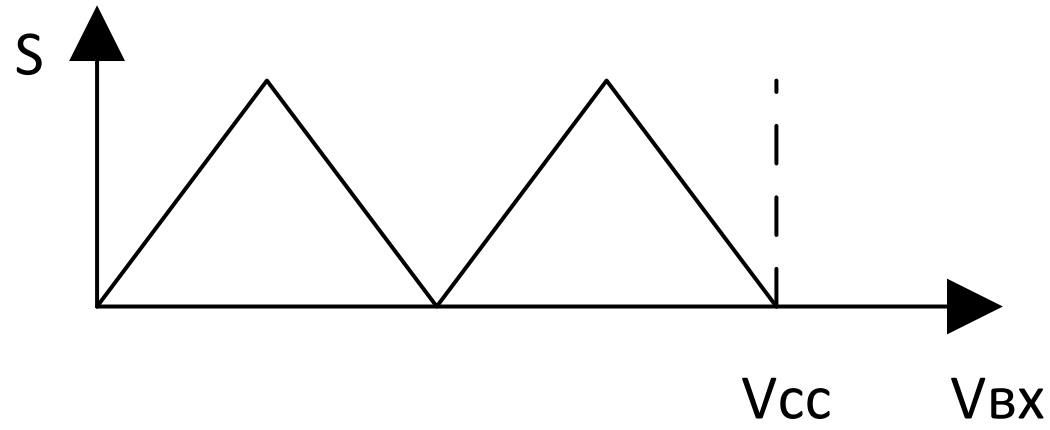
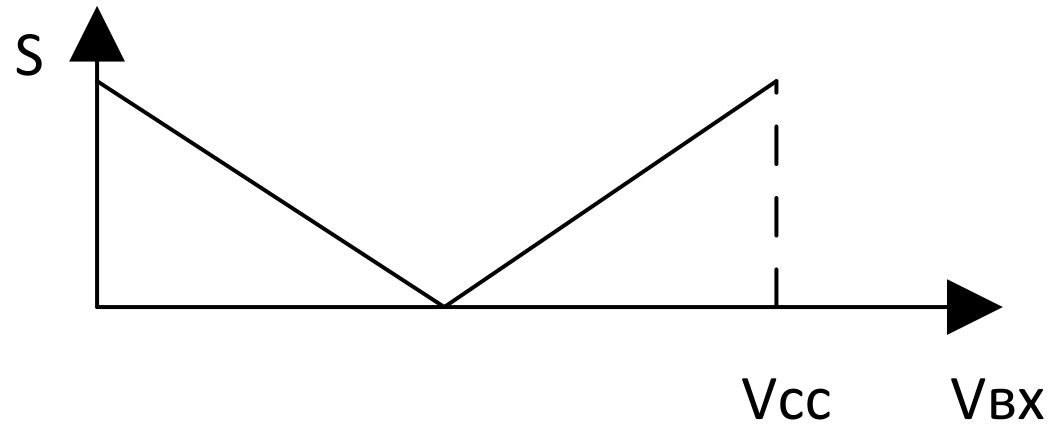
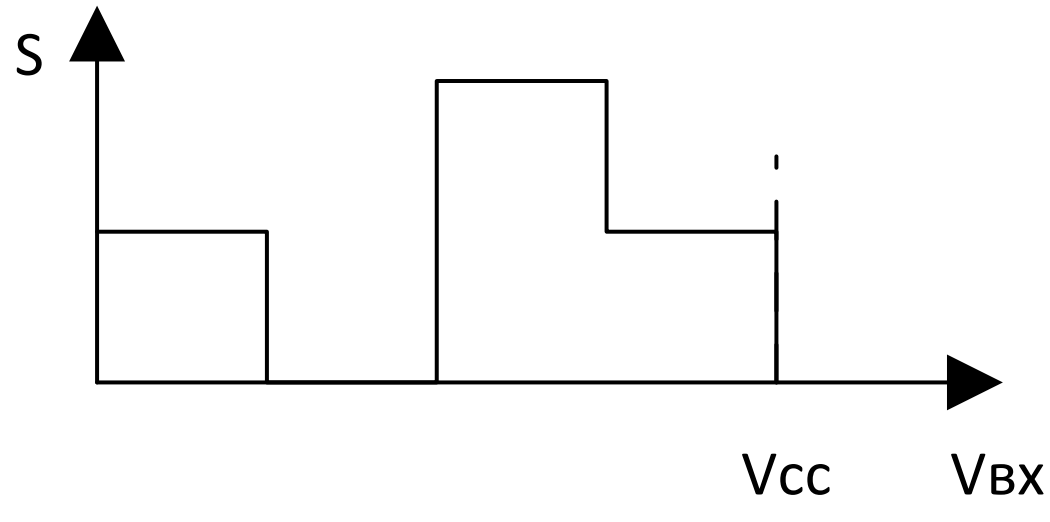
Используя программу 6.1 в качестве образца, обеспечить при выполнении индивидуального задания зависимость выводимых значений *S* от напряжения на потенциометре $V_{вх}$, задаваемую в соответствии с табл. 6.1.

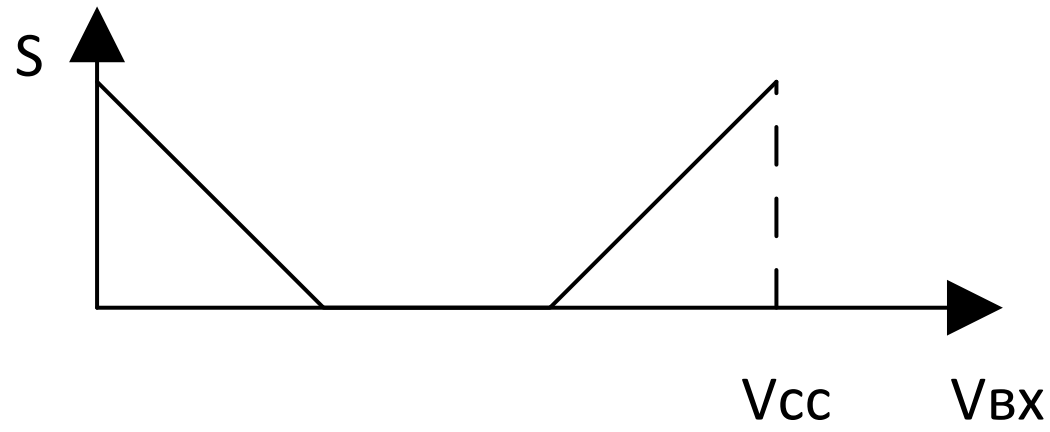
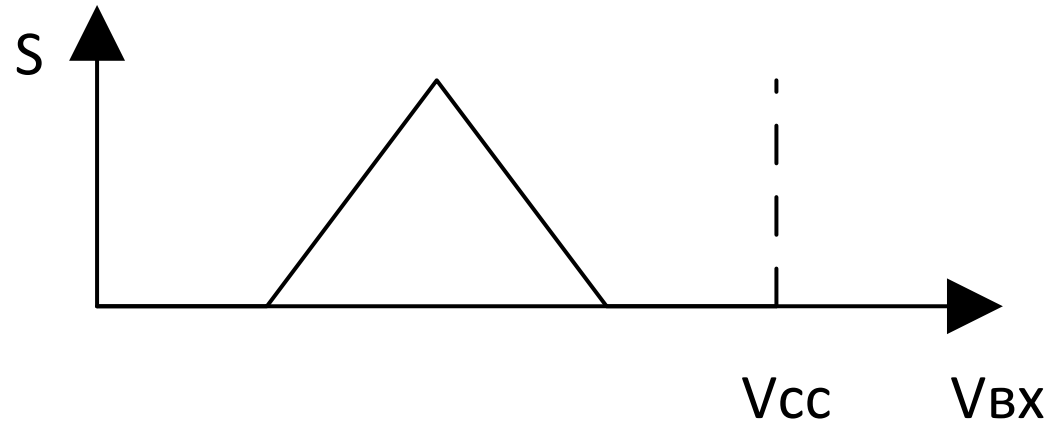
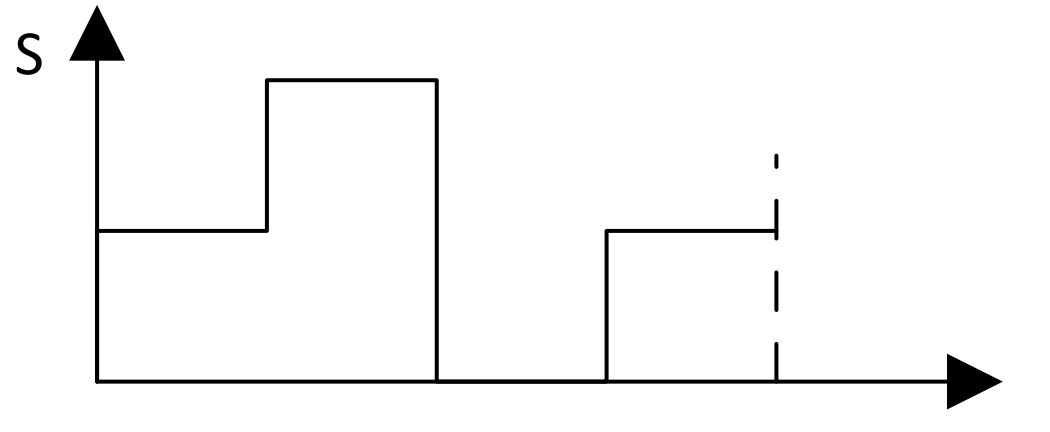
Таблица 6.1 — Индивидуальное задание по лабораторной работе № 4

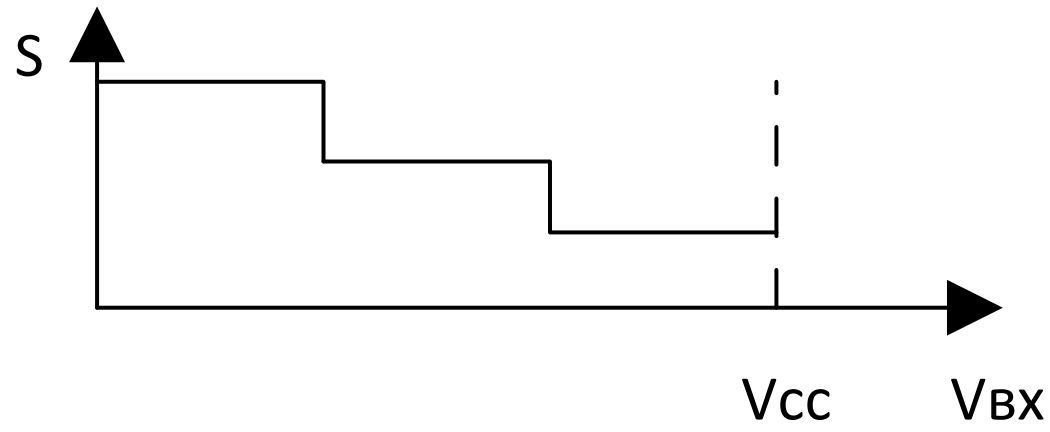
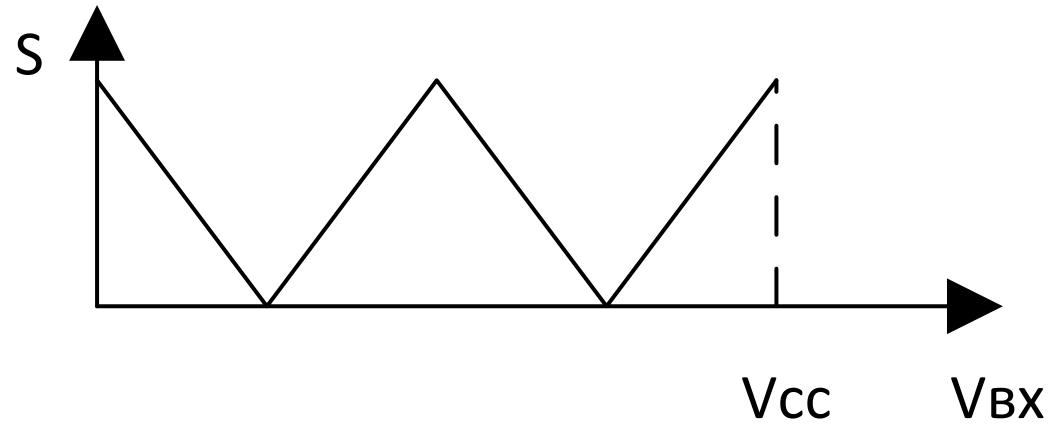
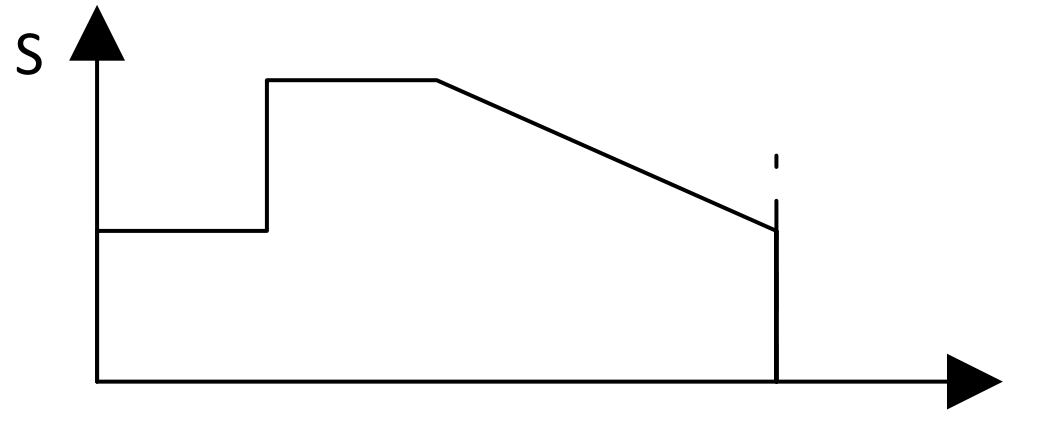
№ ва-рианта	**Задание**
1	

№ варианта	Задание
2	
3	
4	

№ варианта	Задание
5	
6	
7	

№ варианта	Задание
8	
9	
10	

№ варианта	Задание
11	
12	
13	

№ варианта	Задание
14	
15	
16	

6.4. Содержание отчета

Отчет по работе оформляется в соответствии с требованиями, приведенными во введении.

6.5. Вопросы для самоконтроля

6.5.1. Сколько каналов АЦП в МК STM32F103x8?

6.5.2. Какова разрядность АЦП в МК STM32F103x8?

6.5.3. Для чего при аналогово-цифровом преобразовании используется опорное напряжение?

6.5.4. С какого источника подается напряжение на АЦП микроконтроллера в отладочной плате Pinboard II?

6.5.5. Какие регистры управляют работой АЦП МК Cortex-M3?

6.5.6. Формирует ли АЦП Cortex-M3 запрос на прерывание?

6.5.7. Как связаны число уровней дискретизации и разрядность АЦП?

6.5.8. Чем определяется скорость аналогово-цифрового преобразования?

7. ЛАБОРАТОРНАЯ РАБОТА № 5

РАБОТА С МАТРИЧНОЙ КЛАВИАТУРОЙ

7.1. Цель работы

Целью работы является изучение принципов организации ввода данных с матричной клавиатуры в контроллерах, построенных на базе *STM*32 ARM микроконтроллеров семейства *Cortex-M*3.

7.2 Теоретические сведения

Для ввода данных в микроконтроллерную систему часто используется клавиатура. В лабораторной работе № 3 был приведен пример использования кнопочного ввода, однако в случае, когда требуется подключить к МК несколько кнопок, например, клавиатуру 3x4 (12 кнопок), непосредственное подключение каждой кнопки к выводу МК становится неоправданным. В такой ситуации целесообразно использовать матричную клавиатуру, поскольку это уменьшает требуемое количество выводов МК. На рис. 7.1 приведена схема, поясняющая принцип работы матричной клавиатуры.

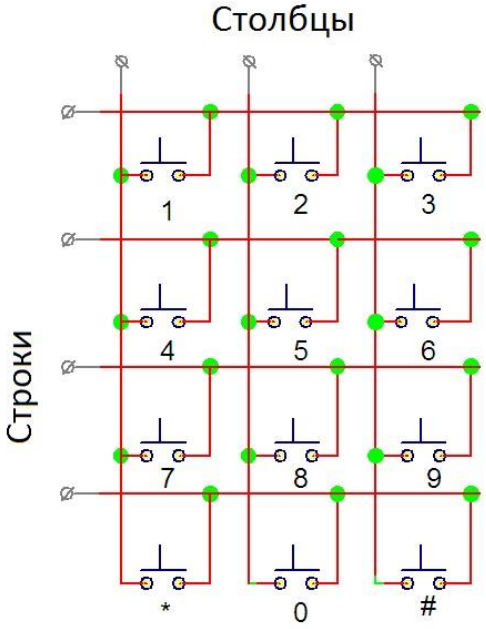


Рис. 7.1 — Схема матричной клавиатуры

Выводы строк и столбцов подключаются к МК. Опрос матричной клавиатуры осуществляется либо по строкам, либо по столбцам. Для опроса первой строки на ее вывод подается логический уровень 0, при этом на всех выводах столбцов включена подтяжка к высокому логическому уровню 1. Нажатие первой, второй или третьей кнопок замкнет соответствующий столбец на землю (логический уровень 0), что регистрируется МК. Опрос остальных строк аналогичен.

На плате Pinboard II реализована матричная клавиатура 3x4, расположение которой показано на рис. 8.2.

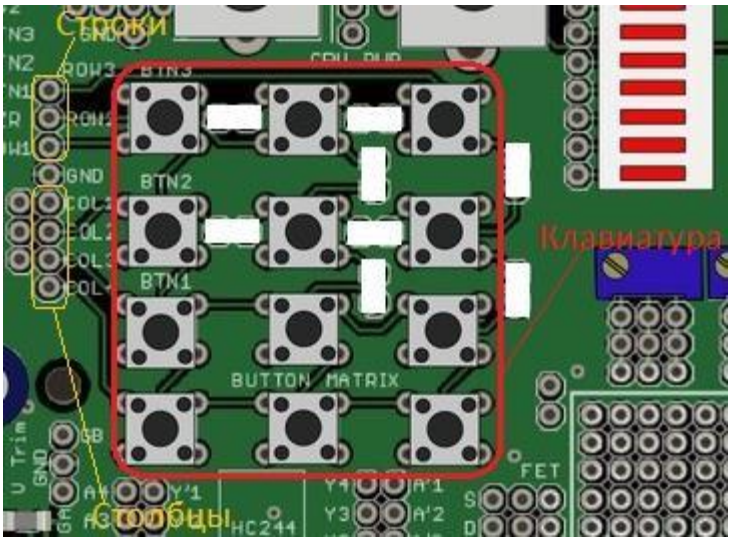


Рис. 8.2 — Матричная клавиатура на Pinboard II

Внимание! Одновременное нажатие нескольких кнопок недопустимо, поскольку приведет к конфликту логических уровней МК

Программа, приведенная в примере, требует такого подключения ССИ и клавиатуры к процессорному модулю:

*D*7<–>*A*05; *D*5<–>*A*07; *D*3<–>A10; *D*1 <–>*A*09; *D*0 <–>*A*04; *D*6<–>*A*06;

*D*4<–>A08; *ROW1*<–>*B05*; *ROW2*<–>B06; *ROW3*<–>*B*07; *COL1*<–>*A*03; *COL2*<–>*A*02; *COL3*<–>A01; *COL4* <–>*A*00;

При создании проекта в разделе *Repository* следует поставить отметку на пункте *GPIO* , чтобы в проект были добавлены необходимые компоненты.

Программа 8.1 — Программа отображает номер нажатой кнопки на ССИ

```
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"

#define HIGH  GPIO_SetBits // Команда установки бита в 1
#define LOW GPIO_ResetBits // Команда сброса бита в 0
#define READ GPIO_ReadInputDataBit // Команда считывающая состояние пина
#define IND_PORT GPIOA
// К какой ноге какой сегмент подключен
#define SEG_A GPIO_Pin_5
#define SEG_B GPIO_Pin_7
#define SEG_C GPIO_Pin_10
#define SEG_D GPIO_Pin_9
#define SEG_E GPIO_Pin_4
#define SEG_F GPIO_Pin_6
#define SEG_G GPIO_Pin_8
//Собираем цифры из сегментов
```

```
#define DIG0 ( SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F )
#define DIG1 ( SEG_B | SEG_C )
#define DIG2 ( SEG_A | SEG_B | SEG_G | SEG_E | SEG_D )
#define DIG3 ( SEG_A | SEG_B | SEG_G | SEG_C | SEG_D )
#define DIG4 ( SEG_F | SEG_G | SEG_B | SEG_C)
#define DIG5 ( SEG_A | SEG_F | SEG_G | SEG_C | SEG_D )
#define DIG6 ( SEG_A | SEG_C | SEG_D | SEG_E | SEG_F | SEG_G )
#define DIG7 ( SEG_A | SEG_B | SEG_C )
#define DIG8 ( SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F | SEG_G)
#define DIG9 ( SEG_A | SEG_B | SEG_C | SEG_D | SEG_F | SEG_G)
#define A    ( SEG_A | SEG_B | SEG_C | SEG_E | SEG_F|  SEG_G)
#define B    ( SEG_C | SEG_D | SEG_E | SEG_F | SEG_G)
#define NONE (SEG_G)// прочерк
 int
n=12;
//Функция формирующая небольшую задержку
void delay(void)
{
      int i=0;
      for (i=0; i != 0x5000; i++);
}
void initGPIO(void)
{
 RCC_APB2PeriphClockCmd( RCC_APB2Periph_GPIOA|RCC_APB2Periph_GPIOB , ENABLE);

     GPIO_InitTypeDef PORTA; //Структура, содержащая настройки порта
      PORTA.GPIO_Pin = GPIO_Pin_All; //Указываем, какие выводы нужно настроить
      PORTA.GPIO_Mode = GPIO_Mode_Out_PP; // Настраиваем как выход Push-pull
      PORTA.GPIO_Speed = GPIO_Speed_2MHz; // Частота 2 МГц
 GPIO_Init(GPIOA, &PORTA); //Вызываем функцию настройки порта
 GPIO_InitTypeDef ROWS; //Структура, содержащая настройки порта
 ROWS.GPIO_Pin = GPIO_Pin_5|GPIO_Pin_6|GPIO_Pin_7; //Указываем, какие выводы
нужно настроить
 ROWS.GPIO_Mode = GPIO_Mode_IPU; // Настраиваем как вход с подтяжкой к лог 1
      ROWS.GPIO_Speed = GPIO_Speed_2MHz; // Частота 2 МГц
      GPIO_Init(GPIOB, &ROWS); //Вызываем функцию настройки порта
}
void indication(int n)
{
// объявляем массив с элементами соответствующими цифрам 0-9
 unsigned int
digit[]={DIG0,DIG1,DIG2,DIG3,DIG4,DIG5,DIG6,DIG7,DIG8,DIG9,A,B,NONE};
      HIGH(IND_PORT,GPIO_Pin_12);// включаем первый разряд
      HIGH(IND_PORT,digit[n]); // включаем нужные сегменты
      delay();
      LOW(IND_PORT,digit[n]); // выключаем их
}
void read_keyboard(void)
{
      // установка всех столбцов в лог. 1
 HIGH (GPIOA,GPIO_Pin_0|GPIO_Pin_1|GPIO_Pin_2|GPIO_Pin_3);
 LOW (GPIOA,GPIO_Pin_3);// опрашиваем первый столбец
if (!READ(GPIOB,GPIO_Pin_5))   n=1;
 if (!READ(GPIOB,GPIO_Pin_6))
 n=2;
      if (!READ(GPIOB,GPIO_Pin_7))
      n=3;
```

```
      HIGH (GPIOA,GPIO_Pin_3);//возвращаем как было
//-------------------------------------
 LOW (GPIOA,GPIO_Pin_2);// опрашиваем второй столбец
if (!READ(GPIOB,GPIO_Pin_5))
      n=4;
      if (!READ(GPIOB,GPIO_Pin_6))
      n=5;
      if (!READ(GPIOB,GPIO_Pin_7))
      n=6;
      HIGH (GPIOA,GPIO_Pin_2);
//-------------------------------------
 LOW (GPIOA,GPIO_Pin_1);// опрашиваем третий столбец
if (!READ(GPIOB,GPIO_Pin_5))
      n=7;
      if (!READ(GPIOB,GPIO_Pin_6))
      n=8;
      if (!READ(GPIOB,GPIO_Pin_7))
      n=9;
      HIGH (GPIOA,GPIO_Pin_1);
      //-------------------------------------
      LOW (GPIOA,GPIO_Pin_0); // опрашиваем четвертый столбец
      if (!READ(GPIOB,GPIO_Pin_5))
      n=10;
      if (!READ(GPIOB,GPIO_Pin_6))
      n=0;
      if (!READ(GPIOB,GPIO_Pin_7))
      n=11;
      HIGH (GPIOA,GPIO_Pin_0);
}
// Основная программа
int main(void)
{
      initGPIO();
      while(1)
            {
            read_keyboard();
            indication(n);
            }
}
```

7.3. Порядок выполнения работы

Используя программу 7.1 в качестве образца, выполнить индивидуальное задание в соответствии с таблицей 7.1.

Таблица 7.1 — Индивидуальное задание по лабораторной работе № 5

№ варианта	**Задание**
1	С помощью клавиатуры ввести трехзначное целое десятичное число. Отобразить это число на семисегментном индикаторе. Если введенное число больше 512, то вывести на линейку светодиодов бегущий огонек. Если введенное число равно 333, то вся линейка светодиодов должна мигать с частотой 1 Гц. Если введенное число меньше 700, то его изображение на семисегментных индикаторах должно мигать с частотой 1 Гц. Одна кнопка должна использоваться для сброса введенного числа, при ее нажатии на семисегментный индикатор должен погаснуть.
2	Разработать калькулятор, способный складывать введенные с клавиатуры двузначные десятичные числа. Предусмотреть очистку.
3	Разработать калькулятор, переводящий введенное двоичное число (должно отображаться на линейке светодиодов) в шестнадцатеричное число, отображаемое на семисегментных индикаторах. При вводе числа использовать две кнопки: 0 и 1. Предусмотреть очистку введенных значений.
4	Разработать калькулятор, переводящий введенное двузначное десятичное число (должно отображаться семисегментном индикаторе) в двоичное число, отображаемое на линейке светодиодов. Предусмотреть очистку введенных значений.
5	Разработать калькулятор, переводящий введенное двузначное десятичное число (должно отображаться семисегментном индикаторе) в шестнадцатеричное число, отображаемое на семисегментных индикаторах. Преобразование должно происходить по нажатию кнопки. Предусмотреть очистку введенных значений.
6	Разработать калькулятор, способный принимать числа в двоичном коде (отображаются на линейке светодиодов) и выполнять над ними операции сложения и вычитания.
7	Разработать калькулятор, складывающий вводимые одноразрядные десятичные числа. Вводимые числа должны отображаться в двоичном коде на линейке светодиодов, а результат сложения — в десятичном коде на семисегментном индикаторе.

№ варианта	Задание
8	Разаработать калькулятор, складывающий два вводимых в двоичном коде числа. Вводимые числа и результат сложения должны отображаться в двоичном коде на линейке светодиодов. Если получившееся число нечетное, то на семисегментном индикаторе должно отображаться «111», если четное — мигающее с частотой 1 Гц значение «000».
9	Запрограммировать на клавиатуре кнопки таким образом, чтобы при нажатии одной из них движение бегущего огонька на линейке светодиодов было слева направо, при нажатии другой — справа налево; нажатия на третью приводили бы к увеличению скорости движения огонька (чем больше нажатий, тем больше скорость); нажатия на четвертую приводили бы к уменьшению скорости движения огонька (чем больше нажатий, тем меньше скорость). При этом скорость движения огонька (в условных единицах) должна отображаться на семисегментном индиаторе.
10	Ввести двузначное число в десятичном коде (должно отображаться на семисегмендтном индикаторе), которое в двоичном коде (по нажатию выбранной кнопки) должно циклически перемещаться на линейке светодиодов.
11	Разработать калькулятор, способный принимать числа с клавиатуры в двоичном коде (отображаются на линейке светодиодов) и выполнять над ними операции умножения и сложения. Результат должен отображаться на семисегментном индикаторе в шестнадцатеричном коде.
12	С помощью клавиатуры ввести двузначное целое десятичное число. Отобразить это число в двоичном коде на линейке светодиодов. Если введенное число больше 32, то вывести семисегментный индикатор «000». Если введенное число равно 21, то это число должно быть выведено на семисегментный индикатор и мигать с частотой 1 Гц. Если введенное число меньше 70, то его изображение на семисегментных индикаторах отобразить «333». Одна кнопка должна использоваться для сброса введенного числа.

№ варианта	Задание
13	Разработать программу, выполняющую действия A*B+C над тремя введенными с клавиатуры двузначными десятичными числами A, B, C. Вводимые числа и результат отображать на семисегментром индикаторе в десятичном коде и на линейке светодиодов в двоичном коде.
14	На семисегментном индикаторе должно отображаться число, изменяющееся на величину A со скоростью одно изменение в секунду. Значение A должно вводиться с клавиатуры о начала счета. Направление счета должно меняться нажатием выбранной кнопки клавиатуры во время счета.
15	Разработать калькулятор, вычисляющий значения синуса и косинуса введенного с клавиатуры числа в радианах в формате «x.xx», где x — десятичные цифры, отображаемые на семисегментром индикаторе. Результат должен выводиться на семисегментном индикаторе в виде «0.xx».
16	Разработать калькулятор, рассчитывающий длину вектора A+B, где векторы A и B имеют вид: A=(x, 0), B=(0, y). Значения x, y должны вводиться с клавиатуры в виде двузначных десятичных чисел. Целая часть результата в виде десятичного трехзначного числа должна отображаться на семисегментном индикаторе.

7.4. Содержание отчета

Отчет по работе оформляется в соответствии с требованиями, приведенными во введении.

7.5. Вопросы для самоконтроля

7.5.1. то нужно сделать, чтобы можно было обрабатывать ситуацию одновременного нажатия нескольких клавиш?

7.5.2. Как осуществляется определение столбца клавиатуры, в котором нажата клавиша?

7.5.3. Как производится идентификация клавиши в пределах одного столбца клавиатуры?

7.5.4. Как организовать однократное выполнение какого-нибудь события при однократном кратковременном нажатии клавиши клавиатуры?

7.5.5. Как организовать циклическое выполнение события на время удержания клавиши нажатой?

7.5.6. Как организовать выполнение события заданное число раз при кратковременном однократном нажатии клавиши?

8. ЛАБОРАТОРНАЯ РАБОТА № 6

ПРОГРАММИРОВАНИЕ ТАЙМЕРА-СЧЕТЧИКА

8.1. Цель работы

Целью работы является изучение базовых возможностей таймера-счетчика в МК семейства *STM*32.

8.2 Теоретические сведения

Одним из важнейших модулей в МК являются таймеры-счетчики. В МК *STM*32*F*103*C*8 есть несколько типов таймеров*: general-purpose timers, advancedcontrol timer, watchdog timers, SysTick timer*. В данной работе исследуются *generalpurpose timers*. Таймер-счетчик в МК позволяет формировать точные интервалы времени, осуществлять ШИМ-модуляцию и подсчет внешних импульсов и т.д.

Рассмотрим принцип формирования точных временных интервалов и генерацию аналоговых сигналов методом ШИМ.

Программа, приведенная в первом примере, требует установки джампера на процессорном модуле так, как показано на рис 8.1, при этом пьезоизлучатель подключается к выводу A00 (PA0) МК.

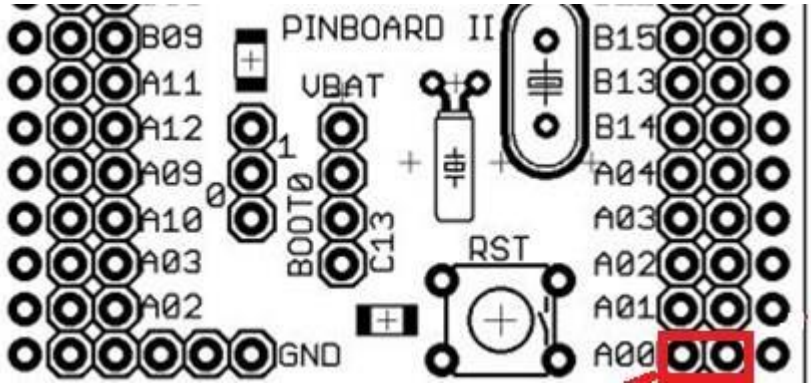


Рис. 8.1 — Необходимые соединения

Программа 8.1 — Программа осуществляет генерацию сигнала в виде меандра с скважностью 2 с частотой, линейно изменяющейся от 1 кГц до 9 кГц.

```
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
int n=0
void delay(void)// объявление функции временной задержки
{
      int i;
      for(i=0;i<0x50000;i++);
}
```

```
int main()
{
  GPIO_InitTypeDef PORT;//Включаем порт A и таймер 4
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA , ENABLE);
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM4,ENABLE);
  PORT.GPIO_Pin = (GPIO_Pin_0);
  PORT.GPIO_Mode = GPIO_Mode_Out_PP;
  PORT.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_Init(GPIOA, &PORT);

  TIM4->PSC =1; // Настраиваем делитель таймера
  TIM4->DIER |= TIM_DIER_UIE; //Чтобы таймер генерировал прерывания
  TIM4->CR1 |= TIM_CR1_CEN; // Запуск таймера
  NVIC_EnableIRQ(TIM4_IRQn); //Разрешение прерывания от таймера
while(1)
  {// обеспечивание в цикле линейное изменение частоты
         n=3000;
         while (n<25000)
    {
n+=1000;
          TIM4->ARR = n;
delay();
         }
  }// от содержимого ARR зависит частота следования импульсов
}
// Обработчик прерывания TIM4_DAC
void TIM4_IRQHandler(void)
{
  TIM4->SR &= ~TIM_SR_UIF; //Сбрасываем флаг прерывания по таймеру 4
  GPIOA->ODR^=(GPIO_Pin_0); //Инвертируем состояние вывода PA0 }
```

Программа, приведенная во втором примере, требует установки джампера на отладочной платете так, как показано на рис 8.2, а также соединения контактов *A*02 с *PW LED*0, при этом обеспечивается подача ШИМ сигнала через *RC* ФНЧ на пьезоизлучатель.

Рис. 8.2 — Необходимые соединения

Программа 8.2 — Программа осуществляет генерацию гармонического сигнала с частотой 1 кГц.

```
#include "stm32f10x.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_rcc.h"
void delay(void)
```

```
{
  volatile uint32_t i;
for (i=1; i != 580; i++);
}
int main()
{
  //Включаем порт A
  RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA , ENABLE);
  //Включаем Таймер 2
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,ENABLE);

  GPIO_InitTypeDef PORT;
  // Настроим вывод на выход
  PORT.GPIO_Pin = (GPIO_Pin_2);
  //Будем использовать альтернативный режим(вывод ШИМ)
  PORT.GPIO_Mode = GPIO_Mode_AF_PP;
  PORT.GPIO_Speed = GPIO_Speed_2MHz;
  GPIO_Init(GPIOA, &PORT);
  //Разрешаем таймеру использовать вывод PA1 для ШИМа
  TIM2->CCER |= (TIM_CCER_CC3E);
  TIM2->CCMR2|=( TIM_CCMR2_OC3M_1 | TIM_CCMR2_OC3M_2);
  //Запускаем таймер
  TIM2->CR1 |= TIM_CR1_CEN;
  TIM2->ARR = 255 ;// таймер считает до 255
  //что аналогично 8-ми разрядному таймеру,однако по факту счетчик 16разрядный
  // массив отсчетов синусоиды
  int sin[16]={127,176,217,244,254,244,217,176,127,79,37,10,0,10,37,78};
int i;
while(1)
  {
        for (i=1;i<=15;i++)
        {
        // заносим в регистр сравнения требуемый коэффициент заполнения
        TIM2->CCR3=sin[i];
   // ждем перед тем как выдать следующий отсчет
delay();
        }
  }
}
```

8.3. Порядок выполнения работы

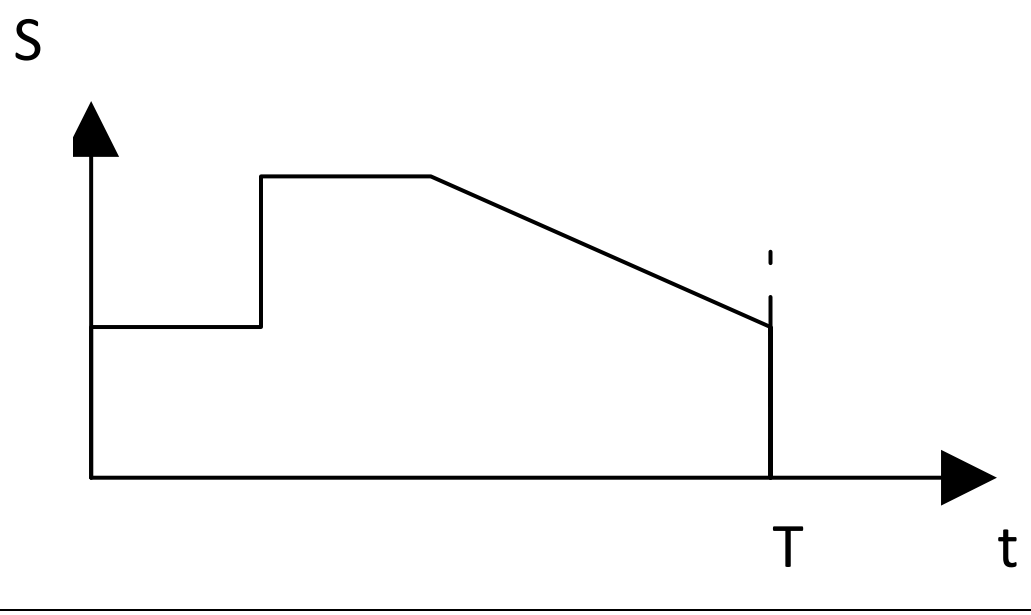
Пользуясь программами 8.1 и 8.2 в качестве образца, создать периодический сигнал с периодом T (табл. 8.1) и зависимостью выводимых значений *S* от времени в соответствии с табл. 8.2. Синтезированный сигнал вывести на осциллограф.

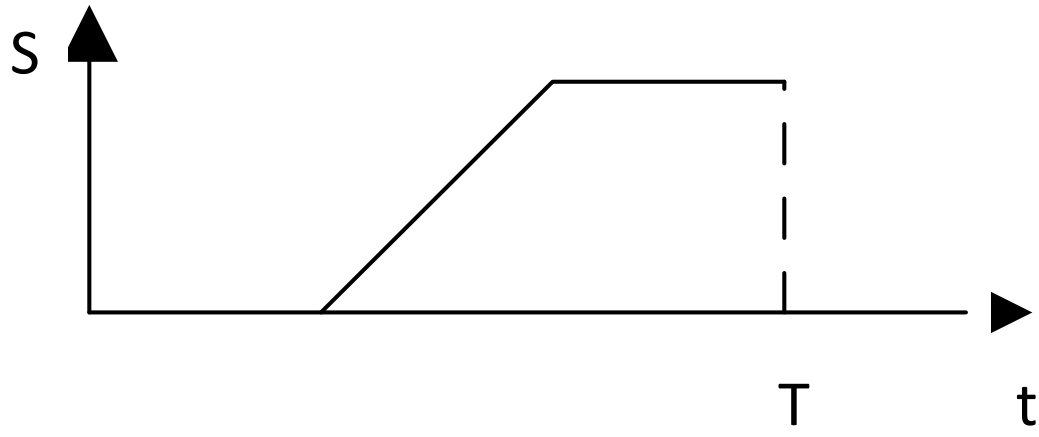
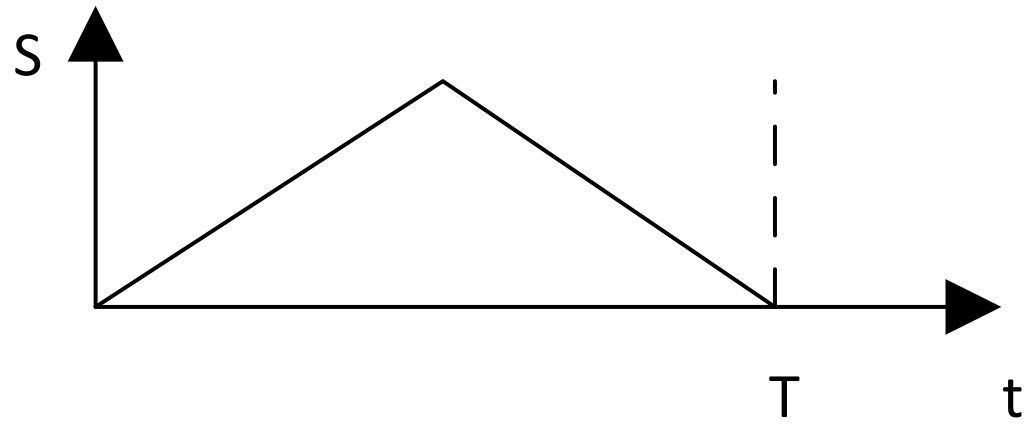
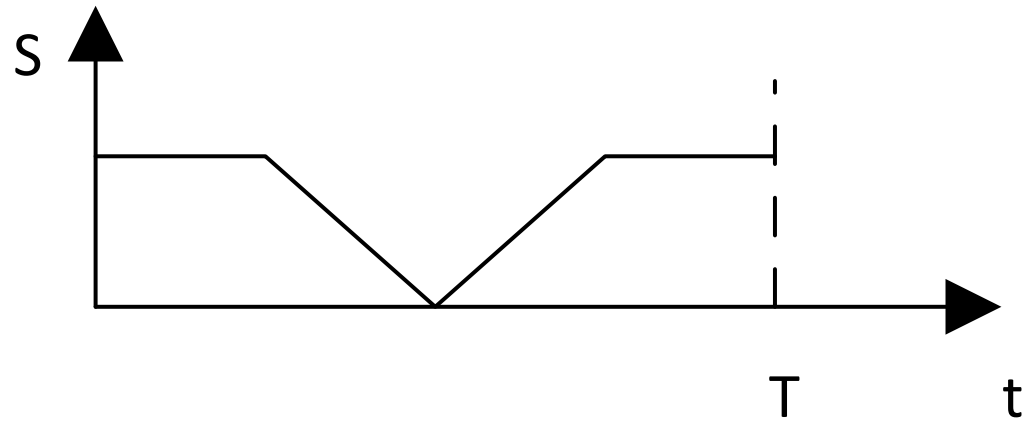
Таблица 8.1 — Индивидуальные задания периода сигнала

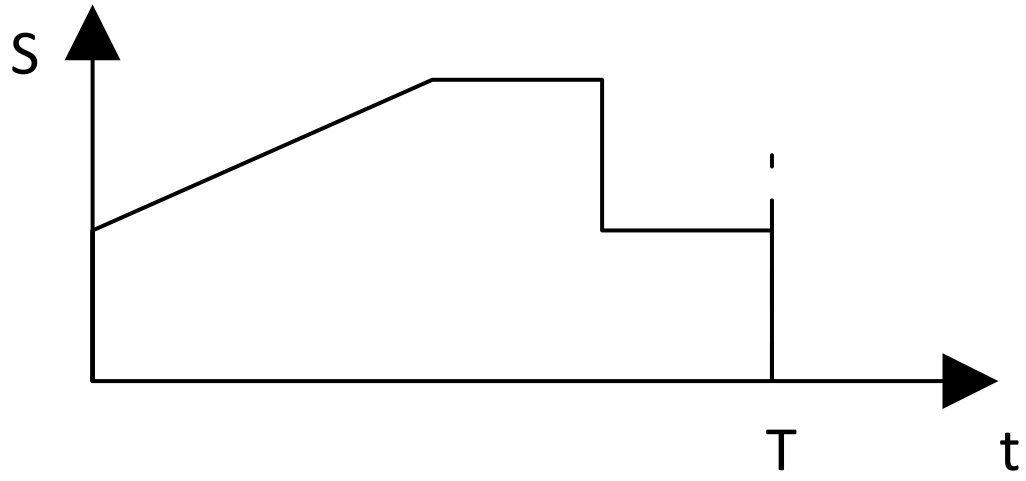
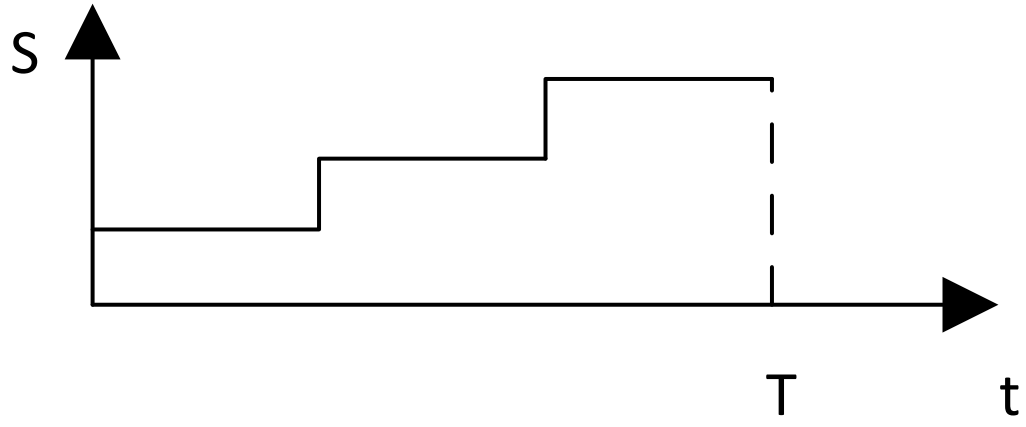
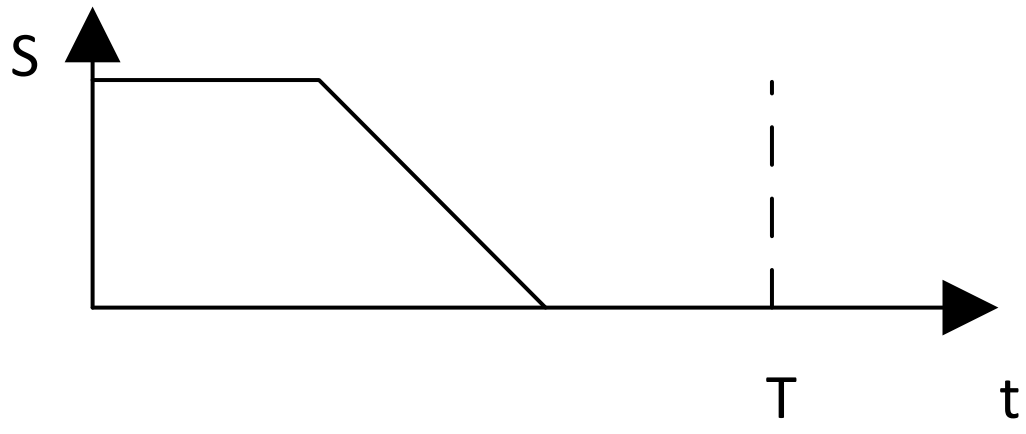
№ п/п	Период T сигнала, кГц
1	2
2	4
3	15

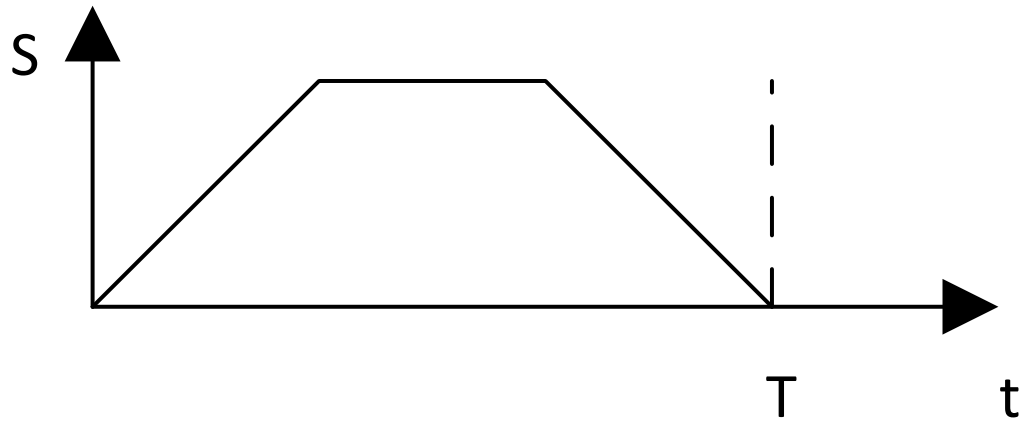
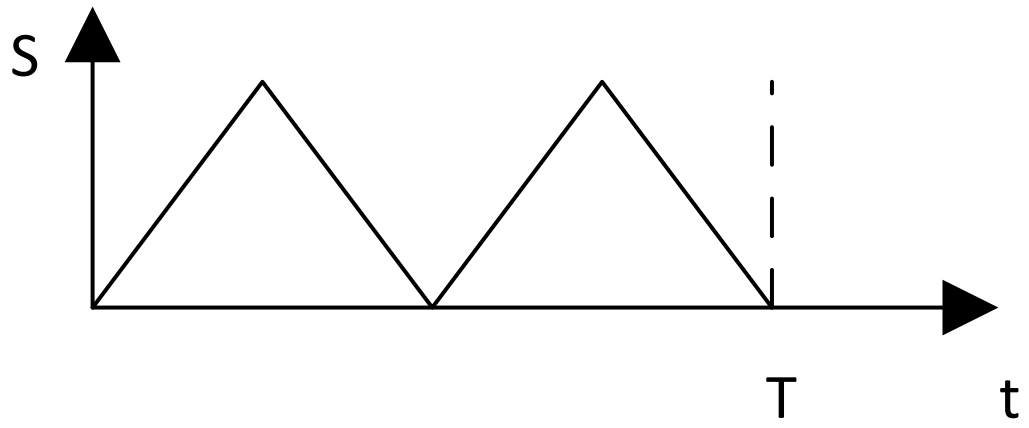
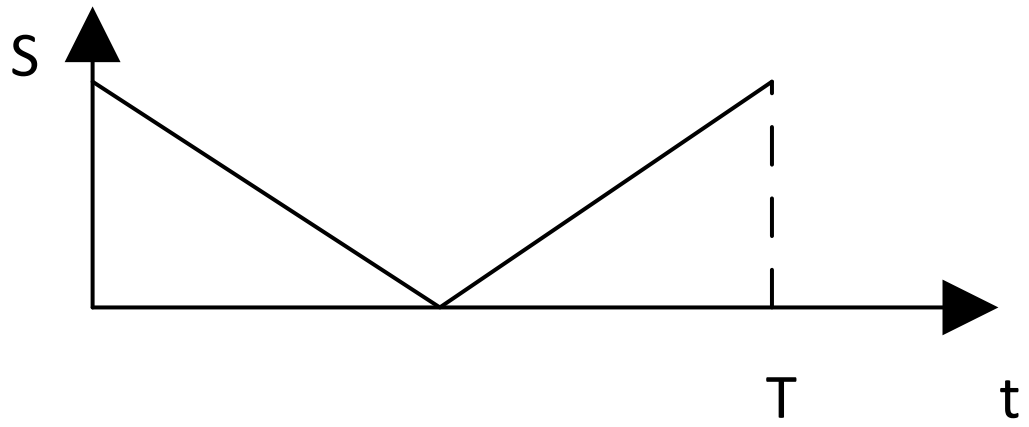
№ п/п	Период Т сигнала, кГц
4	5
5	1
6	6
7	10
8	9
9	7
0	8
11	11
12	14
13	16
14	12
15	4
16	13

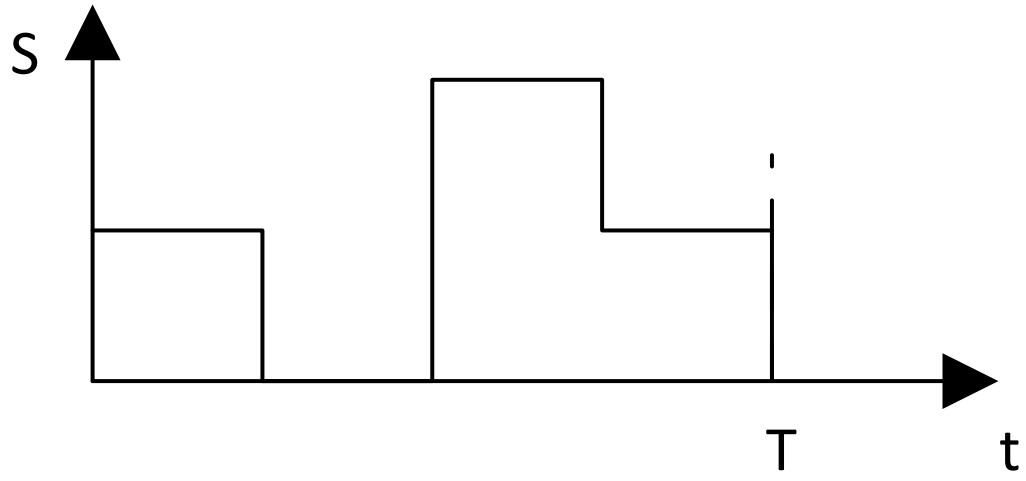
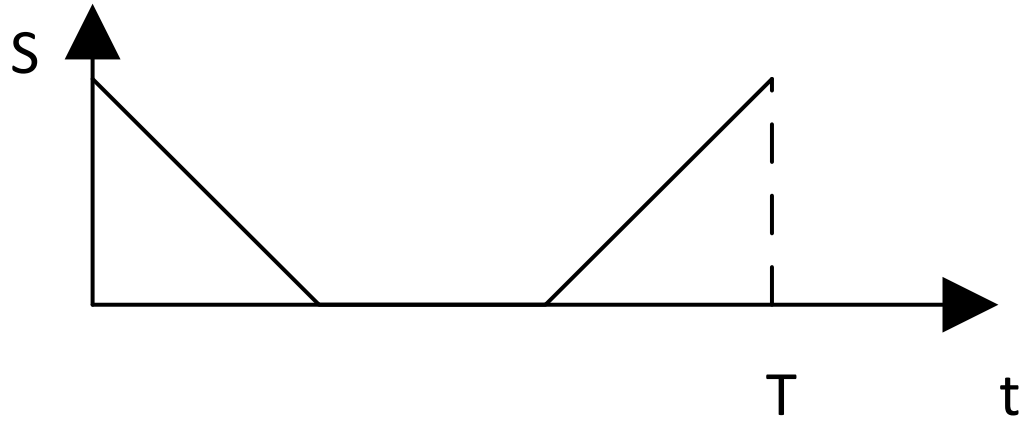
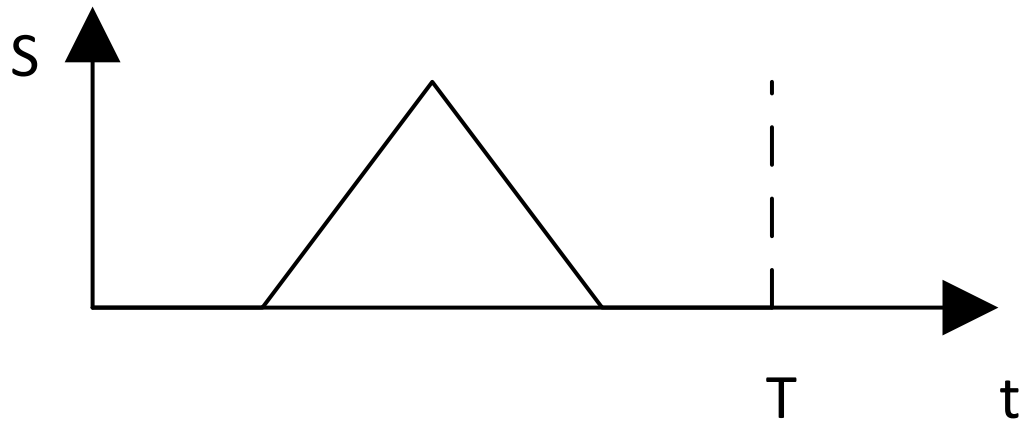
Таблица 8.2 — Индивидуальные задания зависимости выводимых значений *S* от времени

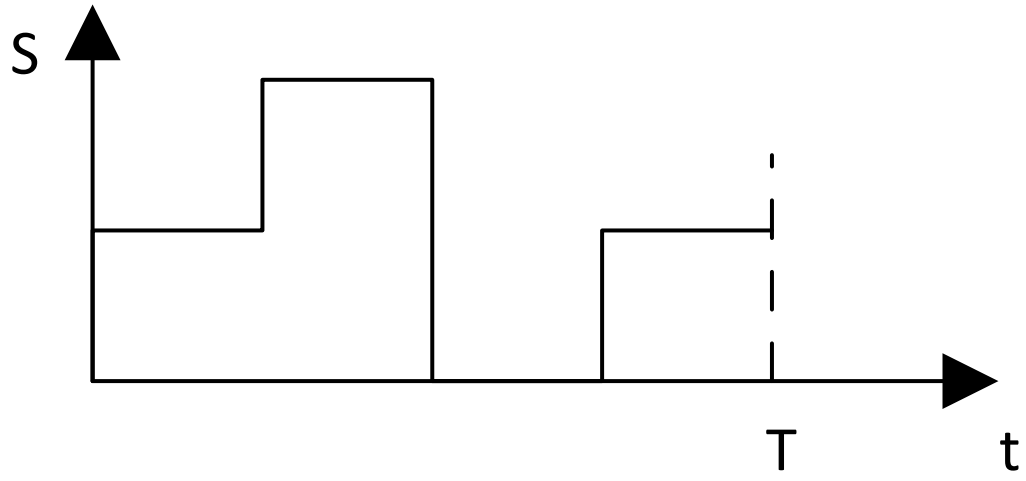
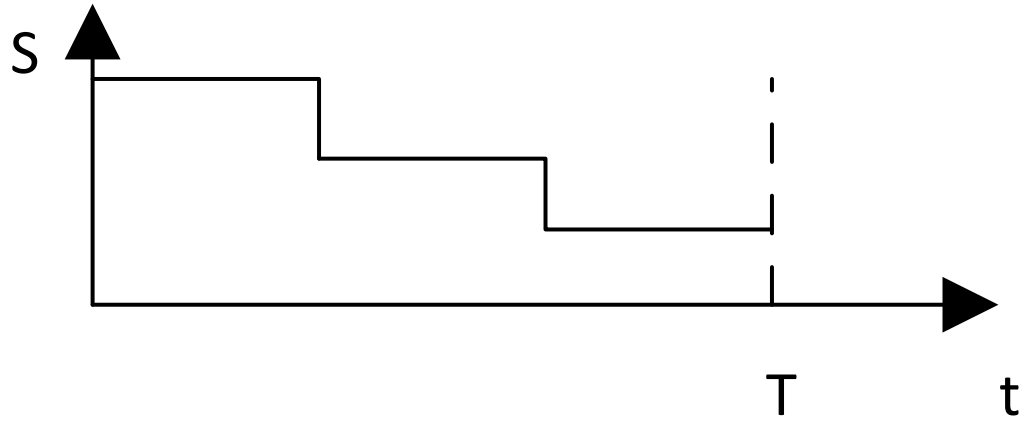
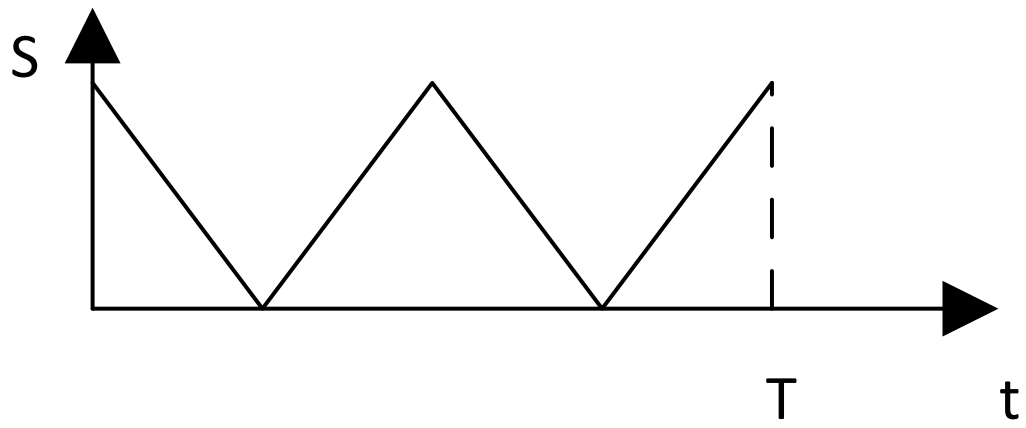
№ ва-рианта	**Задание**
1	

№ варианта	Задание
2	
3	
4	

№ варианта	Задание
5	
6	
7	

№ варианта	Задание
8	
9	
10	

№ варианта	Задание
11	
12	
13	

№ варианта	Задание
14	
15	
16	

8.4. Содержание отчета

Отчет по работе оформляется в соответствии с требованиями, приведенными во введении.

8.5. Вопросы для самоконтроля

8.5.1. Сколько и каких таймеров/счетчиков входит в состав микроконтроллера *STM*32*F*103*C*8? 8.5.2. Какие типы прерываний могут генерировать таймеры-счетчики?

8.5.3. В чем состоит отличие работы таймера/счетчика в режиме таймера и в режиме счетчика?

8.5.4. Для чего используется сторожевой таймер?

8.5.5. В какой области памяти находятся регистры таймеров-счетчиков?

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Оформление текстовых работ: методические указания / СевГУ; сост. В. Г. Слезкин, П. П. Ермолов. — Севастополь : Изд-во СевГУ, 2015. — 19 с.

2. Электроника для всех [Электронный ресурс]. — Режим доступа: http://shop.easyelectronics.ru/index.php (дата обращения: 15.05.2015)

3. CooCox Free/open ARM Cortex-M Development Tool-chain [Электронный ресурс]. — Режим доступа: http://www.coocox.org/ (дата обращения: 15.05.2015)

4. Джозеф, Ю. Ядро CortexM3 компании ARM. Полное руководство / Ю. Джозеф; пер. с англ. А. В. Евстифеева. М. — : ДодэкаXXI, 2012. — 552 с.

ПРОГРАММИРОВАНИЕ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ.
Лабораторный практикум по дисциплине

Учебно-методическое пособие

Составители : **Щекатурин** Андрей Алексеевич,
Руднев Евгений Петрович

В авторской редакции

Изд. № 42/2021. Объем 4 п.л.
РИИЦМ ФГАОУ ВО «Севастопольский государственный университет»