МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральные государственные автономные образовательные учреждения высшего образования
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»**
**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ТОМСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**
Федеральное государственное бюджетное учреждение науки
**ИНСТИТУТ ОПТИКИ АТМОСФЕРЫ ИМ. В.Е. ЗУЕВА
СИБИРСКОГО ОТДЕЛЕНИЯ РОССИЙСКОЙ АКАДЕМИИ НАУК**

**С.Н. Торгаев, М.В. Тригуб,
И.С. Мусоров, Д.С. Чертихина**

ПРАКТИЧЕСКОЕ РУКОВОДСТВО ПО ПРОГРАММИРОВАНИЮ STM-МИКРОКОНТРОЛЛЕРОВ

*Рекомендовано в качестве учебного пособия
Редакционно-издательским советом
Томского политехнического университета*

Издательство
Томского политехнического университета
2015

УДК 681.322 (075.8)
ББК 32.973.26-04я73
Т60

Торгаев С.Н.

Т60 Практическое руководство по программированию STM-микроконтроллеров : учебное пособие / С.Н. Торгаев, М.В. Тригуб, И.С. Мусоров, Д.С. Чертихина ; Томский политехнический университет. – Томск : Изд-во Томского политехнического университета, 2015. – 111 с.

В пособии рассмотрены вопросы программирования микроконтроллеров STM8S, STM32F10x и STM32F40x. Представлено большое количество примеров программ по настройке основных периферийных устройств для данных микроконтроллеров.

Предназначено для студентов, обучающихся по направлениям 110304 «Электроника и наноэлектроника», 120304 «Биотехнические системы и технологии».

УДК 681.322 (075.8)
ББК 32.973.26-04я73

Рецензенты

Кандидат физико-математических наук
научный сотрудник отдела высоких плотностей энергии
Института сильноточной электроники СО РАН
Д.В. Рыбка

Кандидат технических наук
младший научный сотрудник
лаборатории МПКМ ИФПМ СО РАН
М.В. Бурков

Содержание

Введение

На сегодняшний день большую популярность среди разработчиков электронной аппаратуры различного назначения получили микроконтроллеры компании *STMicroelectronics*. Это связано с тем, что данные микроконтроллеры имеют ряд преимуществ перед существующими аналогами. В частности микроконтроллеры *STM* в модельном ряду сохраняют так называемую совместимость *pin-to-pin*, т. е. есть возможность замены микроконтроллера на более современную модель, имеющую большую память и более «богатую» периферию. Для сохранения совместимости создается набор периферийных устройств на весь модельный ряд. При этом при разработке конкретного микроконтроллера из модельного ряда для него используется определенная периферия с сохранением нумерации.

Данное учебное пособие посвящено вопросам настройки периферийных устройств микроконтроллеров *STM8S*, *STM32F10x* и *STM32F40x* и их программированию. В пособии представлено большое количество примеров программ для вышеуказанных микроконтроллеров, написанных на языке С.

Глава 1. Программирование микроконтроллеров STM

1.1. Загрузка программы в микроконтроллер STM8S

Для программирования микроконтроллеров семейства *STM*8 используется интерфейс *SWIM* (*single wire interface module*). Данный интерфейс позволяет осуществлять прошивку и отладку микроконтроллера, находящегося непосредственно в схеме. Загрузка программы в память микроконтроллера осуществляется с помощью программатора *ST-LINK* [1].

На отладочной плате *STM*8*SVLDISCOVERY* [1] модуль *ST-LINK* находится в верхней части (рис. 1.1). Для программирования используется 4 вывода (разъем *CN*7, см. рис. 1.1):

1. *VDD*;
2. *SWIM* (вывод *PD*1 порта *D* микроконтроллера);
3. *GND*;
4. *RESET*.

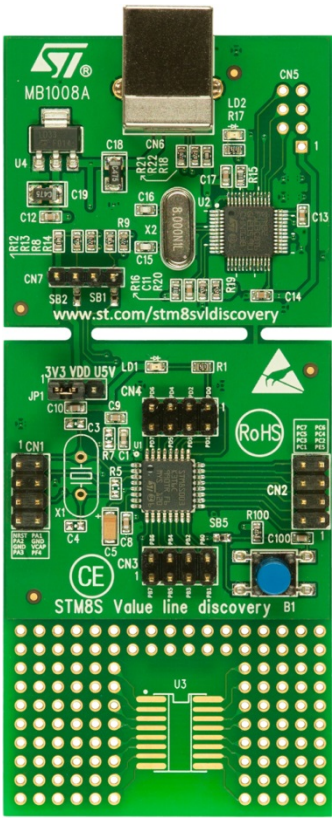

Рис. 1.1. Отладочная плата STM8SVLDISCOVERY

Для того чтобы использовать данный программатор для прошивки внешнего микроконтроллера, необходимо либо отделить его от основной платы микроконтроллера, либо удалить перемычки *SB*1 и *SB*2.

Кроме того, для программирования можно использовать программатор *ST-LINK/V*2 (рис. 1.2) [2]. Верхний разъем (20 выводов) используется для прошивки микроконтроллеров серии *STM*32, а нижний разъем, состоящий из четырех выводов и идентичный с разъемом *CN*7 (см. рис. 1.1), – для программирования микроконтроллеров *STM*8.

Рис. 1.2. Программатор ST-LINK/V2

На рис. 1.3 представлена схема для прошивки и отладки микроконтроллеров семейства *STM*8*S* с помощью *ST-LINK*. Конденсатор *C*1 – конденсатор по питанию микросхемы (0,1–2,2 мкФ); конденсатор *C*2 следует выбирать из диапазона 470–3300 нФ. На плате *STM*8*SVLDISCOVERY* конденсатор *C*2 – электролитический конденсатор с емкостью 680 нФ [1]. Первый и третий выводы *ST-LINK* (*VDD* и *GND*) не являются источниками питания для микроконтроллера, их соединяют с выводами питания микроконтроллера для согласования напряжений.

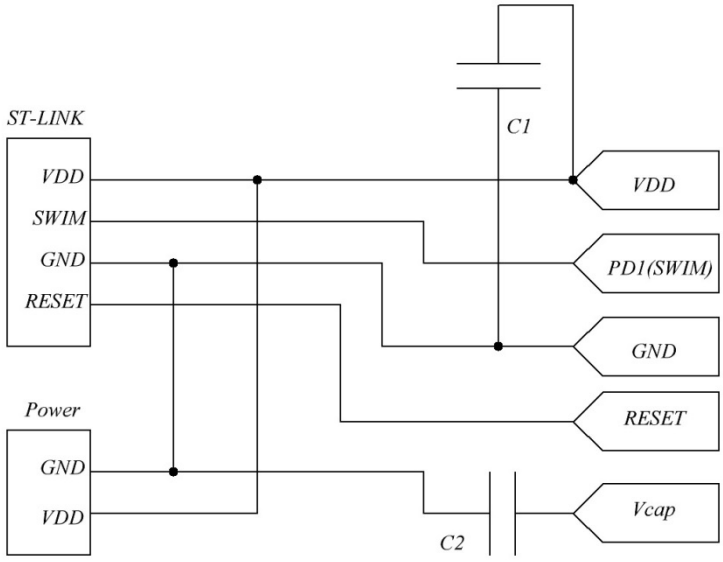

Рис. 1.3. Схема включения микроконтроллера при загрузке программы

1.2. Загрузка программ в микроконтроллер STM32

Для программирования микроконтроллеров семейства *STM*32 используются интерфейсы *JTAG* (*Joint Test Action Group*) и *SWD* (*Serial Wire Debugging*). На рис. 1.4 изображена отладочная плата *STM*32*VLDISCOVERY* [3]. В верхней ее части находится отладчик *ST-LINK*. Для прошивки микроконтроллера, установленного на данную отладочную плату, на разъеме *CN*3 должны быть установлены 2 перемычки, как показано на рис. 1.4. Прошивка и отладка осуществляются по интерфейсу *SWD*. При удалении перемычек с разъема *CN*3 отладчик *ST-LINK* будет загружать программу в микроконтроллер, подключенный к разъему *CN*2. Для этого необходимо соединить общие точки микроконтроллера и *ST-LINK*, а также выводы микроконтроллера *SWDIO* (*PA*13) и *SWDCLK* (*PA*14) соединить с контактами 4 и 2 разъема *CN*2 соответственно (рис. 1.5) [3].

Рис. 1.4. Отладочная плата STM32VLDISCOVERY

Также для прошивки и отладки микроконтроллеров *STM32* можно использовать программатор *ST-LINK/V*2 (разъем на 20 контактов). На рис. 1.5 показана схема подключения микроконтроллеров *STM*32 к *ST-LINK/V*2 для прошивки по интерфейсу *SWD* [3].

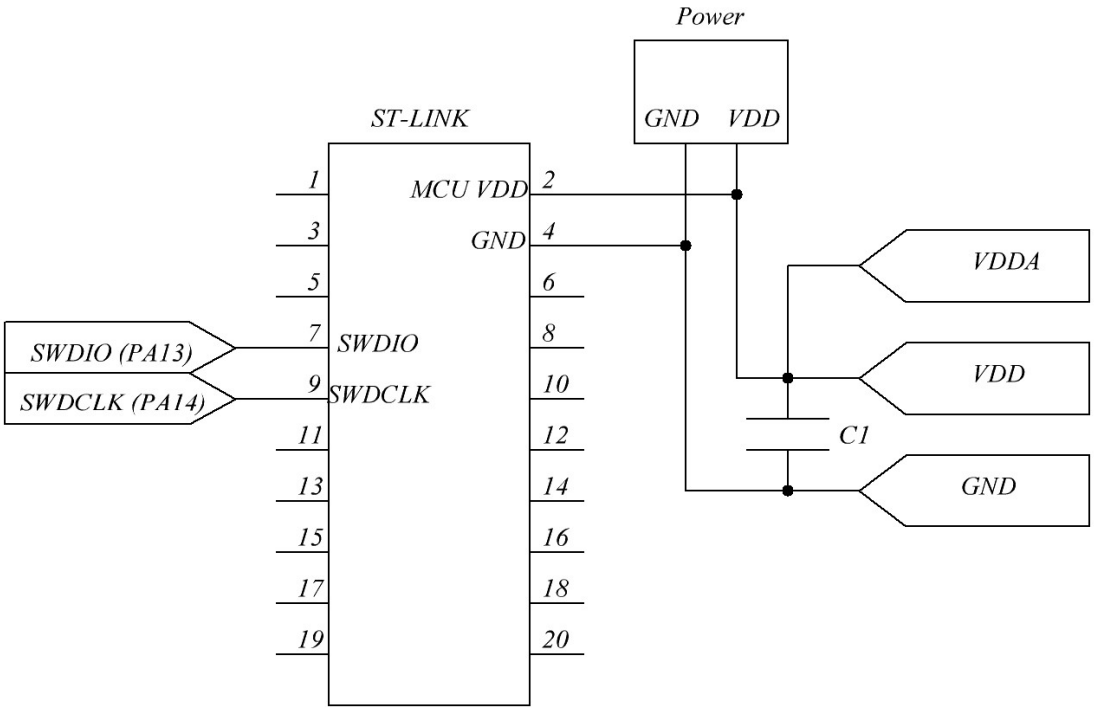

Рис. 1.5. Схема подключения микроконтроллеров STM32 к ST-LINK/V2 для прошивки по интерфейсу SWD

Контроллеры серии *STM*32 обычно имеют несколько выводов питания (*VDD*_1 (вывод питания) – *VSS*_1 (общий вывод), *VDD*_2 – *VSS*_2 и т. д., а также *VDDA* – *VSSA*). Перед прошивкой контроллера необходимо подать напряжение питания на все выводы питания цифровой части (*VDD*_1, *VDD*_2 и т. д.) и аналоговой части (*VDDA*).

Глава 2. Микроконтроллер STM8S

2.1. Создание проекта в программе IAR Embedded Workbench

Создание проекта в среде *IAR Embedded Workbench* осуществляется по следующему алгоритму:

1. Запускаем среду *IAR Embedded Workbench for STMicroelectronics STM*8. На рис. 2.1 представлен внешний вид стартового окна программы.

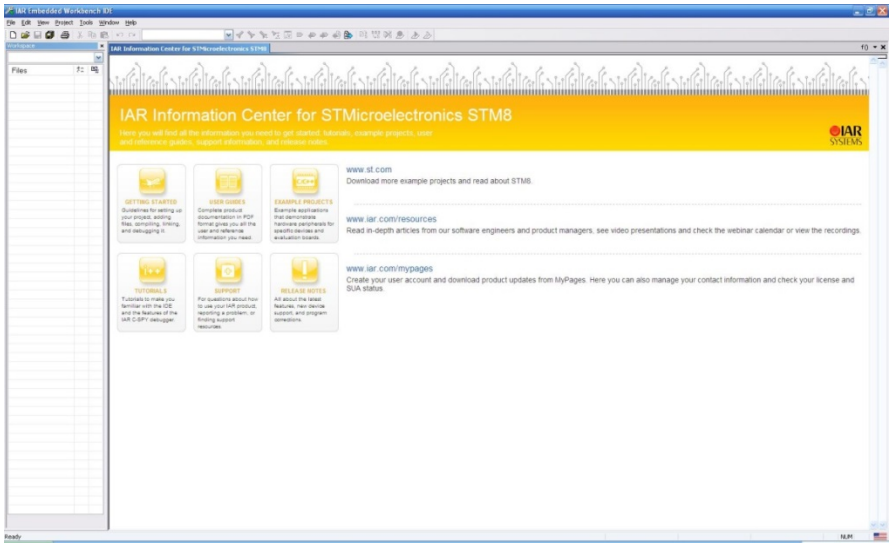

Рис. 2.1. Стартовое окно программы

2. Для создания нового проекта необходимо зайти в меню *Project* и выбрать пункт *Create new project*… (рис. 2.2).

3.

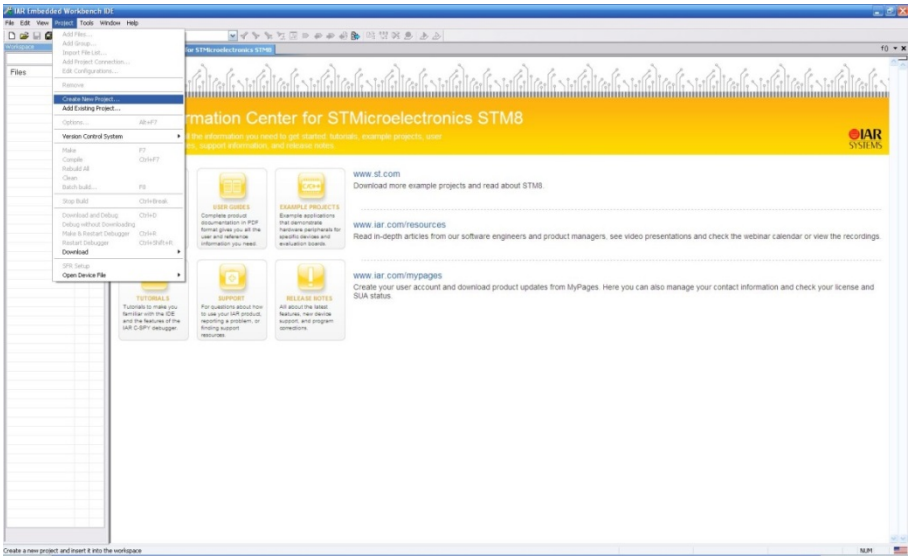

Рис. 2.2. Окно создания нового проекта

4. В появившемся окне (рис. 2.3) выбираем шаблон для языка С и тип микроконтроллера и далее сохраняем рабочую область – *Workspace* (рис. 2.4).

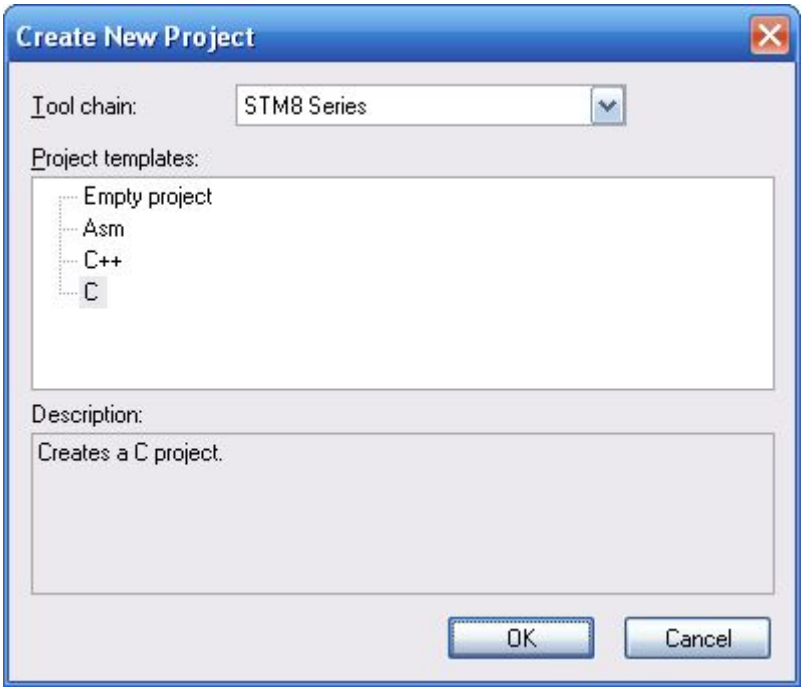

Рис. 2.3. Окно выбора языка программирования и микроконтроллера

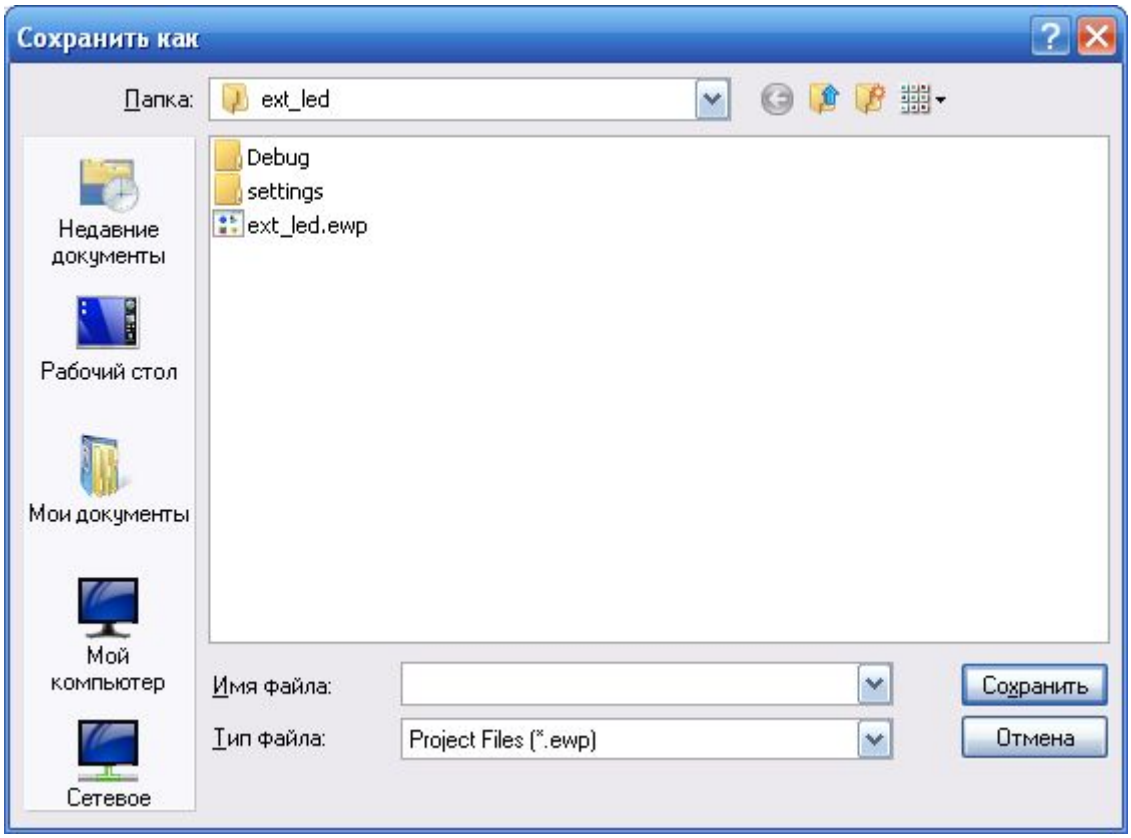

Рис. 2.4. Окно сохранения проекта

5. После сохранения проекта будет открыто рабочее окно проекта (рис. 2.5).

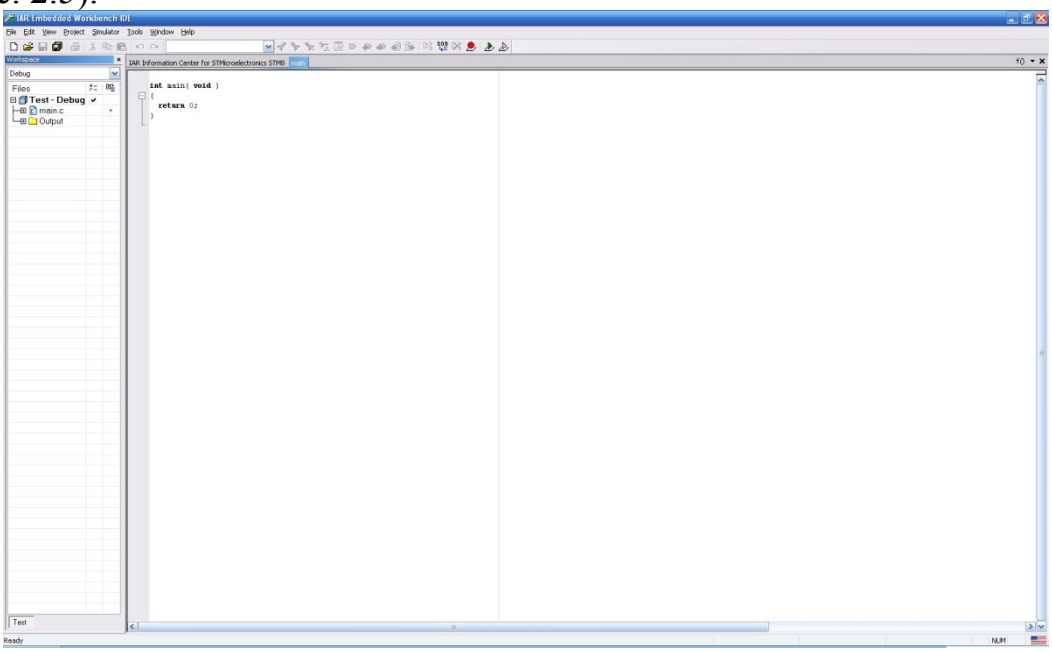

Рис. 2.5. Рабочее окно проекта

6. Для примера работы программы записываем следующий код и сохраняем проект (рис. 2.6):

```
#include "iostm8s003k3.h"
int main( void )
{
}
```

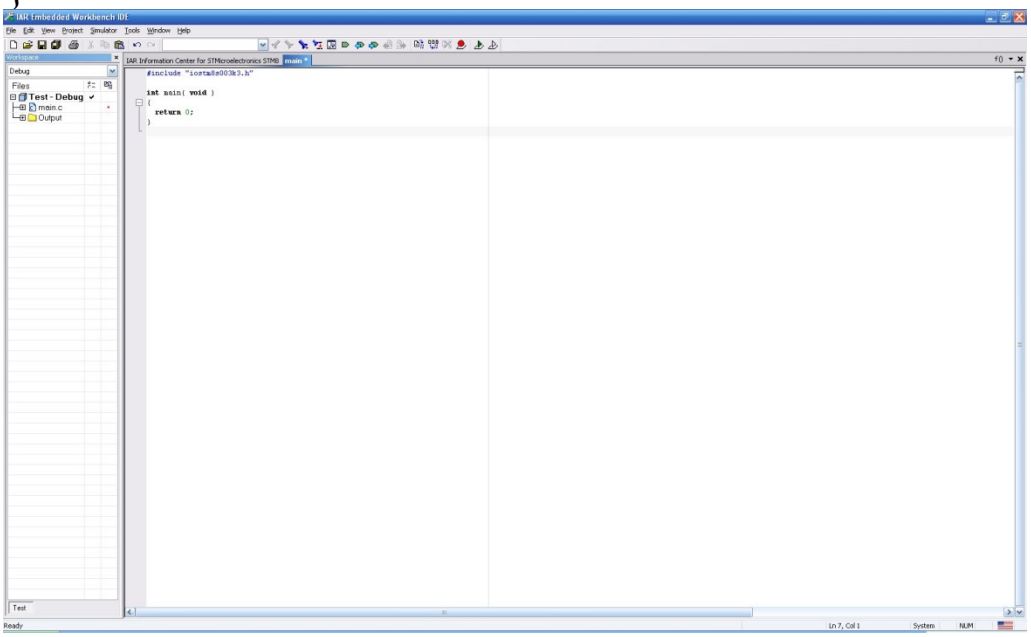

Рис. 2.6. Рабочее окно проекта

7. Далее необходимо настроить проект. Для этого в окне *Workspace* выбираем пункт контекстного меню *Options* (рис. 2.7).

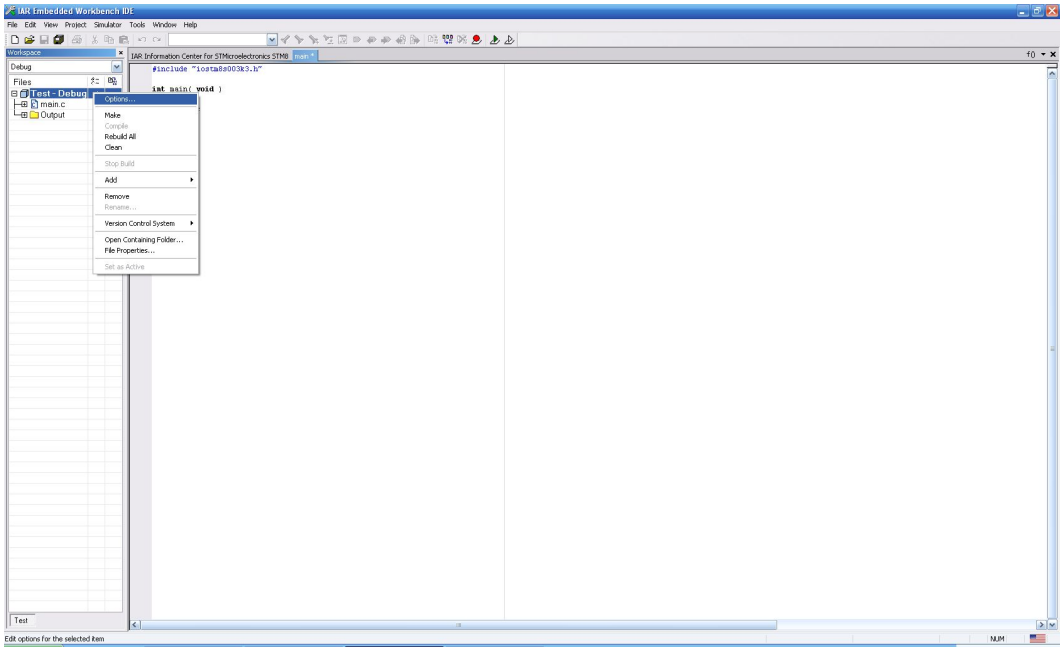

Рис. 2.7. Окно настройки проекта

8. На странице *General*, вкладке *Target*, выбираем модель контроллера: *STM8S→STM8S003K3* (рис. 2.8).

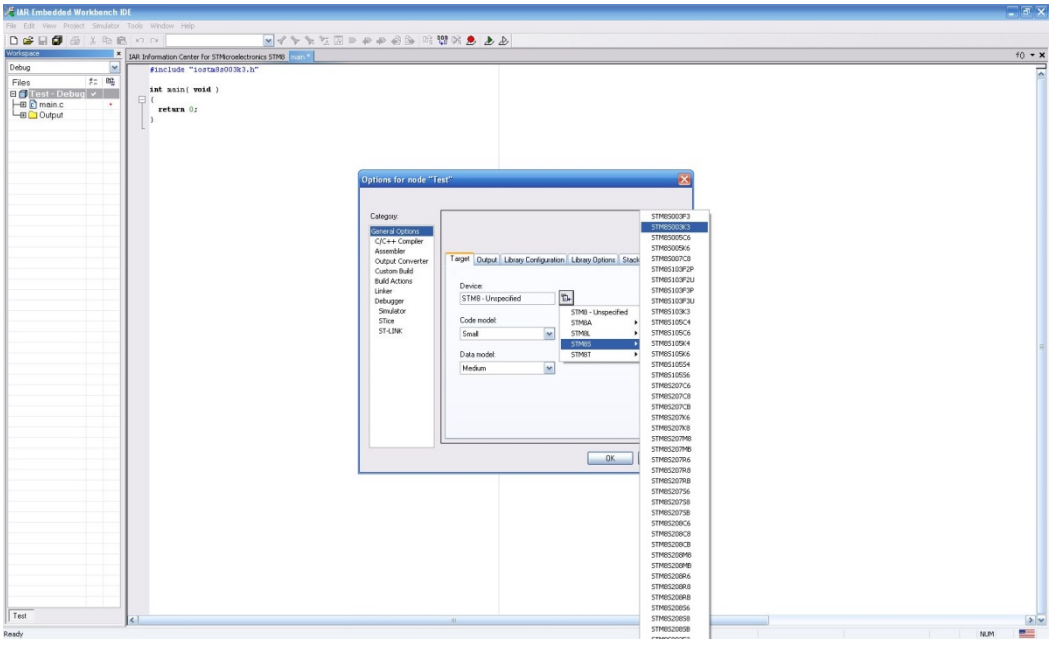

Рис. 2.8. Окно выбора микроконтроллера

9. На странице *Debugger*, вкладке *Setup*, выбираем отладчик *ST-Link* (рис. 2.9).

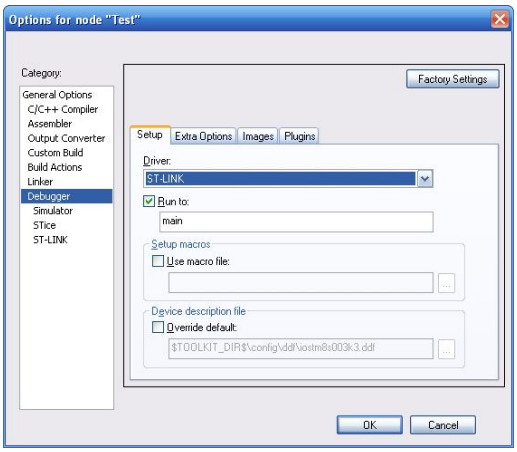

Рис. 2.9. Окно выбора отладчика

10. Загрузка программы в микроконтроллер осуществляется в три этапа (рис. 2.10): компиляция (*Compile*), создание (*Make*), загрузка и отладка (*Download and Debug*).

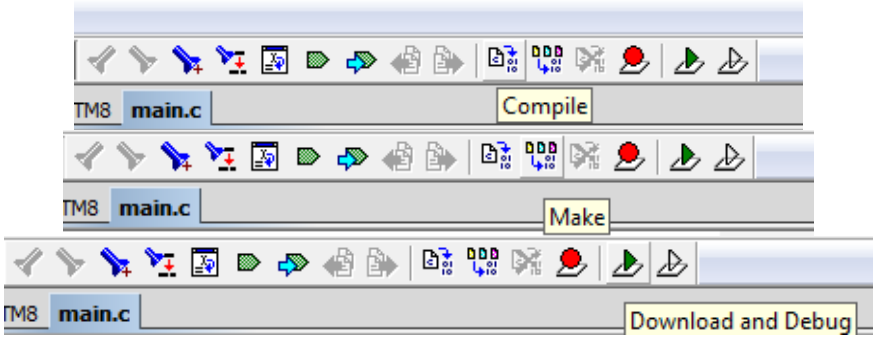

Рис. 2.10. Панели компиляции и загрузки программы

В результате выполненных действий появится окно, показанное на рис. 2.11.

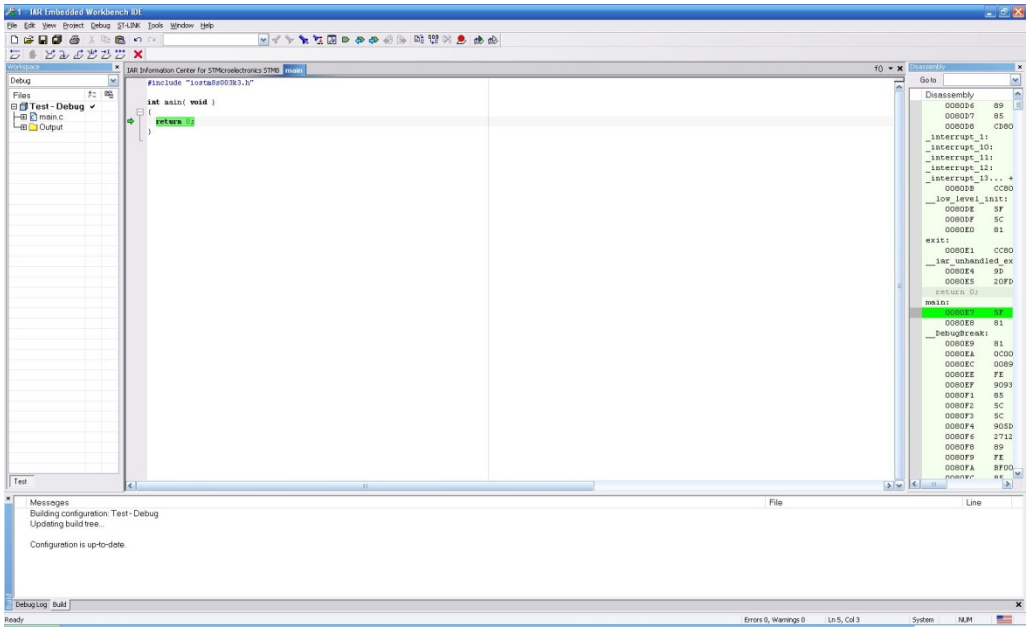

Рис. 2.11. Окно отладчика программы

11. Далее можно запускать программу, как показано на рис. 2.12.

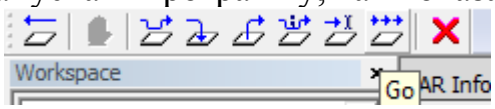

Рис. 2.12. Запуск программы

2.2. Примеры программ для микроконтроллера STM8S

2.2.1. Программа настройки портов ввода-вывода

Программа настройки выводов порта *D* на различные режимы работы включает в себя: *PD*0 – выход типа *push-pull* со скоростью переключения до 10 МГц, *PD*2 – выход с открытым стоком со скоростью переключения до 10 МГц, *PD*3 – дифференциальный вход, не являющийся источником внешних прерываний, *PD*4 – вход с подтягивающим резистором, не являющийся источником внешних прерываний [4].

```
#include <iostm8s003k3.h>      //подключение заголовочного файла с объ-
                               явлениями регистров, масок и битов
int main( void )
{
//Настройка нулевого бита порта D
PD_DDR_bit.DDR0 = 1;           //выход
PD_CR1_bit.C10 = 1;            //выход типа push-pull
PD_CR2_bit.C20 = 0;            //скорость переключения до 10 МГц
PD_ODR_bit.ODR0 = 1;           //вывод на порт логической "1"
//Настройка второго бита порта D
PD_DDR_bit.DDR2 = 1;           //выход
PD_CR1_bit.C12 = 0;            //выход с открытым стоком
PD_CR2_bit.C22 = 0;            //скорость переключения до 10 МГц
PD_ODR_bit.ODR2 = 0;           //вывод на порт логического "0"
//Настройка третьего бита порта D
PD_DDR_bit.DDR3 = 0;           //вход
PD_CR1_bit.C13 = 0;            //дифференциальный вход
PD_CR2_bit.C23 = 0;            //запретить внешние прерывания
//Настройка четвертого бита порта D
PD_DDR_bit.DDR4 = 0;           //вход
PD_CR1_bit.C14 = 1;            //с подтягивающим резистором
PD_CR2_bit.C24 = 0;            //запретить внешние прерывания
  while (1);                   //бесконечный цикл
}
```

2.2.2. Программа, реализующая эффект маятника

Программа реализует эффект маятника на восьми светодиодах, подключенных к порту *PD* (рис. 2.13).

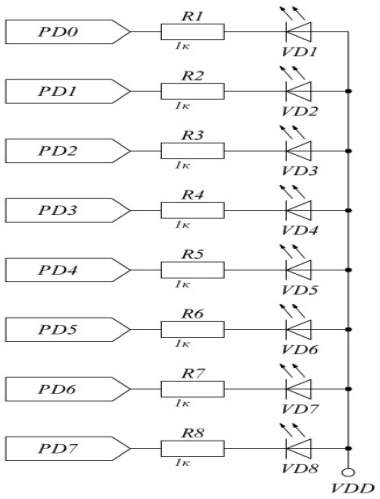

Рис. 2.13. Подключение восьми светодиодов к порту PD

```
#include <iostm8s003k3.h>          //подключение заголовочного файла с
                                   //объявлениями регистров, масок и битов
int i,j,k;                         //объявление переменных

int main( void )
{
//Настройка порта D
PD_DDR = 0xFF;                     //выход
PD_CR1 = 0xFF;                     //выход типа push-pull
PD_CR2 = 0xFF;                     //скорость переключения до 10 МГц
k=0xFE;
PD_ODR = k;
  while (1)                        //бесконечный цикл
  {
    for (j=0;j<=6;j++)
    {
    k<<=1;
    PD_ODR=k;                      //вывод на порт логической "1"
    for (i=0;i<20000;i++);
    }
      for (j=0;j<=6;j++)
      {
      k>>=1;
      PD_ODR = k;                  //вывод на порт логической "1"
      for (i=0;i<20000;i++);
      }
    }
  }
```

2.2.3. Программа, реализующая отслеживание состояния кнопки

Программа отслеживает состояние кнопки на выводе *PB*7 порта *B*. При нажатии на кнопку зажигается светодиод; при отпускании кнопки светодиод гаснет. Светодиод подключен к нулевому выводу порта *D* (рис. 2.14). Внешние прерывания запрещены, отслеживание состояния кнопки производится с помощью считывания входного регистра порта *B*.

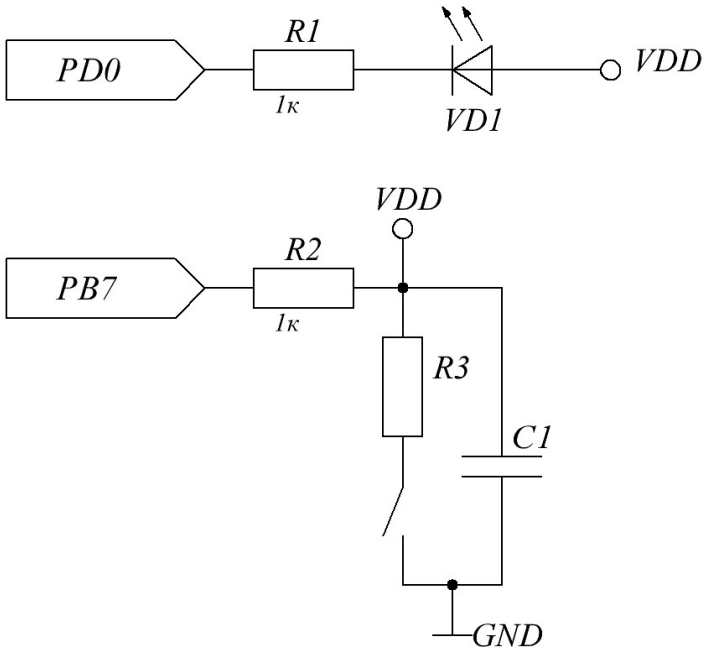

Рис. 2.14. Подключение светодиода и кнопки к микроконтроллеру STM8S

```
#include <iostm8s003k3.h>      //подключение заголовочного файла с
                               //объявлениями регистров, масок и битов
int main( void )
{
//настройка нулевого бита порта D
PD_DDR_bit.DDR0 = 1;           //выход
PD_CR1_bit.C10 = 1;            //выход типа push-pull
PD_CR2_bit.C20 = 0;            //скорость переключения до 10 МГЦ
PD_ODR_bit.ODR0 = 1;           //вывод на порт логической "1"

//настройка третьего бита порта D
PB_DDR_bit.DDR7 = 0;           //вход
PB_CR1_bit.C17 = 0;            //дифференциальный вход
PB_CR2_bit.C27 = 0;            //запретить внешние прерывания

  while (1)                    //бесконечный цикл
```

```
{
 if (PB_IDR_bit.IDR7==0)
 {
 PD_ODR_bit.ODR0 =0x00;   //сбрасываем нулевой бит порта D
 }
PD_ODR_bit.ODR0 =0x01;    //устанавливаем нулевой бит порта D
}
```

2.2.4. Программа, реализующая инверсию состояния светодиода по внешнему прерыванию

Программа реализует инверсию состояния светодиода по нажатию на одну из кнопок (по внешнему прерыванию). Светодиод подключен к нулевому выводу порта *D*, две кнопки подключены к выводам *PB*7 и *PC*7 портов *B* и *C* соответственно (рис. 2.15).

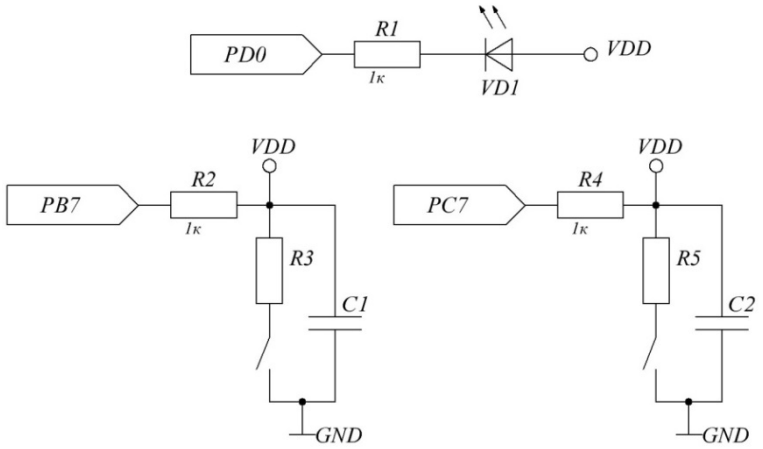

Рис. 2.15. Подключение светодиода и кнопок к микроконтроллеру STM8S

```
#include "iostm8.h"                 //подключение заголовочного файла с
                                    //объявлениями регистров, масок и битов
int i,j;                            //объявление переменных

void interrupt_init(void);          //объявление подпрограммы настройки
                                    //прерываний


#pragma vector=0x06
__interrupt void EXTI_PB7(void);    //имя вектора внешнего прерывания
                                    //порта B
#pragma vector=0x07
__interrupt void EXTI_PC7(void);    //имя вектора внешнего прерывания
                                    //порта C
int main( void )                    //основная программа
{
```

```
PB_DDR_bit.DDR7=0;          //выход
PB_CR1_bit.C17 = 0;         //дифференциальный вход
PB_CR2_bit.C27 = 1;         //прерывания разрешены

PC_DDR_bit.DDR7=0;          //вход
PC_CR1_bit.C17 = 0;         //дифференциальный вход
PC_CR2_bit.C27 = 1;         //прерывания разрешены

PD_DDR_bit.DDR0=1;          //выход
PD_CR1_bit.C10 = 1;         //выход типа push-pull
PD_CR2_bit.C20 = 0;         //скорость переключения до 2 МГц
PD_ODR_bit.ODR0= 0;         //зажигание светодиода

interrupt_init();           //вызов подпрограммы настройки
                            //прерываний

  for (;;)                  //бесконечный цикл
  {
  }

}

//подпрограмма настройки прерываний
void interrupt_init(void)
{
EXTI_CR1|=0x08;             //настройка прерываний: на порту B по
                            //спадающему фронту; на порту C по
                            //низкому уровню и спадающему фронту
asm("rim");                 //глобальное разрешение прерываний
}

//обработчик прерывания
__interrupt void EXTI_PB7(void)
{
PD_ODR_bit.ODR0 =! PD_ODR_bit.ODR0;     //инвертирование
                                        //нулевого бита порта D
  for (i=0;i<=30000;i++);   //задержка
  {
    for (j=0;j<=30000;j++); //задержка
    {
    }
  }
```

```
}

//обработчик прерывания
__interrupt void EXTI_PC7(void)
{
PD_ODR_bit.ODR0 =! PD_ODR_bit.ODR0;      //инвертирование
                                          //нулевого бита порта D
  for (i=0;i<=30000;i++);            //задержка
  {
    for (j=0;j<=30000;j++);          //задержка
    {
    }
  }
}
```

2.2.5. Программа, реализующая эффект бегущей единицы с переменным направлением

Программа реализует эффект бегущей единицы на восьми светодиодах, подключенных к порту *D*. Направление задается с помощью двух кнопок по прерыванию. Кнопки подключены к выводам *PB*7 и *PC*7 портов *B* и *C* соответственно (рис. 2.16).

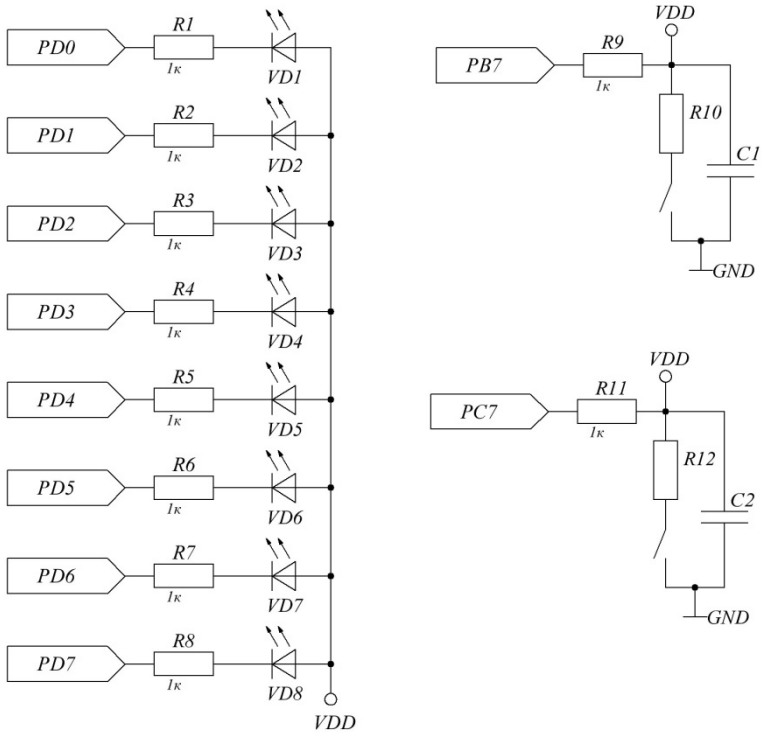

Рис. 2.16. Подключение светодиодов и кнопок к микроконтроллеру STM8S

```
#include <iostm8s003k3.h>      //подключение заголовочного файла с
                               //объявлениями регистров, масок и битов

void interrupt_init(void);     //объявление подпрограммы настройки
                               //прерываний
#pragma vector=0x06
__interrupt void EXTI_PB7(void);      //имя вектора внешнего прерывания

#pragma vector=0x07
__interrupt void EXTI_PC7(void);      //имя вектора внешнего прерывания

int i,j,k,napr;                //объявление переменных

int main( void )
{
PB_DDR_bit.DDR7=0;             //вход
PB_CR1_bit.C17 = 0;            //дифференциальный вход
PB_CR2_bit.C27 = 1;            //0 – прерывания запрещены; 1 – прерывания
                               //разрешены
PC_DDR_bit.DDR7=0;             //вход
PC_CR1_bit.C17 = 0;            //дифференциальный вход
PC_CR2_bit.C27 = 1;            //прерывания разрешены
//Инициализация порта D
PD_DDR = 0xFF;                 //выход
PD_CR1 = 0xFF;                 //выход типа push-pull
PD_CR2 = 0xFF;                 //скорость переключения до 10 МГц
k = 0x01;                      //переменная для бегущей "1"
PD_ODR = k;                    //установка нулевого бита порта D
napr = 0;                      //переменная, задающая направление

interrupt_init();              //вызов подпрограммы настройки
                               //прерываний
  while (1)                    //бесконечный цикл
  {
   if (napr==1)                //проверка направления
   {
   k=0x80;                     //обновление переменной
   PD_ODR = k;                 //вывод в порт значения переменной
     for (j=0;j<=7;j++)
     {
     PD_ODR = k;               //вывод в порт значения переменной
     k>>=1;                    //сдвиг единицы вправо
```

```
      for (i=0;i<20000;i++);        //временная задержка
     }
    }

    if (napr==0)                    //проверка направления
    {
    k = 0x01;                       //переменная для бегущей "1"
    PD_ODR = k;                     //вывод в порт значения переменной
     for (j=0;j<=7;j++)
     {
     PD_ODR = k;                    //вывод в порт значения переменной
     k <<=1;                        //сдвиг единицы влево
      for (i=0;i<20000;i++);        //временная задержка
     }
    }
  }
}

//подпрограмма настройки прерываний
void interrupt_init(void)
{
EXTI_CR1|=0x08;                 // настройка прерываний: на порту B по
                                //спадающему фронту; на порту C по
                                //низкому уровню и спадающему фронту
asm("rim");                     //глобальное разрешение прерываний
}

//обработчик прерывания
__interrupt void EXTI_PB7(void)
{
  // for (i=0;i<=30000;i++);      //временная задержка
napr=1;                         //задание направления вправо
}

//обработчик прерывания
__interrupt void EXTI_PC7(void)
{
  // for (i=0;i<=30000;i++);      //временная задержка
napr=0;                         //задание направления влево
}
```

2.2.6. Программа, реализующая инверсию состояния светодиода по прерыванию таймера 1

Программа реализует инверсию состояния светодиода с использованием прерываний таймера 1 по переполнению. Светодиод соединен с нулевым выводом порта *D* (рис. 2.14).

```
#include "iostm8s003k3.h"                  //подключение заголовочного файла с
                                           //объявлениями регистров, масок и битов
void portD_init(void);
void timer_init(void);
void interrupt_init(void);
#pragma vector=0x0D                        //в таблице прерываний у таймера 1 номер
                                           //прерывания равен 11, при сложении с
                                           //числом 2 получим 13, в hex-формате:
                                           0x0D
__interrupt void TIM1_OVR_UIF(void);       //объявление вектора //прерывания
                                           таймера
int main( void )                           //основная программа
{
portD_init();                              //вызов подпрограммы настройки порта
timer_init();                              //вызов подпрограммы настройки таймера
interrupt_init();                          //вызов подпрограммы настройки
                                           //прерываний
  for (;;)                                 //бесконечный цикл
  {
  }
}
void portD_init(void)
{
//Настройка нулевого бита порта D
PD_DDR_bit.DDR0=1;                         //выход
PD_CR1_bit.C10 = 1;                        //выход типа push-pull
PD_CR2_bit.C20 = 0;                        //скорость переключения до 2 МГц
PD_ODR_bit.ODR0= 0;                        //включаем светодиод
}
void timer_init(void) //подпрограмма настройки таймера
{
CLK_PCKENR2=0xff;                          //тактирование таймера
TIM1_CR1_bit.URS=1;                        //прерывание таймера только при
                                           //переполнении
TIM1_CR1_bit.DIR=0;                        //увеличение значений переменной
                                           //таймера
TIM1_CR1_bit.CMS=0;                        //выбор режима счета
TIM1_IER_bit.UIE=1;                        //разрешение прерывания по
                                           //переполнению по установке бита
TIM1_PSCRH=0x00;                           //старший регистр предварительного
                                           //делителя
```

```
TIM1_PSCRL=0x10;                //младший регистр предварительного
                                //делителя
TIM1_CR1_bit.CEN=1;             //запуск таймера
}

//Настройка прерываний
void interrupt_init(void)
{
asm("rim");                     //глобальное разрешение прерываний
}

//вектор прерывания таймера TIM1
__interrupt void TIM1_OVR_UIF(void)
{
TIM1_SR1_UIF = 0;               //очистка флага переполнения
PD_ODR_bit.ODR0=! PD_ODR_bit.ODR0; //инвертирование нулевого //бита
                                порта D
TIM1_CR1_bit.CEN=1;             //запуск таймера
}
```

2.2.7. Программа, реализующая инверсию состояния светодиода по прерыванию таймера 2

Программа реализует инверсию состояния светодиода, с использованием прерываний таймера 2 по переполнению. Светодиод присоединен к нулевому выводу порта *D* (рис. 2.14).

```
#include "iostm8s003k3.h"       //подключение заголовочного файла с
                                //объявлениями регистров, масок и битов
void portD_init(void);
void timer_init(void);
void interrupt_init(void);
#pragma vector=0x0F             //в таблице прерываний у таймера 2 номер
                                //прерывания равен 13, при сложении с
                                //числом 2 получим 15, в hex-формате: 0x0F
__interrupt void TIM2_OVR_UIF(void); //объявление вектора
                                //прерывания таймера
int main( void )                //основная программа
{
portD_init();                   //вызов подпрограммы настройки порта
timer_init();                   //вызов подпрограммы настройки таймера
interrupt_init();               //вызов подпрограммы настройки
                                //прерываний
  for (;;)                      //бесконечный цикл
  {
  }
}
```

```
//настройка нулевого бита порта D
void portD_init(void)
{
PD_DDR_bit.DDR0=1;          //выход
PD_CR1_bit.C10 = 1;         //выход типа push-pull
PD_CR2_bit.C20 = 0;         //скорость переключения до 2 МГц
PD_ODR_bit.ODR0= 0;         //зажигание светодиода
}

//подпрограмма настройки таймера
void timer_init(void)
{
CLK_PCKENR2=0xff;           //тактирование таймера
TIM2_CR1_bit.URS=1;         //прерывание таймера только при
                            //переполнении
TIM2_IER_bit.UIE=1;         // прерывание по переполнению разрешено
                            //по установке бита
TIM2_PSCR=0x13;             //младший регистр предварительного
                            //делителя
TIM2_CR1_bit.CEN=1;         //запуск таймера
}

//Настройка прерываний
void interrupt_init(void)
{
asm("rim");                 //глобальное разрешение прерываний
}

//вектор прерывания таймера TIM2
__interrupt void TIM2_OVR_UIF(void)
{
TIM2_SR1_UIF = 0;           //очистка флага переполнения
PD_ODR_bit.ODR0=! PD_ODR_bit.ODR0; //инвертирование нулевого
                            //бита порта D
TIM2_CR1_bit.CEN=1;         //запуск таймера
}
```

2.2.8. Программа, реализующая генерацию импульса по прерыванию двух таймеров

Программа реализует генерацию импульса, с использованием прерываний по переполнению двух таймеров. Таймер 2 отвечает за время

импульса, пауза формируется с помощью таймера 1. Импульс генерируется на нулевом выводе порта *D*.

```
#include "iostm8s003k3.h"          //подключение заголовочного файла с
                                   //объявлениями регистров, масок и битов
void portD_init(void);
void timer_init(void);
void interrupt_init(void);
#pragma vector=0x0D                //в таблице прерываний у таймера 1 номер
                                   //прерывания равен 11, при сложении с
                                   //числом 2 получим 13, в hex-формате:
                                   0x0D
__interrupt void TIM1_OVR_UIF(void); //объявление вектора
                                   //прерывания таймера
#pragma vector=0x0F                //в таблице прерываний у таймера 2 номер
                                   //прерывания равен 13, при сложении с
                                   //числом 2 получим 15, в hex-формате:
                                   0x0F
__interrupt void TIM2_OVR_UIF(void); //объявление вектора
                                   //прерывания таймера
int main( void )                   //основная программа
{
portD_init();                      //вызов подпрограммы настройки порта
timer_init();                      //вызов подпрограммы настройки таймера
interrupt_init();                  //вызов подпрограммы настройки преры-
                                   ваний
  for (;;)                         //бесконечный цикл
  {
  }
}

//настройка нулевого бита порта D
void portD_init(void)
{
PD_DDR_bit.DDR0=1;                 //выход
PD_CR1_bit.C10 = 1;                //выход типа push-pull
PD_CR2_bit.C20 = 0;                //скорость переключения до 2 МГц
PD_ODR_bit.ODR0= 0;                //включение светодиода
}

//подпрограмма настройки таймера
void timer_init(void)
{
CLK_PCKENR2=0xff;                  //тактирование таймера
```

```
TIM1_CR1_bit.URS=1;          //прерывание таймера только при
                             //переполнении
TIM1_CR1_bit.DIR=0;          //таймер считает вверх
TIM1_CR1_bit.CMS=0;          //выбор режима счета
TIM1_CR1_bit.OPM=1;          //режим одного импульса
TIM1_IER_bit.UIE=1;          //установка бита разрешает прерывание по
                             //переполнению
TIM1_PSCRH=0x00;             //старший регистр предделителя
TIM1_PSCRL=0x15;             //младший регистр предделителя

TIM2_CR1_bit.URS = 1;        //прерывание таймера только при
                             //переполнении
TIM2_IER_bit.UIE = 1;        //установка бита разрешает прерывание по
                             переполнению
TIM2_PSCR=0x15;              //младший регистр предделителя
TIM2_CR1_bit.OPM=1;          //режим одного импульса

TIM1_CR1_bit.CEN = 1;        //запуск таймера
TIM2_CR1_bit.CEN = 1;        //запуск таймера
}

//Настройка прерываний
void interrupt_init(void)
{
asm("rim");                  //глобальное разрешение прерываний
}

//вектор прерывания таймера TIM1
__interrupt void TIM1_OVR_UIF(void)
{
TIM1_SR1_UIF = 0;            //очистка флага переполнения
PD_ODR_bit.ODR0 =1;          //инвертирование нулевого бита порта D
TIM2_CR1_bit.CEN = 1;        //запуск таймера
TIM1_CR1_bit.CEN = 1;        //запуск таймера
}
//вектор прерывания таймера TIM2
__interrupt void TIM2_OVR_UIF(void)
{
TIM2_SR1_UIF = 0;            //очистка флага переполнения
PD_ODR_bit.ODR0 =0;          //инвертирование нулевого бита порта D
}
```

2.2.9. Программа, реализующая ШИМ таймера 1

Программа реализует ШИМ на втором канале таймера 1. Второй канал таймера 1 соединен со вторым выводом порта *C* [Pdf]. Изменение времени импульса производится с помощью двух кнопок, которые соединены с выводами 7 портов *B* и *C* (рис. 2.15).

```
#include "iostm8.h"          //подключение заголовочного файла с
                             //объявлениями регистров, масок и битов
int i,j;                     //объявление переменных

void interrupt_init(void);   //объявление подпрограммы настройки
                             //прерываний
void PWM_TIM1_CH2 (void);

#pragma vector=0x06
__interrupt void EXTI_PB7(void);   //имя вектора внешнего прерывания

#pragma vector=0x07
__interrupt void EXTI_PC7(void);   //имя вектора внешнего прерывания

int main( void )             //основная программа
{
PB_DDR_bit.DDR7=0;           //вход
PB_CR1_bit.C17 = 0;          //дифференциальный вход
PB_CR2_bit.C27 = 1;          //прерывания разрешены

PC_DDR_bit.DDR7=0;           //вход
PC_CR1_bit.C17 = 0;          //дифференциальный вход
PC_CR2_bit.C27 = 1;          //прерывания разрешены

interrupt_init();            //вызов подпрограммы настройки
                             //прерываний
PWM_TIM1_CH2();              //вызов подпрограммы настройки ШИМ
                             таймера 1 канал 2 – PC2
  for (;;)
  {
  }
}

//подпрограмма настройки прерываний
void interrupt_init(void)
{
```

```
EXTI_CR1|=0x28;                 //прерывания на портах B и C по
                                //спадающему фронту
asm("rim");                     //глобальное разрешение прерываний
}

//обработчик прерывания
__interrupt void EXTI_PB7(void)
{
 if (TIM1_CCR2L<0xF5)
  {
 TIM1_CCR2L=TIM1_CCR2L+10;    //увеличение времени импульса
  }
   for (i=0;i<300;i++);
}

//обработчик прерывания
__interrupt void EXTI_PC7(void)
{
 if (TIM1_CCR2L>0x10)
  {
 TIM1_CCR2L=TIM1_CCR2L-10;     //уменьшение времени импульса
  }
   for (i=0;i<300;i++);
}

//настройка таймера
void PWM_TIM1_CH2 (void)
{
CLK_PCKENR2=0xff;               //тактирование таймера 1
TIM1_PSCRL = 0x00;              //предварительный делитель
TIM1_PSCRH = 0x0F;              //предварительный делитель
TIM1_BKR = 0x80;                //разрешение каналов таймера
TIM1_CCMR2 = 0x68;              //режим ШИМ1 и разрешение
                                //предварительной загрузки
TIM1_CCER1_bit.CC2P = 0;        //активный уровень высокий
TIM1_CCER1_bit.CC2E = 1;        //включение канала 2
TIM1_ARRH = 0x00;               // старший байт периода ШИМ
TIM1_ARRL = 0xFF;               // младший байт периода ШИМ т
TIM1_CCR2H = 0x00;              // старший байт времени импульса ШИМ
TIM1_CCR2L = 0x20;              // младший байт времени импульса ШИМ
TIM1_CR1_bit.CEN = 1;           //запуск таймера
}
```

2.2.10. Программа, реализующая эффект маятника по прерыванию таймера

Программа реализует эффект маятника на восьми светодиодах, подключенных к порту *D* (рис. 2.13). Все задержки организованы по прерыванию таймера 4.

```
#include "iostm8s003k3.h"        //подключение заголовочного файла с
                                 //объявлениями регистров, масок и битов
void portD_init(void);
void timer_init(void);
void interrupt_init(void);
int i=0, k,j;
#pragma vector=0x19              //в таблице прерываний у таймера 4 номер
                                 //прерывания равен 23, при сложении с
                                 //числом 2 получим 25, в hex-формате:
                                 //0x19
__interrupt void TIM4_OVR_UIF(void);  //объявление вектора
                                 //прерывания таймера
int main( void )                 //основная программа
{
portD_init();                    //вызов подпрограммы настройки порта
timer_init();                    //вызов подпрограммы настройки таймера
interrupt_init();                //вызов подпрограммы настройки
                                 //прерываний
k=0xFE;
PD_ODR = k;
  while (1)                      //бесконечный цикл
  {
    for (j=0;j<=6;j++)
    {
    k <<=1;                      //вывод на порт логической "1"
    PD_ODR = k;
    TIM4_CR1_bit.CEN=1;          //запуск таймера
      while (i<5)
      {
      }
    i=0;
    }
    for (j=0;j<=6;j++)
    {
    k>>=1;                       //вывод на порт логической "1"
    PD_ODR = k;
    TIM4_CR1_bit.CEN=1;          //запуск таймера
```

```
      while (i<5)
      {
      }
    i=0;
    }
  }
}

//подпрограмма настройки порта D
void portD_init(void)

{
PD_DDR=0xFF;                //выход
PD_CR1 = 0xFF;              //выход типа push-pull
PD_CR2 = 0x00;              //скорость переключения до 2 МГц
}

//подпрограмма настройки таймера
void timer_init(void)
{
CLK_PCKENR2=0xff;           //тактирование таймера
TIM4_CR1_bit.URS=1;         //прерывание таймера только при
                            //переполнении
TIM4_IER_bit.UIE=1;         //установка бита разрешает прерывание по
                            //переполнению
TIM4_PSCR=0x1F;             //регистр предделителя
//TIM4_CR1_bit.CEN=1;       //запуск таймера
}

//Настройка прерываний
void interrupt_init(void)
{
asm("rim");                 //глобальное разрешение прерываний
}

//вектор прерывания таймера TIM4
__interrupt void TIM4_OVR_UIF(void)
{
TIM4_SR_UIF = 0;            //очистка флага переполнения
i++;
TIM4_CR1_bit.CEN=1;         //запуск таймера
}
```

2.2.11. Программа, реализующая работу модуля АЦП

Программа реализует вывод результата АЦП на восемь светодиодов, соединенных с портом *D* (рис. 2.13). АЦП совершает преобразование данных на канале *AIN*0, который соединен с нулевым выводом порта *B* [Pdf]. На вход АЦП подается напряжение с выхода операционного усилителя (рис. 2.17).

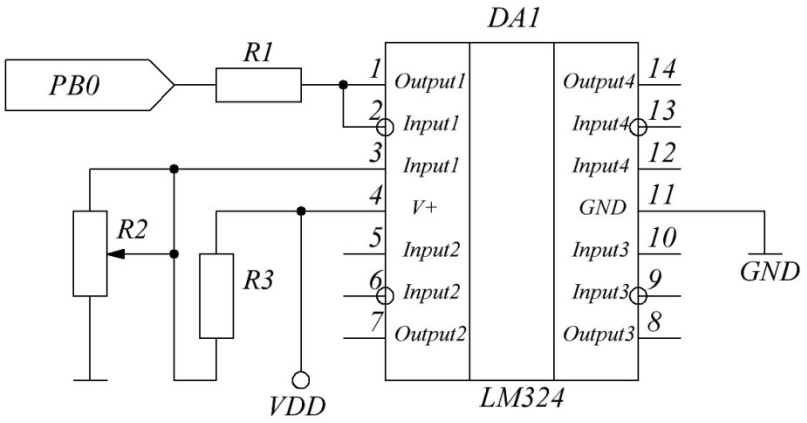

Рис. 2.17. Схема формирования аналогового сигнала на входе АЦП микроконтроллера STM8S

```
#include "iostm8s003k3.h"        //подключение заголовочного файла с
                                 //объявлениями регистров, масок и битов
void interrupt_init(void);
void adc_init(void);
int i,res;
#pragma vector=0x18
__interrupt void ADC_end(void); //объявление вектора прерывания АЦП
int main( void )                 //основная программа
{
//Настройка порта D
PD_DDR=0xFF;                     //выход
PD_CR1 = 0xFF;                   //выход типа push-pull
PD_CR2 = 0x00;                   //скорость переключения до 2 МГц

adc_init();                      //вызов подпрограммы настройки АЦП
interrupt_init();                //вызов подпрограммы настройки
                                 //прерываний
  for (;;)                       //бесконечный цикл
  {
  }
}
```

```
//подпрограмма настройки АЦП
void adc_init(void)
{
CLK_PCKENR2=0xff;          //тактирование АЦП
ADC_CSR_bit.AWD=0;         //запрет ожидания сигнала от аналогового
                           //сторожевого таймера
ADC_CSR_bit.EOCIE=1;       //прерывание по окончанию
                           //преобразования разрешено
ADC_CSR_bit.AWDIE=0;       //прерывание от сторожевого таймера
                           //запрещено
ADC_CSR_bit.CH=0x0;        //канал AIN0
ADC_CR1_bit.SPSEL=0x0;     //выбор частоты работы АЦП
ADC_CR1_bit.CONT=0;        //одиночное преобразование
ADC_CR2_bit.EXTTRIG=0;     //преобразование по внешнему событию
                           //запрещено
ADC_CR3_bit.DBUF=0;        //результат преобразования в ADC_DRH и
                           //ADC_DRL
ADC_CR3_bit.OVR=0;         //очистка флага завершения
                           //преобразования
ADC_CR1_bit.ADON=1;        //подключение АЦП к источнику питания
i=0;                       //временная задержка
do {i++;}
while (i<1000);
ADC_CR1_bit.ADON=1;        //разрешение начала преобразования
}

//Настройка прерываний
void interrupt_init(void)
{
asm("rim");                //глобальное разрешение прерываний
}

//Вектор прерывания АЦП
__interrupt void ADC_end(void)
{
ADC_CSR_bit.EOC=0;         //очистка флага завершения
                           //преобразования
res = ADC_DRH;
//res =! res;
PD_ODR = res;
ADC_CR1_bit.ADON=1;        //разрешение начала преобразования
}
```

2.2.12. Программа, реализующая ШИМ таймера 1 с регулируемой длительностью сигнала посредством АЦП

Программа реализует генерацию ШИМ на втором канале таймера 1. Второй канал таймера 1 соединен со вторым выводом порта *C*. Изменение времени импульса производится с помощью АЦП.

```
#include "iostm8.h"          //подключение заголовочного файла с
                             //объявлениями регистров, масок и битов

int i,j,res;                 //объявление переменных

void interrupt_init(void);   //объявление подпрограммы настройки
                             прерываний
void PWM_TIM1_CH2 (void);
void adc_init(void);
#pragma vector=0x18
__interrupt void ADC_end(void); //объявление вектора прерывания АЦП
#pragma vector=0x06
__interrupt void EXTI_PB7(void);    //имя вектора внешнего прерывания
#pragma vector=0x07
__interrupt void EXTI_PC7(void);    //имя вектора внешнего прерывания

int main( void )             //основная программа
{
interrupt_init();            //вызов подпрограммы настройки
                             //прерываний
PWM_TIM1_CH2();              //вызов подпрограммы настройки ШИМ
                             //таймера 1 канал 2 – PC2
adc_init();                  //вызов подпрограммы настройки АЦП

  for (;;)
  {
  }
}

//подпрограмма настройки ШИМ таймера 1
void PWM_TIM1_CH2 (void)
{
CLK_PCKENR2=0xff;            //тактирование таймера 1
TIM1_PSCRL = 0x00;           //предварительный делитель
TIM1_PSCRH = 0x0F;           //предварительный делитель
TIM1_BKR = 0x80;             //подключение каналов таймера
```

```
TIM1_CCMR2 = 0x68;          //режим ШИМ1 и разрешение
                            //предварительной загрузки
TIM1_CCER1_bit.CC2P = 0;    //активный уровень высокий
TIM1_CCER1_bit.CC2E = 1;    //включение канала 2
TIM1_ARRH = 0x00;           // старший байт периода ШИМ
TIM1_ARRL = 0xFF;           // младший байт периода ШИМ
TIM1_CCR2H = 0x00;          // старший байт времени импульса ШИМ
TIM1_CCR2L = 0x20;          // младший байт времени импульса ШИМ
TIM1_CR1_bit.CEN = 1;       //запуск таймера
}

//настройка АЦП
void adc_init(void)
{
CLK_PCKENR2=0xff;           //тактирование АЦП
ADC_CSR_bit.AWD=0;          //запрет ожидания сигнала от аналогового
                            //сторожевого таймера
ADC_CSR_bit.EOCIE=1;        //прерывание по окончанию
                            //преобразования разрешено
ADC_CSR_bit.AWDIE=0;        //прерывание от аналогового сторожевого
                            //таймера запрещено
ADC_CSR_bit.CH=0x0;         //канал AIN0
ADC_CR1_bit.SPSEL=0x0;      //выбор частоты работы АЦП
ADC_CR1_bit.CONT=0;         //одиночное преобразование
ADC_CR2_bit.EXTTRIG=0;      //преобразование по внешнему событию
                            //запрещено
ADC_CR3_bit.DBUF=0;         //результат преобразования в ADC_DRH и
                            //ADC_DRL
ADC_CR3_bit.OVR=0;          //очистка флага завершения
                            //преобразования
ADC_CR1_bit.ADON=1;         //подключение АЦП к источнику
                            //напряжению питания
i=0;                        //временная задержка
  do {i++;}
  while (i<1000);
ADC_CR1_bit.ADON=1;         //разрешение начала преобразования
}

//подпрограмма настройки прерываний
void interrupt_init(void)
{
asm("rim");                 //глобальное разрешение прерываний
}
```

```
//вектор прерывания АЦП
__interrupt void ADC_end(void)
{
ADC_CSR_bit.EOC=0;          //очистка флага завершения
                            //преобразования
res = ADC_DRH;              //результат преобразования
TIM1_CCR2L = res;           //младший байт времени импульса ШИМ
ADC_CR1_bit.ADON=1;         //разрешение начала преобразования
}
```

2.2.13. Программа, реализующая настройку UART

Программа реализует настройку модуля *UART*. При получении по *UART ASCI* кода клавиши *s* микроконтроллер запускает преобразование, осуществляемое АЦП. При получении по *UART ASCI* кода клавиши *v* микроконтроллер отправляет результат преобразования по *UART*.

```
#include "iostm8.h"          //подключение заголовочного файла с
                             объявлениями регистров, масок и битов

void interrupt_init(void);
void adc_init(void);
void uart_init (void);
int i, res;
int j,p;
int res1;
int res2;
#pragma vector=0x18
__interrupt void ADC_end(void);  //объявление вектора прерывания АЦП
int main( void )                 //основная программа
{
//Настройка пятого бита порта D5 – передатчик UART1
PD_DDR_bit.DDR5=1;                          //0 – вход; 1 – выход
PD_CR1_bit.C15 = 0;      //0 – выход с открытым стоком; 1 – выход типа
                         push-pull
PD_CR2_bit.C25 = 0;      //скорость переключения: от 0 до 2 MHz; от 1 до
                         10 MHz

//настройка пятого бита порта D6 – приемник UART1
PD_DDR_bit.DDR6=0;                          //0 – вход; 1 – выход
PD_CR1_bit.C16 = 0;          //0 – floating input; 1 – input with pull-up
PD_CR2_bit.C26 = 0;      //скорость переключения:
                         // от 0 до 2 MHz; от 1 до 10 MHz
```

```
uart_init();                        //вызов подпрограммы настройки uart
adc_init();                         //вызов подпрограммы настройки АЦП
interrupt_init();       //вызов подпрограммы настройки прерываний
for (;;) //бесконечный цикл
{
if (((UART1_SR)&0x20)==0x20)                //ожидание нажатия кнопки
{
res=UART1_DR;
if ((res&0x73)==0x73)                       //если нажата кнопка 's'
{
ADC_CR1_bit.ADON=1;         //разрешение  начала преобразования
}
if ((res&0x76)==0x76)                       //если нажата 'v'
{
UART1_DR= res2;
i=0;                                        //задержка
do {i++;}
while (i<1000);

UART1_DR= res1;
i=0;                                        //задержка
do {i++;}
while (i<1000);
UART1_DR=0x00;
}
}
}
}

void uart_init (void)
{
  CLK_PCKENR1 |= 0xff;          //тактирование uart
 //все параметры (стоп биты, чётность, количество байт данных) уже
//настроены
//как надо (по умолчанию, во все эти биты записаны нули)
  UART1_CR1 = 0;
  UART1_CR3 = 0;
  UART1_CR4 = 0;
  UART1_CR5 = 0;
  UART1_CR2_bit.TEN = 1; //разрешение работы передатчика
  UART1_CR2_bit.REN = 1; //разрешение работы приемника
```

```
//скорость передачи 2400
UART1_BRR2 = 0x01;
UART1_BRR1 = 0x34;

void adc_init(void)          //настройка АЦП
{
CLK_PCKENR2=0xff; //тактирование АЦП
ADC_CSR_bit.AWD=0; //0 – Нет сигнала события от аналогового
                   //сторожевого таймера
ADC_CSR_bit.EOCIE=1; //прерывание по окончанию преобразования
                     //разрешено;
ADC_CSR_bit.AWDIE=0; //прерывание от сторожевого таймера
                       //запрещено;
ADC_CSR_bit.CH=0x0; //канал AIN0;
ADC_CR1_bit.SPSEL=0x0; //выбор частоты работы АЦП;
ADC_CR1_bit.CONT=0; //0 – одиночное преобразование; 1 – несколько
                     //преобразований
ADC_CR2_bit.EXTTRIG=0;  //преобразование по внешнему событию
                         //0 – выкл; 1 – вкл
ADC_CR3_bit.DBUF=0; //результат преобразования ADC_DRH и
                     //ADC_DRL
ADC_CR3_bit.OVR=0; //очистка флага завершения преобразования
ADC_CR1_bit.ADON=1; //разрешение начать преобразования
i=0; //Задержка
do {i++;}
while (i<1000);
ADC_CR1_bit.ADON=1; //разрешение начать преобразования
}

void interrupt_init(void)
//Настройка прерываний
{
asm("rim"); //глобальное разрешение прерываний
}

__interrupt void ADC_end(void)
{
res2=ADC_DRH; //результаты преобразования
res1=ADC_DRL; //результаты преобразования
ADC_CSR_bit.EOC=0; //очистка флага завершения преобразования
}
```

Глава 3. Микроконтроллер STM32F1x

3.1. Создание проекта в программе IAR Embedded Workbench

Создание проекта в среде *IAR Embedded Workbench* осуществляется по следующему алгоритму [5].

1. Запускаем среду программирования *IAR Embedded Workbench for STMicroelectronics ARM*. На рис. 3.1 представлен внешний вид стартового окна программы.

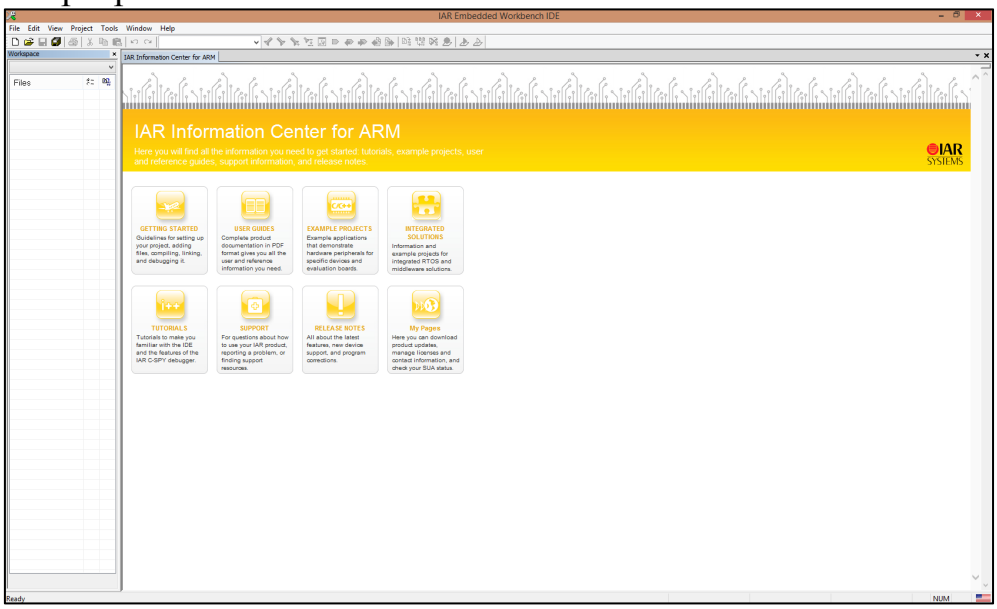

Рис. 3.1. Стартовое окно программы

2. Для создания нового проекта необходимо зайти в меню *Project* и выбрать пункт *Create new project*... (рис. 3.2).

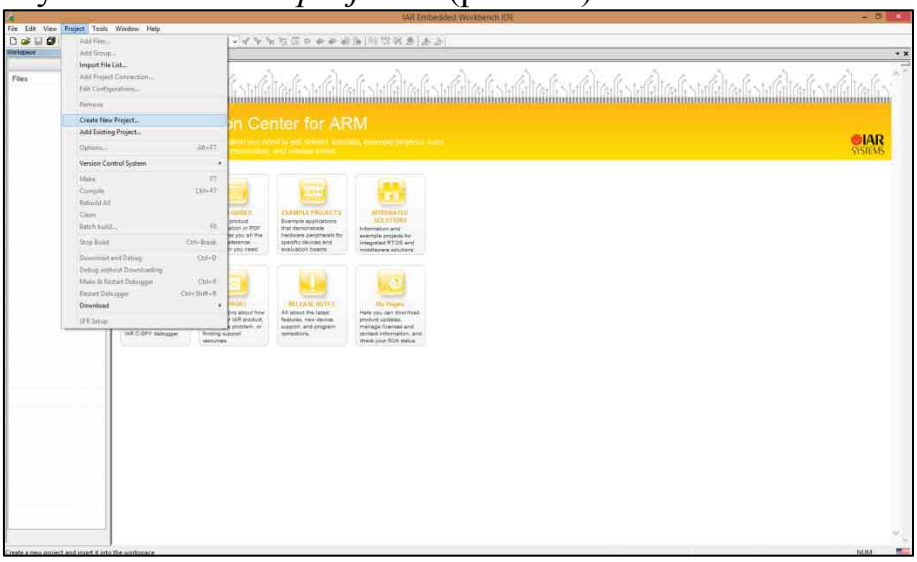

Рис. 3.2. Окно создания нового проекта

3. В появившемся окне (рис. 3.3) необходимо выбрать шаблон для языка C и тип микроконтроллера (*ARM*). Далее следует сохранить рабочую область *Workspace* (рис. 3.4).

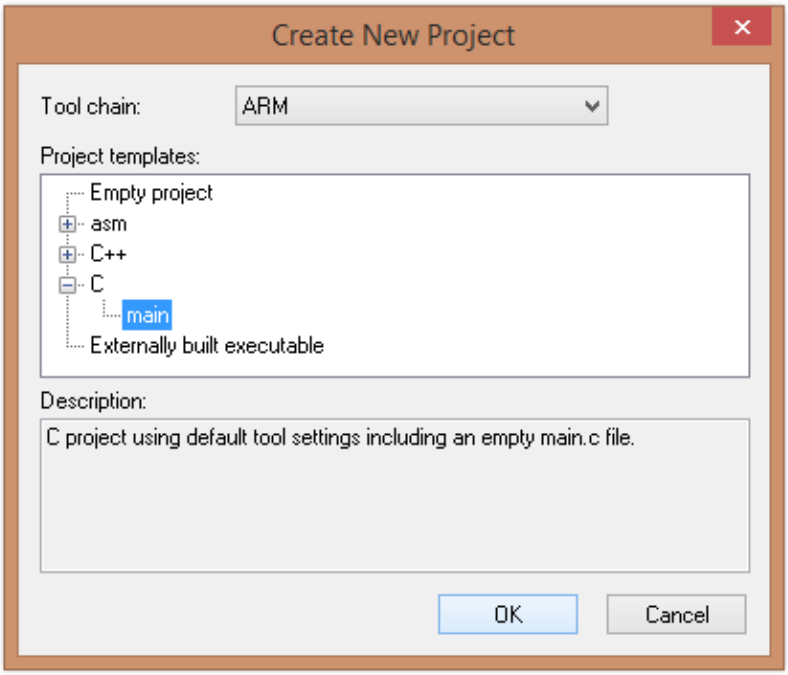

Рис. 3.3. Окно выбора языка программирования и микроконтроллера

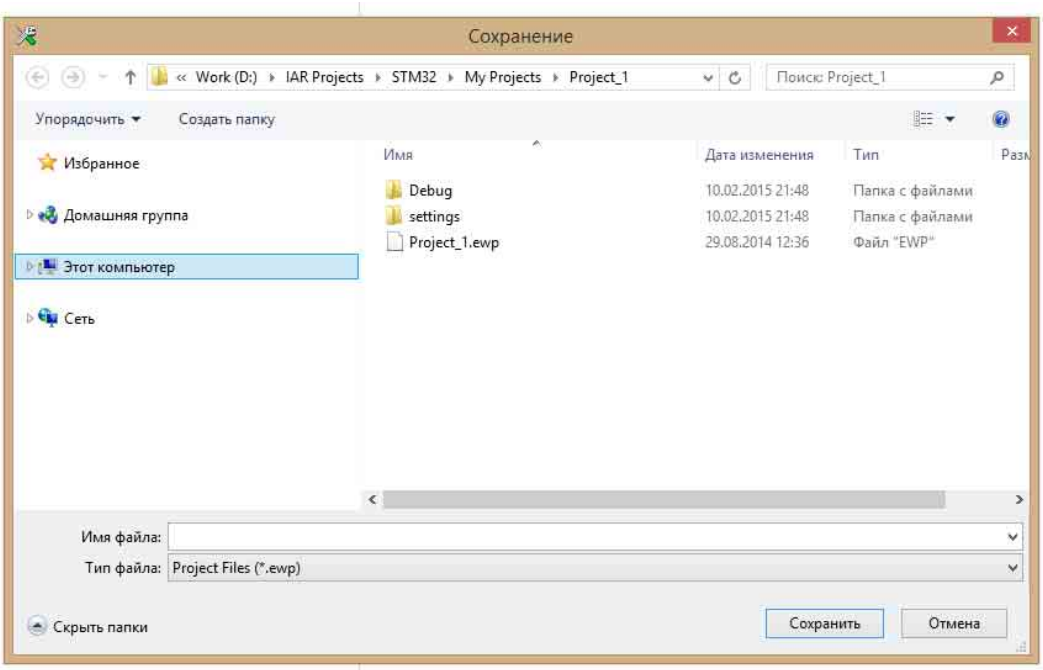

Рис. 3.4. Окно сохранения проекта

4. После сохранения проекта будет открыто его рабочее окно (рис. 3.5).

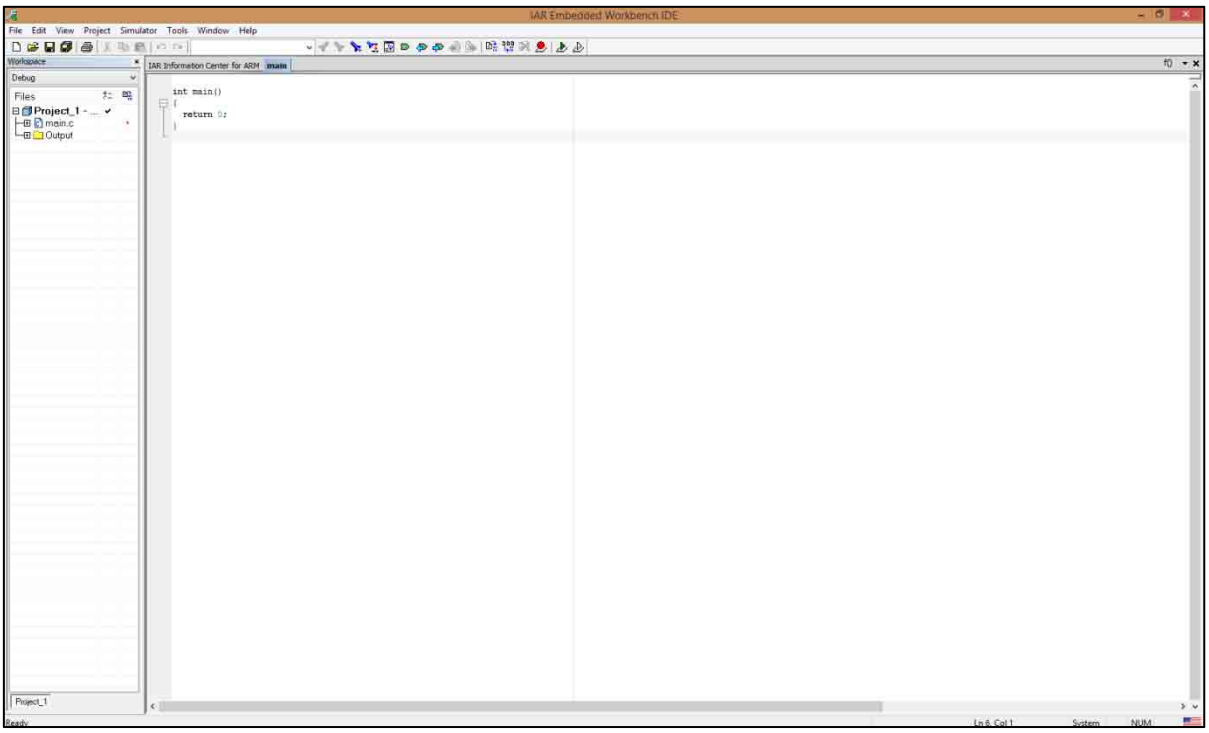

Рис. 3.5. Рабочее окно проекта

5. Далее необходимо настроить проект. Для этого в окне *Workspace* выбирается пункт контекстного меню *Options* (рис. 3.6).

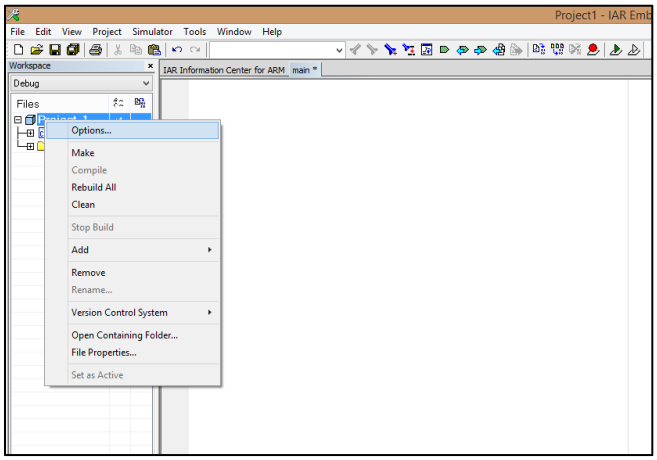

Рис. 3.6. Окно настройки проекта

6. На странице *General Options* во вкладке *Target* выбирается модель контроллера: *Device—>ST—>STM*32*F*100*—> STM*32*F*100*xB* (рис. 3.7).

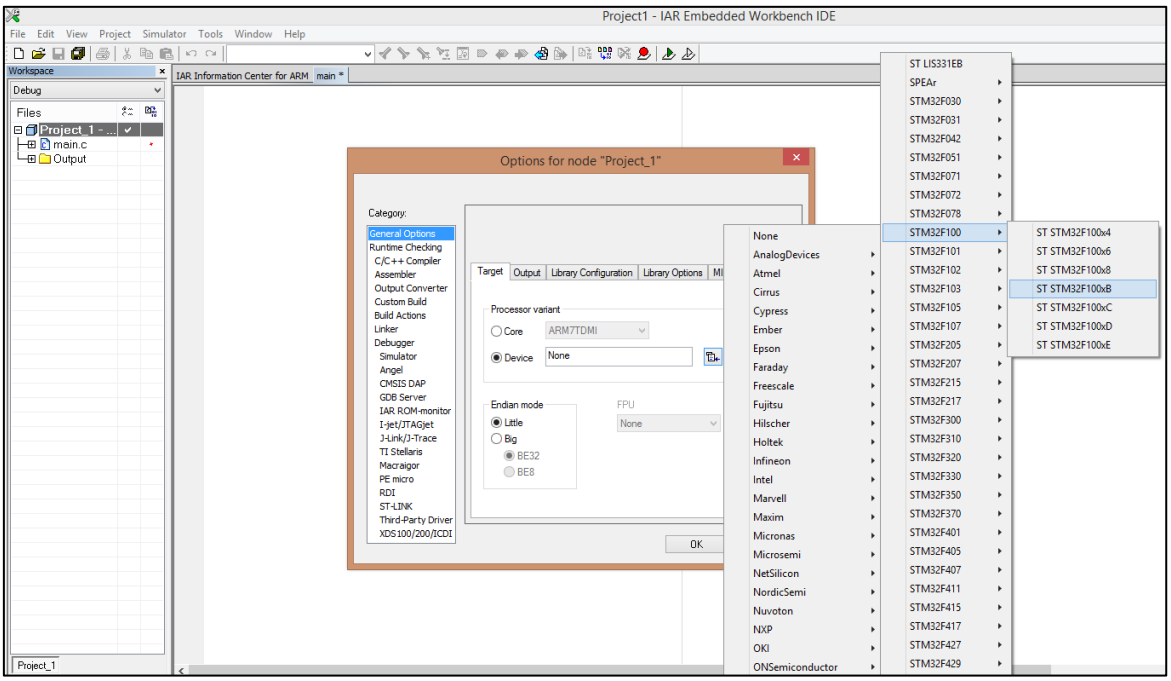

Рис. 3.7. Окно выбора микроконтроллера

7. Для дальнейшей работы с микроконтроллером необходимо подключить библиотеку, содержащую описание регистров, масок и битов. Существуют стандартные библиотеки ядра микроконтроллеров серии *Cortex – CMSIS*. Данная аббревиатура расшифровывается как *Cortex Microcontroller Software Interface*. Помимо этого существует еще одна библиотека для *STM32F10x* под названием *Standard Peripherals Library* (*SPL*). Библиотека *SPL* может использоваться в дополнение к *CMSIS*, обеспечивая более быстрый и удобный доступ к периферии. Библиотеку *SPL* часто называют набором драйверов к периферийным модулям.

Для загрузки библиотек нужно перейти на страницу *C/C++ Compiler*, выбрать вкладку *Preprocessor* и в соответствующем окне указать пути к следующим файлам библиотеки *CMSIS* (рис. 3.8):

- **core_cm3.c:** Libraries\CMSIS\CM3\CoreSupport\
- **core_cm3.h:** Libraries\CMSIS\CM3\CoreSupport\
- **stm32f10x.h:** Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x
- **system_stm32f10x.h:**Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x
- **system_stm32f10x.c:** Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x
- **startup_stm32f10x_md_vl.s:** Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x\startup\iar

Также в дополнение к *CMSIS* следует указать пути к ряду файлов библиотеки *SPL*:

- **Все содержимое папки *inc*:** Libraries\STM32F10x_StdPeriph_Driver\inc
- **Все содержимое папки *scr*:** Libraries\STM32F10x_StdPeriph_Driver\src

Наконец, необходимо добавить файл ***stm32f10x_conf***, расположенный в папке ***inc***, которая находится в корне библиотеки.

В результате содержимое окна *Additional includes direction* должно соответствовать перечню файлов, приведенному на рис. 3.8.

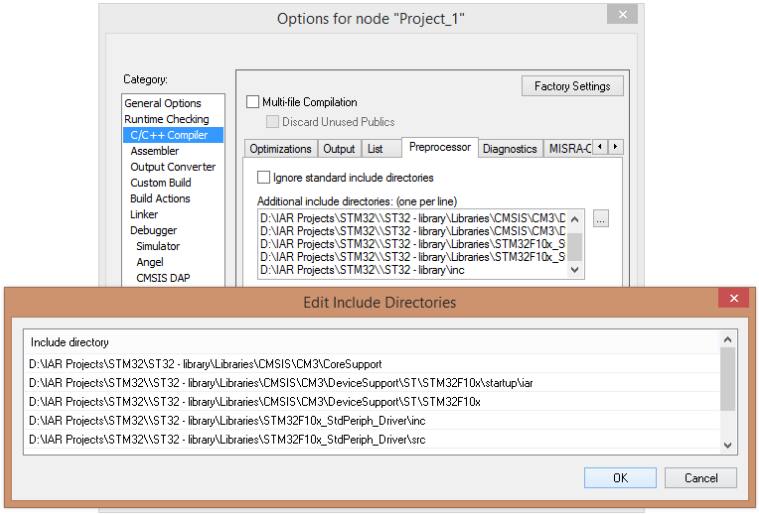

Рис. 3.8. Содержимое окна Additional includes direction

8. Далее необходимо перейти на страницу *Debbuger*. Во вкладке *Setup* расположено поле *Driver*, в котором следует выбрать *ST-LINK* (рис. 3.9).

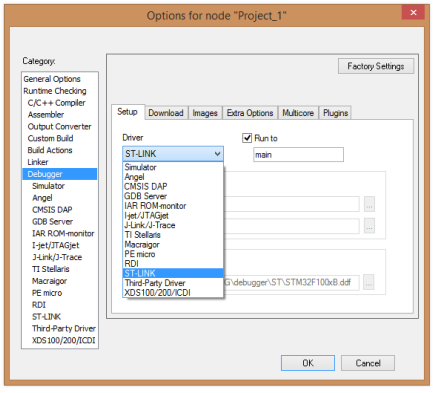

Рис. 3.9. Страница Debbuger, вкладка Setup

9. Во вкладке *Download* нужно поставить галочку возле поля *Use flash loader(s)* (рис. 3.10).

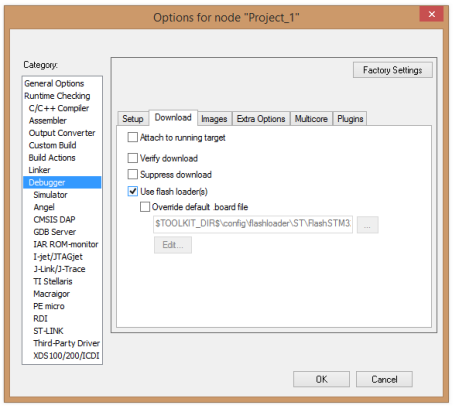

Рис. 3.10. Страница Debbuger, вкладка Download

Далее необходимо перейти на страницу *ST-LINK*. В поле *Reset* следует выбрать *Normal* и в качестве интерфейса отметить *SWD* (рис. 3.11).

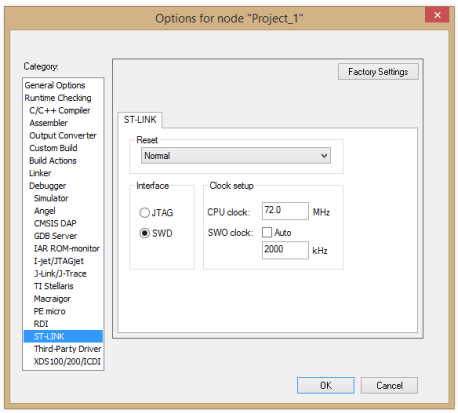

Рис. 3.11. Страница Debbuger, вкладка Setup

10. После осуществления всех настроек в контекстном меню *Options* необходимо нажать правой кнопкой мыши на окно *Workspace* и создать папку: *Add—>Add Group*. Назовем эту папку *CMSIS*. Далее путем нажатия правой кнопки мыши по созданной папке вызываем окно, в котором выбираем *Add—>Add Files*. Необходимо добавить перечень файлов из библиотеки *CMSIS*, к которым ранее были указаны пути (рис. 3.12):

- **core_cm3.c:** Libraries\CMSIS\CM3\CoreSupport\
- **core_cm3.h:** Libraries\CMSIS\CM3\CoreSupport\
- **stm32f10x.h:** Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x
- **system_stm32f10x.h:**Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x
- **system_stm32f10x.c:** Libraries\CMSIS\CM3\DeviceSupport\ST\STM32F10x
- **startup_stm32f10x_md_vl.s:** Libraries\CMSIS\CM3\DeviceSupport\ST\ STM32F10x\startup\iar

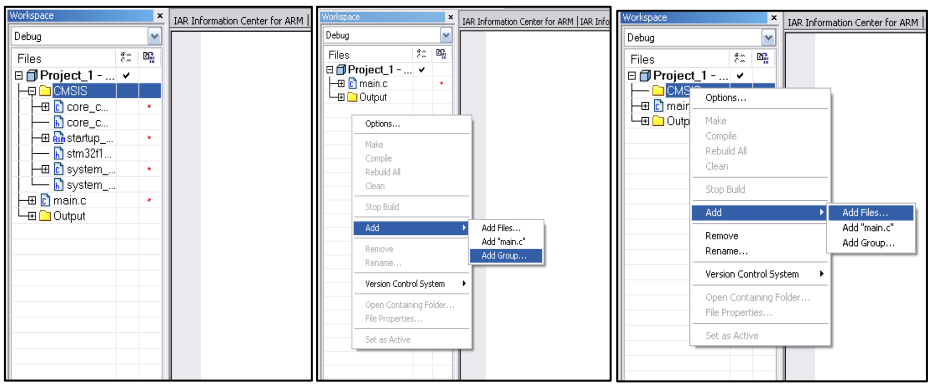

Рис. 3.12. Создание кода программы

11. Для примера работы программы напишем следующий код и сохраним проект (рис. 3.13):

```
#include "stm32f10x.h"
int main( void )
{
}
```

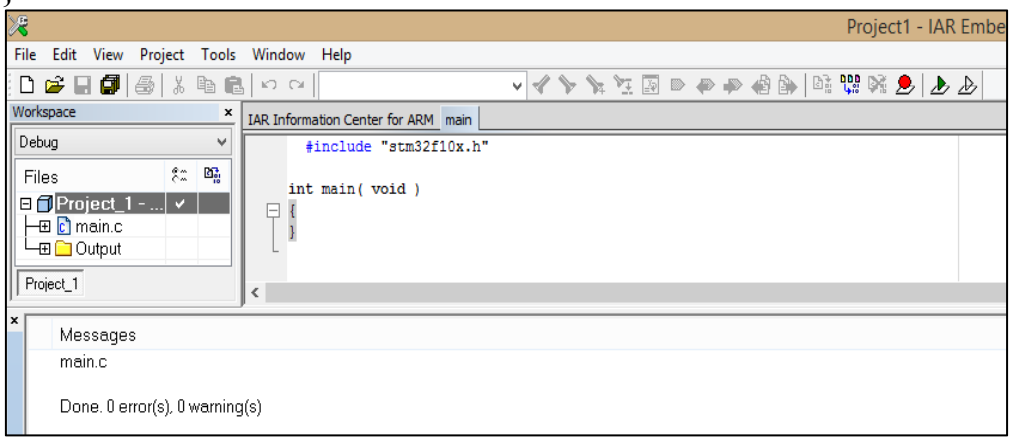

Рис. 3.13. Создание кода программы

12. Загрузка программы в микроконтроллер осуществляется в три этапа (рис. 3.14): компиляция (*Compile*), создание (*Make*), загрузка и отладка (*Download and Debug*).

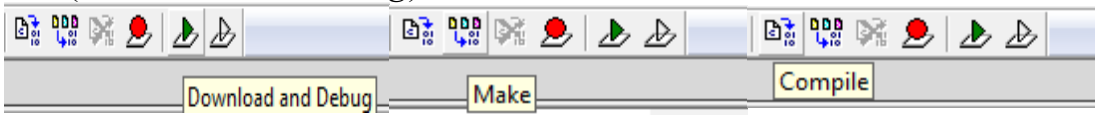

Рис. 3.14. Панели компиляции и загрузки программы

13. Далее можно запускать программу, как показано на рис. 3.15.

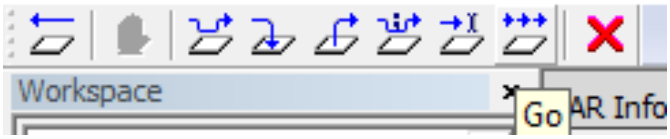

Рис. 3.15. Запуск программы

3.2. Создание проекта в программе *CooCox CoIDE*

Для создания проекта необходимо сделать следующее.

1. Запустить среду *CooCox CoIDE*. Внешний вид стартового окна программы представлен на рис. 3.16 [5].

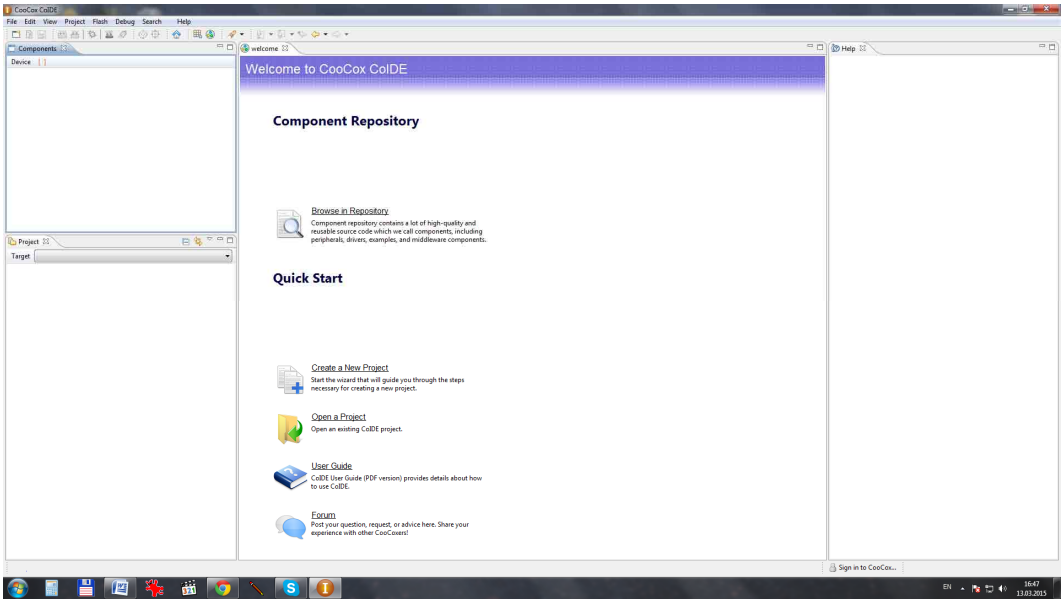

Рис. 3.16. Внешний вид стартового окна программы CooCox CoIDE

2. Для создания нового проекта необходимо выполнить *Project->NewProject*.

3. В открывшемся диалоговом окне (рис. 3.17) необходимо ввести название проекта, а также указать полный путь к папке, в которой он будет храниться. (По умолчанию проект сохраняется в C:\CooCox\CoIDE\workspace.)

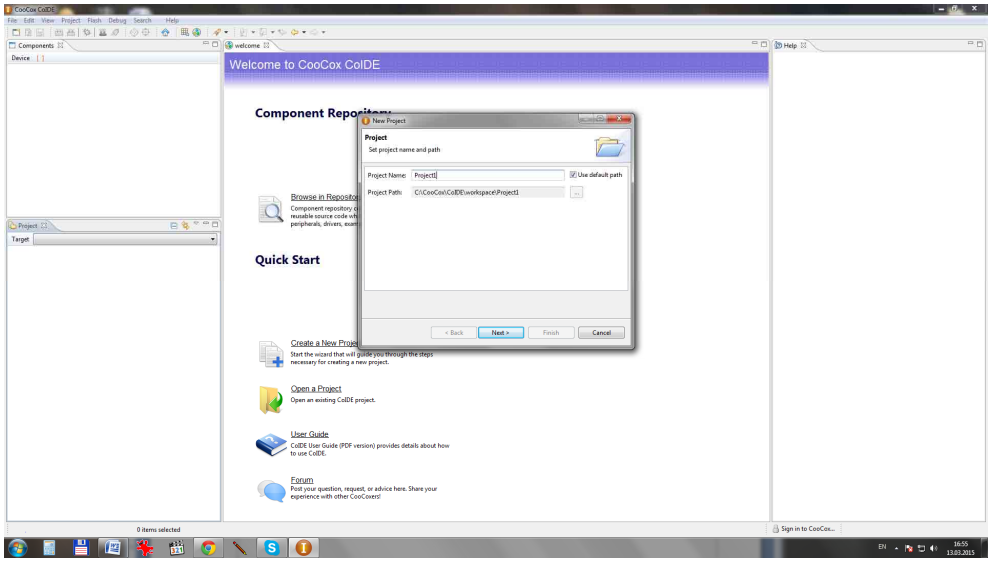

Рис. 3.17. Окно создания нового проекта

4. Далее нужно выбрать, что будет использоваться в данном проекте, – чип или отладочная плата (рис. 3.18).

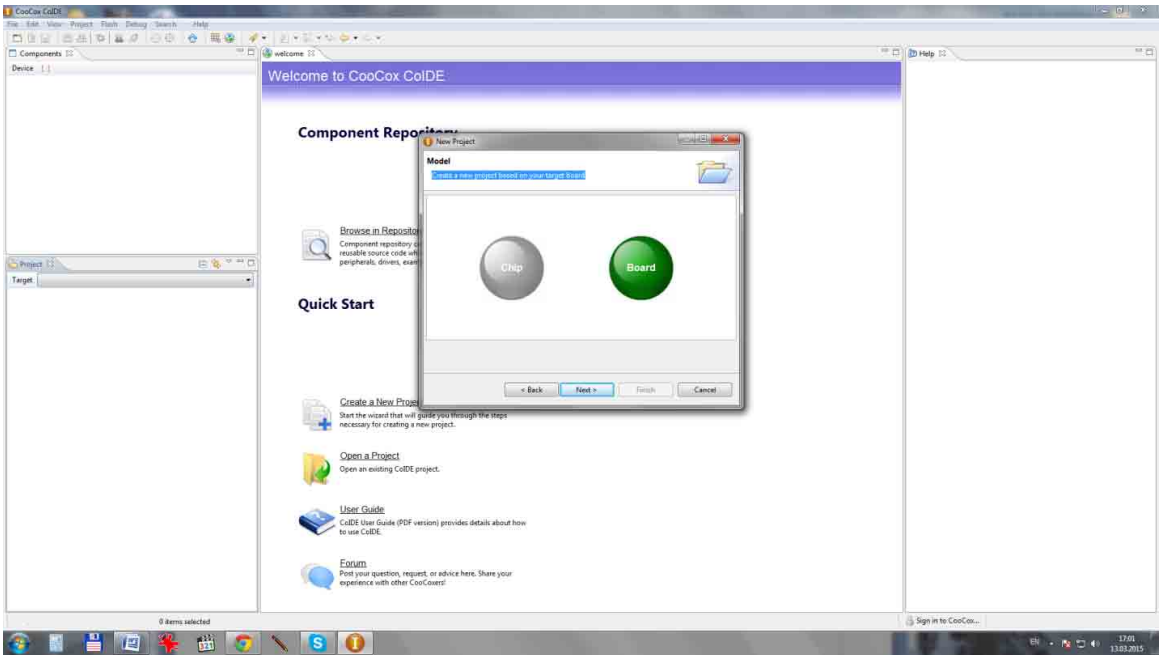

Рис. 3.18. Окно создания нового проекта

5. В следующем окне следует выбрать нужный микроконтроллер (рис. 3.19*) (STM32F100RB)*.

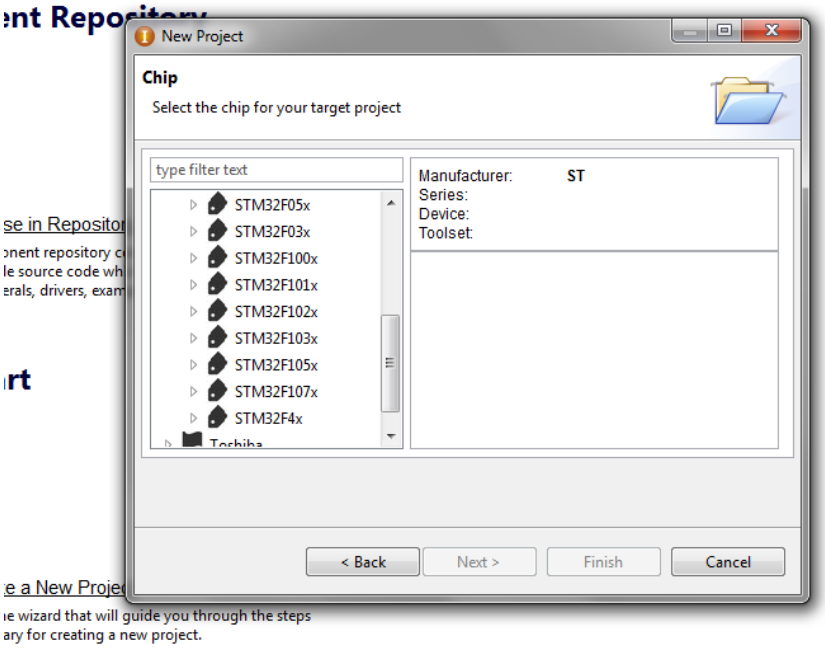

Рис. 3.19. Выбор микроконтроллера

6. В открывшейся странице *Repository* требуется выбрать базовые компоненты, необходимые для использования в проекте (рис. 3.20).

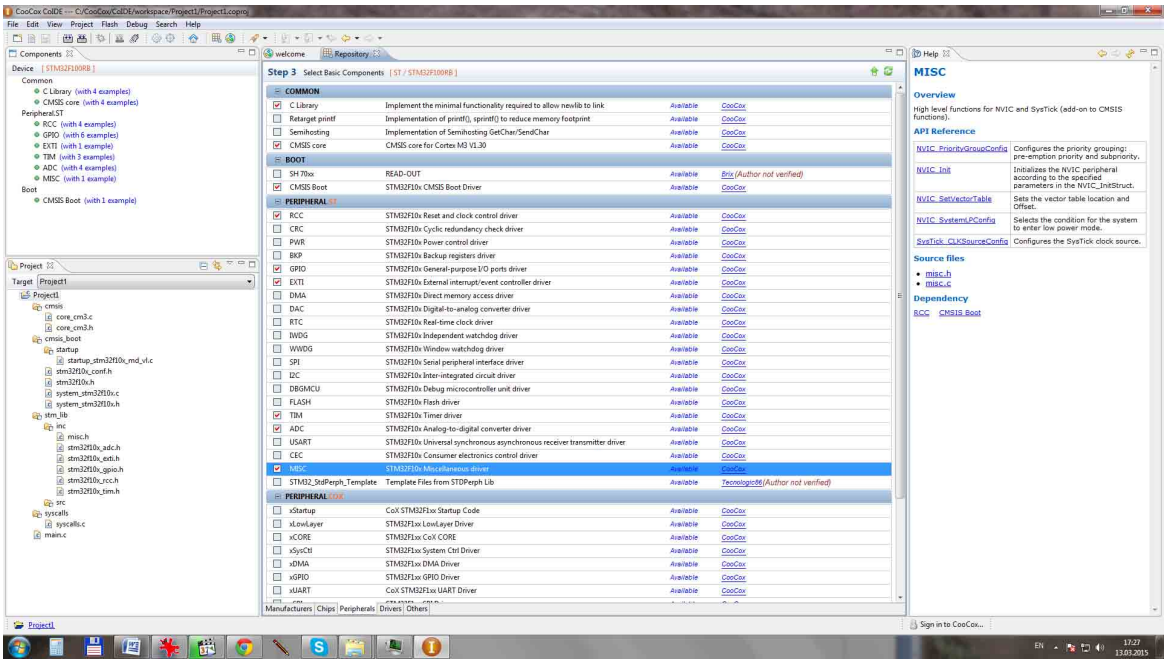

Рис. 3.20. Страница Repository

7. Далее необходимо открыть файл *main.c* (рис. 3.21). В открывшемся окне пишется программа (рис. 3.22).

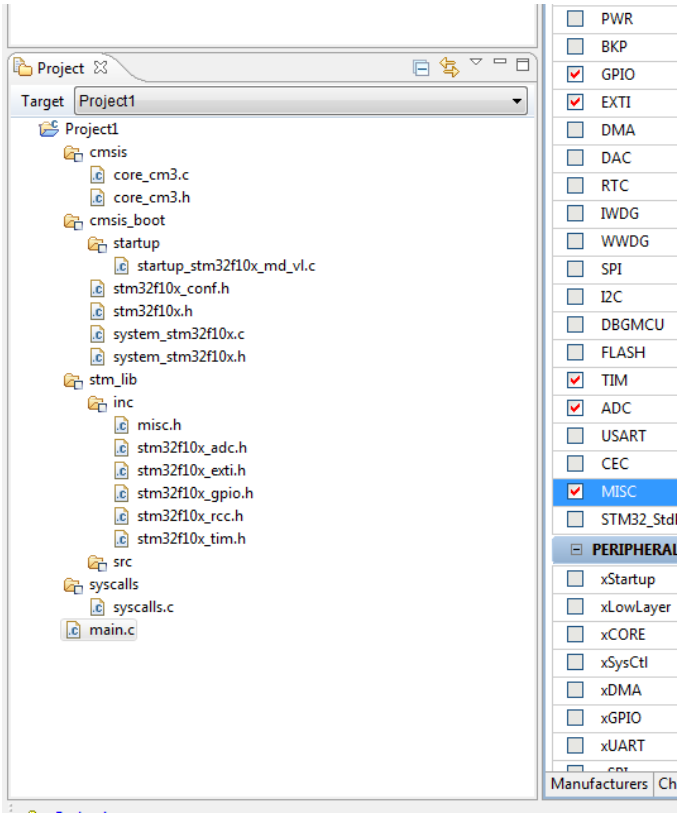

Рис. 3.21. Каталог Project

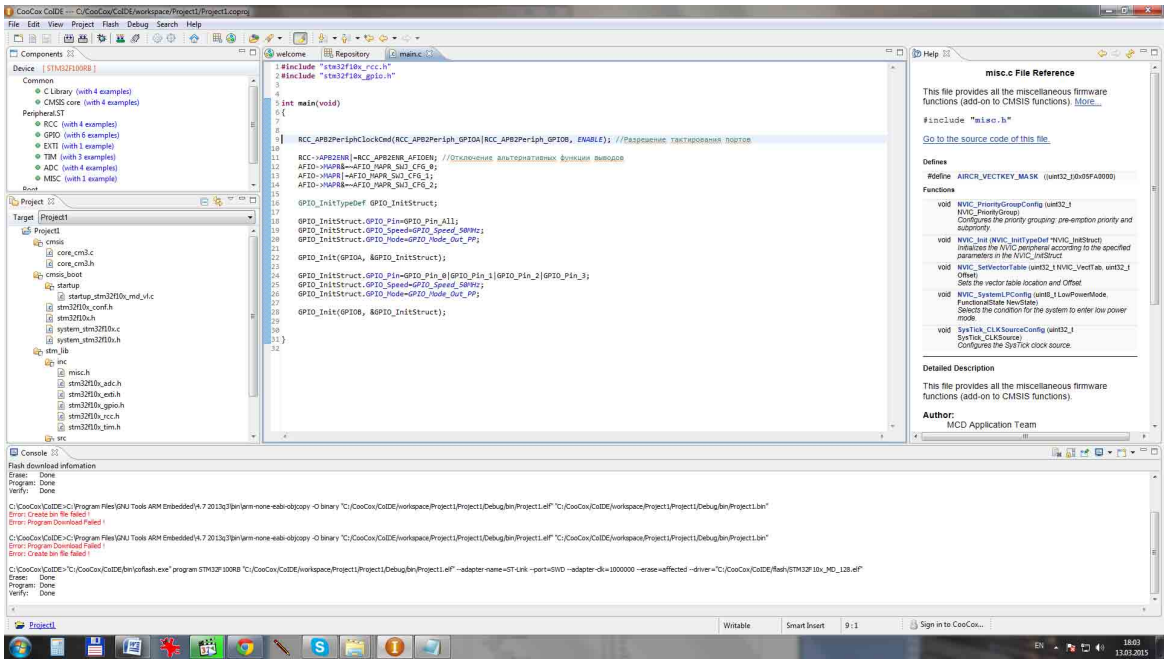

Рис. 3.22. Окно проекта main.c

8. Для прошивки микроконтроллера следует выбрать *ST-Link*, а для этого нужно выбрать *View->Configuration->Debugger->Adapter->ST-Link* (рис. 3.23).

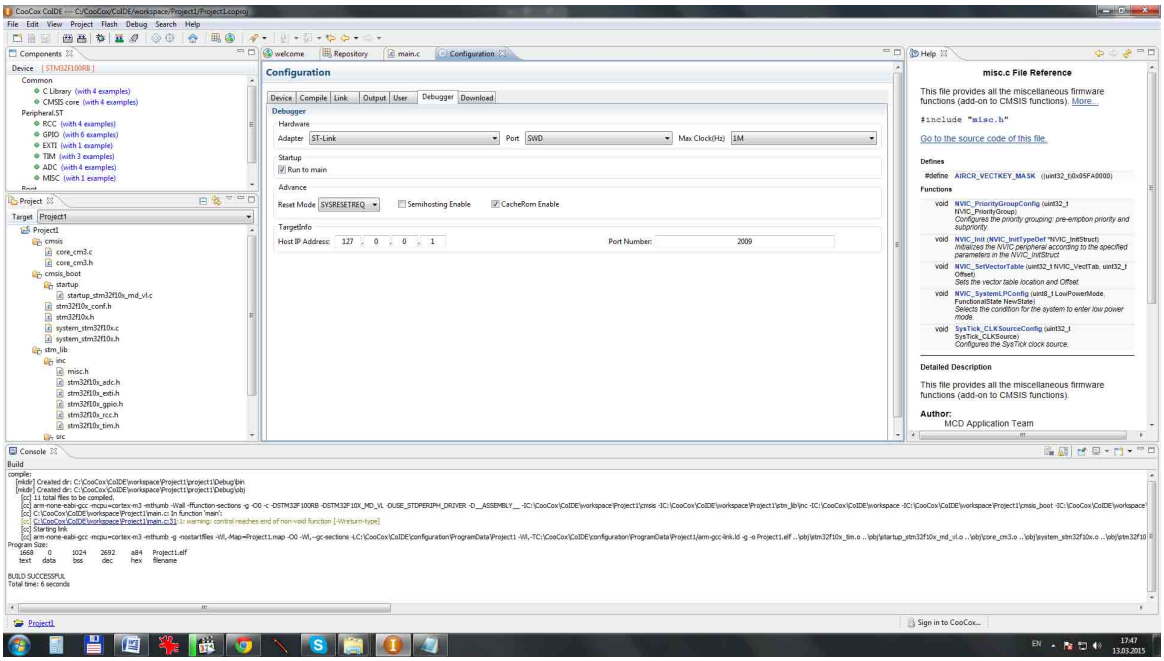

Рис. 3.23. Окно Configuration

9. Для загрузки программы в микроконтроллер необходимо выполнить *Build(F7)*, затем – *Download code to flash* (рис. 3.24).

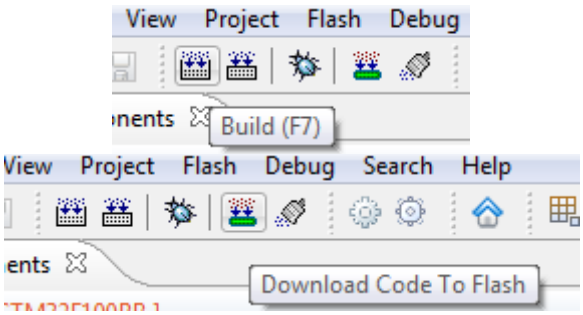

Рис. 3.24. Панели компиляции и загрузки программы

3.3. Примеры программ

3.3.1. Программа настройки портов

Программа реализует настройку выводов порта *B* на различные режимы работы: *PB*0 – выход типа push-pull со скоростью переключения до 50 МГц, *PB*1 – выход с открытым стоком со скоростью переключения до 50 МГц, *PB*2 – дифференциальный вход, не являющийся источником внешних прерываний, *PB*3 – вход с подтягивающим резистором, не являющийся источником внешних прерываний.

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

//основная программа
int main(void) {
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB, ENABLE);
//включение тактирования

GPIO_InitTypeDef GPIO_InitStruct; //объявление структуры настройки портов

GPIO_InitStruct.GPIO_Pin=GPIO_Pin_0; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //определение
                                              //максимальной частоты
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_Out_PP; //выход типа push-pull
GPIO_Init(GPIOB, &GPIO_InitStruct); //заполнение структуры

GPIO_InitStruct.GPIO_Pin=GPIO_Pin_1; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //определение
                                              //максимальной частоты
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_Out_OD; //выход с открытым
                                             //стоком
GPIO_Init(GPIOB, &GPIO_InitStruct); //заполнение структуры
```

```
GPIO_InitStruct.GPIO_Pin=GPIO_Pin_2; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //определение
                                                    //максимальной частоты
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_IN_FLOATING;
//дифференциальный вход
GPIO_Init(GPIOB, &GPIO_InitStruct); //заполнение структуры

GPIO_InitStruct.GPIO_Pin=GPIO_Pin_3; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //определение
                                                    //максимальной частоты
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_IPU; //вход с подтягивающим
                                                  //резистором
GPIO_Init(GPIOB, &GPIO_InitStruct); //заполнение структуры

  while (1) //бесконечный цикл
  {
  }
}
```

3.3.2. Программа инверсии состояния светодиода с использованием системной задержки

Программа реализует инверсию состояния светодиода с использованием системной задержки. Светодиод соединен с девятым выводом порта *C*.

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

GPIO_InitTypeDef GPIO_InitStruct; //объявление структуры настройки портов

static __IO uint32_t TimingDelay; //переменная для функции задержки

//основная программа
int main(void)
{
SysTick_Config(240); //настройка системного таймера
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
//включение тактирования

GPIO_InitStruct.GPIO_Pin=GPIO_Pin_9; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz; //определение макси-
мальной частоты
```

```
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP; //режим – вывод, тип – push-pull
GPIO_Init(GPIOC, &GPIO_InitStruct); //заполнение объявленной структуры

  while(1) //бесконечный цикл
  {
  Delay_x10us (50000); //задержка 0.5 с
  GPIO_ResetBits(GPIOC, GPIO_Pin_9);
  Delay_x10us (50000); //задержка 0.5 с
  GPIO_SetBits(GPIOC, GPIO_Pin_9);
  } //while (1)
} //main

//функция временной задержки
void Delay_x10us(__IO uint32_t nTime)
{
TimingDelay = nTime;
while(TimingDelay != 0);
}
//обработчик прерывания системного таймера
void SysTick_Handler(void)
{
TimingDelay_Decrement(); //вызов подпрограммы декремента переменной для
                         //задержки
}

//подпрограмма декремента переменной для задержки
void TimingDelay_Decrement(void)
{
  if (TimingDelay != 0x00)
  {
  TimingDelay--; //декремент переменной для задержки
  }
}
```

3.3.3. Программа, реализующая эффект маятника

Программа реализует эффект маятника на восьми светодиодах, подключенных к порту *C* (рис. 3.25).

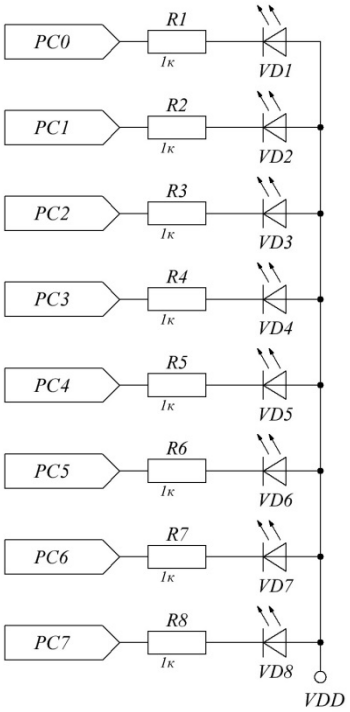

Рис. 3.25. Схема подключения восьми светодиодов к микроконтроллеру STM32F10x

```
#include "stm32f10x_rcc.h"
#include "stm32f10x_gpio.h"
u16 a; //объявление переменной
static __IO uint32_t TimingDelay; //объявление переменной для системной
                                  //задержки

//основная программа
int main(void)
{
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC,  ENABLE);
//включение тактирования

GPIO_InitTypeDef GPIO_InitStruct; //объявление структуры настройки портов

GPIO_InitStruct.GPIO_Pin=GPIO_Pin_All; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //определелеление
                                             //максимальной частоты

GPIO_InitStruct.GPIO_Mode=GPIO_Mode_Out_PP; //режим – вывод с
                                            //открытым стоком
GPIO_Init(GPIOC, &GPIO_InitStruct); //заполнение объявленной структуры
```

```
SysTick_Config(240); //настройка системного таймера
a=0x0001;
GPIOC->ODR=a; //вывод переменной в порт С
m1:
  while(a!=0x80)
  {
  a=a*2;
  GPIOC->ODR=a;
  Delay_x10us(20000); //задержка
  }

  while(a!=0x0001)
  {
  a=a/2;
  GPIOC->ODR=a;
  Delay_x10us(20000);
  }
goto m1;
} //main

//функция временной задержки
void Delay_x10us(__IO uint32_t nTime)
{
TimingDelay = nTime;
while(TimingDelay != 0);
}

//обработчик прерывания системного таймера
void SysTick_Handler(void)
{
TimingDelay_Decrement(); //вызов подпрограммы декремента переменной для задержки
}
//подпрограмма декремента переменной для задержки
void TimingDelay_Decrement(void)
{
  if (TimingDelay != 0x00)
  {
  TimingDelay--; //декремент переменной для задержки
  }
}
```

3.3.4. Программа, реализующая отслеживание состояния кнопки

Программа реализует отслеживание состояния кнопки. При нажатии на кнопку зажигается светодиод, а при отпускании светодиод гаснет. Светодиод подключен к восьмому выводу порта *C*, кнопка соединена с нулевым выводом порта *A*. Внешние прерывания запрещены, отслеживание состояния кнопки производится с помощью считывания входного регистра порта *A*.

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

u16 a; //объявление переменной
static __IO uint32_t TimingDelay; //объявление переменной для системной
                                  //задержки

//основная программа
int main(void)
{

RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph
_GPIOC, ENABLE); //включение тактирования

GPIO_InitTypeDef GPIO_InitStruct; //объявление структуры настройки портов

RCC_APB2PeriphClockCmd(RCC_APB2ENR_AFIOEN, ENABLE);
//включение тактирования

GPIO_InitStruct.GPIO_Pin=GPIO_Pin_8; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //определяем
                                             //максимальную частоту
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_Out_PP; //режим – вывод,
                                            //тип – push-pull
GPIO_Init(GPIOC, &GPIO_InitStruct); //заполнение объявленной структуры

GPIO_InitStruct.GPIO_Pin=GPIO_Pin_0; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_IN_FLOATING; //режим –
                                          //дифференциальный вход
GPIO_Init(GPIOA, &GPIO_InitStruct); //заполнение объявленной структуры

  while(1) //бесконечный цикл
  {
  a=GPIO_ReadInputDataBit (GPIOA,GPIO_Pin_0 ); //присвоение переменной
                                               //значения бита
```

```
    if(a==0)
    {
    GPIO_ResetBits(GPIOC, GPIO_Pin_8); //сброс бита
    }

    else
    {
    GPIO_SetBits(GPIOC, GPIO_Pin_8); //установка бита
    }
  }
}
```

3.3.5. Программа, реализующая переключение светодиода по внешнему прерыванию

Программа реализует инверсию состояния светодиода по нажатию кнопки, по внешнему прерыванию. Светодиод подключен к восьмому выводу порта *C*, кнопка соединена с нулевым выводом порта *A*.

```
#include "stm32f10x_rcc.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_exti.h"

u16 a; //объявление переменной

//основная программа
int main(void)
{
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph
_GPIOC, ENABLE); //включение тактирования

RCC_APB2PeriphClockCmd(RCC_APB2ENR_AFIOEN , ENABLE);
//включение тактирования

GPIO_InitTypeDef GPIO_InitStruct; //объявление структуры настройки портов
GPIO_InitStruct.GPIO_Pin=GPIO_Pin_8; //выбор настраиваемых выводов

GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //определение
                                              //максимальной частоты
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_Out_PP; //режим – вывод,
                                              //тип –  push-pull
GPIO_Init(GPIOC, &GPIO_InitStruct); //заполнение объявленной структуры
```

```
GPIO_InitStruct.GPIO_Pin=GPIO_Pin_0; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_IN_FLOATING; //режим –
                                                  //дифференциальный вход
GPIO_Init(GPIOA, &GPIO_InitStruct); //заполнение объявленной структуры

GPIO_EXTILineConfig(GPIO_PortSourceGPIOA,        GPIO_PinSource0);
//назначение вывода 0 порта А источником внешнего прерывания

EXTI_InitTypeDef EXTI_InitStruct; //объявление структуры настройки
                                  //внешних прерываний

EXTI_InitStruct.EXTI_Line=EXTI_Line0; //выбор линии внешнего прерывания
EXTI_InitStruct.EXTI_LineCmd=ENABLE; //разрешение прерывания на
                                     //выбранной линии
EXTI_InitStruct.EXTI_Mode=EXTI_Mode_Interrupt; //режим прерывания
EXTI_InitStruct.EXTI_Trigger=EXTI_Trigger_Rising; //прерывание по
                                                  //переднему фронту
EXTI_Init(&EXTI_InitStruct); //заполнение объявленной структуры

NVIC_EnableIRQ (EXTI0_IRQn); //разрешение прерывания
  while(1) //бесконечный цикл
  {
  } //while (1)
} //main

//обработчик прерывания EXTI0
void EXTI0_IRQHandler(void)
{
GPIOC->ODR^=GPIO_Pin_8; //инверсия бита вывода, к которому подключен
                        //светодиод
EXTI->PR|=0x01; //очистка флага прерывания
}
```

3.3.6. Программа, реализующая эффект бегущей единицы

Программа реализует эффект бегущей единицы на восьми светодиодах, подключенных к порту *C*. Направление задается с помощью кнопки, по прерыванию. Кнопка подключена к нулевому выводу порта *A*.

```
#include "stm32f10x_rcc.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_exti.h"
```

```
u16 a; //объявление переменной
int k; //объявление переменной
static __IO uint32_t TimingDelay; //объявление переменной для системной
                                  //задержки

//основная программа
int main(void)
{
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA|RCC_APB2Periph
_GPIOC, ENABLE); //включение тактирования
RCC_APB2PeriphClockCmd(RCC_APB2ENR_AFIOEN , ENABLE);
//включение тактирования

GPIO_InitTypeDef GPIO_InitStruct; //объявление структуры настройки портов
GPIO_InitStruct.GPIO_Pin=GPIO_Pin_All; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //определение
                                             //максимальной частоты
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_Out_PP; //режим – вывод,
                                            //тип – push-pull
GPIO_Init(GPIOC, &GPIO_InitStruct); //заполнение объявленной структуры

GPIO_InitStruct.GPIO_Pin=GPIO_Pin_0; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_IN_FLOATING; //режим –
                                                 //дифференциальный вход
GPIO_Init(GPIOA, &GPIO_InitStruct); //заполнение объявленной структуры

GPIO_EXTILineConfig(GPIO_PortSourceGPIOA, GPIO_PinSource0);
//назначение вывода 0 порта А источником внешних прерываний

EXTI_InitTypeDef EXTI_InitStruct; //объявление структуры настройки
                                  //внешних прерываний

EXTI_InitStruct.EXTI_Line=EXTI_Line0; //выбор настраиваемой линии
EXTI_InitStruct.EXTI_LineCmd=ENABLE; //разрешение внешнего
                                     //прерывания на выбранной линии
EXTI_InitStruct.EXTI_Mode=EXTI_Mode_Interrupt; //режим прерывания
EXTI_InitStruct.EXTI_Trigger=EXTI_Trigger_Rising; //прерывание по
                                                  //переднему фронту
EXTI_Init(&EXTI_InitStruct); //заполнение объявленной структуры
NVIC_EnableIRQ (EXTI0_IRQn); //разрешение прерывания
SysTick_Config(240); //настройка системного таймера
k=0x01;
```

```
a=0x00;
GPIOC->ODR=k; //вывод переменной в порт
  while(1) //бесконечный цикл
  {
    if(a==0x0000)
    {
      if(k!=0x80)
      {
      k <<=1; //побитный сдвиг влево
      GPIOC->ODR=k;
      Delay_x10us(20000); //задержка
      }
      else
      {
      k=0x01;
      }
    }
    if(a==0xFFFF)
    {
      if(k!=0x01)
      {
      k >>=1; //побитный сдвиг вправо
      GPIOC->ODR=k;
      Delay_x10us(20000); //задержка
      }
      else
      {
      k=0x80;
      }
    }
  }
}



//обработчик прерывания EXTI0
void EXTI0_IRQHandler(void)
{

a=~a; //инверсия переменной
EXTI->PR|=0x01; //очистка флага прерывания
```

```
}

//функция временной задержки
void Delay_x10us(__IO uint32_t nTime)
{
TimingDelay = nTime;
while(TimingDelay != 0);
}

//подпрограмма декремента переменной для задержки
void TimingDelay_Decrement(void)
{
  if (TimingDelay != 0x00)
  {
  TimingDelay--; //декремент переменной для задержки
  }
}

//обработчик прерывания системного таймера
void SysTick_Handler(void) {
{
TimingDelay_Decrement(); //вызов подпрограммы декремента переменной для задержки
}}
```

3.3.7. Программа, реализующая мерцание светодиода с использованием прерываний таймера 2

Программа реализует инверсию состояния светодиода с использованием прерываний таймера 2 по переполнению. Светодиод подсоединен к восьмому выводу порта *C*.

```
#include "stm32f10x_rcc.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_tim.h"
#include <misc.h>

//основная программа
int main(void)
{
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
//включение тактирования портов
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
//включение тактирования таймера
```

```
GPIO_InitTypeDef GPIO_InitStruct; //объявление структуры настройки портов
GPIO_InitStruct.GPIO_Pin=GPIO_Pin_8; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //определение
                                                    //максимальной частоты
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_Out_PP; //режим – вывод, тип –
                                                    //push-pull
GPIO_Init(GPIOC, &GPIO_InitStruct); //заполнение объявленной структуры

TIM_TimeBaseInitTypeDef TIM_InitStructure; //объявление структуры
                                                    //настройки таймеров
TIM_TimeBaseStructInit(&TIM_InitStructure); //инициализация структуры
TIM_InitStructure.TIM_Prescaler = 15000; //предварительный делитель
TIM_InitStructure.TIM_Period = 1000; //период таймера
TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE); //прерывание по
                                                    //переполнению
TIM_TimeBaseInit(TIM2, &TIM_InitStructure); //заполнение объявленной
                                                    //структуры
NVIC_EnableIRQ(TIM2_IRQn); //разрешение прерывания от таймера
TIM_Cmd(TIM2, ENABLE); //включение таймера

  while(1) //бесконечный цикл
  {
  } //while(1)
} //main

//вектор прерывания таймера
void TIM2_IRQHandler (void)
{
GPIOC->ODR^=GPIO_Pin_8; //инверсия бита вывода, к которому подключен
                                    //светодиод
TIM_ClearITPendingBit(TIM2, TIM_IT_Update); //сброс флага переполнения
}
```

3.3.8. Программа, реализующая инверсию состояния светодиода, с использованием прерываний таймера 6.

Программа реализует инверсию состояния светодиода с использованием прерываний таймера 6 по переполнению. Светодиод соединен с восьмым выводом порта *C*.

```
#include "stm32f10x_rcc.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_tim.h"
#include <misc.h>
```

```
//основная программа
int main(void)
{
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC,          ENABLE);
//включение тактирования
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6,           ENABLE);
//включение тактирования
GPIO_InitTypeDef GPIO_InitStruct; //создание переменной структуры
GPIO_InitStruct.GPIO_Pin=GPIO_Pin_8; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //определение
                                                    //максимальной частоты
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_Out_PP; //режим – вывод,
                                                     //тип – push-pull
GPIO_Init(GPIOC, &GPIO_InitStruct); //заполнение объявленной структуры

TIM_TimeBaseInitTypeDef TIM_InitStructure; //объявление структуры
                                                  //настройки таймера
TIM_TimeBaseStructInit(&TIM_InitStructure); //инициализация структуры
TIM_InitStructure.TIM_Prescaler = 15000; //предделитель
TIM_InitStructure.TIM_Period = 1000; //период таймера
TIM_ITConfig(TIM6, TIM_IT_Update, ENABLE); //прерывание по
                                                    //переполнению
TIM_TimeBaseInit(TIM6, &TIM_InitStructure); //функция настройки таймера
NVIC_EnableIRQ(TIM6_DAC_IRQn); //разрешение прерывания от таймера
TIM_Cmd(TIM6, ENABLE); //включение таймера

  while(1) //бесконечный цикл
  {
  } //while(1)
}//main

//вектор прерывания таймера
void TIM6_DAC_IRQHandler (void)
{
GPIOC->ODR^=GPIO_Pin_8; //инверсия бита вывода, к которому подключен
                                  //светодиод
TIM_ClearITPendingBit(TIM6, TIM_IT_Update); //сброс флага переполнения
}
```

3.3.9. Программа, реализующая генерацию импульса с использованием прерываний по переполнению двух таймеров

Программа реализует генерацию импульса с использованием прерываний по переполнению двух таймеров. Таймер 2 отвечает за время импульса, пауза формируется с помощью таймера 6. Импульс генерируется на восьмом выводе порта *PC*, к которому подсоединен светодиод.

```
#include "stm32f10x_rcc.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_tim.h"
#include <misc.h>

//основная программа
int main(void)
{
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
//включение тактирования
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM6|RCC_APB1Periph_
TIM2, ENABLE); //включение тактирования
GPIO_InitTypeDef GPIO_InitStruct; //объявление структуры настройки портов
GPIO_InitStruct.GPIO_Pin=GPIO_Pin_8; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //определение
                                                //максимальной частоты
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_Out_PP; //режим – вывод,
                                                //тип – push-pull

GPIO_Init(GPIOC, &GPIO_InitStruct); //заполнение объявленной структуры

TIM_TimeBaseInitTypeDef TIM_InitStructure; //объявление структуры
                                                //настройки таймеров
TIM_TimeBaseStructInit(&TIM_InitStructure); //инициализация структуры
TIM_InitStructure.TIM_Prescaler = 20000; //предделитель
TIM_InitStructure.TIM_Period = 2000; //период таймера
TIM_ITConfig(TIM6, TIM_IT_Update, ENABLE); //прерывание по
                                                //переполнению
TIM_TimeBaseInit(TIM6, &TIM_InitStructure); //функция настройки таймера
NVIC_EnableIRQ(TIM6_DAC_IRQn); //разрешение прерывания от таймера

TIM_TimeBaseStructInit(&TIM_InitStructure); //инициализация структуры
TIM_InitStructure.TIM_Prescaler = 15000; //предделитель
TIM_InitStructure.TIM_Period = 1000; //период таймера
TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE); //прерывание по
                                                //переполнению
```

```
TIM_TimeBaseInit(TIM2, &TIM_InitStructure); //заполнение объявленной
                                            //структуры
NVIC_EnableIRQ(TIM2_IRQn); //разрешение прерывания от таймера
TIM_Cmd(TIM2, ENABLE); //включение таймера

  while(1) //бесконечный цикл
  {
  } //while(1)
} //main

//вектор прерывания таймера
void TIM6_DAC_IRQHandler (void)
{
GPIO_ResetBits(GPIOC, GPIO_Pin_8); //сброс бита
TIM_ClearITPendingBit(TIM6, TIM_IT_Update); //сброс флага переполнения
TIM_Cmd(TIM2, ENABLE); //включение таймера
}

//вектор прерывания таймера
void TIM2_IRQHandler (void)
{
GPIO_SetBits(GPIOC, GPIO_Pin_8); //установка бита
TIM_ClearITPendingBit(TIM2, TIM_IT_Update); //сброс флага переполнения
TIM_Cmd(TIM6, ENABLE); //включение таймера
}
```

3.3.10. Программа, реализующая эффект маятника по прерыванию таймера

Программа реализует эффект маятника на восьми светодиодах, подключенных к порту *C* (рис. 3.25). Все задержки организованы с помощью таймера 2.

```
#include "stm32f10x_rcc.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_tim.h"
#include <misc.h>
int j;//Объявление переменной
int k;
int i;

//основная программа
int main(void)
{
```

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC,          ENABLE); //включение тактирования
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2,           ENABLE); //включение тактирования

GPIO_InitTypeDef GPIO_InitStruct; //объявление структуры настройки портов
GPIO_InitStruct.GPIO_Pin=GPIO_Pin_All; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //определение
                                                    //максимальной частоты
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_Out_PP; //режим – вывод,
                                                    //тип – push-pull
GPIO_Init(GPIOC, &GPIO_InitStruct); //заполнение объявленной структуры

TIM_TimeBaseInitTypeDef TIM_InitStructure; //объявление структуры
                                                  //настройки таймеров
TIM_TimeBaseStructInit(&TIM_InitStructure); //инициализация структуры
TIM_InitStructure.TIM_Prescaler = 10000; //предделитель
TIM_InitStructure.TIM_Period = 500; //период таймера
TIM_ITConfig(TIM2, TIM_IT_Update, ENABLE); //прерывание по
                                                  //переполнению
TIM_TimeBaseInit(TIM2, &TIM_InitStructure); //заполнение объявленной
                                                  //структуры
NVIC_EnableIRQ(TIM2_IRQn); //разрешение прерывания от таймера
TIM_Cmd(TIM2, ENABLE); //включение таймера

k=0xFE;
GPIOC->ODR=k;
  while(1) //бесконечный цикл
  {
    for (j=0;j<=6;j++)
    {
    k <<=1; //вывод на порт логической "1"
    GPIOC->ODR=k;
    TIM_Cmd(TIM2, ENABLE); //включение таймера

      while (i<5)
      {
      }
    i=0;
    }
    for (j=0;j<=6;j++)
    {
```

```
    k>>=1; //вывод на порт логической "1"
    GPIOC->ODR=k;
    TIM_Cmd(TIM2, ENABLE); //включение таймера

      while (i<5)
      {
      }
    i=0;
    }
  }
}

//вектор прерывания таймера
void TIM2_IRQHandler (void)
{
GPIOC->ODR^=GPIO_Pin_8; //инверсия бита вывода, к которому подключен светодиод
TIM_ClearITPendingBit(TIM2, TIM_IT_Update); //сброс флага переполнения
i++;
TIM_Cmd(TIM2, ENABLE); //включение таймера
}
```

3.3.11. Программа, реализующая отправку данных по UART

По прерыванию от нажатия кнопки включается светодиод и на ПК выводится сообщение *on*, при повторном нажатии кнопки светодиод гаснет и на ПК выводится сообщение *off*.

```
#include "stm32f10x_gpio.h"
#include "stm32f10x_exti.h"
#include "stm32f10x_rcc.h"
#include <misc.h>
#include "stm32f10x_usart.h"

int i,a; //объявление переменных
u32 hh; //объявление переменной

void port(void); //подпрограмма настройки портов
void ext(void); //подпрограмма настройки внешних прерываний

//основная программа
int main(void)
{
```

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC|RCC_APB2Periph
_GPIOB|RCC_APB2Periph_GPIOA, ENABLE); //включение тактирования
RCC_APB2PeriphClockCmd(RCC_APB2ENR_AFIOEN, ENABLE);
//включение тактирования
RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3, ENABLE);
//включение тактирования

port(); //вызов подпрограммы настройки портов
ext(); //вызов подпрограммы настройки внешних прерываний

USART_InitTypeDef USART_InitStructure; //объявление структуры настройки
USART
USART_InitStructure.USART_BaudRate = 9600; //скорость
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
//8 бит данных
USART_InitStructure.USART_StopBits = USART_StopBits_1;
//один стоп-бит
USART_InitStructure.USART_Parity = USART_Parity_No;
//четность – нет
USART_InitStructure.USART_HardwareFlowControl =
USART_HardwareFlowControl_None; //управление потоком – нет
USART_InitStructure.USART_Mode = USART_Mode_Rx |
USART_Mode_Tx; //разрешение приема и передачи
USART_Init(USART3, &USART_InitStructure); //заполнение объявленной
                                          //структуры
NVIC_EnableIRQ (USART3_IRQn); //разрешение прерываний UART3
a=0x0000;
  while(1) //бесконечный цикл
  {
  }
}

//подпрограмма настройки портов
void port(void)
{
GPIO_InitTypeDef GPIO_InitStruct; //объявление структуры настройки портов

GPIO_InitStruct.GPIO_Pin=GPIO_Pin_8|GPIO_Pin_9; //выбор
                                          //настраиваемых выводов
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //определение
                                          //максимальной частоты
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_Out_PP; //режим – вывод,
```

```
                                                    //тип – push-pull
GPIO_Init(GPIOC, &GPIO_InitStruct); //заполнение объявленной структуры

GPIO_InitStruct.GPIO_Pin=GPIO_Pin_10; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //определение
                                                    //максимальной частоты
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_Out_PP; //режим – вывод,
                                                    //тип – push-pull
GPIO_Init(GPIOB, &GPIO_InitStruct); //заполнение объявленной структуры
GPIO_InitStruct.GPIO_Pin=GPIO_Pin_11; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Speed=GPIO_Speed_50MHz; //определение
                                                    //максимальной частоты
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_IN_FLOATING; //режим –
                                                  //дифференциальный вход
GPIO_Init(GPIOB, &GPIO_InitStruct); //заполнение объявленной структуры
GPIO_InitStruct.GPIO_Pin=GPIO_Pin_0; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Mode=GPIO_Mode_IN_FLOATING; //режим –
                                                    //дифференциальный вход
GPIO_Init(GPIOA, &GPIO_InitStruct); //заполнение объявленной структуры

GPIO_EXTILineConfig(GPIO_PortSourceGPIOA,        GPIO_PinSource0);
//назначение вывода 0 порта А источником внешних прерываний
}

//подпрограмма настройки внешних прерываний
void ext(void)
{
EXTI_InitTypeDef EXTI_InitStruct; //объявление структуры настройки
                                        //внешних прерываний

EXTI_InitStruct.EXTI_Line=EXTI_Line0; //выбор настраиваемой линии
EXTI_InitStruct.EXTI_LineCmd=ENABLE; //разрешение прерывания на
                                        //выбранной линии
EXTI_InitStruct.EXTI_Mode=EXTI_Mode_Interrupt; //режим прерывания
EXTI_InitStruct.EXTI_Trigger=EXTI_Trigger_Rising; //прерывание по
                                                    //переднему фронту
EXTI_Init(&EXTI_InitStruct); //заполнение объявленной структуры
NVIC_EnableIRQ (EXTI0_IRQn); //разрешение прерывания
USART_ITConfig(USART3, USART_IT_RXNE, ENABLE); // разрешение
                                                    //прерывания приемника
USART_Cmd(USART3, ENABLE); //включение USART3
}
//подпрограмма отправки 1 байта по UART
```

```
void send_to_uart(uint8_t data)
{
while(!(USART3->SR & USART_SR_TC)); //при равенстве 1 бита TC в
                                    //регистре SR
USART3->DR=data; //отправка байта через UART
}

//Обработчик прерывания EXTI0
void EXTI0_IRQHandler(void)
{
 if(a==0x0001)
 {
 GPIOC->ODR^=GPIO_Pin_8; //инверсия состояния светодиода
 EXTI->PR|=0x01; //очистка флага
 send_to_uart('o'); // отправка данных
 send_to_uart('f');
 send_to_uart('f');
 a--; //декремент переменной
 }

 else
 {
 GPIOC->ODR^=GPIO_Pin_8; //инверсия состояния светодиода
 EXTI->PR|=0x01; //очистка флага
 send_to_uart('o'); //отправка данных
 send_to_uart('n');
 a++; //инкремент переменной
 }
}
```

3.3.12. Программа настройки и запуска ШИМ

Программа реализует генерацию сигналов ШИМ на двух каналах таймера 1.

```
# include "stm32f10x_gpio.h"
# include "stm32f10x_rcc.h"
# include "stm32f10x_tim.h"

//основная программа
int main(void)
{
GPIO_InitTypeDef GPIO_InitStructure; //объявление структуры настройки
                                     //портов
```

```
TIM_TimeBaseInitTypeDef  TIM_TimeBaseStructure; //объявление структуры
                                                //настройки таймеров
TIM_OCInitTypeDef  TIM_OCInitStructure; //объявление структуры настройки
                                        //выходных каналов таймеров

RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1 |
RCC_APB2Periph_GPIOA | RCC_APB2Periph_GPIOC|
RCC_APB2Periph_GPIOB|RCC_APB2Periph_AFIO, ENABLE);
//тактирование периферии
RCC_APB1PeriphClockCmd(RCC_APB1Periph_TIM2, ENABLE);
//тактирование периферии

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_8|GPIO_Pin_9;  //выбор
                                                //настраиваемых выводов
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF_PP; //режим
                                        //альтернативной функции, push-pull
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz; //определение
                                                //максимальной частоты
GPIO_Init(GPIOA, &GPIO_InitStructure); //заполнение объявленной
                                        //структуры

TIM_TimeBaseStructure.TIM_Prescaler = 0; // предделитель счетчика
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
//режим счета вверх
TIM_TimeBaseStructure.TIM_Period = 5000; //период счета
TIM_TimeBaseStructure.TIM_ClockDivision = 0; //деление частоты
TIM_TimeBaseStructure.TIM_RepetitionCounter = 0; //обнуление счетчика
                                                //повторений

TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure); //заполнение
                                                //объявленной структуры

TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1; //режим
                                                //ШИМ1
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
//разрешение работы неинверсного канала
TIM_OCInitStructure.TIM_Pulse = 1000; //время импульса

TIM_OC1Init(TIM1, &TIM_OCInitStructure); //заполнение объявленной
                                        //структуры

TIM_OCInitStructure.TIM_Pulse = 3000;
TIM_OC2Init(TIM1, &TIM_OCInitStructure); //заполнение объявленной
                                        //структуры
```

```
TIM_Cmd(TIM1, ENABLE); //запуск таймера 1

TIM_CtrlPWMOutputs(TIM1, ENABLE); //включение выходов ШИМ

  while(1)
  {
  }
}
```

3.3.13. Программа настройки и запуска АЦП

Программа реализует инверсию состояния светодиода, в зависимости от результата преобразования АЦП. Преобразования происходят на первом канале АЦП (первый вывод порта *A*).

```
#include "stm32f10x_rcc.h"
#include "stm32f10x_gpio.h"
#include "stm32f10x_adc.h"

void ADC_init (void);

u16 adc_res,y;

GPIO_InitTypeDef GPIO_InitStruct;
ADC_InitTypeDef ADC_InitStructure;

//основная программа
int main(void)
{
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC|RCC_APB2Periph
_GPIOA|RCC_APB2Periph_AFIO, ENABLE); //тактирование периферии
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1,  ENABLE);
//тактирование ADC1

GPIO_InitStruct.GPIO_Pin = GPIO_Pin_8|GPIO_Pin_9; //выбор
                                                  //настраиваемых выводов
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz; //определение
                                                  //максимальной частоты
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP; //режим – вывод,
                                                  //тип – push-pull
GPIO_Init(GPIOC, &GPIO_InitStruct); //заполнение объявленной структуры

GPIO_InitStruct.GPIO_Pin = GPIO_Pin_1; //выбор настраиваемых выводов
```

```
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AIN; //режим – аналоговый вход
GPIO_Init(GPIOA, &GPIO_InitStruct); //заполнение объявленной структуры

ADC_init(); //вызов подпрограммы настройки АЦП

  while(1)
  {
  for (y=0;y<(adc_res*10);y++);
  GPIO_ResetBits(GPIOC, GPIO_Pin_8);
  for (y=0;y<(adc_res*10);y++);
  GPIO_SetBits(GPIOC, GPIO_Pin_8);
  }
}

//подпрограмма настройки АЦП
void ADC_init (void)
{
NVIC_EnableIRQ(ADC1_IRQn); //разрешение прерываний АЦП1
NVIC_SetPriority (ADC1_IRQn, 1); //приоритет прерываний

ADC_InitStructure.ADC_Mode = ADC_Mode_Independent; //независимый
                                                   //режим работы АЦП
ADC_InitStructure.ADC_ScanConvMode = DISABLE; //запрещение режима
                                              //сканирования

ADC_InitStructure.ADC_ContinuousConvMode = ENABLE;
//режим непрерывных преобразований
ADC_InitStructure.ADC_ExternalTrigConv                    =
ADC_ExternalTrigConv_None; //отсутствие внешнего источника запуска
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
//выравнивание данных по правому краю
ADC_InitStructure.ADC_NbrOfChannel = 1; //номер канала
ADC_Init (ADC1, &ADC_InitStructure); //заполнение объявленной структуры

ADC_RegularChannelConfig (ADC1, ADC_Channel_1, 1,
ADC_SampleTime_239Cycles5); //настройка регулярного канала
ADC_ITConfig (ADC1, ADC_IT_EOC, ENABLE ); //разрешение прерываний
                                          //по окончанию преобразования

ADC_Cmd (ADC1,ENABLE); //запуск АЦП

ADC_ResetCalibration(ADC1); //сброс предыдущего преобразования
```

```
while(ADC_GetResetCalibrationStatus(ADC1));
ADC_StartCalibration(ADC1); //старт нового преобразования
while(ADC_GetCalibrationStatus(ADC1));
ADC_Cmd (ADC1,ENABLE); //запуск АЦП
}

//вектор прерывания АЦП1
void ADC1_IRQHandler (void)
{
ADC_ClearFlag (ADC1, ADC_FLAG_EOC); //сброс флага окончания
                                    //преобразования АЦП1
adc_res = ADC1->DR; //считывание результата преобразования
}
```

Глава 4. Микроконтроллер STM32F40x

4.1. Примеры программ

4.1.1. Программа настройки портов на ввод-вывод

Программа реализует настройку тринадцатого вывода порта *D*, к которому подключен светодиод, на вывод информации. Нулевой вывод, с которым соединена кнопка (рис. 4.1), настраивается на ввод информации. При нажатии на кнопку происходит включение светодиода, при отпускании кнопки светодиод отключается. Состояние нажатия кнопки отслеживается постоянно, внешние прерывания отсутствуют [6, 7].

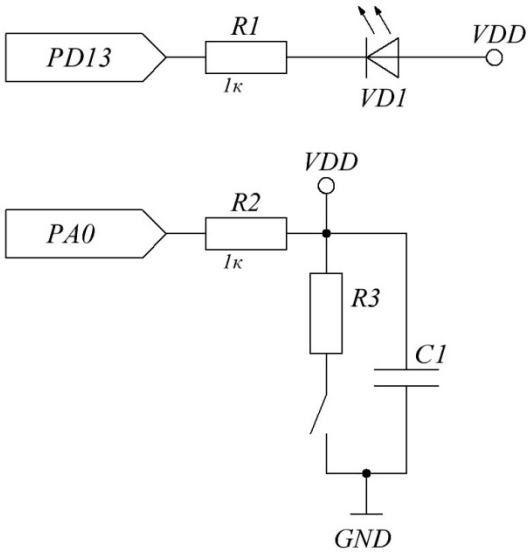

Рис. 4.1. Схема подключения светодиода и кнопки к микроконтроллеру STM32F407

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

GPIO_InitTypeDef GPIO_InitStructure; //объявление структуры настройки
                                     //портов
//основная программа
int main(void)
{
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD|RCC_AHB1Perip
h_GPIOA, ENABLE); //тактирование портов и периферии

//настройка порта D на вывод (светодиоды платы STM32F4Discovery)
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13; //выбор настраиваемых
                                           //выводов
```

```
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT; //выбор режима –
                                                //вывод
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; //тип – push/pull
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz; //определение
                                                //максимальной частоты
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL; //без
                                                //подтягивающего резистора
GPIO_Init(GPIOD, &GPIO_InitStructure); //заполнение объявленной
                                                //структуры

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0; // выбор настраиваемых выводов
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN; // выбор режима – вход
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; // тип – push-pull
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_DOWN; //подтяжка к земле
GPIO_Init(GPIOA, &GPIO_InitStructure); //заполнение объявленной
                                                //структуры
  while(1) //бесконечный цикл
  {
    if (GPIO_ReadInputDataBit(GPIOA,GPIO_Pin_0)==1) //считывание
                                                //состояния нажатия кнопки
    {
    GPIO_SetBits(GPIOD,GPIO_Pin_13); //отключение светодиода
    }
  GPIO_ResetBits(GPIOD,GPIO_Pin_13); //включение светодиода
  } //while
} //main
```

4.1.2. Программа настройки внешнего прерывания

Программа реализует настройку внешнего прерывания на нулевом выводе порта *A*. По нажатию кнопки происходит инверсия состояния светодиода, соединенного с тринадцатым выводом порта *D*. Нажатие кнопки отслеживается с помощью внешнего прерывания.

```
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_exti.h"
#include "misc.h"

GPIO_InitTypeDef GPIO_InitStructure; //объявление структуры настройки
                                                //портов
NVIC_InitTypeDef NVIC_InitStructure; //объявление структуры настройки
                                                //контроллера прерываний
```

```
EXTI_InitTypeDef EXTI_InitStructure; //объявление структуры настройки
                                     //внешних прерываний

//основная программа
int main(void)
{
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD|RCC_AHB1Perip
h_GPIOA, ENABLE); //тактирование портов и периферии

//настройка порта D на вывод (светодиоды платы STM32F4Discovery)
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13; //выбор настраиваемых
                                           //выводов
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT; //выбор режима –
                                              //вывод
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; //тип – push/pull
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz; //определение
                                                   //максимальной частоты
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL; //без подтяжки
GPIO_Init(GPIOD, &GPIO_InitStructure); //заполнение объявленной
                                       //структуры

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0; //выбор настраиваемых выводов
GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_IN; // выбор режима – вход
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; // тип – push-pull
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_DOWN; //подтяжка к земле
GPIO_Init(GPIOA, &GPIO_InitStructure); //заполнение объявленной
                                       //структуры

NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1); //настройка группы
                                                //приоритетов
NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn; //выбор источника
                                                 //прерывания
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1; //основной
                                                          //приоритет
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0; //вложенный
                                                   //приоритет
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //разрешение
                                                //прерывания
NVIC_Init(&NVIC_InitStructure); //заполнение объявленной структуры

EXTI_InitStructure.EXTI_Line=EXTI_Line0; //номер линии внешнего
                                         //прерывания
EXTI_InitStructure.EXTI_LineCmd=ENABLE; //разрешение прерывания на
                                        //линии
```

```
EXTI_InitStructure.EXTI_Mode=EXTI_Mode_Interrupt;
//прерывание/событие
EXTI_InitStructure.EXTI_Trigger=EXTI_Trigger_Rising;  //по переднему
                                                      //фронту
EXTI_Init(&EXTI_InitStructure); //заполнение объявленной структуры
  while(1) //бесконечный цикл
  {
  } //while
} //main
//обработчик внешнего прерывания на нулевой линии
void EXTI0_IRQHandler(void)
{
GPIO_ToggleBits(GPIOD,GPIO_Pin_13); //инверсия состояния светодиодов
EXTI_ClearITPendingBit(EXTI_Line0); //очистка флага прерывания
}
```

4.1.3. Программа настройки USART

Программа реализует настройку *USART*. При нажатии кнопки происходит инверсия состояния светодиода и отправка числа 2 по *USART* в *Terminal*. При отправке из терминала в микроконтроллер числа 1 происходит инверсия состояния светодиода, подключенного к 12 выводу порта *D*. В случае отправки числа 0 происходит инверсия состояния светодиода, подключенного к выводу 13 порта *D*.

```
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_usart.h"
#include "stm32f4xx_exti.h"
#include "misc.h"

GPIO_InitTypeDef GPIO_InitStructure; //объявление структуры настройки
                                     //портов
USART_InitTypeDef USART_InitStructure; //объявление структуры настройки
                                       //USART
NVIC_InitTypeDef NVIC_InitStructure; //объявление структуры настройки
                                     //контроллера прерываний
EXTI_InitTypeDef EXTI_InitStructure; //объявление структуры настройки
                                     //внешних прерываний

int bufer=0; //объявление переменной для приемника USART

//основная программа
```

```
int main(void)
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA|RCC_AHB1Perip
h_GPIOD, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);
RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART3,        ENABLE);
//тактирование портов и периферии

//настройка порта D на вывод (светодиоды платы STM32F4Discovery)
GPIO_InitStructure.GPIO_Pin      =      GPIO_Pin_12      |      GPIO_Pin_13|
GPIO_Pin_14| GPIO_Pin_15; //выбор настраиваемых выводов
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT; //выбор режима –
                                                    //вывод
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; //тип – push/pull
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz; //определение
                                                    //максимальной частоты
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL; //без
                                                    //подтягивающего резистора
GPIO_Init(GPIOD, &GPIO_InitStructure); //заполнение объявленной
                                                    //структуры

//настройка выводов USART (PD8-Tx,PD9-Rx)
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF; //выбор режима –
                                                    //альтернативная функция
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9|GPIO_Pin_8; //выбор
                                                    //настраиваемых выводов
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_50MHz; //определение
                                                    //максимальной частоты
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; //тип – push/pull
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_UP; //с подтягивающим рези-
стором к питанию
GPIO_Init(GPIOD, &GPIO_InitStructure); //заполнение объявленной
                                                    //структуры

GPIO_PinAFConfig(GPIOD,    GPIO_PinSource9,    GPIO_AF_USART3);
//настройка альтернативной функции 9 вывода порта D
GPIO_PinAFConfig(GPIOD,    GPIO_PinSource8,    GPIO_AF_USART3);
//настройка альтернативной функции 8 вывода порта D

NVIC_EnableIRQ(USART3_IRQn); //разрешение прерываний от USART

USART_InitStructure.USART_Mode      =      USART_Mode_Rx      |
USART_Mode_Tx; //использование приема и передачи
USART_InitStructure.USART_BaudRate = 115200; //скорость
```

```
USART_InitStructure.USART_Parity = USART_Parity_No; //без проверки
                                                    //четности
USART_InitStructure.USART_StopBits = USART_StopBits_1; //количество
                                                    //стоп-бит – 1
USART_InitStructure.USART_WordLength = USART_WordLength_8b;
//длина передаваемого слова – 8 бит
USART_Init(USART3, &USART_InitStructure); //заполнение объявленной
                                          //структуры

GPIO_InitStructure.GPIO_Pin  = GPIO_Pin_0; //выбор настраиваемых
                                          //выводов
GPIO_InitStructure.GPIO_Mode  = GPIO_Mode_IN; //выбор режима – вход
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; //тип – push/pull
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_DOWN;  //с подтягивающим
резистором к земле
GPIO_Init(GPIOA, &GPIO_InitStructure); //заполнение объявленной
                                       //структуры

NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1); //настройка группы
                                                //приоритетов
NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn; //выбор источника
                                                 //прерывания
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1; //основной
                                                          //приоритет
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0; //вложенный
                                                   //приоритет
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //разрешение
                                                //прерывания
NVIC_Init(&NVIC_InitStructure); //заполнение объявленной структуры

EXTI_InitStructure.EXTI_Line=EXTI_Line0; //номер линии
EXTI_InitStructure.EXTI_LineCmd=ENABLE; //разрешение прерывания на
                                        //линии
EXTI_InitStructure.EXTI_Mode=EXTI_Mode_Interrupt;
//прерывание/событие
EXTI_InitStructure.EXTI_Trigger=EXTI_Trigger_Rising; //по переднему
                                                     //фронту
EXTI_Init(&EXTI_InitStructure); //заполнение объявленной структуры

USART_ITConfig(USART3, USART_IT_RXNE, ENABLE); //разрешение
                                               //прерывания приемника
USART_Cmd(USART3, ENABLE); //включение USART3
```

```
  while(1)//бесконечный цикл
  {
  } //while
} //main
//обработчик внешнего прерывания
void EXTI0_IRQHandler(void)
{
GPIO_ToggleBits(GPIOD,GPIO_Pin_14|GPIO_Pin_15); //инверсия состояния
                                                //светодиодов
USART_SendData (USART3,'2'); //отправка числа 2 по USART
EXTI_ClearITPendingBit(EXTI_Line0); //очистка флага прерывания
}
//обработчик прерывания USART
void USART3_IRQHandler(void)
{
// проверка флага приемника
  if( USART_GetITStatus(USART3, USART_IT_RXNE) )
  {
  bufer = USART_ReceiveData(USART3); //считывание данных с USART3
    if (bufer=='1')
    {
    GPIO_ToggleBits(GPIOD,GPIO_Pin_12); //инверсия состояния светодиодов
    }
    if (bufer=='0')
    {
    GPIO_ToggleBits(GPIOD, GPIO_Pin_13);//инверсия состояния светодиодов
    }
  }
}
```

4.1.4. Программа настройки таймера TIM8

Программа реализует настройку таймера *TIM*8. Таймер считает в сторону увеличения от 0 до 0xFFFF; по переполнению таймера генерируется прерывание, в обработчике которого осуществляется инверсия состояния светодиодов.

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

```
GPIO_InitTypeDef GPIO_InitStructure; //объявление структуры настройки
                                     //портов
TIM_TimeBaseInitTypeDef TIM_InitStructure; //объявление структуры
                                           //настройки таймера
NVIC_InitTypeDef NVIC_InitStructure; //объявление структуры настройки
                                     //контроллера прерываний

//основная программа
int main(void)
{
RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM8, ENABLE);
//тактирование таймера TIM8
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
//тактирование порта D

//настройка порта D на вывод (светодиоды платы STM32F4Discovery)
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13|
GPIO_Pin_14| GPIO_Pin_15; //выбор настраиваемых выводов
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT; //выбор режима –
                                              //вывод
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; //тип – push/pull
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz; //определение
                                                   //максимальной частоты
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL; //без
                                                 //подтягивающего резистора
GPIO_Init(GPIOD, &GPIO_InitStructure); //заполнение объявленной
                                       //структуры настройки порта D
                                       //вышеперечисленными
                                       //параметрами

//настройка таймера
TIM_InitStructure.TIM_RepetitionCounter = 0; //значение счетчика повторений
TIM_InitStructure.TIM_Prescaler = 200; //значение предварительного делителя
TIM_InitStructure.TIM_Period = 0xFFFF; //величина периода счетчика
TIM_InitStructure.TIM_ClockDivision = 0; //деление частоты
TIM_InitStructure.TIM_CounterMode = TIM_CounterMode_Up; //режим
                                                        //счета вверх
TIM_TimeBaseInit(TIM8, &TIM_InitStructure); //заполнение объявленной
                                            //структуры

TIM_ITConfig(TIM8,TIM_IT_Update,ENABLE); //разрешение прерываний
                                         //по переполнению таймера
```

```
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);
NVIC_InitStructure.NVIC_IRQChannel = TIM8_UP_TIM13_IRQn ; //выбор
                                                    //источника прерывания
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1; //основной
                                                    //приоритет
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0; //вложенный
                                                    //приоритет
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //разрешение
                                                    //прерывания
NVIC_Init(&NVIC_InitStructure); //заполнение объявленной структуры

TIM_Cmd(TIM8, ENABLE); //включение таймера 8

  while(1)
  {
  } //while
} //main

//обработчик прерывания таймера 8
void TIM8_UP_TIM13_IRQHandler(void)
{
GPIO_ToggleBits(GPIOD,GPIO_Pin_12|                GPIO_Pin_13|
GPIO_Pin_14|GPIO_Pin_15); //инверсия состояния светодиодов
TIM_ClearITPendingBit(TIM8, TIM_IT_Update); //очистка флага прерывания
}
```

4.1.5. Программа настройки таймера для генерации ШИМ

Программа реализует генерацию ШИМ на первом канале таймера TIM1 (8 вывод порта *A*).

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

GPIO_InitTypeDef GPIO_InitStructure; //объявление структуры
                                                    //настройки портов
TIM_TimeBaseInitTypeDef TIM_TimeBaseStructure; //объявление структуры
                                                    //настройки таймера
TIM_OCInitTypeDef TIM_OCInitStructure; //объявление структуры настройки
                                                    //выходных каналов таймера
//основная программа
```

```
int main(void)
{
RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE); //тактирование таймера TIM1
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA|RCC_AHB1Periph_GPIOE, ENABLE); //тактирование портов

GPIO_InitStructure.GPIO_Pin=GPIO_Pin_8; //выбор настраиваемых выводов
GPIO_InitStructure.GPIO_Mode=GPIO_Mode_AF; //выбор режима –
                                           //альтернативная функция
GPIO_InitStructure.GPIO_OType=GPIO_OType_PP; //тип – push/pull
GPIO_InitStructure.GPIO_Speed=GPIO_Speed_50MHz; //определение
                                                //максимальной частоты
GPIO_InitStructure.GPIO_PuPd=GPIO_PuPd_UP; //подтяжка к питанию
GPIO_Init(GPIOA,&GPIO_InitStructure); //заполнение объявленной структуры

GPIO_PinAFConfig(GPIOA,GPIO_PinSource8,GPIO_AF_TIM1);
//настройка альтернативной функции 8 вывода порта A

TIM_TimeBaseStructure.TIM_Period = 1000; //значение периода
TIM_TimeBaseStructure.TIM_Prescaler = 10; //величина предварительного
                                          //делителя
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
//режим счета вверх
TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure); //заполнение
                                                //объявленной структуры

TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1; //выбор
                                                  //режима – ШИМ1
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
//включение выхода
TIM_OCInitStructure.TIM_Pulse = 500; //время импульса
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_High;
//полярность выхода
TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Disable;
//инверсный выход выключен
TIM_OCInitStructure.TIM_OCIdleState = TIM_OCIdleState_Reset; //низкий
                                                             //неактивный уровень
TIM_OC1Init(TIM1, &TIM_OCInitStructure); //заполнение объявленной
                                         //структуры

TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Enable); //разрешение
                                                  //предварительной загрузки CCR1
```

```
TIM_ARRPreloadConfig(TIM1,ENABLE); //разрешение предварительной
                                   //загрузки ARR

TIM_CtrlPWMOutputs(TIM1, ENABLE); //включение выходов ШИМ, только
                                  //для таймеров TIM8 и TIM1

TIM_Cmd(TIM1,ENABLE); //запуск таймера

  while(1)
  {

  }//while
}//main
```

4.1.6. Программа настройки АЦП

Программа реализует мерцание светодиодами с частотой, зависящей от результата преобразования АЦП2. Преобразование на первом канале АЦП2 запускается программно и однократно по нажатию кнопки.

```
#include "stm32f4xx.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_rcc.h"
#include "misc.h"
#include "stm32f4xx_adc.h"
#include "stm32f4xx_exti.h"

int i,ADC_Data; //объявление переменных

GPIO_InitTypeDef GPIO_InitStructure; //объявление структуры настройки
                                     //портов
NVIC_InitTypeDef NVIC_InitStructure; //объявление структуры настройки
                                     //контроллера прерываний
EXTI_InitTypeDef EXTI_InitStructure;  //объявление структуры настройки
                                     //внешних прерываний
ADC_InitTypeDef ADC_InitStructure; //объявление структуры настройки АЦП
ADC_CommonInitTypeDef ADC_CommonInitStructure; //объявление
                                     //структуры общей настройки АЦП

//объявление подпрограммы получения результата преобразования АЦП
void GetDataFromTheChannel(ADC_TypeDef* ADCx, uint8_t
ADC_Channel, uint8_t Rank, uint8_t ADC_SampleTime);
//основная программа
```

```
int main(void)
{
ADC_Data = 0xFF00; //начальная частота мерцания светодиодов

RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG|RCC_APB2Periph_ADC2, ENABLE); //включение тактирования внешних прерываний
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD|RCC_AHB1Periph_GPIOA, ENABLE); //тактирование портов

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12| GPIO_Pin_13| GPIO_Pin_14| GPIO_Pin_15; //выбор настраиваемых выводов
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT; //режим – выход
GPIO_Init(GPIOD, &GPIO_InitStructure); //заполнение объявленной
                                       //структуры

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_0; //выбор настраиваемых выводов
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_IN; //режим – вход
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; //тип – push-pull
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_DOWN; //с подтягивающим
                                               //резистором к земле
GPIO_Init(GPIOA, &GPIO_InitStructure); //заполнение объявленной
                                       //структуры

GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1; //выбор настраиваемых выводов
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN; //режим – аналоговый
                                             //(для АЦП)

GPIO_Init(GPIOA, &GPIO_InitStructure); //заполнение объявленной
                                       //структуры

NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1); //настройка группы
                                                //приоритетов
NVIC_InitStructure.NVIC_IRQChannel = EXTI0_IRQn; //выбор источника
                                                 //прерывания
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1; //основной
                                                          //приоритет
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0; //вложенный
                                                   //приоритет
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE; //разрешение
                                                //прерывания
NVIC_Init(&NVIC_InitStructure); //заполнение объявленной структуры
```

```
EXTI_InitStructure.EXTI_Line=EXTI_Line0; //номер линии внешнего
                                          //прерывания
EXTI_InitStructure.EXTI_LineCmd=ENABLE; //разрешение прерывания на
                                        //линии
EXTI_InitStructure.EXTI_Mode=EXTI_Mode_Interrupt;
//прерывание/событие
EXTI_InitStructure.EXTI_Trigger=EXTI_Trigger_Rising; //прерывание по
                                                     //переднему фронту
EXTI_Init(&EXTI_InitStructure); //заполнение объявленной структуры

ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
//выбор независимого режима преобразований
ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div2;
//установка предварительного делителя
ADC_CommonInitStructure.ADC_DMAAccessMode =
ADC_DMAAccessMode_Disabled; //без прямого доступа к памяти
ADC_CommonInitStructure.ADC_TwoSamplingDelay =
ADC_TwoSamplingDelay_5Cycles; //задержка между двумя выборками
ADC_CommonInit(&ADC_CommonInitStructure); //заполнение объявленной
                                          //структуры

ADC_InitStructure.ADC_ContinuousConvMode = DISABLE; //запрещение
                                        //непрерывного режима преобразований
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Left; //выравнивание
                                                      //по левому краю
ADC_InitStructure.ADC_NbrOfConversion = 1; //количество преобразований
ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b; //настройка
                                                       //разрядности АЦП
ADC_InitStructure.ADC_ScanConvMode = DISABLE; //запрещение режима
                                              //сканирования
ADC_Init(ADC2, &ADC_InitStructure); //заполнение объявленной структуры

ADC_Cmd(ADC2 , ENABLE); //запуск АЦП

  while(1) //бесконечный цикл
  {
  for (i=0;i<(2*ADC_Data);i++);
  GPIO_ToggleBits(GPIOD,GPIO_Pin_12| GPIO_Pin_13|
  GPIO_Pin_14|GPIO_Pin_15); //инверсия состояния светодиодов
  } //while
} //main

//обработчик внешнего прерывания
```

```
void EXTI0_IRQHandler(void)
{
GetDataFromTheChannel(ADC2, ADC_Channel_1, 1,
ADC_SampleTime_3Cycles); //запуск выбранного канала
EXTI_ClearITPendingBit(EXTI_Line0); //очистка флага прерывания
}

// подпрограмма получения результата преобразования АЦП
void GetDataFromTheChannel(ADC_TypeDef* ADCx, uint8_t
ADC_Channel, uint8_t Rank, uint8_t ADC_SampleTime)
{
ADC_TypeDef* ADC_number = ADCx;
uint8_t Channel_number = ADC_Channel;
uint8_t Rank1 = Rank;
uint8_t Sample_Time = ADC_SampleTime;

ADC_RegularChannelConfig(ADC_number,Channel_number,Rank1,Sample
_Time); //выбор и настройка канала
ADC_SoftwareStartConv (ADC_number); //программный запуск АЦП
while(ADC_GetFlagStatus(ADCx , ADC_FLAG_EOC) == RESET);
//ожидание окончания преобразования
ADC_Data= ADC_GetConversionValue(ADC2); //считывание результата
                                        //преобразования
}
```

4.1.7. Программа запуска преобразований АЦП с помощью таймера

Программа реализует мерцание светодиодами с частотой, зависящей от результата преобразований АЦП. АЦП запускается внешним источником, в качестве которого выступает запускающий выход таймера *TIM*8.

```
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_adc.h"
#include "stm32f4xx_tim.h"

int data=0xFFFF,i=0; //объявление переменных

ADC_InitTypeDef ADC_InitStructure; //объявление структуры настройки АЦП
ADC_CommonInitTypeDef ADC_CommonInitStructure; //объявление
                                        //структуры общей настройки АЦП
```

```
GPIO_InitTypeDef GPIO_InitStructure; //объявление структуры
                                     //настройки портов
TIM_TimeBaseInitTypeDef TIM_InitStructure; //объявление структуры
                                           //настройки таймера

//основная программа
int main(void)
{
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA|RCC_AHB1Perip
h_GPIOD, ENABLE); //тактирование портов
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1|RCC_APB2Periph_
TIM8, ENABLE); //тактирование таймера TIM8 и АЦП1

//настройка PA1 на аналоговый вход (вход АЦП)
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1; //выбор настраиваемых выводов
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN; //режим – аналоговый
                                             //(для АЦП)
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL; //без
                                                 //подтягивающего резистора
GPIO_Init(GPIOA, &GPIO_InitStructure); //заполнение объявленной
                                       //структуры

//настройка порта D на вывод (светодиоды платы STM32F4Discovery)
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13|
GPIO_Pin_14| GPIO_Pin_15; //выбор настраиваемых выводов
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT; //режим – вывод
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; //тип – push-pull
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz; //выбор
                                                   //максимальной частоты
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL; // без
                                                 //подтягивающего резистора

GPIO_Init(GPIOD, &GPIO_InitStructure); //заполнение объявленной
                                       //структуры

//настройка таймера
TIM_InitStructure.TIM_RepetitionCounter = 0; //значение счетчика повторений
TIM_InitStructure.TIM_Prescaler = 267; //значение предварительного делителя
TIM_InitStructure.TIM_Period = 1; //величина периода
TIM_InitStructure.TIM_ClockDivision = 0; //деление частоты
TIM_InitStructure.TIM_CounterMode = TIM_CounterMode_Up; //режим
                                                        //счета вверх
TIM_TimeBaseInit(TIM8, &TIM_InitStructure); //заполнение объявленной
                                            //структуры
```

```
TIM_SelectOutputTrigger(TIM8, TIM_TRGOSource_Update); //выбор
                                                      //выхода запуска

//настройка АЦП
ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
//режим независимого преобразования
ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div2;
//предварительный делитель
ADC_CommonInitStructure.ADC_DMAAccessMode =
ADC_DMAAccessMode_Disabled; //без прямого доступа к памяти
ADC_CommonInitStructure.ADC_TwoSamplingDelay =
ADC_TwoSamplingDelay_5Cycles; //задержка между двумя выборками
ADC_CommonInit(&ADC_CommonInitStructure); //заполнение объявленной
                                          //структуры

ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b; //12-битный
                                                       //результат
ADC_InitStructure.ADC_ScanConvMode = DISABLE; //без сканирования
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE; //непрерывное
                                                    //преобразование запрещено
ADC_InitStructure.ADC_ExternalTrigConvEdge =
ADC_ExternalTrigConvEdge_Rising; //запуск передним фронтом сигнала
ADC_InitStructure.ADC_ExternalTrigConv =
ADC_ExternalTrigConv_T8_TRGO; //выбор запускающего сигнала
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
//выравнивание по правому краю
ADC_InitStructure.ADC_NbrOfConversion = 1; //одно преобразование
ADC_Init(ADC1, &ADC_InitStructure); //заполнение объявленной структуры

//настройка регулярного канала АЦП1, канал 1, PA1
ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1,
ADC_SampleTime_3Cycles);
ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE); //включение прерываний
                                        //по завершению
                                        //преобразования
//разрешение прерывания от АЦП
NVIC_EnableIRQ (ADC_IRQn);
TIM_Cmd(TIM8, ENABLE); //включение таймера

ADC_Cmd(ADC1, ENABLE); //включение АЦП 1

  while(1)
```

```
  {
  for (i=0;i<(30*data);i++);
  GPIO_ToggleBits(GPIOD,GPIO_Pin_12|                GPIO_Pin_13|
  GPIO_Pin_14|GPIO_Pin_15); //инверсия состояния светодиодов
  } //while(1)
} //main

//обработчик прерывания АЦП
void ADC_IRQHandler(void)
{
data=ADC_GetConversionValue(ADC1); //получение результата
                                   //преобразования
ADC_ClearFlag(ADC1, ADC_FLAG_EOC); //сброс флага
}
```

4.1.8. Программа настройки ЦАП

Программа реализует вывод импульсов треугольной формы с помощью ЦАП.

```
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_dac.h"

int i=0; //объявление переменных

GPIO_InitTypeDef GPIO_InitStructure; //объявление структуры
                                     //настройки портов
DAC_InitTypeDef DAC_InitStructure; //объявление структуры настройки ЦАП

//основная программа
int main(void)
{
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA|RCC_AHB1Perip
h_GPIOD, ENABLE); //тактирование портов
RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC,        ENABLE);
//тактирование ЦАП

//настраиваем выход ЦАП
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN; //режим – аналоговый
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP; //тип – push-pull
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4; //выбираем настраиваемые
                                          //выводы
```

```
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL; //без подтяжки
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz; //выбор
                                                    //максимальной частоты
GPIO_Init(GPIOA, &GPIO_InitStructure); //заполнение объявленной
                                        //структуры

DAC_InitStructure.DAC_Trigger = DAC_Trigger_Software; //программный
                                                      //запуск
DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_None;
//отсутствие генерации сигнала по умолчанию
DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
//включение выходного буфера
DAC_Init(DAC_Channel_1, &DAC_InitStructure); //заполнение объявленной
                                              //структуры

DAC_Cmd(DAC_Channel_1, ENABLE); //включение ЦАП

  while(1) //бесконечный цикл
  {
    for (i=0;i<0xFFF;i++)
    {
    DAC_SetChannel1Data(DAC_Align_12b_R, i); //заполнение буфера
    DAC_SoftwareTriggerCmd(DAC_Channel_1, ENABLE); //включение
                                                   //ЦАП программно
    }
    for (i=0xFFF;i>0;i--)
    {
    DAC_SetChannel1Data(DAC_Align_12b_R, i); //заполнение буфера
    DAC_SoftwareTriggerCmd(DAC_Channel_1, ENABLE); //включение
                                                   //ЦАП программно
    }
  } //while(1)
} //main
```

Приложение 1
Программа вывода символов на ЖК-дисплей WH1602 на микроконтроллере STM32F100RB

Программа реализует вывод строки «Томский Политех» на ЖК-дисплей. Данная программа работает с использованием библиотеки *LCD.h* Схема подключения дисплея указана на рис. П.1 [8].

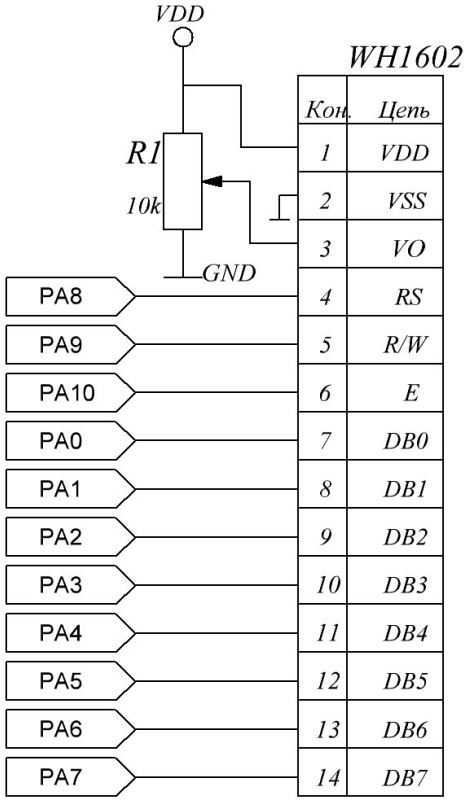

Рис. П.1. Схема подключения ЖК-дисплея WH1602 к микроконтроллеру

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

GPIO_InitTypeDef GPIO_InitStruct; //объявление структуры настройки портов

static __IO uint32_t TimingDelay; //переменная для функции задержки

//основная программа
int main(void)
{
SysTick_Config(240); //настройка системного таймера
```

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB|RCC_APB2Periph
_GPIOA, ENABLE);

GPIO_InitStruct.GPIO_Pin=GPIO_Pin_All; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz; //определение
                                                //максимальной частоты
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP; //режим – вывод,
                                              //тип – push-pull
GPIO_Init(GPIOA, &GPIO_InitStruct); //заполнение объявленной структуры

Delay_x10us(4000); //задержка перед началом инициализации дисплея
Display_init(); //вызов подпрограммы инициализации дисплея
vivod_na_display(); //вызов подпрограммы вывода информации на дисплей

  while(1) //бесконечный цикл
  {
  } //while (1)
} //main

//подпрограмма вывода информации на дисплей
void vivod_na_display (void)
{
vivod_simvola(84);      //вывод буквы Т
vivod_simvola(111);     //вывод буквы о
vivod_simvola(188);     //вывод буквы м
vivod_simvola(99);      //вывод буквы с
vivod_simvola(107);     //вывод буквы к
vivod_simvola(184);     //вывод буквы и
vivod_simvola(185);     //вывод буквы й
vivod_simvola(32);      //вывод пробела
vivod_simvola(168);     //вывод буквы П
vivod_simvola(111);     //вывод буквы о
vivod_simvola(187);     //вывод буквы л
vivod_simvola(184);     //вывод буквы и
vivod_simvola(191);     //вывод буквы т
vivod_simvola(101);     //вывод буквы е
vivod_simvola(120);     //вывод буквы х
}

//функция временной задержки
void Delay_x10us(__IO uint32_t nTime)
```

```
{
TimingDelay = nTime;
while(TimingDelay != 0);
}

//обработчик прерывания системного таймера
void SysTick_Handler(void)
{
TimingDelay_Decrement(); //вызов подпрограммы декремента переменной для
                         //задержки
}

//подпрограмма декремента переменной для задержки
void TimingDelay_Decrement(void)
{
  if (TimingDelay != 0x00)
  {
  TimingDelay--; //декремент переменной для задержки
  }
}
```

Содержание подключаемой библиотеки LCD.h

```
//подпрограмма инициализации дисплея
void Display_init(void)
{
// Function set:
GPIOA->ODR = 0x003C;
Delay_x10us(10);
GPIO_SetBits(GPIOA, GPIO_Pin_10); //E=1
Delay_x10us(10);
GPIO_ResetBits(GPIOA, GPIO_Pin_10); //E=0
Delay_x10us(10);

// Function set:
GPIOA->ODR = 0x003C;
Delay_x10us(10);
GPIO_SetBits(GPIOA, GPIO_Pin_10); //E=1
Delay_x10us(10);
GPIO_ResetBits(GPIOA, GPIO_Pin_10); //E=0
Delay_x10us(10);
```

```
// Display on/off control:
GPIOA->ODR = 0x000F; //экран включен
Delay_x10us(10);
GPIO_SetBits(GPIOA, GPIO_Pin_10); //E=1
Delay_x10us(10);
GPIO_ResetBits(GPIOA, GPIO_Pin_10); //E=0
Delay_x10us(10);

//Display Clear :
GPIOA->ODR = 0x0001; //очистка экрана
Delay_x10us(10);
GPIO_SetBits(GPIOA, GPIO_Pin_10); //E=1
Delay_x10us(200);
GPIO_ResetBits(GPIOA, GPIO_Pin_10); //E=0
Delay_x10us(10);

//Entry mode set:
GPIOA->ODR = 0x0006; //сдвиг курсора после вывода символа
Delay_x10us(10);
GPIO_SetBits(GPIOA, GPIO_Pin_10); //E=1
Delay_x10us(10);
GPIO_ResetBits(GPIOA, GPIO_Pin_10); //E=0
Delay_x10us(2000);
}

//подпрограмма вывода символа с использованием системной задержки
void vivod_simvola (u32 D)
{
GPIOA->ODR = D ;
GPIO_SetBits(GPIOA, GPIO_Pin_8); // RS=1
Delay_x10us(10);
GPIO_SetBits(GPIOA, GPIO_Pin_10); //E=1
Delay_x10us(10);
GPIO_ResetBits(GPIOA, GPIO_Pin_10); //E=0
Delay_x10us(10);
}

//подпрограмма вывода символа без использования системной задержки
void vivod_simvola2 (u32 D)
{
u8 ink;
```

```
GPIOA->ODR = D ;
GPIO_SetBits(GPIOA, GPIO_Pin_8); // RS=1
for (ink=0;ink<0xFF;ink++);
GPIO_SetBits(GPIOA, GPIO_Pin_10); //E=1
for (ink=0;ink<0xFF;ink++);
GPIO_ResetBits(GPIOA, GPIO_Pin_10); //E=0
for (ink=0;ink<0xFF;ink++);
}

//подпрограмма возвращения курсора в начальное положение
void go_home (void)
{
GPIOA->ODR = 0x0002;
Delay_x10us(10);
GPIO_SetBits(GPIOA, GPIO_Pin_10); //E=1
Delay_x10us(200);
GPIO_ResetBits(GPIOA, GPIO_Pin_10); //E=0
Delay_x10us(10);
}

//подпрограмма очистки дисплея
void Display_clear (void)
{
GPIOA->ODR = 0x0001;
Delay_x10us(10);
GPIO_SetBits(GPIOA, GPIO_Pin_10); //E=1
Delay_x10us(200);
GPIO_ResetBits(GPIOA, GPIO_Pin_10); //E=0
Delay_x10us(10);
}
```

Приложение 2
Программа управления шаговым двигателем на микроконтроллере STM8S

Программа реализует управление шаговым двигателем. При нажатии одной из кнопок шаговый двигатель совершает вращение по часовой стрелке, при нажатии второй кнопки вращение осуществляется в противоположную сторону. Прерывания от кнопок организованы не по фронту, а по уровню сигнала на выводе порта, поэтому при нажатии одной из кнопок вращение будет осуществляться до того момента, пока кнопка не будет отпущена. Схема управления шаговым двигателем представлена на рис. П.2.

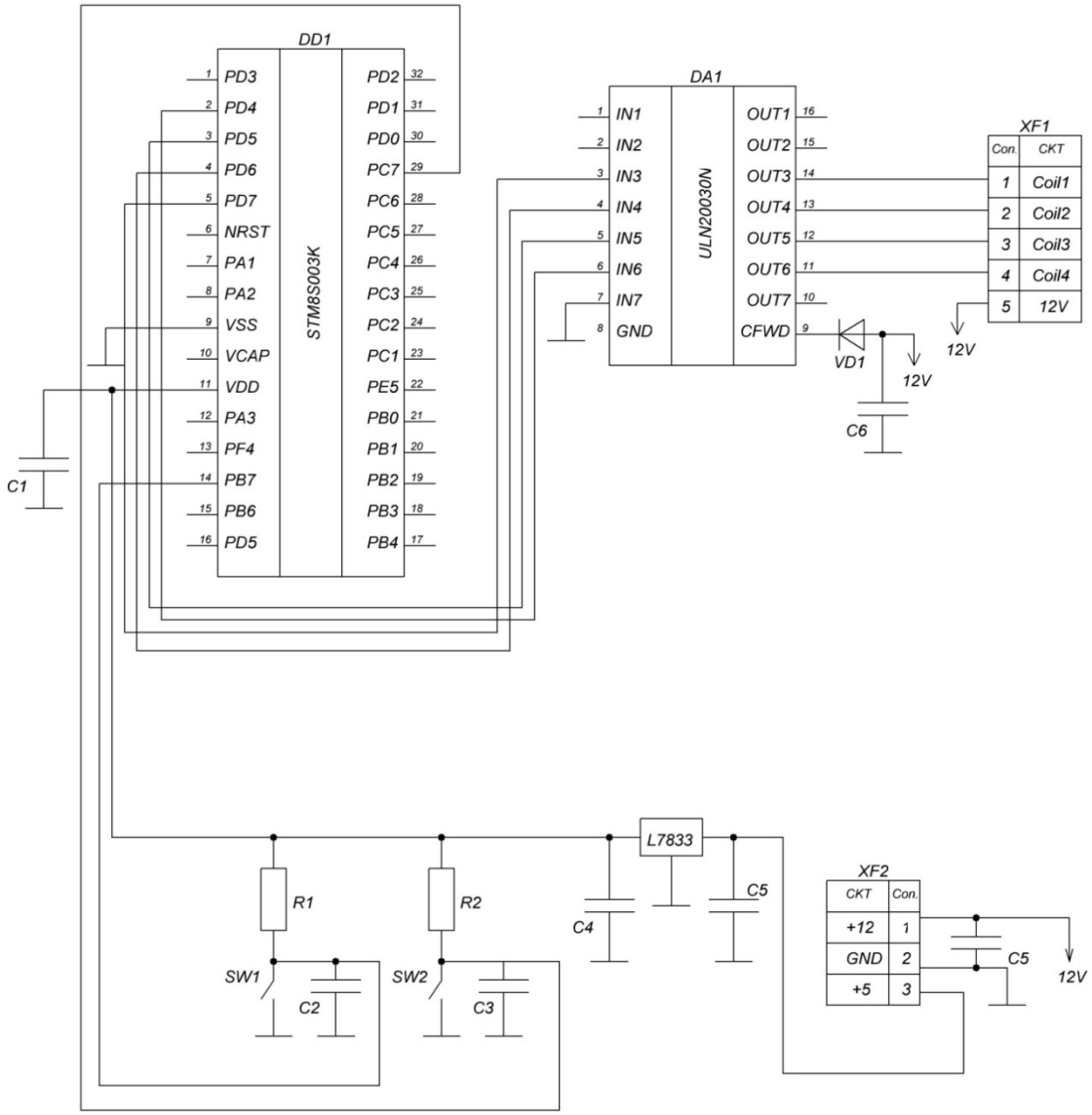

Рис. П.2. Схема подключения шагового двигателя к микроконтроллеру

```
#include "iostm8.h"

void portD_init(void);
void interrupt_init(void);
void portB_init(void);
#pragma vector=0x06
__interrupt void EXTI_PB7(void);
#pragma vector=0x07
__interrupt void EXTI_PC7(void);

int m[] = {160, 96, 80, 144}; //массив для управления шаговым двигателем
int s=0,k,l;

int main( void )
{
asm("sim"); //глобальное запрещение прерываний
ports_init();
interrupt_init();
  for (;;)
  {
  }
}

void ports_init(void)
{
PD_DDR_bit.DDR7=1;
PD_CR1_bit.C17 = 1;
PD_CR2_bit.C27 = 0;

PD_DDR_bit.DDR6=1;
PD_CR1_bit.C16 = 1;
PD_CR2_bit.C26 = 0;

PD_DDR_bit.DDR5=1;
PD_CR1_bit.C15 = 1;
PD_CR2_bit.C25 = 0;

PD_DDR_bit.DDR4=1;
PD_CR1_bit.C14 = 1;
PD_CR2_bit.C24 = 0;

PB_DDR_bit.DDR7=0;
PB_CR1_bit.C17 = 0;
PB_CR2_bit.C27 = 1;

PC_DDR_bit.DDR7=0;
PC_CR1_bit.C17 = 0;
PC_CR2_bit.C27 = 1;
```

```
}

void interrupt_init(void)
{
 asm("rim"); //глобальное разрешение прерываний
EXTI_CR1 = 0x28;
}

__interrupt void EXTI_PB7(void) //обработчик прерывания кнопки, задающей
                                //вращение против часовой стрелки
{
asm("sim");
PD_ODR = m[s];
s++;
if (s>3)
{
s=0;
}
for (k=0;k<255; k++)
{
for (l=0;l<255; l++)
{
}
}
asm("rim");
}

__interrupt void EXTI_PC7(void) //обработчик прерывания кнопки, задающей
                                //вращение по часовой стрелке
{
asm("sim");
if (s==0)
{
s=3;
}
s--;
PD_ODR = m[s];
for (k=0;k<255; k++)
{
for (l=0;l<255; l++)
{
}
}
asm("rim");
}
}
```

Приложение 3
Программа настройки цифрового датчика температуры D18B20 и вывод температуры на ЖК-дисплей WH1602

Программа реализует процесс измерения температуры и вывод результата на ЖК-дисплей. Данная программа работает с использованием библиотеки *LCD.h.* Схема подключения датчика температуры и ЖК-дисплея показана на рис. П.3 [8, 9].

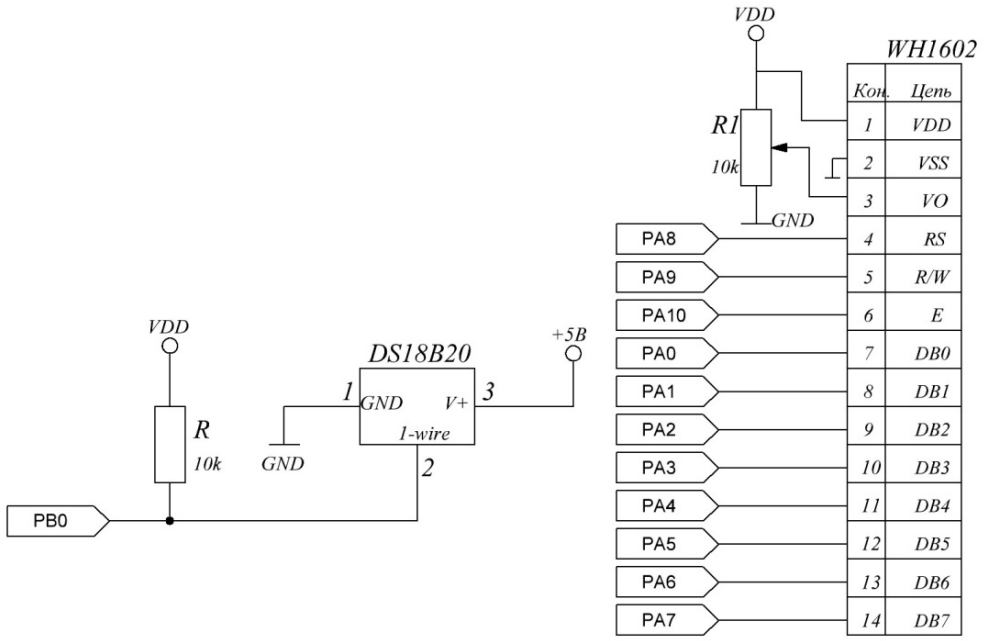

Рис. П.3. Схема подключения цифрового датчика температуры DS18B20 и ЖК-дисплея WH1602 к микроконтроллеру

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

GPIO_InitTypeDef GPIO_InitStruct; //объявление структуры настройки порта

static __IO uint32_t TimingDelay; //объявление переменной для системной
                                  //задержки
u8 t1, t2, res, z; //объявление переменных

//основная программа
int main(void)
{
SysTick_Config(240); //настройка системного таймера
```

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOB|RCC_APB2Periph
_GPIOA, ENABLE); //тактирование портов

GPIO_InitStruct.GPIO_Pin = GPIO_Pin_0; //выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz; //определение
                                                //максимальной частоты
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_OD; //режим – вывод с
                                                //открытым стоком
GPIO_Init(GPIOB, &GPIO_InitStruct); //заполнение объявленной структуры

GPIO_InitStruct.GPIO_Pin=GPIO_Pin_All; // выбор настраиваемых выводов
GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz; //определение
                                                //максимальной частоты
GPIO_InitStruct.GPIO_Mode = GPIO_Mode_Out_PP; //режим – вывод,
                                                //тип – push-pull
GPIO_Init(GPIOA, &GPIO_InitStruct); //заполнение объявленной структуры

Delay_x10us(4000); //задержка перед началом инициализации дисплея
Display_init(); //вызов подпрограммы инициализации дисплея

  while(1) //бесконечный цикл
  {
  W1_Reset(); //вызов подпрограммы сброса датчика
  W1_SendByte(0xCC); //вызов подпрограммы отправки команды датчику
  W1_SendByte(0x44); //вызов подпрограммы отправки команды датчику
  Delay_1s(); //задержка 1 с для отклика датчика
  W1_Reset(); //вызов подпрограммы сброса датчика
  W1_SendByte(0xCC); //вызов подпрограммы отправки команды датчику
  W1_SendByte(0xBE); //вызов подпрограммы отправки команды датчику
  W1_RecByte(); //вызов подпрограммы приема данных от датчика
  t2 = res;
  W1_RecByte(); //вызов подпрограммы приема данных от датчика
  t1 = res;
  vivod_na_display(); //вызов подпрограммы вывода температуры
  } //while (1)
} //main

//подпрограмма сброса датчика
void W1_Reset(void)
{
u8 t;
```

```
GPIO_ResetBits(GPIOB, GPIO_Pin_0);
Delay_x10us(50);
GPIO_SetBits(GPIOB, GPIO_Pin_0);
Delay_65us();
t = GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_0);
Delay_x10us(50);
}

//подпрограмма приема данных от датчика
void W1_RecByte(void)
{
u8 mask= 0x01, i,t=0;
res = 0x00;
  for (i = 0; i <= 7; i++)
  {
  GPIO_ResetBits(GPIOB, GPIO_Pin_0);
  Delay_2us();

  GPIO_SetBits(GPIOB, GPIO_Pin_0);
  Delay_8us();

  t = GPIO_ReadInputDataBit(GPIOB, GPIO_Pin_0);
  Delay_70us();

    if (t!= 0)
    {
   res |= mask;
    }
  mask <<= 1;
  }
}

//подпрограмма отправки команды датчику
void W1_SendByte(u8 byte)
{
u8 mask, i;

mask = 0x01;
  for (i = 0; i <= 7; i++)
  {
    if ((byte & mask) == 0)
```

```
    {
    GPIO_ResetBits(GPIOB, GPIO_Pin_0);
    Delay_x10us(6);
    GPIO_SetBits(GPIOB, GPIO_Pin_0);
    Delay_2us();
    mask <<= 1;
    }
    else
    {
    GPIO_ResetBits(GPIOB, GPIO_Pin_0);
    Delay_2us();
    GPIO_SetBits(GPIOB, GPIO_Pin_0);
    Delay_x10us(6);
    mask <<= 1;
    }
  }
}

//подпрограмма вывода температуры
void vivod_na_display (void)
{
u8 t3, d; //локальное объявление переменных
//определение знака температуры:
  if (((t1)&0xF8) == 0xF8)
  {
  z = 0x002D; //знак -
  }
  else
  {
  z = 0x002B; //знак +
  }
go_home (); //вызов подпрограммы возвращения курсора в начальное положение
Display_clear(); //вызов подпрограммы очистки дисплея
go_home (); //вызов подпрограммы возвращения курсора в начальное положение

vivod_simvola ('T'); //вывод буквы T
vivod_simvola ('='); //вывод знака        =
vivod_simvola (z); //вывод знака температуры

//формирование значения температуры путем объединения двух переменных в одну
и избавления от лишних битов
```

```
t2 =t2>>4;
t2 = t2&0x0f;
t1 = t1<<4;
t1 = t1&0xf0;
t3 = t2|t1;

//выделение десятков и единиц
d=0;
  while (t3>=10)
  {
  t3=t3-10;
  d=++d;
  }
d = d + 0x30; //формирование ASCII кода
vivod_simvola (d); //вывод десятков
t3 = t3 + 0x30; //формирование ASCII кода
vivod_simvola (t3); //вывод единиц
}

//функция временной задержки
void Delay_x10us(__IO uint32_t nTime)
{
TimingDelay = nTime;
while(TimingDelay != 0);
}

//обработчик прерывания системного таймера
void SysTick_Handler(void)
{
TimingDelay_Decrement(); //вызов подпрограммы декремента переменной для задержки
}

//подпрограмма декремента переменной для задержки
void TimingDelay_Decrement(void)
{
  if (TimingDelay != 0x00)
  {
  TimingDelay--; //декремент переменной для задержки
  }
}
```

```
void Delay_65us()
{
u32 i;
for (i = 0; i <= 150; i++);
}

void Delay_2us()
{
u32 i;
i = 1;
i = 2;
i = 3;
}

void Delay_8us()
{
u32 i;
for (i = 0; i <= 13; i++);
}

void Delay_1s(void)
{
u32 i;
for (i = 0; i <= 2400000; i++);
}

void Delay_70us()
{
u32 i;
for (i = 0; i <= 175; i++);
}
```

Приложение 4
Программа реализации цифрового фильтра на микроконтроллере STM32F407

Программа реализует цифровой фильтр нижних частот Баттерворта 5-го порядка с использованием микроконтроллера *STM32F407.*

```
#include "stm32f4xx.h"
#include "stm32f4xx_rcc.h"
#include "stm32f4xx_gpio.h"
#include "stm32f4xx_adc.h"
#include "stm32f4xx_tim.h"
#include "stm32f4xx_dac.h"

void Config(void);
void ADC_IRQHandler(void);

int date=0, i=0, dateout=0;
int in[6];
float out[6];

const float a[6] = {
0,
4.009,
-6.822,
6.111,
-2.872,
0.566
};

const float b[6] = {
0.000251,
0.001257,
0.002515,
0.002515,
0.001257,
0.000251,
};

int main(void)
{
```

```
SystemInit(); //настройка частоты тактирования
Config();
__enable_irq (); //глобальное разрешение прерываний
 while (1)
{
}
}

void ADC_IRQHandler(void)
{
for(i=0; i<5; i++)
{
in[i]=in[i+1];
out[i]=out[i+1];
}
out[5]=0;

date=ADC_GetConversionValue(ADC1); //получение результата
ADC_ClearFlag(ADC1, ADC_FLAG_EOC); //сброс флага
in[5]=date;
for(i=0; i<6; i++)
{
out[5]=out[5]+a[i]*out[5-i]+b[i]*in[5-i];
}
dateout=out[5]*0.7;
DAC_SetChannel1Data(DAC_Align_12b_R, dateout);
DAC_SoftwareTriggerCmd(DAC_Channel_1, ENABLE);
TIM_ClearFlag(TIM8, TIM_TRGOSource_Update);
}

void Config(void) //настройка
{
ADC_InitTypeDef ADC_InitStructure;
ADC_CommonInitTypeDef ADC_CommonInitStructure;
GPIO_InitTypeDef GPIO_InitStructure;
TIM_TimeBaseInitTypeDef TIM_InitStructure;
DAC_InitTypeDef DAC_InitStructure;

//включение тактирования
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOA|RCC_AHB1Perip
h_GPIOD, ENABLE);
```

```
RCC_APB1PeriphClockCmd(RCC_APB1Periph_DAC, ENABLE);
RCC_APB2PeriphClockCmd(RCC_APB2Periph_ADC1|RCC_APB2Periph_
TIM8, ENABLE);

//настройка PA1 на аналоговый вход (вход АЦП)
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_1;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL ;
GPIO_Init(GPIOA, &GPIO_InitStructure);

//настройка выхода ЦАП
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AN;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_4;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_Init(GPIOA, &GPIO_InitStructure);

//настройка порта D на вывод (светодиоды платы STM32F4Discovery)
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_12 | GPIO_Pin_13|
GPIO_Pin_14| GPIO_Pin_15;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
GPIO_Init(GPIOD, &GPIO_InitStructure);

//настройка таймера
TIM_InitStructure.TIM_RepetitionCounter = 0;
TIM_InitStructure.TIM_Prescaler = 267;
TIM_InitStructure.TIM_Period = 1;
TIM_InitStructure.TIM_ClockDivision = 0;
TIM_InitStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseInit(TIM8, &TIM_InitStructure);

TIM_SelectOutputTrigger(TIM8, TIM_TRGOSource_Update);
TIM_Cmd(TIM8, ENABLE);


//настройка АЦП
ADC_CommonInitStructure.ADC_Mode = ADC_Mode_Independent;
//одиночный режим
```

```
ADC_CommonInitStructure.ADC_Prescaler = ADC_Prescaler_Div2;
//предделитель 8
ADC_CommonInitStructure.ADC_DMAAccessMode =
ADC_DMAAccessMode_Disabled; //без DMA
ADC_CommonInitStructure.ADC_TwoSamplingDelay =
ADC_TwoSamplingDelay_5Cycles; //задержка
ADC_CommonInit(&ADC_CommonInitStructure);

ADC_InitStructure.ADC_Resolution = ADC_Resolution_12b; //12-битный
                                                       //результат
ADC_InitStructure.ADC_ScanConvMode = DISABLE; //без сканирования
ADC_InitStructure.ADC_ContinuousConvMode = DISABLE;
// преобразования одно за другим
ADC_InitStructure.ADC_ExternalTrigConvEdge =
ADC_ExternalTrigConvEdge_Rising;
ADC_InitStructure.ADC_ExternalTrigConv =
ADC_ExternalTrigConv_T8_TRGO;
ADC_InitStructure.ADC_DataAlign = ADC_DataAlign_Right;
//выравнивание по правому краю
ADC_InitStructure.ADC_NbrOfConversion = 1; //один канал
ADC_Init(ADC1, &ADC_InitStructure);

//настройка регулярного канала АЦП1, канал 1, PA1
ADC_RegularChannelConfig(ADC1, ADC_Channel_1, 1,
ADC_SampleTime_3Cycles);
ADC_ITConfig(ADC1, ADC_IT_EOC, ENABLE); //включение прерываний
                                        //от АЦП по биту EOC
NVIC_EnableIRQ (ADC_IRQn);

ADC_Cmd(ADC1, ENABLE); //включение АЦП 1

DAC_InitStructure.DAC_Trigger = DAC_Trigger_Software;
DAC_InitStructure.DAC_WaveGeneration = DAC_WaveGeneration_None;
DAC_InitStructure.DAC_OutputBuffer = DAC_OutputBuffer_Enable;
DAC_Init(DAC_Channel_1, &DAC_InitStructure);
DAC_Cmd(DAC_Channel_1, ENABLE);
}
```

Список литературы

1. STlife.augmented [Электронный ресурс] / User manual. – Режим доступа : http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00040810.pdf, свободный. – Загл. с экрана. – Яз. англ.
2. ST-LINK/V2 [Электронный ресурс] / ST-LINK/V2 in-circuit debugger/programmer for STM8 and STM32. – Режим доступа : http://lib.chipdip.ru/163/DOC001163688.pdf, свободный. – Загл. с экрана. – Яз. англ.
3. STlife.augmented [Электронный ресурс] / User manual. – Режим доступа : http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/CD00267113.pdf, свободный. – Загл. с экрана. – Яз. англ.
4. Основы микропроцессорной техники : микроконтроллеры STM8S : учебное пособие / С.Н. Торгаев, И.С. Мусоров, Д.С. Чертихина и др. – Томск : Изд-во ТПУ, 2014 – 130 с.
5. STlife.augmented [Электронный ресурс] / User manual. – Режим доступа : http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/CD00283778.pdf, свободный. – Загл. с экрана. – Яз. англ.
6. STlife.augmented [Электронный ресурс] / User manual. – Режим доступа : http://www.st.com/st-web-ui/static/active/en/resource/technical/document/technical_note/DM00039768.pdf, свободный. – Загл. с экрана. – Яз. англ.
7. STlife.augmented [Электронный ресурс] / User manual. – Режим доступа : http://www.st.com/st-web-ui/static/active/en/resource/technical/document/user_manual/DM00039084.pdf, свободный. – Загл. с экрана. – Яз. англ.
8. Winstar Display Co., Ltd. [Электронный ресурс] / Specification. – Режим доступа : http://aquacontrol.narod.ru/spravka/WH1602A-YGH-CTK.pdf, свободный. – Загл. с экрана. – Яз. англ.
9. Dallas Semiconductor [Электронный ресурс] / DS18B20. – Режим доступа : http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Temp/DS18B20.pdf, свободный. – Загл. с экрана. – Яз. англ.

Учебное издание

ТОРГАЕВ Станислав Николаевич
ТРИГУБ Максим Викторович
МУСОРОВ Илья Сергеевич
ЧЕРТИХИНА Дарья Сергеевна

ПРАКТИЧЕСКОЕ РУКОВОДСТВО ПО ПРОГРАММИРОВАНИЮ STM-МИКРОКОНТРОЛЛЕРОВ

Учебное пособие

Корректура *В.Ю. Пановица*
Компьютерная верстка *О.Ю. Аршинова*

Зарегистрировано в Издательстве ТПУ
Размещено на корпоративном портале ТПУ

Национальный исследовательский Томский политехнический университет
Система менеджмента качества
Издательства Томского политехнического университета
Сертифицирована в соответствии с требованиями ISO 9001:2008

ИЗДАТЕЛЬСТВО ТПУ. 634050, г. Томск, пр. Ленина, 30
Тел./факс: 8(3822)56-35-35, www.tpu.ru