

Программируемый логический контроллер

**Программируемые логические контроллеры
серии MC
с дискретными входами-выходами**

**MC-12D4R4O, MC-12D6R, MC-12D8O,
MC-8D2S, MC-8D2R, MC-4A7D4R4O,
MC-8U8O, MC-8U6R**

**Библиотеки функций
для программирования на языке Си**

Уважаемый покупатель!

Научно-исследовательская лаборатория автоматизации проектирования (НИЛ АП) благодарит Вас за покупку и просит сообщать нам свои пожелания по улучшению этого руководства или описанной в нем продукции. Ваши пожелания можно направлять по почтовому или электронному адресу, а также сообщать по телефону:

НИЛ АП, ул. Биржевой спуск, 8, Таганрог, 347900,

Тел.: (495) 26-66-700,

e-mail: info@reallab.ru • <http://www.reallab.ru>.

Вы можете также получить консультации по применению нашей продукции, воспользовавшись указанными выше координатами.

Пожалуйста, внимательно изучите настоящее руководство. Это позволит вам в кратчайший срок и наилучшим образом использовать приобретенное изделие.

НИЛ АП оставляет за собой право изменять данное руководство и модифицировать изделия без уведомления покупателей.

Представленную здесь информацию мы старались сделать максимально достоверной и точной, однако НИЛ АП не несет какой-либо ответственности за результат ее использования, поскольку невозможно гарантировать, что данное изделие пригодно для всех целей, в которых оно применяется покупателем.

Программное обеспечение, поставляемое в комплекте с прибором, продается без доработки для нужд конкретного покупателя и в том виде, в котором оно существует на дату продажи.

Авторские права на программное обеспечение, ПЛК, товарный знак "RealLab!" и настоящее руководство принадлежат НИЛ АП.

Оглавление

Введение	4
1. Описание библиотек.....	5
1.1. Библиотека LED.h.....	5
1.2. Библиотека Key.h.....	9
1.3. Библиотека IOPort.h.....	12
1.4. Библиотека Shim.h	18
1.5. Библиотека COM.h	20
1.6. Библиотека RTC.h.....	37
1.7. Библиотека IWire.h.....	43
1.8. Библиотека ADS1248.h.....	44
1.9. Библиотека Analog.h.....	59
1.10. Библиотека COM_SLAVE.h.....	61
2. Заключительная часть	81

Введение

В комплект поставки контроллеров серии MC (далее - "контроллер") входят описанные ниже библиотеки для разработки программного обеспечения пользователя. Библиотеки предназначены для использования в среде программирования Atmel Studio 7. Последняя версия среды программирования Atmel Studio 7 доступна для бесплатного копирования с сайта компании ATMEL <http://www.atmel.com/tools/atmelstudio.aspx>. Библиотеки также тестировались в AVR Studio 6 и AVR Studio 5.

Библиотеки позволяют управлять устройством индикации (далее-индикатором), дискретными линиями ввода/вывода, формировать сигналы ШИМ на них, управлять интерфейсом RS485 с поддержкой протоколов DCON и Modbus RTU, а также проводить опрос элементов ручного управления (далее кнопок), датчика температуры платы и микросхемы часов реального времени.

Для каждого контроллера создан специальный заголовочный файл описаний, позволяющий правильно сконфигурировать библиотеки под конкретную модификацию контроллера. В начале исходного текста программы необходимо обязательно указать тип модификации контроллера, путем подключения соответствующего заголовочного файла.

Перечень библиотек комплекта:

- MC12D4R40.h - файл конфигурации для контроллера MC-12D4R40;
- MC12D6R.h - файл конфигурации для контроллера MC-12D6R;
- MC12D8O.h - файл конфигурации для контроллера MC-12D8O;
- MC8D2S2R.h - файл конфигурации для контроллеров MC-8D2S и MC-8D2R;
- MC8U8O.h – файл конфигурации для контроллеров MC-8U8O и MC-8U6R;
- LED.h – библиотека для работы с индикатором;
- Key.h – библиотека для работы с кнопками контроллера;
- IOPort.h – библиотека для работы с дискретными вх./вых.;
- Shim.h – библиотека для управления генератором ШИМ сигналов.
- COM.h – библиотека для работы с COM-портами контроллера по протоколам DCON и Modbus RTU;

Описание библиотек

- 1Wire.h – библиотека для работы с шиной 1Wire и чтения температуры платы;
- RTC.h – библиотека для работы с микросхемой часов реального времени;
- Analog.h – библиотека функций управления аналоговыми входами контроллера MC-4A7D4R4O.
- ADS1248.h – библиотека функций работы с АЦП контроллеров MC-8U8O и MC-8U6R

Все необходимые библиотеки и заголовочные файлы должны храниться в корневой папке проекта или путь к ним должен быть указан в настройках проекта.

1. Описание библиотек

1.1. Библиотека LED.h

Функции работы с индикатором контроллера

Данная библиотека содержит функции управления индикатором и светодиодом. Функции управления индикатором позволяют обеспечить вывод на индикатор информации, заданной в символьном виде. При использовании функций данной библиотеки рекомендуется использовать оптимизацию кода программы. Уровень оптимизации может быть произвольным.

При использовании функции циклического сдвига информации на индикаторе (режим «бегущая строка»), в начале исходного текста (до подключения библиотеки) необходимо объявить директиву **#define TIMER2**. Данная константа включает в исходный текст программы обработчик прерывания таймера-счетчика 2, при этом в дальнейшем, не допускается перепрограммировать данный таймер-счетчик. Если данную константу не объявить, компилятор не выдаст ошибки, но циклический сдвиг информации на индикаторе осуществляться не будет.

В библиотеке предусмотрены следующие константы, упрощающие использование функций.

```
#define LD_OFF 0      //Константа "Выключить светодиод"  
#define LD_ON  1     //Константа "Включить светодиод"
```

1.1. Библиотека LED.h

```
#define LD_SW 2      //Константа "Переключить светодиод"  
#define ENABLE 1    //Константа "Включено"  
#define DISABLE 0   //Константа "Выключено"
```

Ниже представлены прототипы функций данной библиотеки.

void LED_Init(void);

Функция предназначена для инициализации индикатора. Без предварительного вызова данной функции управление индикатором невозможно. Данная функция вызывается один раз, при инициализации контроллера. Для управления светодиодом вызывать данную функцию нет необходимости.

void LED_Clear(void);

Функция предназначена для очистки индикатора. После выполнения функции, DDRAM память индикатора заполняется символами пробела (код 0x20), позиция курсора и сдвиг индикатора не изменяются.

void LED_Return_Home(void);

Функция предназначена для сброса сдвига индикатора и установки курсора в левый верхний угол. Информация на индикаторе при этом не удаляется.

void LED_CursorEnable(uint8_t EN);

Функция включение/выключение отображения курсора.

Параметры:

EN — флаг управления курсором. При значении 0 курсор не отображается, при любом другом отображается.

Если курсор находится за пределами видимой в данный момент области индикатора, то он отображаться не будет, пока не попадет в видимую область, путем сдвига индикатора или самого курсора.

void LED_SetPos(uint8_t Y, uint8_t X);

Функция предназначена для установки курсора в заданную позицию. Если в данной позиции находится какой-либо символ, он будет отображаться с прерывистым свечением (мигать). Сам курсор, в случае активации, так же будет отображаться прерывисто.

Параметры:

Y — номер строки. Счет строк ведется сверху вниз с 0 до 1.

X — номер столбца. Счет столбцов ведется слева направо с 0 до 63.

Описание библиотек

Необходимо учитывать, что на индикаторе одновременно может быть отображено только 2 строки по 16 символов в каждой.

void LED_Write_String(char *buf);

Функция осуществляет вывод текстовой строки в текущую позицию курсора. Курсор автоматически будет смещен вправо на количество символов в строке.

Параметры:

***buf** — строка символов, выводимая на индикатор. Строка может быть задана в качестве массива символьных переменных или указана непосредственно в виде строковой константы. Признаком окончания строки по стандарту языка СИ является символ с ASCII кодом 0x00 (ноль-символ).

Если позиция курсора находится не в видимой области индикатора, равно как и если строка не помещается полностью в видимую область, отображены будут только те символы, которые уместятся в данную область. Однако, остальные символы будут размещены в оперативной памяти индикатора, и могут быть выведены в дальнейшем путем сдвига индикатора.

Отображаемая строка не должна превышать 64 символа. Если длина строки превысит данную величину, все лишние символы будут отброшены.

В случае если сумма горизонтальной позиции курсора и длины строки превысит 64 байта, то данные будут автоматически перенесены на смежную строку.

void LED_Cycle_Shift(char time);

Функция осуществляет запуск циклического сдвига строки (режим «бегущая строка»).

Параметры:

time — интервал времени по истечении которого дисплей будет сдвинут на 1 разряд. Фактически характеризует скорость движения текста в «бегущей строке». Одна единица соответствует 10 мс. При значении аргумента равном нулю, движение останавливается. Для комфортного визуального восприятия информации на индикаторе, значение параметра должно быть в районе 20-25 единиц.

void Led(char power);

Функция позволяет включить или выключить светодиод на лицевой панели контроллера.

1.1. Библиотека LED.h

Параметры:

power — параметр управляющий состоянием светодиода (0-выключить, 1-включить, 2-переключить). Переключение осуществляет включение светодиода, если он был выключен и наоборот. Для удобства работы с данным параметром целесообразно применять константы, описанные в начале раздела.

Пример LED_1:

Данная программа выводит на индикатор две текстовых строки. Первая строка превышает допустимую длину 64 символа, поэтому лишние символы отбрасываются. Также программа осуществляет переключение светодиода с интервалом 500 мс.

```
#define F_CPU 14745600UL /*Стандартная константа, задающая частоту микроконтроллера*/

#include "MC12D4R4O.h" //Тип контроллера

#define TIMER2 //Константа, разрешающая функциям библиотеки использовать таймер 2

#include <avr/io.h> //Стандартная библиотека ввода-вывода
#include "LED.h" //Библиотека работы с LED дисплеем

//=====
int main(void)
{
    LED_Init(); //Инициализация индикатора

    LED_SetPos(0,0); //Установка курсора в левую позицию верхней строки

    //Вывод сообщения на индикатор
    LED_Write_String("Научно-исследовательская лаборатория автоматизации проектирования");

    LED_SetPos(1,0); //Установка курсора в левую позицию нижней строки

    //Вывод сообщения на индикатор
    LED_Write_String("общество с ограниченной ответственностью");

    LED_Cycle_Shift(25);/*Запуск циклического сдвига индикатора с интервалом 250
мс*/

    while(1) //Бесконечный цикл
    {
        Led(LD_SW); //Переключение светодиода
        _delay_ms(500);
    }
}
```

1.2. Библиотека Key.h

Функции работы с кнопками контроллера

Функции данной библиотеки позволяют определять состояния кнопок контроллера. Отдельно для каждой кнопки можно включить режим фиксации нажатия, который будет подробно описан ниже.

В библиотеке предусмотрены следующие константы, упрощающие использование функций.

```
#define UP          0          /*Кнопка в состоянии «Не нажата» */
#define DOWN       1          /*Кнопка в состоянии «Нажата» */
#define LATCH_ON   1          /*Включить фиксацию кнопки*/
#define LATCH_OFF  0          /*Выключить фиксацию кнопки*/
```

При использовании режима фиксации нажатия, в начале исходного текста (до подключения библиотеки) необходимо объявить константу **TIMER2**, которая включает в исходный текст программы обработчик прерывания таймера-счетчика 2. При этом в дальнейшем, не допускается перепрограммировать данный таймер-счетчик. Если данную константу не объявить и включить режим фиксации нажатия, компилятор не выдаст ошибки, но кнопка опрашиваться не будет. Пример объявления константы, включающей в исходный текст программы обработчик прерывания таймера-счетчика 2.

```
#define TIMER2     /* подключить обработчик прерывания таймера 2*/
#include "Key.h"   //Подключить библиотеку работы с кнопками
```

Далее представлены прототипы функций для определения состояния кнопок.

void Key_Init(char sw);

Функция проводит инициализацию входных линий кнопок. Также, в случае если включен режим фиксации нажатия, функция проводит инициализацию таймера-счетчика 2, задействованного в опросе состояния кнопок. Без предварительного вызова данной функции, корректное определение состояния кнопок невозможно. Данную функцию достаточно вызвать один раз при инициализации контроллера. Существуют два режима работы кнопок: текущее состояние и с фиксацией нажатия кнопки. В режиме текущего состояния, функция чтения статуса кнопки возвращает «0», если кнопка в данный момент отпущена, и «1» если кнопка нажата. При нажа-

1.2. Библиотека Key.h

тии кнопки, работающей в режиме с фиксацией нажатия, устанавливается специальный флаг, который считывается как статус кнопки. Данный флаг сбрасывается автоматически после каждого выполнения функции чтения статуса кнопки.

Режим с фиксацией позволяет определять состояние кнопок, не пользуясь постоянными циклическими опросами, что может быть полезно в системах реального времени. Также данный режим позволяет избежать эффекта «дребезга контактов» и не применять дополнительных мер по борьбе с ним. Режим текущего состояния кнопок предоставляет большую свободу действий и может быть полезен, когда необходимо определить не только нажатие, но и удержание кнопки или когда требуется разделять события нажатия и отпускания кнопки. По умолчанию у всех кнопок включен режим текущего состояния кнопки.

Параметры:

sw — параметр, включающий/отключающий режим фиксации кнопки. При нулевом значении параметра, будет включен режим текущего состояния кнопки, при любом другом значении, будет включен режим фиксации нажатия. Для удобства указания данного параметра целесообразно применять константы, описанные в начале раздела.

char Key(char key);

Функция позволяет прочитать состояние выбранной кнопки (нажата или отпущена) в режиме текущего состояния кнопки. В случае, если включен режим фиксации нажатия, данная функция производит чтение специального флага, который устанавливается автоматически, когда происходит нажатие и сбрасывается по завершении выполнения данной функции.

Параметры:

key - номер кнопки, состояние которой необходимо определить. Может принимать значения от 0 до 3. Нумерация кнопок на контроллере идет справа налево.

Возвращаемое значение:

Функция возвращает состояние кнопки или флага фиксации. При отпущенной кнопке возвращаемое значение будет равно нулю, при нажатой единице. Для удобства работы с данным параметром целесообразно применять константы, описанные в начале раздела.

Пример Key_1:

Описание библиотек

Данная программа проводит постоянный опрос состояния кнопок и выводит результат на индикатор. В случае если кнопка нажата, соответствующий разряд индикатора будет отображать единицу, в противном случае ноль.

```
#define F_CPU 14745600UL /*Стандартная константа, задающая частоту микроконтроллера*/

#include "MC12D4R40.h" //Тип контроллера

#define TIMER2 //Константа, разрешающая функциям библиотеки использовать таймер 2

#include <avr/io.h> //Стандартная библиотека ввода-вывода
#include "LED.h" //Библиотека работы с LED дисплеем
#include "Key.h" //Библиотека работы с кнопками

int main(void)
{
    LED_Init(); //Инициализация индикатора
    Key_Init(LATCH_OFF); //Инициализация кнопок

    char buf[5]={0,0,0,0,0};

    while(1)
    {
        LED_Return_Home(); //Курсор индикатора в начало экрана

        buf[0]=Key(3)+0x30; //Состояние левой кнопки
        buf[1]=Key(2)+0x30;
        buf[2]=Key(1)+0x30;
        buf[3]=Key(0)+0x30; //Состояние правой кнопки

        LED_Write_String(buf); //Отобразить на экране

        _delay_ms(100); //Задержка на 100 мс
    }
}
```

Пример Key_2:

Данная программа демонстрирует использование кнопок для ввода каких-либо параметров. На индикатор выводится числовое значение. При нажатии на кнопку 3, значение уменьшается на 10. При нажатии на кнопку 2, значение уменьшается на 1. При нажатии на кнопку 1, значение увеличивается на 1. При нажатии на кнопку 0, значение увеличивается на 10.

1.3. Библиотека IOPort.h

```
#define F_CPU 14745600UL /*Стандартная константа, задающая частоту микроконтроллера*/

#include "MC12D4R4O.h" //Тип контроллера

#include <avr/io.h> //Стандартная библиотека ввода-вывода
#include "LED.h" //Библиотека работы с LED дисплеем
#include "Key.h" //Библиотека работы с кнопками

#include<stdio.h> /*Стандартная библиотека ввода-вывода (в данном примере используется для преобразования числового значения в строку)*/

int main(void)
{
    LED_Init();
    Key_Init(LATCH_OFF);

    char buf[7];

    int Count=0;

    while(1)
    {
        LED_Clear(); //Очистка индикатора
        LED_Return_Home(); //Установка курсора в начало индикатора

        if (Key(3)==DOWN) Count-=10; //Опрос кнопки 3
        else if (Key(2)==DOWN) Count--; //Опрос кнопки 2
        else if (Key(1)==DOWN) Count++; //Опрос кнопки 1
        else if (Key(0)==DOWN) Count+=10; //Опрос кнопки 0

        sprintf(buf,"%d",Count); //Преобразования числа в строку

        LED_Write_String(buf); //Вывод строки на индикатор

        _delay_ms(100); //Задержка
    }
}
```

1.3. Библиотека IOPort.h

Функции управления дискретными входами-выходами контроллера

Функции данной библиотеки предназначены для настройки дискретных входов-выходов контроллера и осуществления их чтения и установки.

Для удобства использования функций в данной библиотеке объявлены следующие константы:

Описание библиотек

```
#define LOW          0      /*Низкое состояние линии*/  
#define HI           1      /*Высокое состояние линии*/  
#define SW           2      /*Переключить состояние линии*/
```

Далее представлены прототипы функций для управления дискретными входами-выходами.

void IO_Init(void);

Функция осуществляет инициализацию дискретных входов-выходов. Без предварительного вызова данной функции работа с дискретными входами-выходами невозможна. Данную функцию достаточно вызвать один раз, при инициализации контроллера.

uint8_t Input_Read(uint8_t Line);

Функция осуществляет чтение состояния указанного дискретного входа контроллера.

Параметры:

Line – номер считываемого дискретного входа.

Возвращаемое значение:

Функция возвращает состояние прочитанного дискретного входа. Ноль соответствует низкому состоянию на входе, единица соответственно высокому.

int Input_Read_All (void);

Функция осуществляет чтение состояния всех дискретных входов контроллера.

Возвращаемое значение:

Функция возвращает состояние прочитанных дискретных входов. Младшему разряду соответствует нулевой дискретный вход. Ноль соответствует низкому состоянию на входе, единица соответственно высокому.

int Input_Led(void);

Функция выполняет чтение состояния всех дискретных входов при помощи функции **Input_Read_All** и осуществляет их вывод на индикатор в двоичном формате (каждый вход представлен отдельно и подписан).

Возвращаемое значение:

1.3. Библиотека IOPort.h

Функция возвращает состояние прочитанных дискретных входов. Младшему разряду соответствует нулевой дискретный вход.

void Output_Write(uint8_t Line, uint8_t Status);

Функция осуществляет управление указанным дискретным выходом.

Параметры:

Line – номер дискретного выхода.

Status – состояние дискретного выхода. Данный параметр может принимать следующие значения:

0-установить дискретный выход в низкое состояние;

1-установить дискретный выход в высокое состояние;

2-изменить состояние дискретного выхода на противоположное.

Для удобства работы с данным параметром целесообразно применять константы, описанные в начале раздела.

void Output_Write_All(uint8_t data);

Функция осуществляет управление сразу всеми дискретными выходами.

Параметры:

data – значение отправляемое на дискретные выходы. Нулевому разряду аргумента соответствует нулевой выход.

uint8_t Output_Read(uint8_t Line);

Функция считывает ранее установленное состояние указанного дискретного выхода.

Параметры:

Line – номер дискретного выхода.

Возвращаемое значение:

Функция возвращает состояние указанного дискретного выхода.

uint8_t Output_Read_All(void);

Функция считывает ранее установленное состояние всех дискретных выходов.

Возвращаемое значение:

Описание библиотек

Функция возвращает состояние ранее установленных дискретных выходов. Младший бит соответствует младшему дискретному выходу.

Пример IOPort_1:

Данная программа отображает на индикаторе состояние всех дискретных входов, используя специализированную функцию.

```
#define F_CPU 14745600UL //Стандартная константа, задающая частоту микроконтроллера

#include "MC12D4R4O.h" //Тип контроллера

#define TIMER2 //Константа, разрешающая функциям библиотеки использовать таймер 2*/

#include <avr/io.h> //Стандартная библиотека ввода-вывода
#include "LED.h" //Библиотека работы с LED дисплеем
#include "IOPort.h" //Библиотека работы с дискретными входами/выходами

int main(void)
{
    LED_Init(); //Инициализация индикатора
    IO_Init(); //Инициализация дискретных входов-выходов

    while(1) //Бесконечный цикл программы
    {
        Input_Led(); //Вызов функции опроса дискретных входов и отображения их состояния на индикаторе*/

        Led(LD_SW); //Переключить светодиод
        _delay_ms(500); //Программная задержка
    }
}
```

Пример IOPort_2:

Данная программа отображает на индикаторе состояние всех дискретных входов, используя функцию одиночного чтения дискретных входов.

```
#define F_CPU 14745600UL //Стандартная константа, задающая частоту микроконтроллера

#include "MC12D4R4O.h" //Тип контроллера

#define TIMER2 //Константа, разрешающая функциям библиотеки использовать таймер 2*/

#include <avr/io.h> //Стандартная библиотека ввода-вывода
#include "LED.h" //Библиотека работы с LED дисплеем
#include "IOPort.h" //Библиотека работы с дискретными входами/выходами
```

1.3. Библиотека IOPort.h

```
int main(void)
{
    uint8_t temp;           //Вспомогательные переменные

    LED_Init();           //Инициализация индикатора
    IO_Init();           //Инициализация дискретных входов-выходов

    while(1)             //Бесконечный цикл программы
    {
        LED_SetPos(0,0);  //Установить курсор в верхний левый угол

        for (temp=0; temp<12; temp++) //Вывод состояния входов
        {
            /*Если логическая "1"*/
            if (Input_Read(temp)) _LED_Write_Byte('1');

            /*иначе, логический "0"*/
            else _LED_Write_Byte('0');
        }
        Led(LD_SW);       //Переключить светодиод
        _delay_ms(500);   //Программная задержка
    }
}
```

Пример IOPort_3:

Данная программа периодически переключает дискретные выходы (бегущая единица) и отображает их состояние на индикаторе.

```
#define F_CPU 14745600UL /*Стандартная константа, задающая частоту микроконтроллера*/

#include "MC12D4R4O.h" //Тип контроллера

#include <avr/io.h> //Стандартная библиотека ввода-вывода
#include "LED.h" //Библиотека работы с LED дисплеем
#include "IOPort.h" //Библиотека работы с дискретными входами/выходами

int main(void)
{
    LED_Init(); //Инициализация индикатора
    IO_Init(); //Инициализация дискретных входов-выходов

    char output=1; //Переменная для управления входами
    int temp; //Вспомогательная переменная
    char buf[9]={0,0,0,0,0,0,0,0,0}; //Строка для отображения на индикаторе

    while(1) //Бесконечный цикл
    {
        Output_Write_All(output); //Вывести значение на дискретные выходы
        Led(LD_SW); //Переключить светодиод
    }
}
```

Описание библиотек

```
        LED_Return_Home();           //Установка курсора в начало индикатора

        for (temp=0; temp<8; temp++) /*Сформировать строку с состоянием
дискретных выходов*/
        {
                buff[temp]=((output>>(7-temp)) & 0x01)+0x30;
        }
        LED_Write_String(buff);       //Вывод строки на индикатор
        output=output<<1;              //Переключится на следующий дискретный выход
        if (output==0) output=1;      /*Если достигнут последний выход, пе-
рейти на начало*/
        _delay_ms(500);               //Программная задержка
    }
}
```

Пример IOPort_4:

Данная программа, при нажатии на одну из кнопок, переводит соответствующий ей дискретный выход в противоположное состояние, при этом текущее состояние дискретных выходов отображается на индикаторе.

```
#define F_CPU 14745600UL /*Стандартная константа, задающая частоту микроконтроллера*/

#include "MC12D4R40.h" //Тип контроллера

#include <avr/io.h> //Стандартная библиотека ввода-вывода
#include "LED.h" //Библиотека работы с LED дисплеем
#include "IOPort.h" //Библиотека работы с дискретными входа-ми/выходами
#include "Key.h" //Библиотека работы с кнопками

int main(void)
{
    LED_Init(); //Инициализация индикатора
    IO_Init(); //Инициализация дискретных входов-выходов
    Key_Init(LATCH_OFF); //Инициализация кнопок

    char output=0; //Переменная для управления входами
    int temp; //Вспомогательная переменная
    char buff[9]={0,0,0,0,0,0,0,0,0}; //Строка для отображения на ин-дикаторе

    while(1) //Бесконечный цикл
    {
        LED_Return_Home(); //Установка курсора в начало индикатора
        output=Output_Read_All(); /*Чтение текущего состояния дискретных
выходов*/

        for (temp=0; temp<8; temp++) /*Сформировать строку с состоянием
дискретных выходов*/
        {
```

1.4. Библиотека Shim.h

```
        buf[temp]=((output>>(7-temp)) & 0x01)+0x30;
    }
    LED_Write_String(buf);          //Вывод строки на индикатор

    if (Key(3)==DOWN)                //Опрос кнопки 3
    {
        while(Key(3)==DOWN);        //Ожидание отпускания кнопки
        Output_Write(3,SW); //Переключение дискретного выхода
    }
    if (Key(2)==DOWN)                //Опрос кнопки 2
    {
        while(Key(2)==DOWN);        //Ожидание отпускания кнопки
        Output_Write(2,SW); //Переключение дискретного выхода
    }
    if (Key(1)==DOWN)                //Опрос кнопки 1
    {
        while(Key(1)==DOWN);        //Ожидание отпускания кнопки
        Output_Write(1,SW); //Переключение дискретного выхода
    }
    if (Key(0)==DOWN)                //Опрос кнопки 0
    {
        while(Key(0)==DOWN);        //Ожидание отпускания кнопки
        Output_Write(0,SW); //Переключение дискретного выхода
    }

    _delay_ms(100);                  //Программная задержка
}
}
```

1.4. Библиотека Shim.h

Функции для работы с генератором ШИМ сигналов

Функции данной библиотеки позволяют осуществлять инициализацию и запуск программного ШИМ. В качестве выводов ШИМ используются дискретные выходы. Каждый из каналов ШИМ может управляться отдельно. Величина периода регулирования и длительность управляющего импульса варьируется от 0 до 10 секунд, с шагом 10 мс.

В начале исходного текста (до подключения библиотеки) необходимо объявить константу **TIMER2**, которая включает в исходный текст программы обработчик прерывания таймера-счетчика 2. При этом в дальнейшем, не допускается перепрограммировать данный таймер-счетчик. Если данную константу не объявить компилятор не выдаст ошибки, но ШИМ

Описание библиотек

генерироваться не будет. Пример объявления константы, включающей в исходный текст программы обработчик прерывания таймера-счетчика 2.

```
#define TIMER2 /* подключить обработчик прерывания таймера 2*/
```

```
#include "Shim.h" //Подключить библиотеку ШИМ
```

Ниже приведены прототипы функций, описанных в данной библиотеке.

char Shim(int Channel, unsigned int Pulse, unsigned int Period);

Функция предназначена для инициализации и запуска генерации ШИМ-сигналов.

Параметры:

Channel - номер канала ШИМ.

Pulse – длительность управляющего импульса. Одна единица соответствует 10 мс. Длительность управляющего импульса не должна превышать длительности периода регулирования. Для остановки генерации ШИМ сигнала, необходимо запустить функцию повторно присвоив данному параметру значение 0 или равное значению длительности периода регулирования (в зависимости от того какое состояние дискретного выхода нужно после остановки генерации ШИМ).

Period – период регулирования. Одна единица соответствует 10 мс. Значение данного параметра не должно быть больше 1000, т.е. период регулирования не может превышать 10 секунд;

Возвращаемое значение:

Функция возвращает 0 в случае успешного завершения работы, либо 0xFF в случае, если длительность периода превышает 10 секунд или длительность управляющего импульса превышает период регулирования.

Пример Shim_1:

Данная программа осуществляет запуск генерации ШИМ сигнала на дискретном выводе 0. Длительность периода 1 секунда, длительность импульса управления 500 мс.

```
#define F_CPU 14745600UL /*Стандартная константа, задающая частоту микроконтроллера*/
```

```
#include "MC12D4R40.h" //Тип контроллера
```

```
#define TIMER2 //Константа, разрешающая функциям библиотеки использовать таймер 2
```

1.5. Библиотека COM.h

```
#include <avr/io.h>           //Стандартная библиотека ввода-вывода
#include "Shim.h"             //Библиотека работы с генератором ШИМ сигналов

int main(void)
{
    Shim(0,50,100);          //Запуск генерации ШИМ

    _delay_ms(10000);        //Задержка на 10с (генерация ШИМ продолжается)

    Shim(0,0,100);           //Остановка генерации ШИМ
    while(1);                 //Бесконечный цикл программы
}
```

1.5. Библиотека COM.h

Функции работы с COM портами

Функции данной библиотеки предназначены для работы с COM портами контроллера в соответствии со стандартами протоколов DCON и Modbus RTU. При программировании порта COM1 с использованием протокола Modbus RTU не допускается перепрограммировать таймер-счетчик 1 микроконтроллера, а при использовании порта COM2 - таймер-счетчик 3. При использовании протокола DCON таймеры-счетчики не задействуются.

Для удобства использования функций в данной библиотеке объявлены следующие константы:

```
#define CRC_EN      1          /*Контрольная сумма в протоколе DCON
используется*/
#define CRC_DIS     0          /*Контрольная сумма в протоколе DCON
не используется*/
#define DCON        0          /*Протокол DCON*/
#define MODBUS      1          /*Протокол Modbus RTU*/
#define COM_PORT1   0          /*Порт COM1*/
#define COM_PORT2   1          /*Порт COM2*/
#define NONE        0          /*Константа "Нет контроля паритета"*/
#define ODD         1          /*Константа "Контроль нечетности"*/
#define EVEN        2          /*Константа "Контроль четности"*/
```

Описание библиотек

При использовании данной библиотеки в начале исходного текста (до подключения библиотеки) необходимо объявить стандартную константу **F_CPU**, указывающую частоту кварца микроконтроллера (в стандартной комплектации контроллера частота кварца равна 14,7456 МГц). Если этого не сделать, компилятор выдаст предупреждение и установит частоту кварца по умолчанию 14,7456 МГц.

Пример объявления стандартной константы, указывающей частоту кварца микроконтроллера:

```
#define F_CPU 14745600UL /*Установить частоту кварца 14,7456 МГц*/
```

При использовании данной библиотеки в начале исходного текста (до подключения библиотеки) необходимо объявить константы **COM_RX_BUFFER_SIZE** и **COM_TX_BUFFER_SIZE** задающие в байтах размер приемного и передающего буфера приемопередатчика соответственно. Размер буфера должен быть такой, чтобы в нем могла полностью разместиться самая длинная посылка. Если этого не сделать, компилятор выдаст предупреждение и установит для каждого буфера размер 64 байта.

Пример объявления констант, указывающих размер приемного и передающего буфера приемопередатчика:

```
#define COM_RX_BUFFER_SIZE 128 /*Размер приемного буфера 128 байт*/
```

```
#define COM_TX_BUFFER_SIZE 128 /*Размер передающего буфера 128 байт*/
```

При использовании данной библиотеки в начале исходного текста (до подключения библиотеки) необходимо объявить константы **COM1_PROTOCOL** и/или **COM2_PROTOCOL** и присвоить им значение **DCON** или **MODBUS** в зависимости от используемого протокола. При этом, если константе **COM1_PROTOCOL** будет присвоено значение **MODBUS**, в исходный текст программы будет добавлен обработчик прерывания таймера-счетчика 1, и его в дальнейшем, не допускается перепрограммировать. Аналогично, если константе **COM2_PROTOCOL** будет присвоено значение **MODBUS**, в исходный текст программы будет включен обработчик прерывания таймера-счетчика 3, и его в дальнейшем, также не допускается перепрограммировать.

Если какую-либо константу не объявить, компилятор не выдаст предупреждений, но обмен информацией по данному COM порту осуществляться не будет.

1.5. Библиотека COM.h

Пример объявления констант, определяющих используемый протокол.

```
#define COM1_PROTOCOL MODBUS /*Установить для порта COM1
протокол Modbus RTU (перепрограммировать таймер-счетчик 1 не допу-
стимо)*/
```

```
#define COM2_PROTOCOL DCON /* установить для порта COM2
протокол DCON*/
```

```
#include "COM.h" //Библиотека работы с COM портами
```

Далее представлены прототипы функций данной библиотеки.

```
char COM_Init(int port, long COM_speed, uint8_t Parity, uint8_t
Stop_Bit);
```

Функция проводит инициализацию указанного COM порта контроллера в соответствии с требованиями стандартов протоколов DCON или Modbus RTU. Данная функция позволяет выбрать один из двух портов, задать скорость связи, установить паритет и количество стоп битов. Если в программе не планируется изменять настройки связи, то данную функцию достаточно вызвать однократно для каждого используемого COM порта, при инициализации контроллера.

Параметры:

port — COM порт контроллера. Для удобства ввода данного параметра рекомендуется использовать константы, описанные в начале раздела.

COM_speed – скорость связи COM порта. Значение параметра скорости может принимать любое целочисленное значение, выраженное в битах в секунду и не превышающее значение 1.000.000. Однако, если частота кварца, указанного в программе, будет слишком низкая, а скорость связи слишком высокая, функция выдаст ошибку. Для более полной информации смотрите таблицу примеров настройки скоростей UART в руководстве по эксплуатации микроконтроллера ATmega128 (колонки с U2X = 0), которая доступна для свободного копирования с сайта компании ATMEL <http://www.atmel.com/Images/doc2467.pdf> . Если в таблице для указанного кварца и выбранной скорости стоит прочерк, значит данная скорость не достижима при текущей частоте передающего сигнала. Для обеспечения безошибочной передачи информации на высоких скоростях необходимо выбирать параметры, при которых ошибка будет равна нулю.

Parity — контроль паритета. Может принимать значения от 0 до 2. При 0 значении контроль паритета выключен, при значении 1 выполняется кон-

Описание библиотек

троль нечетности, при значении 2 выполняется контроль четности. Для удобства работы с данным параметром, целесообразно применять константы, описанные выше.

Stop_Bit — количество стоп битов. Может принимать значения 1 или 2.

Возвращаемое значение:

Функция возвращает 0 в случае успешного завершения работы, либо 0xFF в случае, если входные параметры были заданы неверно.

char CHK_Control(int port, char CHK);

Функция позволяет включить или выключить использование контрольной суммы в протоколе DCON для выбранного COM порта. При использовании протокола Modbus RTU данная функция ни на что не влияет.

Параметры:

port — COM-порт контроллера. Для удобства ввода данного параметра рекомендуется использовать константы, описанные в начале раздела.

CHK – признак включения/выключения контрольной суммы в протоколе DCON. Для удобства ввода данного параметра рекомендуется использовать константы, описанные в начале раздела.

Возвращаемое значение:

Функция возвращает 0 в случае успешного выполнения, либо 0xFF в случае, если указан не допустимый номер COM порта.

char Read_DCON(int port, char *buf);

Функция проверяет приемный буфер выбранного порта на наличие непрочитанных данных. В случае, если данные имеются и используется протокол DCON, функция проводит их считывание в буфер, указанный в качестве входного параметра. При этом нулевой элемент буфера будет содержать количество принятых байт данных, включая контрольную сумму, если она используется. Если контрольная сумма используется, функция автоматически производит ее расчет и проверку (данная опция включается функцией **CHK_Control** описанной выше). Два байта контрольной суммы будут включены в буфер «для чтения» вместе с остальной информацией.

Параметры:

port — COM порт контроллера. Для удобства ввода данного параметра рекомендуется использовать константы, описанные в начале раздела.

1.5. Библиотека COM.h

***buf** – буфер для считывания. Нулевой элемент буфера будет содержать количество принятых байт.

Возвращаемое значение:

Функция возвращает одно из приведенных ниже значений:

0x00 – непрочитанные данные отсутствуют;

0x01 – данные успешно прочитаны;

0xFE – ошибка контрольной суммы;

0xFF – не допустимый номер COM порта.

char Write_DCON(int port, char *buf);

Функция позволяет произвести запись данных в выбранный COM порт в соответствии со стандартом протокола DCON. Если включено вычисление контрольной суммы, функция автоматически производит ее расчет и подстановку в отправляемую посылку (данная опция включается функцией **CHK_Control** описанной выше). Если предыдущая посылка не была полностью отправлена, функция будет ожидать окончания передачи.

Параметры:

port — COM-порт контроллера. Для удобства ввода данного параметра рекомендуется использовать константы, описанные в начале раздела.

***buf** – массив данных, записываемый в COM-порт. Данный массив может представлять собой текстовую строку. По стандарту языка СИ строка должна заканчиваться нулевым символом, но т.к. в данном случае используется протокол DCON, признаком конца строки может выступать символ 0x0D. Функция автоматически определит наличие в строке символа 0x0D и в случае, если он отсутствует, добавит его в конец строки.

Возвращаемое значение:

Функция возвращает 0 в случае успешного выполнения, либо 0xFF в случае, если указан не допустимый номер COM порта.

char Read_Modbus(int port, char *buf);

Функция проверяет приемный буфер выбранного COM порта на наличие непрочитанных данных. В случае, если данные имеются и включен протокол Modbus RTU, функция проводит их считывание в буфер, указанный в качестве входного параметра. При этом нулевой элемент буфера будет

Описание библиотек

содержать количество принятых байт данных, включая контрольную сумму.

Параметры:

port — COM порт контроллера. Для удобства ввода данного параметра рекомендуется использовать константы, описанные в начале раздела.

***buf** – буфер для считывания. Нулевой элемент буфера будет содержать количество принятых байт.

Возвращаемое значение:

Функция возвращает одно из приведенных ниже значений:

0x00 - не прочитанные данные отсутствуют;

0x01 – данные успешно прочитаны;

0xFE – ошибка контрольной суммы;

0xFF – не корректный номер COM порта.

char Write_Request_Modbus(int port, char Address, char Func, unsigned int SubFunc, char *Data, unsigned int Amount);

Функция позволяет отправить команду запроса в выбранный COM-порт. Назначение отправляемых данных определяется кодом функции (в соответствии со стандартом). Контрольная сумма будет автоматически рассчитана и добавлена в отправляемую посылку. Данную функцию допустимо применять только для запросов (когда модуль выступает в качестве ведущего). Для отправки ответов, необходимо применять функцию **Write_Response_Modbus**, описание которой приведено ниже.

Параметры:

port — COM-порт контроллера. Для удобства ввода данного параметра рекомендуется использовать константы, описанные в начале раздела.

Address – адрес устройства (в соответствии со стандартом Modbus RTU).

Func – код функции. Допустимые коды функций: 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x0F, 0x10 (в соответствии со стандартом Modbus RTU).

SubFunc – код подфункции (адрес регистра в соответствии со стандартом Modbus RTU).

***Data** – массив байт записываемых данных.

1.5. Библиотека COM.h

Amount – количество записываемых элементов (регистров или ячеек в зависимости от выбранной функции).

Возвращаемое значение:

Функция возвращает одно из приведенных ниже значений:

0x00 – успешное завершение работы;

0x80 – не поддерживаемый номер функции;

0xFF – не корректный номер COM порта.

char Write_Response_Modbus (int port, char Addres, char Func, unsigned int SubFunc, char *Data, unsigned int Amount);

Функция позволяет отправить команду ответа в выбранный COM-порт. Назначение отправляемых данных определяется кодом функции (в соответствии со стандартом). Контрольная сумма будет автоматически рассчитана и добавлена в отправляемую посылку. Данную функцию допустимо применять только для ответов (когда модуль выступает в качестве ведомого). Для отправки запросов, необходимо применять функцию **Write_Request_Modbus**, описание которой приведено выше.

Параметры:

port — COM-порт контроллера. Для удобства ввода данного параметра рекомендуется использовать константы, описанные в начале раздела.

Addres – адрес устройства (в соответствии со стандартом Modbus RTU).

Func – код функции. Допустимые коды функций: 0x01, 0x02, 0x03, 0x04, 0x05, 0x06, 0x0F, 0x10, а также эти же коды с установленным битом ошибки: 0x81, 0x82, 0x83, 0x84, 0x85, 0x86, 0x8F, 0x90 (в соответствии со стандартом Modbus RTU).

SubFunc – код подфункции (адрес регистра в соответствии со стандартом Modbus RTU).

***Data** – массив байт записываемых данных.

Amount – количество записываемых элементов (регистров или ячеек в зависимости от выбранной функции).

Возвращаемое значение:

Возвращаемое значение:

Функция возвращает одно из приведенных ниже значений:

Описание библиотек

- 0x00 – успешное завершение работы;
- 0x80 – не поддерживаемый номер функции;
- 0xFF – не корректный номер COM порта.

Пример COM_1:

Данная программа выполняет поиск устройств на шине RS485, отправляя команду чтения конфигурации модулям серии NL. Ответы от обнаруженных устройств, программа отображает на индикаторе. В общем виде команда выглядит следующим образом:

Запрос: \$AA2/g

Где:

- \$ - символ идентификации группы команд;
- AA-адрес модуля;
- 2-символ идентификации команды;
- /g-возврат каретки (код 0x0D);

Ответ: !AATTCCFF/g

Где:

- !-признак успешного выполнения команды;
- AA-адрес модуля;
- TT-код диапазона;
- CC-код скорости связи;
- FF-формат команды;

Более подробную информацию по команде, можно найти в руководстве по эксплуатации на модули серии NL.

Программа перебирает все адреса от 1 до 255 и останавливает поиск, когда найдены два устройства или опрошены все адреса. Устройства должны быть подключены к порту COM1, иметь скорость связи 9600 бит/с и быть настроены на работу с протоколом DCON без контрольной суммы.

```
#define F_CPU 14745600UL /*Стандартная константа, задающая частоту микроконтроллера*/  
  
#include "MC12D4R40.h" //Тип контроллера  
  
#define COM_RX_BUFFER_SIZE 32 //Установить размер приемного буфера  
#define COM_TX_BUFFER_SIZE 32 //Установить размер передающего буфера  
  
#define COM1_PROTOCOL DCON // установить для порта COM1 протокол DCON  
  
#include <avr/io.h> //Стандартная библиотека ввода-вывода  
#include "LED.h" //Библиотека работы с LED дисплеем  
#include "COM.h" //Библиотека работы с RS485
```

1.5. Библиотека COM.h

```
//----- Инициализация RAM -----
char buf[16]; //Массив данных
uint8_t temp; //Вспомогательная переменная
uint8_t Find_Ok; //Признак успешного поиска устройства (модуль ответил на команду)
uint8_t Addres=0x01; //Адрес устройств для поиска
char char2[2]; //Массив для преобразования числа в строку

//-----
//Функция перевода двоичного числа в двухсимвольное шестнадцатеричное
//-----
void hex_to_char2(char s1)
{
    char temp; //Вспомогательная переменная

    temp=s1>>4;
    if (temp<=9) char2[0]=temp+0x30; //Если первый символ цифра (0-9)
    else char2[0]=temp+0x37; //Если первый символ буква (A-F)
    temp=s1 & 0x0F;
    if (temp<=9) char2[1]=temp+0x30; //Если второй символ цифра (0-9)
    else char2[1]=temp+0x37; //Если второй символ буква (A-F)
}

//-----
//Основная программа
//-----
int main(void)
{
    _delay_ms(500); /*Начальная задержка для стабилизации питания и инициализации модулей*/

    LED_Init(); //Инициализация индикатора
    COM_Init(COM_PORT1,9600); //Инициализация COM порта

//----- Поиск первого устройства -----
    LED_SetPos(0,0); //Установить курсор индикатора в левый верхний угол
    Find_Ok=0; //Сбросить флаг обнаружения устройства

    while((Find_Ok==0) && (Addres!=0x00)) /*Выполнять поиск пока
    устройство не ответит, либо не будут опрошены все адреса*/
    {
        buf[0]='$'; //Формирование команды
        hex_to_char2(Addres); /*Преобразовать адрес устройства к сим-
        вольному виду*/

        buf[1]=char2[0];
        buf[2]=char2[1];
        buf[3]='2';
        buf[4]=0; //Признак окончания строки команды (можно указы-
        вать символ 0x0D)*/

        Write_DCON(COM_PORT1,buf); //Отправить команду по RS485
    }
}
```

Описание библиотек

```
        _delay_ms(200);                //Ожидание ответа модуля

        if (Read_DCON(COM_PORT1,buf)==1) //Если ответ получен
        {
                temp=buf[0];           //Определить длину принятого ответа
                buf[temp]=0;           /*Вставить в буфер признак окончания
строки (чтобы на экран не попал лишний мусор из буфера)*/
                Find_Ok=1; //Установить флаг обнаружения устройства

                LED_Write_String(&buf[1]); /*Вывести ответ устройства на
индикатор*/
        }
        Address++;                     /*Увеличить адрес для поиска следующе-
го устройства*/
    }
}

//----- Поиск второго устройства -----
LED_SetPos(1,0); //Установить курсор индикатора в левый нижний угол
Find_Ok=0;      //Сбросить флаг обнаружения устройства

while((Find_Ok==0) && (Address!=0x00)) /*Выполнять поиск пока
устройство не ответит, либо не будут опрошены все адреса*/
{
        buf[0]='$';                    //Формирование команды
        hex_to_char2(Address);         /*Преобразовать адрес устройства к сим-
вольному виду*/
        buf[1]=char2[0];
        buf[2]=char2[1];
        buf[3]='2';
        buf[4]=0;                      /*Признак окончания строки команды (можно ука-
зывать символ 0x0D)*/

        Write_DCON(COM_PORT1,buf);    //Отправить команду по RS485
        _delay_ms(200);                //Ожидание ответа модуля

        if (Read_DCON(COM_PORT1,buf)==1) //Если ответ получен
        {
                temp=buf[0];           //Определить длину принятого ответа
                Find_Ok=1;             /*Вставить в буфер признак окончания
строки (чтобы на экран не попал лишний мусор из буфера)*/
                buf[temp]=0;           /*Установить флаг обнаружения устрой-
ства*/

                LED_Write_String(&buf[1]); /*Вывести ответ устройства на
индикатор*/
        }
        Address++; //Увеличить адрес для поиска следующего устройства
    }
    Led(LD_ON); //Включить светодиод
}
```

1.5. Библиотека COM.h

Пример COM_2:

Данный пример является шаблоном программы для модулей серии NL. В нем реализованы такие команды как:

- Чтения и записи адреса модуля;
- Чтения и записи скорости связи;
- Чтения и записи имени модуля;
- Чтения версии программы;

В программе продублирована работа с обоими COM портами. Никаких принципиальных изменений в работе портов нет. Протокол связи Modbus RTU.

/*

Данный пример является шаблоном программы для модулей серии NL. В нем реализованы такие команды как:

- Чтения и записи адреса модуля;
- Чтения и записи скорости связи;
- Чтения и записи имени модуля;
- Чтения версии программы;

В программе продублирована работа с обоими COM портами. Никаких принципиальных изменений в работе портов нет. Протокол связи Modbus RTU.

*/

```
#define F_CPU 14745600UL
#include "MC12D4R4O.h"
#include <avr/io.h>
#include <avr/eeprom.h>
```

```
#define COM_RX_BUFFER_SIZE 32
#define COM_TX_BUFFER_SIZE 32
```

```
#define COM1_PROTOCOL MODBUS
#define COM2_PROTOCOL MODBUS
#include "COM.h"
```

```
//----- Инициализация EEPROM -----
unsigned char EE_Address_device EEMEM=0x01;
unsigned char EE_COM_Speed EEMEM=0x06;
```

```
unsigned char EE_Parity EEMEM=0;
unsigned char EE_Stop_Bit EEMEM=1;
```

```
unsigned char EE_Name[] EEMEM="MC12D4R4O";
unsigned char EE_Version[] EEMEM=__DATE__;
```

```
//----- Инициализация RAM -----
char Address_device;
char COM_Speed;
```

Описание библиотек

```
char Name[10];

char Func;
uint16_t SubFunc;
int Data;

char buf[32];
int Address_EEP;

//===== Функция инициализации модуля =====
void Init(void)
{
    int temp;
    long Speed;
    Address_device=eeprom_read_byte(&EE_Address_device);
    COM_Speed=eeprom_read_byte(&EE_COM_Speed);

    switch (COM_Speed)
    {
        case 0x03: Speed=1200; break;
        case 0x04: Speed=2400; break;
        case 0x05: Speed=4800; break;
        case 0x06: Speed=9600; break;
        case 0x07: Speed=19200; break;
        case 0x08: Speed=38400; break;
        case 0x09: Speed=57600; break;
        case 0x0A: Speed=115200; break;
        default: Speed=9600;
    }

    COM_Init(COM_PORT1,Speed,eeprom_read_byte(&EE_Parity),eeprom_read_byte(&E
E_Stop_Bit));
    COM_Init(COM_PORT2,Speed,eeprom_read_byte(&EE_Parity),eeprom_read_byte(&E
E_Stop_Bit));

    Address_EEP=(int)&EE_Name;

    for ( temp=0; temp<10; temp++)
    Name[temp]=eeprom_read_byte(Address_EEP+(uint8_t*)temp);
}

//==== Функция декодирования команд полученных по COM портам ====
void Command_decode(char Port)
{
    char temp;
    int int_temp;

    if (buf[1]==Address_device)
    {
        Func=buf[2];
        switch(Func)
```

1.5. Библиотека COM.h

```
    {
    case 0x03:
        SubFunc=(buf[3]<<8)|buf[4];
        Data=(buf[5]<<8)|buf[6];

        switch (SubFunc)
        {
//----- Чтение имени модуля -----
        case 0x00C8:
            if (Data!=4)
            {
                buf[0]=0x03;

                Write_Response_Modbus(Port,Address_device,Func | 0x80,SubFunc,buf,1);
                break;
            }

            for (int_temp=0; int_temp<8; int_temp++)
buf[int_temp]=Name[int_temp];
                Write_Response_Modbus(Port,Address_device,Func,SubFunc,buf,4);
                break;

//----- Чтение версии программы -----
        case 0x00D4:
            if (Data>10)
            {
                buf[0]=0x03;

                Write_Response_Modbus(Port,Address_device,Func | 0x80,SubFunc,buf,1);
                break;
            }

            for (int_temp=0; int_temp<6; int_temp++)

            for (int_temp=0; ; int_temp++)
            {

buf[int_temp]=eeprom_read_byte(&EE_Version[0]+int_temp);
                if (buf[int_temp]==0) break;
            }

                Write_Response_Modbus(Port,Address_device,Func,SubFunc,buf,Data);
                break;

//----- Чтение адреса -----
        case 0x0200:
            if (Data!=1)

                //Если количество запрашиваемых регистров не
соответствует команде
```

Описание библиотек

```
        {
            buf[0]=0x03;

Write_Response_Modbus(Port,Address_device,Func | 0x80,SubFunc,buf,1);
//Ответ "Ошибка"
            break;
        }

        temp=eeprom_read_byte(&EE_Address_device);

//Копирование адреса устройства из EEPROM в ОЗУ
        buf[0]=0;
        buf[1]=temp;

Write_Response_Modbus(Port,Address_device,Func,SubFunc,buf,1);
        break;

//----- Чтение скорости -----
        case 0x0201:
            if (Data!=1)

//Если количество за-прашиваемых регистров не
соответствует команде
            {
                buf[0]=0x03;

Write_Response_Modbus(Port,Address_device,Func | 0x80,SubFunc,buf,1);
//Ответ "Ошибка"
                break;
            }

            temp=eeprom_read_byte(&EE_COM_Speed);

//Копирование скорости связи из EEPROM в ОЗУ
            buf[0]=0;
            buf[1]=temp;

Write_Response_Modbus(Port,Address_device,Func,SubFunc,buf,1);
            break;

//----- Чтение контроля паритета и количества стоп бит -----
        case 0x020A:
            if (Data!=1)

//Если количество запрашиваемых регистров не
соответствует команде
            {
                buf[0]=0x03;

Write_Response_Modbus(Port,Address_device,Func | 0x80,SubFunc,buf,1);
//Ответ "Ошибка"
```

1.5. Библиотека COM.h

```
                break;
            }
            buf[0]=eeprom_read_byte(&EE_Parity);

//Чтение контроля паритета
            buf[1]=eeprom_read_byte(&EE_Stop_Bit);

//Чтение количества стоп битов
            Write_Response_Modbus(Port,Address_device,Func,SubFunc,buf,1);
            break;

//----- Не поддерживаемый код подфункции -----
            default:

                buf[0]=0x02;

                //Ответ "Ошибка"
                Write_Response_Modbus(Port,Address_device,Func |
0x80,SubFunc,buf,1);
            }
            break;

            case 0x06:

                //Функция записи одиночного регистра (0x06)
                SubFunc=(buf[3]<<8)|buf[4];

                //Определение кода подфункции
                Data=(buf[5]<<8)|buf[6];

                //Определение данных

                switch (SubFunc)

                //Выбор в зависимости от полученного кода подфункции
                {
//----- Запись адреса -----
                case 0x0200:
                    if ((Data==0) || (Data>0xF7))
                    {
                        buf[0]=0x03;
                        Write_Response_Modbus(Port,Address_device,Func |
0x80,SubFunc,buf,1);
                        break;
                    }

                    eeprom_write_byte(&EE_Address_device,Data);

                    buf[0]=Data>>8;
                    buf[1]=Data & 0x00FF;
```

Описание библиотек

```
Write_Response_Modbus(Port,Address_device,Func,SubFunc,buf,1);
break;

//----- Запись скорости связи -----
case 0x0201:
    if ((Data<0x03) || (Data>0x0A))
    {
        buf[0]=0x03;

        Write_Response_Modbus(Port,Address_device,Func | 0x80,SubFunc,buf,1);
        break;
    }

    eeprom_write_byte(&EE_COM_Speed,Data);

    buf[0]=Data>>8;
    buf[1]=Data & 0x00FF;

    Write_Response_Modbus(Port,Address_device,Func,SubFunc,buf,1);
    break;

//----- Не поддерживаемый код подфункции -----
default:

    buf[0]=0x02;
    Write_Response_Modbus(Port,Address_device,Func |
0x80,SubFunc,buf,1);

//----- Запись контроля паритета и количества стоп битов -----
case 0x020A:
    if (((Data>>8)>2) || ((Data & 0x00FF)==0) || ((Data &
0x00FF)>2))
    {
        buf[0]=0x03;

        Write_Response_Modbus(Port,Address_device,Func | 0x80,SubFunc,buf,1);
        break;
    }

    eeprom_write_byte(&EE_Parity,Data>>8);

    eeprom_write_byte(&EE_Stop_Bit,Data & 0x00FF);

    buf[0]=Data>>8;
    buf[1]=Data & 0x00FF;

    Write_Response_Modbus(Port,Address_device,Func,SubFunc,buf,1);
    break;
}
break;
```

1.5. Библиотека COM.h

```
        case 0x10:
            SubFunc=(buf[3]<<8)|buf[4];
            int_temp=(buf[5]<<8)|buf[6];

            switch (SubFunc)
            {
//----- Запись имени модуля -----
                case 0x00C8:
                    if (int_temp!=4)
                    {
                        buf[0]=0x03;
                        Write_Response_Modbus(Port,Address_device,Func |
0x80,SubFunc,buf,1);

                        break;
                    }

                    Address_EEP=(int)&EE_Name;
                    for (int_temp=0; int_temp<8; int_temp++)
                    {
                        Name[int_temp]=buf[int_temp+8];
                        eeprom_write_byte(Address_EEP+(uint8_t*)int_temp,Name[int_temp]);
                    }

                    Write_Response_Modbus(Port,Address_device,Func,SubFunc,buf,5);
                    break;

//----- Не поддерживаемый код подфункции -----
                default:
                    buf[0]=0x02;
                    Write_Response_Modbus(Port,Address_device,Func |
0x80,SubFunc,buf,1);

                    break;

//----- Не поддерживаемый код функции -----
                default:
                    buf[0]=0x01;
                    Write_Response_Modbus(Port,Address_device,Func |
0x80,SubFunc,buf,1);
            }
        }
    }

//===== Основная программа =====
int main(void)
{
    Init();

    while(1)
    {
        if (Read_Modbus(COM_PORT1,buf)==1)
```

Описание библиотек

```
Command_decode(COM_PORT1);
    if (Read_Modbus(COM_PORT2,buf)==1) Com-
mand_decode(COM_PORT2);
    }
}
```

1.6. Библиотека RTC.h

Функции для работы с часами реального времени

Функции этой библиотеки позволяют осуществлять установку и чтение даты и времени.

Ниже представлены прототипы функций, описанных в данной библиотеке.

char RTC_Init(void);

Функция проводит инициализацию интерфейса I2C для связи микроконтроллера с микросхемой часов реального времени. Данную функцию достаточно вызвать один раз, при инициализации контроллера.

Возвращаемое значение:

Функция возвращает одно из приведенных ниже значений:

0x00 – нет ошибок;

0x01 – ошибка связи с микросхемой часов реального времени;

char RTC_Set_Time(unsigned char hour, unsigned char min, unsigned char sec);

Функция осуществляет установку времени в микросхеме (часы, минуты, секунды).

Параметры:

hour – часы (от 0 до 23).

min – минуты (от 0 до 59);

sec – секунды (от 0 до 59);

Возвращаемое значение:

Функция возвращает одно из приведенных ниже значений:

0x00 – нет ошибок;

1.6. Библиотека RTC.h

0x01 – ошибка связи с микросхемой часов реального времени;

char RTC_Get_Time(unsigned char* hour, unsigned char* min, unsigned char* sec);

Функция осуществляет чтение времени из микросхемы (часы, минуты, секунды).

Параметры:

В качестве параметров указываются адреса переменных, в которых необходимо разместить прочитанную информацию.

&hour – часы (от 0 до 23).

&min – минуты (от 0 до 59);

&sec – секунды (от 0 до 59);

Возвращаемое значение:

Функция возвращает одно из приведенных ниже значений:

0x00 – нет ошибок;

0x01 – ошибка связи с микросхемой часов реального времени;

char RTC_Set_Date(unsigned char day, unsigned char date, unsigned char month, unsigned char year);

Функция осуществляет установку даты в микросхеме (число, месяц, год).

Параметры:

day – день недели (от 1 до 7);

date – число (от 1 до 31);

month – месяц (от 1 до 12);

year – год (от 0 до 99);

Возвращаемое значение:

Функция возвращает одно из приведенных ниже значений:

0x00 – нет ошибок;

0x01 – ошибка связи с микросхемой часов реального времени;

char RTC_Get_Date(unsigned char* day, unsigned char* date, unsigned char* month, unsigned char* year);

Описание библиотек

Функция осуществляет чтение даты из микросхемы (число, месяц, год).

Параметры:

В качестве параметров указываются адреса переменных, в которых необходимо разместить прочитанную информацию.

&day – день недели (от 1 до 7);

&date – число (от 1 до 31);

&month – месяц (от 1 до 12);

&year – год (от 0 до 99);

Возвращаемое значение:

Функция возвращает одно из приведенных ниже значений:

0x00 – нет ошибок;

0x01 – ошибка связи с микросхемой часов реального времени;

Пример RTC_1:

Данная программа превращает ПЛК в часы. Для входа в режим установки часов необходимо нажать левую кнопку, при этом младший разряд значения часов будет подсвечен курсором и начнет мерцать. Нажимая вторую и третью кнопку, можно установить требуемое значение часов. Нажав еще раз левую кнопку, программа перейдет к режиму настройки минут, а при повторном нажатии, к режиму сброса секунд. В режиме сброса секунд, секунды обнуляются, а значение часов и минут корректируются в зависимости от того какое значение имели секунды при сбросе. Если их значение было меньше 30, минуты и часы не изменят свое значение, если же значение секунд было больше 30, показания часов увеличатся на 1 минуту.

```
#define F_CPU 14745600UL           /*Стандартная константа, задающая частоту микро-
контроллера*/

#include "MC12D6R.h"              //Тип контроллера
#include <avr/io.h>                //Стандартная библиотека ввода-вывода
#include "LED.h"                  //Библиотека работы с LED дисплеем
#include "RTC.h"                  /*Библиотека работы с микросхемой часов реально-
го времени*/

#include "Key.h"                  //Библиотека работы с кнопками

uint8_t hour, min, sec;          //Переменные часов, минут, секунд
uint8_t TimeSet=0;               //Флаг режима установки времени
uint8_t Error;                  //Флаг ошибки связи с RTC
```

1.6. Библиотека RTC.h

```
char String[11]; //Буфер для текстовой строки

int main(void)
{
    LED_Init(); //Инициализация индикатора
    RTC_Init(); //Инициализация RTC
    Key_Init(LATCH_OFF); //Инициализация кнопок

    while(1) //Бесконечный цикл
    {
        Error=0; //Обнулить флаг ошибки RTC

        //Прочитать значения часов, минут, секунд
        if (RTC_Get_Time(&hour,&min,&sec)==1) Error=1;

        //----- Кнопка "Настройка" -----
        if (Key(3)==DOWN) //Опрос 3 кнопки (крайне левая)
        {
            while(Key(3)==DOWN); //Ожидание отпускания кнопки
            TimeSet++; //Переключить режим настройки времени
            if (TimeSet>3) TimeSet=0; /*После настройки секунд
программа возвращается в рабочий режим*/
        }
        //----- Кнопка "+" -----
        if (Key(2)==DOWN) //Опрос 2 кнопки
        {
            switch(TimeSet) //Определение изменяемого параметра
            {
                case 0: break; /*Рабочий режим (выход без измене-
ний)*/

                case 1: //Режим изменения часов
                    hour++; //Увеличить значение часов
                    if (hour>23) hour=0; /*Коррекция часов, если их
значение вышло за предел*/

                    //Обновить данные в микросхеме
                    if (RTC_Set_Time(hour,min,sec)==1) Error=1;
                    break;

                case 2: //Режим изменения минут
                    min++; //Увеличить значение минут
                    if (min>59) min=0; /*Коррекция минут, если их
значение вышло за предел*/

                    //Обновить данные в микросхеме
                    if (RTC_Set_Time(hour,min,sec)==1) Error=1;
                    break;

                case 3: //Режим изменения секунд
                    sec=0; //Сброс секунд
                    if (sec>30) min++; /*Коррекция минут, если
округление секунд произошло в большую сторону*/
            }
        }
    }
}
```

Описание библиотек

```
if (min>59) { min=0; hour++;} /*Коррекция минут и
часов, если минуты вышли за предел*/
если они вышли за предел*/
if (hour>23) hour=0; /*Коррекция часов,
//Обновить данные в микросхеме
if (RTC_Set_Time(hour,min,sec)==1) Error=1;
break;
}
_delay_ms(100); //Программная задержка
}
//----- Кнопка "-" -----
if (Key(1)==DOWN) //Опрос 1 кнопки
{
switch(TimeSet) //Определение изменяемого параметра
{
case 0: break; //Рабочий режим (выход без изменений)
case 1: //Режим изменения часов
hour--; //Уменьшить значение часов
if (hour==255) hour=23; /*Коррекция часов,
если их значение вышло за предел*/
//Обновить данные в микросхеме
if (RTC_Set_Time(hour,min,sec)==1) Error=1;
break;
case 2: //Режим изменения минут
min--; //Уменьшить значение минут
if (min==255) min=59; /*Коррекция минут,
если их значение вышло за предел*/
//Обновить данные в микросхеме
if (RTC_Set_Time(hour,min,sec)==1) Error=1;
break;
case 3: //Режим изменения секунд
sec=0; //Сброс секунд
if (sec>30) min--; /*Коррекция минут, если
округление секунд произошло в большую сторону*/
if (min==255) { min=59; hour--;} /*Коррекция минут
и часов, если минуты вышли за предел*/
if (hour==255) hour=0; /*Коррекция часов,
если они вышли за предел*/
//Обновить данные в микросхеме
if (RTC_Set_Time(hour,min,sec)==1) Error=1;
break;
}
_delay_ms(100); //Программная задержка
}
String[0]=hour/10+0x30; /*Привести значение часов к
символьному виду*/
String[1]=hour%10+0x30;
String[2]=': '; //Вставить разделитель
```

1.6. Библиотека RTC.h

```
String[3]=min/10+0x30;           /*Привести значение минут к
символьному виду*/
String[4]=min%10+0x30;
String[5]=':';                   //Вставить разделитель

String[6]=sec/10+0x30;           /*Привести значение секунд к
символьному виду*/
String[7]=sec%10+0x30;
String[8]=0x20;                  //Очистка буфера
String[9]=0x20;                  //Очистка буфера
String[10]=0;                    //Признак конца строки

if (Error==1)                    //Если была ошибка RTC
{
    LED_SetPos(0,0);             /*Установка курсора в левую
позицию верхней строки*/
    LED_Write_String("Ошибка RTC"); /*Вывод времени на
индикатор*/
}
else
{
    LED_SetPos(0,0);             /*Установка курсора в левую
позицию верхней строки*/
    LED_Write_String(String);     //Вывод времени на индикатор

    switch(TimeSet)              //Определение положения курсора
    {
        //Выключить отображение курсора
        case 0: LED_CursorEnable(0); break;

        //Установить курсор на значение часов
        case 1: LED_CursorEnable(1); LED_SetPos(0,1); break;

        //Установить курсор на значение минут
        case 2: LED_CursorEnable(1); LED_SetPos(0,4); break;

        //Установить курсор на значение секунд
        case 3: LED_CursorEnable(1); LED_SetPos(0,7); break;

    }

    Led(LD_SW);                  //Переключить светодиод
    _delay_ms(100);              //Программная задержка
}
}
```

1.7. Библиотека 1Wire.h

Функции для работы датчиком температуры платы

Функции этой библиотеки позволяют измерить температуру платы цифровым датчиком DS18B20.

Ниже представлены прототипы функций, описанных в данной библиотеке.

void Temperature_Board_Start(void);

Функция выполняет инициализацию датчика температуры платы и запускает измерение температуры. Данную функцию необходимо вызывать для каждого нового измерения температуры.

uint8_t Temperature_Board_Read(int* Temperature);

Функция выполняет чтение данных с датчика температуры платы. Для преобразования температуры датчиком требуется 750мс, поэтому пользователю необходимо выдержать данный временной интервал, между функциями запуска преобразования и чтения температуры. Функция осуществляет проверку контрольной суммы прочитанных данных с датчика и возвращает нулевое значение в случае ошибки.

Параметры:

В качестве параметра указывается адрес переменной, в которую необходимо поместить прочитанную температуру.

Temperature – прочитанная температура.

Возвращаемое значение:

Функция возвращает 1 в случае успешного завершения работы, либо 0 в случае, если возникла ошибка контрольной суммы.

Пример 1Wire_1:

Данная программа проводит измерения температуры платы и отображает ее значение на индикаторе.

```
#define F_CPU 14745600UL //Стандартная константа задающая частоту микроконтроллера

#include "MC12D4R4O.h" //Тип контроллера

#include <avr/io.h> //Стандартная библиотека ввода-вывода
#include "LED.h" //Библиотека работы с LED дисплеем
#include "1Wire.h" //Библиотека работы с шиной 1-Wire
```

1.8. Библиотека ADS1248.h

```
#include <stdio.h>           /*Стандартная библиотека ввода-вывода (в данном примере
используется для преобразования числового значения в строку)*/

int main(void)
{
    LED_Init();             //Инициализация индикатора

    int temperature;       //Переменная для прочитанного значения температуры
    char String[16];       //Текстовая строка для вывода на индикатор

    while (1)              //Бесконечный цикл
    {
        Temperature_Board_Start(); //Запуск измерения температуры платы
        _delay_ms(750);          /*Задержка 750мс на преобразование
температуры датчиком*/

        if (Temperature_Board_Read(&temperature)) //Чтение температуры
        {
            LED_SetPos(0,0);      /*Установка курсора в левую
позицию верхней строки*/
            LED_Write_String("Температура "); /*Вывод сообщения
"Температура"*/
            sprintf(String, "%d", temperature/10); /*Преобразования
числа в строку*/
            LED_Write_String(String); //Вывод целой части
            LED_Write_String("."); //Вывод разделительной точки
            sprintf(String, "%d", temperature%10); /*Преобразования
числа в строку*/
            LED_Write_String(String); //Вывод дробной части
        }
        else                    //Если была ошибка чтения контрольной суммы
        {
            LED_SetPos(0,0);      /*Установка курсора в левую
позицию верхней строки*/
            LED_Write_String("Ошибка!"); /*Вывод сообщения
об ошибке*/
        }
    }
}
```

1.8. Библиотека ADS1248.h

Данная библиотека содержит функции управления АЦП для работы в режиме TI и RTD. Функции управления АЦП позволяют обеспечить настройку АЦП, получения данных и калибровку. При использовании

Описание библиотек

функций данной библиотеки рекомендуется использовать оптимизацию кода программы. Уровень оптимизации может быть произвольным.

В библиотеке предусмотрены следующие определения, упрощающие использование функций.

```
#define TYPE_SETTINGS_FOR_RTD_CH_0 0x00 /*тип настроек для RTD
канала 0*/
#define TYPE_SETTINGS_FOR_RTD_CH_1 0x01 /*тип настроек для RTD
канала 1*/
#define TYPE_SETTINGS_FOR_TI_CH_0 0x10 /*тип настроек для TI
канала 0*/
#define TYPE_SETTINGS_FOR_TI_CH_1 0x11 /*тип настроек для TI
канала 1*/
#define TYPE_SETTINGS_FOR_TI_CH_2 0x12 /*тип настроек для TI
канала 2*/
#define TYPE_SETTINGS_FOR_TI_CH_3 0x13 /*тип настроек для TI
канала 3*/
```

Ниже представлены прототипы функций и процедур данной библиотеки.

void Init_ADS1248(void);

Процедура осуществляет инициализацию работы с АЦП. Без предварительного вызова данной процедуры работа с АЦП невозможна. Данную функцию достаточно вызвать один раз, при инициализации контроллера.

void Reset_ADS1248(void);

Процедура осуществляет перезагрузку и сброс АЦП до значений по умолчанию. Рекомендуется использовать данную процедуру после инициализации для исключения ошибок связанных с неверной настройкой АЦП.

void write_ADS1248(uint8_t type);

Процедура записи настроек в регистры АЦП. После завершения записи в регистры запускает одиночное преобразование, которое занимает около 75 мс по истечению которого, результат можно получить функцией чтения данных uint32_t read_ADS1248(void).

Параметры:

type – Тип настроек, которые будут записаны в регистры АЦП, в данной библиотеке реализованы настройки для следующих типов работы:

1. С термосопротивлением типа Pt100 с возможным 2 каналов (канал 0 использует входы Ain0 – Ain3, канал 1 – входы Ain4 – Ain7), на рис. 1 представлено подключение 2 каналов RTD по четырех проводной схеме подключения.

1.8. Библиотека ADS1248.h

2. С термопарами типа J с возможным 4 каналов (канал 0 использует входы Ain0 – Ain1, канал 1 – входы Ain2 – Ain3, канал 2 – входы Ain4 - Ain5, канал 3 – входы Ain6 – Ain7), на рис. 2 представлено подключение 4 каналов ТТ.

В случае необходимости использования других типов или другого расположения контактов необходимо внесение изменений или добавление нового типа с Вашими настройками согласно Data sheet ADS1248.

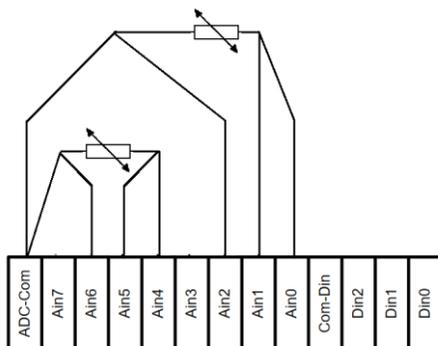


Рис. 1 – Пример подключения 2 каналов RTD

uint32_t read_ADS1248(void);

Функция чтения данных, полученных от АЦП. Запуск преобразования начинается после записи в регистры АЦП настроек процедурой записи настроек в регистры АЦП (`void write_ADS1248(uint8_t type)`). Время преобразования зависит от установленных настроек.

Возвращаемое значение:

Функция возвращает значение от АЦП, полученное после последнего преобразования. АЦП возвращает данные в формате 24 значащих бита. Возможные значение 00000000h – 00FFFFFFh. В случае использования настройки биполярного отображения данных значения 00800000h – 00FFFFFFh являются представлением в дополнительном коде.

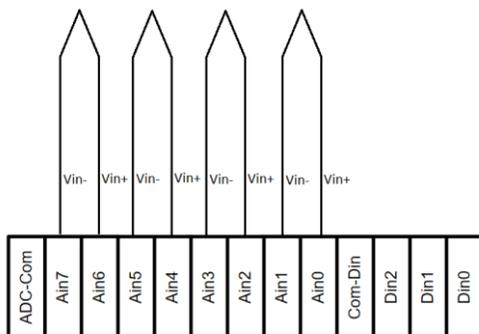


Рис. 2 – Пример подключения 4 каналов TI

uint32_t Calibrov_Max_RTD(uint8_t channel);

Функция калибровки усиления сигнала АЦП выбранного канала RTD. Значение усиления сохраняется в переменную и используется при записи в регистры настроек для АЦП, а также передается в виде возвращаемого значения.

Параметры:

channel – номер канала RTD на котором производится калибровка.

Возвращаемое значение:

Функция возвращает значение усиления сигнала АЦП. Возвращаемое значение имеет 24 значащих бита. Возможные значения 00000000h – 00FFFFFFh. Данное значение используется в регистре FSC АЦП для преобразования сигнала

uint32_t Calibrov_Min_RTD(uint8_t channel);

Функция калибровки смещения нуля сигнала АЦП выбранного канала RTD. Значение усиления сохраняется в переменную и используется при записи в регистры настроек для АЦП, а также передается в виде возвращаемого значения.

Параметры:

channel – номер канала RTD на котором производится калибровка.

Возвращаемое значение:

1.8. Библиотека ADS1248.h

Функция возвращает значение смещения нуля сигнала АЦП. Возвращаемое значение имеет 24 значащих бита. Возможные значения 00000000h – 00FFFFFFh. Данное значение используется в регистре OFC АЦП для преобразования сигнала

uint32_t Calibrov_Max_TI(uint8_t channel);

Функция калибровки усиления сигнала АЦП выбранного канала RTD. Значение усиления сохраняется в переменную и используется при записи в регистры настроек для АЦП, а также передается в виде возвращаемого значения.

Параметры:

channel – номер канала RTD на котором производится калибровка.

Возвращаемое значение:

Функция возвращает значение усиления сигнала АЦП. Возвращаемое значение имеет 24 значащих бита. Возможные значения 00000000h – 00FFFFFFh. Данное значение используется в регистре FSC АЦП для преобразования сигнала

uint32_t Calibrov_Min_TI(uint8_t channel);

Функция калибровки смещения нуля сигнала АЦП выбранного канала RTD. Значение усиления сохраняется в переменную и используется при записи в регистры настроек для АЦП, а также передается в виде возвращаемого значения.

Параметры:

channel – номер канала RTD на котором производится калибровка.

Возвращаемое значение:

Функция возвращает значение смещения нуля сигнала АЦП. Возвращаемое значение имеет 24 значащих бита. Возможные значения 00000000h – 00FFFFFFh. Данное значение используется в регистре OFC АЦП для преобразования сигнала

float calktemper(uint32_t adc_buf);

Функция для перерасчета полученных данных от АЦП в температуру для RTD для типа Pt100.

Параметры:

adc_buf – Данные полученные от функции uint32_t read_ADS1248(void).

Описание библиотек

Возвращаемое значение:

Функция возвращает значение температуры в формате числа с плавающей точкой с одинарной точности.

float TI_to_temperature(float voltage);

Функция для перерасчета напряжения в температуру для TI для типа J.

Параметры:

voltage – значение напряжение в формате числа с плавающей точкой с одинарной точности.

Возвращаемое значение:

Функция возвращает значение температуры в формате числа с плавающей точкой с одинарной точности. В случаях выхода за пределы предусмотренного аппроксимированными функциями согласно ГОСТ Р 8.585-2001 значение будет равно -INF или INF согласно формату чисел с плавающей точкой.

Пример Analog2:

Данная программа выводит значения с АЦП и данные о напряжении от термопары. В данной программе предусмотрено два режима работы ПЛК:

1. Рабочий режим. В данном режиме производится вывод на экран информации о текущих данных от АЦП в HEX и значение напряжения с точностью до двух знаков после запятой;
2. Режим калибровки. Нажатиями клавиш, расположенными на лицевой панели, производится калибровка смещения нуля канал 0 (клавиша вправо), усиления канал 0 (клавиша вниз), смещения нуля канал 1 (клавиша вверх), усиления канал 1 (клавиша влево).

Переход между режимами осуществляется нажатием всех 4 клавиш одновременно.

```
#define F_CPU 14745600UL //Стандартная константа, задающая частоту микроконтроллера
#include <avr/io.h> //Стандартная библиотека ввода-вывода
#include <avr/eeprom.h> //Стандартная библиотека работы с EEPROM
#include <avr/wdt.h> //Библиотека сторожевого таймера
#include <stdio.h>
#include <avr/interrupt.h> //Библиотека обработки прерываний
```

1.8. Библиотека ADS1248.h

```
#include "main.h"

#include "MC8U80.h"           //Тип контроллера
#include "LED.h"             //Библиотека работы с LED дисплеем
#include "Key.h"             //Библиотека кнопок
#include "ADS1248.h"

/*****
*****
*****

                               Инициализация таймера 0
*****
*****
*****/

void Init_Tim0(void)
{
    TCCR0 = (1<<WGM01) | (1<<CS02) | (1<<CS00);    //Устанавливаем
                                                    делитель (1024) и режим
                                                    таймера (сброс при совпадении
                                                    с регистром сравнения)

    TCNT0 = 0;                                     //Обнуляем счетный регистр
    OCR0 = 144;                                    //Устанавливаем регистр
                                                    сравнения (10 мс)

    TIMSK |= (1<<OCIE0);                          //Разрешаем прерывание по
                                                    совпадению с регистром
                                                    сравнения
}

/*****
*****
*****

                               Остановка таймера 0
*****
*****
*****/

void Stop_Tim0(void)
```

Описание библиотек

```
{
    TCCR0 = 0;
    TIMSK &= ~(1<<OCIE0);
}

char String[17]; //Буфер для текстовой строки
uint16_t cnt_tim0 = 0; //Счетчик для отслеживания 1
                        сек при помощи таймера 0
uint8_t Calibrationmode=0;

extern uint32_t OFC1, OFC0, FSC1, FSC0;

#define EEPROM_ADR_OFC0 (uint32_t*)0x00000200
#define EEPROM_ADR_OFC1 (uint32_t*)0x00000204
#define EEPROM_ADR_FSC0 (uint32_t*)0x00000208
#define EEPROM_ADR_FSC1 (uint32_t*)0x0000020C

uint32_t dataformADC[3] = {0,0,0};
float temp, temp2;
//===== Прерывание таймера =====//
ISR(TIMER0_COMP_vect)
{
    if (Calibrationmode == 0) //не в режиме калибровки
        switch(cnt_tim0) //проверка счетчика таймера
        {
            case100:
                write_ADS1248(TYPE_SETTINGS_FOR_TI_CH_0); break;
                //Записываем настройки для
                //работы с каналом 0 TI Type J
            case 200:
                dataformADC[0] = read_ADS1248();
                //считываем данные от АЦП
        }
    }
}
```

1.8. Библиотека ADS1248.h

```
        sprintf(String, "CH0 %4X%4X",
(uint16_t) (dataformADC[0]>>16), (uint16_t) (dataformADC[0]>>0));
        //Подготовка строки для вывода
        на экран
        LED_SetPos(0,0);        //Установка курсора в
        строке 0 на позицию 0
        LED_Write_String(String);    //Вывод на
        экран последнего полученного значения
        temp = (float)dataformADC[0]*70/0x7FFFFFFF;
        //Пересчет значения в мВ (70 указано
значение используемое в калибровке)
        sprintf(String, "CH0 %07d",
(int16_t) (temp*100));        //Подготовка строки для вывода
        на экран с точность 0.01
        LED_SetPos(1,0);        //Установка курсора в
        строке 1 на позицию 0
        LED_Write_String(String);    //Вывод на
        экран последнего значения в мВ
        cnt_tim0 = 0;        //Сброс счетчика
        таймера
        LED_RED_TUGGLE;
        break;
    }
    cnt_tim0++;        //наращивание счетчика таймера
    TCNT0 = 0;        //обнуления счетного регистра
        таймера
}
int main(void)        //Основная программа
{
    DDR_Led_Red |= (1<<P_Led_Red);    //Настраиваем красный
        светодиод на выход
    Init_ADS1248 ();        //Инициализация АЦП
```

Описание библиотек

```
_delay_ms(200);
Reset_ADS1248();           //Сброс до заводских настроек
                             АЦП

_delay_ms(200);
Init_Tim0();               //Инициализация таймера
LED_Init();                //Инициализация дисплея
Key_Init(0);               //Инициализация клавиш

sprintf(String, " Старт теста ");           //вывод на
                                             экран начального сообщения
LED_SetPos(0,0);           //вывод на экран начального
                             сообщения
LED_Write_String(String);  //вывод на экран начального
                             сообщения
sprintf(String, " ");       //вывод на
                             экран начального сообщения
LED_SetPos(1,0);           //вывод на экран начального
                             сообщения
LED_Write_String(String);  //вывод на экран начального
                             сообщения
sei();                      //Разрешаем прерывания
while (1)                   //Бесконечный цикл
{
    if(Key(0)==1 && Key(1)== 1 && Key(2)== 1 && Key(3)==
1 && Calibrationmode == 0) //проверка нажатия всех 4
                             кнопок для перехода в режим
                             калибровки
    {
        _delay_ms(500);     //проверка надребезг
                             контактов
        if (Key(0)==1 && Key(1)== 1 && Key(2)== 1 &&
Key(3)== 1) //проверка нажатия всех 4
```

1.8. Библиотека ADS1248.h

```
        кнопок для перехода в режим
        калибровки
    {
        Calibrationmode=1;    //установка
        флага о переходе в режим
        калибровки
        cli();                //запрет прерываний
        _delay_ms(100);
        sprintf(String, "Калибровка      ");
        //вывод на экран сообщения о
        переходе в режим калибровки
        LED_SetPos(0,0);      //вывод на
        экран сообщения о переходе в
        режим калибровки
        LED_Write_String(String); //вывод на
        экран сообщения о переходе в
        режим калибровки
        sprintf(String, "Начата          ");
        //вывод на экран сообщения о
        переходе в режим калибровки
        LED_SetPos(1,0);     //вывод на
        экран сообщения о переходе в
        режим калибровки
        LED_Write_String(String); //вывод на
        экран сообщения о переходе в
        режим калибровки
        _delay_ms(500);
    }
}

if(Calibrationmode == 1)    //проверка флага
    нахождения в режиме калибровки
{
    // если установлен флаг
    uint8_t temp[10];      //временная переменная
```

Описание библиотек

```
temp[0] =
Key(3)<<3|Key(2)<<2|Key(1)<<1|Key(0); //фиксируем состояния
всех клавиш
_delay_ms(500); //проверка надребезг
контактов

temp[1] =
Key(3)<<3|Key(2)<<2|Key(1)<<1|Key(0); //фик-
сируем состояния всех клавиш

if (temp[0] == temp[1]) //если пройдена
проверка надребезг контактов
{
switch (temp[1]) //выбор комби-
нации нажатых клавиш
{
case 0x0f://Нажаты все клавиши
Calibrationmode=0;
//выход из режима калибровки
_delay_ms(100);
printf(String, "Ка-
либровка "); //Вывод на экран сообщения о
выходе из режима калибровки
LED_SetPos(0,0);
//Вывод на экран сообщения о
выходе из режима калибровки
LED_Write_String(String); //Вывод на экран сообщения о
выходе из режима калибровки
printf(String, "За-
вершена "); //Вывод на экран сообщения о
выходе из режима калибровки
LED_SetPos(1,0);
//Вывод на экран сообщения о
выходе из режима калибровки
LED_Write_String(String); //Вывод на экран сообщения о
выходе из режима калибровки
```

1.8. Библиотека ADS1248.h

```
        _delay_ms(500);
        sei();
        break;
    case 0b0001: //Нажата
        клавиша 0
            _delay_ms(100);
            OFC0 =
Calibrov_Min_TI(0); //Выполнить калибровку смещения 0 для
                    канала 0 TI type J
            _delay_ms(100);
            eeprom_write_dword(EEPROM_ADR_OFC0, OFC0); //Сохранить
                    значение калибровки в eeprom
            _delay_ms(100);
            sprintf(String, "Ка-
либровка CH0 "); //Вывод на экран значения калибровки
                    LED_SetPos(0,0);
                    //Вывод на экран значения калибровки
            LED_Write_String(String); //Вывод на экран значения
                    калибровки
            sprintf(String, "Смещ
0 %4X%4X", (uint16_t)(OFC0>>16), (uint16_t)(OFC0>>0));
                    //Вывод на экран значения калибровки
                    LED_SetPos(1,0);
                    //Вывод на экран значения калибровки
            LED_Write_String(String); // Вывод на экран значения
                    калибровки

            break;
    case 0b0010:
        _delay_ms(100);
        FSC0 =
Calibrov_Max_TI(0); //Выполнить калибровку усиления для
                    канала 0 TI type J
        _delay_ms(100);
        eeprom_write_dword(EEPROM_ADR_FSC0, FSC0);
                    //Сохранить значение калибровки в
```

Описание библиотек

```
        eeprom

                _delay_ms(100);

                sprintf(String, "Ка-
либровка CH0 "); //Вывод на экран значения калибровки

                LED_SetPos(0,0);
                //Вывод на экран значения калибровки

                LED_Write_String(String); //Вывод на экран значения
                калибровки

                sprintf(String, "Усил
0 %4X%4X", (uint16_t) (FSC0>>16), (uint16_t) (FSC0>>0));
                //Вывод на экран значения калибровки

                LED_SetPos(1,0);
                //Вывод на экран значения калибровки

                LED_Write_String(String); //Вывод на экран значения
                калибровки

                break;

                case 0b0100:

                        _delay_ms(100);

                        OFC1 =
Calibrov_Min_TI(1); //Призвести калибровку смещения 0 для
                        канала 1 TI type J

                        _delay_ms(100);

                eeprom_write_dword(EEPROM_ADR_OFC1, OFC1);
                //Сохранить значение калибровки в
                eeprom

                        _delay_ms(100);

                        sprintf(String, "Ка-
либровка CH1 "); //Вывод на экран значения калибровки

                        LED_SetPos(0,0);
                        //Вывод на экран значения калибровки

                LED_Write_String(String); //Вывод на экран значения
                калибровки

                        sprintf(String, "Смещ
1 %4X%4X", (uint16_t) (OFC1>>16), (uint16_t) (OFC1>>0));
                        // Вывод на экран значения калибровки
```

1.8. Библиотека ADS1248.h

```
        LED_SetPos(1,0);
        //Вывод на экран значения калибровки
LED_Write_String(String);    //Вывод на экран значения
        калибровки

        break;

        case 0b1000:
            _delay_ms(100);

            FSC1 =
Calibrov_Max_TI(1);    //Выполнить калибровку усиления для
        канала 1 TI type J

            _delay_ms(100);

            eeprom_write_dword(EEPROM_ADR_FSC1, FSC1);
            //Сохранить значение калибровки в
            eeprom

            _delay_ms(100);
            //Вывод на экран значения калибровки
            sprintf(String, "Ка-
либровка CH1 ");
            //Вывод на экран значения калибровки
            LED_SetPos(0,0);

            //Вывод на экран значения калибровки
LED_Write_String(String);    //Вывод на экран значения
        калибровки

            sprintf(String, "Усил
1 %4X%4X", (uint16_t) (FSC1>>16), (uint16_t) (FSC1>>0));
            //Вывод на экран значения калибровки

            LED_SetPos(1,0);
            //Вывод на экран значения калибровки

LED_Write_String(String);    //Вывод на экран значения
        калибровки

            break;

        }

    }

}
}
```

1.9. Библиотека Analog.h

Функции управления аналоговыми входами панельного контроллера MC-4A7D4R4O

Функции данной библиотеки позволяют осуществлять запуск преобразования АЦП и чтение значения напряжения или тока на аналоговых входах панельного контроллера MC-4A7D4R4O. Выбор входного типа сигнала осуществляется подключением шунтирующего резистора при помощи движкового переключателя расположенного на обратной стороне печатного узла модуля. Когда переключатель находится в положении «On», на соответствующем ему аналоговом входе подключен шунтирующий резистор 124 Ома. Аналоговые входы позволяют измерять напряжение в диапазоне 0-2,5В или ток в диапазоне 0-20мА.

Для удобства использования функций в данной библиотеке объявлены следующие константы:

```
#define AIN0 0 /*Константа "0 аналоговый вход"*/
#define AIN1 1 /*Константа "1 аналоговый вход"*/
#define AIN2 2 /*Константа "2 аналоговый вход"*/
#define AIN3 3 /*Константа "3 аналоговый вход"*/

#define VOLTAGE 0 /*Константа "Входной диапазон напряжение 0-2.5 В"*/

#define CURRENT 1 /*Константа "Входной диапазон ток 0-20 мА"*/
```

Ниже представлены прототипы функций, описанных в данной библиотеке.

char ADC_Start_Measuring(char Channel, char Range);

Функция осуществляет настройку указанной линии аналогового входа на требуемый диапазон и запускает АЦП на преобразование. Время первого преобразования составляет 217мкс. Все последующие преобразования выполняются за 113мкс.

Параметры:

Channel - номер аналогового входа (от 0 до 3);

Range – входной диапазон (0-соответствует напряжению 0-2,5 В. 1-соответствует току 0-20 мА). Для удобства работы с данным параметром целесообразно применять константы, описанные в начале раздела.

1.9. Библиотека Analog.h

Возвращаемое значение:

Функция возвращает 0 в случае успешного завершения работы, либо 0xFF, в случае если параметры заданы не верно.

char ADC_Read(unsigned int *Data);

Функция осуществляет проверку данных на готовность и чтение значения преобразованного АЦП.

Параметры:

В качестве параметра указывается адрес переменной, в которую необходимо поместить прочитанное значение.

***Data** – прочитанное значение напряжения или тока представленное в милливольтгах или микроамперах в зависимости от выбранного диапазона.

Возвращаемое значение:

Функция возвращает 0 в случае успешного завершения работы, либо 1 в случае, если преобразование еще не завершено.

Пример Analog_1:

Данная программа проводит измерения тока по 0 каналу и измерение напряжения на каналах 1-3. Программа выводит на индикатор значения измеренных величин.

```
#define F_CPU 14745600UL           /*Стандартная константа, задающая частоту микро-
контроллера*/

#include <avr/io.h>                //Стандартная библиотека ввода-вывода

#include "MC4A7D4R4O.h"           //Тип контроллера
#include "LED.h"                   //Библиотека работы с LED дисплеем
#include "Analog.h"               //Библиотека работы с АЦП
#include <stdio.h>                 /*Стандартная библиотека ввода-вывода (в данном
примере используется для преобразования числового значения в строку)*/

char String[17];                  //Буфер для текстовой строки

int main(void)                    //Основная программа
{
    unsigned int result[4];       //Массив для чтения результата с АЦП.
    int Channel=0;                //Переменная для нумерации каналов
    LED_Init();                   //Инициализация индикатора

    while (1)                     //Бесконечный цикл
    {
```

Описание библиотек

```
/*Запуск преобразования АЦП для выбранного канала (для 0 канала токовый диапазон, для остальных потенциальный)*/
    if (Channel==0) ADC_Start_Measuring(Channel,CURRENT);
    else ADC_Start_Measuring(Channel,VOLTAGE);

/*Ожидание окончания преобразования и чтение результата*/
    while (ADC_Read(&result[Channel]));

/*Установка курсора в левую позицию верхней строки*/
    LED_SetPos(Channel/2,(Channel % 2)*9);

/*Вывод на индикатор целой части*/
    sprintf(String,"%d",result[Channel]/1000);
    LED_Write_String(String);

/*Вывод на индикатор разделительной точки*/
    LED_Write_String(".");

/*Преобразование дробной части в символьный вид*/
    result[Channel]=result[Channel] % 1000;
    String[5]=0;
    String[4]='.';
    String[3]=' ';
    String[2]=(result[Channel] % 10)+0x30;

/*Вывод на индикатор дробной части*/
    result[Channel]=result[Channel]/10;
    String[1]=(result[Channel] % 10)+0x30;
    String[0]=result[Channel]/10+0x30;
    LED_Write_String(String);

/*Переключение на следующий канал*/
    if (Channel<3) Channel++;
    else Channel=0;

/*Задержка на 100мс.*/
    _delay_ms(100);
}
}
```

1.10. Библиотека COM_SLAVE.h

Библиотека COM SLAVE позволяет реализовать ведомое устройство в протоколе Modbus RTU.

Библиотека поддерживает следующие функции Modbus RTU:

- 0x01 – чтение Coils;
- 0x02 – чтение discrete input;

1.10. Библиотека COM_SLAVE.h

- 0x03 – чтение holding register;
- 0x04 – чтение input register;
- 0x05 – запись Coils;
- 0x06 – запись holding register;
- 0x0F – запись нескольких Coils;
- 0x10 – запись нескольких holding register.

В библиотеке предусмотрены следующие константы, упрощающие использование функций:

```
#define ON 1 /*разрешает использование порта в режиме
ведомого устройства*/
#define OFF 0 /*запрещает использование порта в режиме
ведомого устройств

#define SLAVE_COM1 OFF /*константа, задающая режим работы порта
COM1. Для включения порта как ведомого
необходимо переопределить данную кон-
станту. Допустимые значения ON и OFF*/

#define SLAVE_COM2 OFF /*константа, задающая режим работы порта
COM2. Для включения порта как ведомого
необходимо переопределить данную кон-
станту. Допустимые значения ON и OFF */

#define COM_ERROR 0xFF /*ошибки определения номера порта*/
#define REG_ERROR 0xFE /*ошибка определения типа регистра*/
#define POSITION_ERROR 0xFC /*некорректный адрес регистра*/
#define LENGTH_ERROR 0xFA /*ошибка определения допустимого количе-
ства регистров*/
#define VALUE_ERROR 0xFB /*некорректное значение*/
#define GOOD 0 /*успешное выполнение функции*/
#define ERROR 1 /*ошибка выполнения функции*/
#define MODBUS_FUNC_ERROR 0x01 /*Принятый код функции не может быть
обработан*/
#define MODBUS_ADD_ERROR 0x02 /*некорректный адрес регистра */
#define MODBUS_DATA_ERROR 0x03 /*ошибка определения количества запраши-
ваемых регистров*/
#define MODBUS_NO_DATA_RECIVED F9 /*данных по modbus не поступало*/
#define SIZE_COIL_COM1 10 /*константа, определяющая размер дискрет-
ного выхода для порта COM 1 */
#define SIZE_COIL_COM2 10 /*константа, определяющая размер дискрет-
ного выхода для порта COM 2*/
```

Описание библиотек

```
#define SIZE_DISCRETE_INPUT_COM1 10 /*константа, определяющая размер дискретного
входа для порта COM 1*/

#define SIZE_DISCRETE_INPUT_COM2 10 /*константа, определяющая размер дискретного
входа для порта COM 2*/

#define SIZE_ANALOG_INPUT_COM1 10 /*константа, определяющая размер аналогового
входа для порта COM 1*/

#define SIZE_ANALOG_INPUT_COM2 10 /*константа, определяющая размер аналогового
входа для порта COM 2*/

#define SIZE_ANALOG_OUTPUT_COM1 10 /*константа, определяющая размер аналогового
выхода для порта COM 1*/

#define SIZE_ANALOG_OUTPUT_COM2 10 /*константа, определяющая размер аналогового
выхода для порта COM 2*/

#define COIL 0 /*тип регистра - дискретный выход (Coils) */

#define DISCRETE_INPUT 1 /*тип регистра - дискретный вход (Discrete
Input)*/

#define ANALOG_INPUT 2 /*тип регистра - аналоговый вход (Input
Registers)*/

#define ANALOG_OUTPUT 3 /*тип регистра - аналоговый выход (Holding
Registers) */
```

Также были предусмотрены 2 вспомогательные переменные ADDRESS_COM1 и ADDRESS_COM2 типа uint8_t. Они предназначены для определения адреса ведомого устройства. По умолчанию ADDRESS_COM1 = 1, ADDRESS_COM2 = 2.

При использовании данной библиотеки в начале исходного текста (до подключения библиотеки) необходимо объявить стандартную константу F_CPU, указывающую частоту кварца микроконтроллера (в стандартной комплектации контроллера частота кварца равна 14,7456 МГц). Если этого не сделать, компилятор выдаст предупреждение и установит частоту кварца по умолчанию 14,7456 МГц.

Ниже представлен пример объявления константы, задающая частоту микроконтроллера:

```
#define F_CPU 14745600UL /*Стандартная константа, задающая частоту
микроконтроллера*/
```

Также необходимо подключить библиотеку COM.h и выполнить инициализацию необходимых портов и констант согласно требованиям, описанным в разделе подключаемой библиотеки.

Ниже представлен пример:

1.10. Библиотека COM_SLAVE.h

```
#define COM_RX_BUFFER_SIZE 256    /*Размер приемного буфера 256
                                   байт*/
#define COM_TX_BUFFER_SIZE 256    /*Размер передающего буфера 256
                                   байт*/
#define COM1_PROTOCOL MODBUS     /*Установка работы COM1-порта по
                                   протоколу Modbus RTU*/

int main(){
COM_Init(COM_PORT1, 9600);        /*Инициализация COM1 порта*/
}
```

Для определения режима работы порта имеются две константы SLAVE_COM1 и SLAVE_COM2. Их также необходимо объявить до подключения библиотеки, если этого не сделать, то компилятор выдаст предупреждения. По умолчанию порты имеют значение OFF.

Ниже представлен пример объявления константы, отвечающая за режим работы порта COM1:

```
#define SLAVE_COM1 ON             /*Включение порта COM1 в режиме работы ведомого*/
```

Для определения размера Coils предусмотрены две константы SIZE_COIL_COM1 и SIZE_COIL_COM2, их необходимо объявить до подключения библиотеки, если этого не сделать компилятор выдаст предупреждение, значение по умолчанию установлено равным 10.

Ниже представлен пример определения размера для Coils:

```
#define SIZE_COIL_COM1 32        /*Стандартная константа, задающая размер Coils для COM1*/
#define SIZE_COIL_COM2 32        /*Стандартная константа, задающая размер Coils для COM2*/
```

Для определения размера Discrete Input предусмотрены две константы SIZE_DISCRETE_INPUT_COM1 и SIZE_DISCRETE_INPUT_COM2, их необходимо объявить до подключения библиотеки, если этого не сделать компилятор выдаст предупреждение, значение по умолчанию установлено равным 10.

Ниже представлен пример определения размера для Discrete Input:

```
#define SIZE_DISCRETE_INPUT_COM1 32    /*Стандартная константа, задающая размер Discrete Input для COM1*/
#define SIZE_DISCRETE_INPUT_COM2 32    /*Стандартная константа, задающая размер Discrete Input для COM2*/
```

Для определения размера Input Register предусмотрены две константы SIZE_ANALOG_INPUT_COM1 и SIZE_ANALOG_INPUT_COM2, их

Описание библиотек

необходимо объявить до подключения библиотеки, если этого не сделать компилятор выдаст предупреждение, значение по умолчанию установлено равным 10.

Ниже представлен пример определения размера для Input Register:

```
#define SIZE_ANALOG_INPUT_COM1 32      /*Стандартная константа, задающая раз-  
мер Input Register для COM1*/  
  
#define SIZE_ANALOG_INPUT_COM2 32      /*Стандартная константа, задающая раз-  
мер Input Register для COM2*/
```

Для определения размера Holding Register предусмотрены две константы SIZE_ANALOG_OUTPUT_COM1 и SIZE_ANALOG_OUTPUT_COM2, их необходимо объявить до подключения библиотеки, если этого не сделать компилятор выдаст предупреждение, значение по умолчанию установлено равным 10.

Ниже представлен пример определения размера для Holding Register:

```
#define SIZE_ANALOG_OUTPUT_COM1 32      /*Стандартная константа, задающая  
размер Holding Register для COM1*/  
  
#define SIZE_ANALOG_OUTPUT_COM2 32      /*Стандартная константа, задающая  
размер Holding Register для COM2*/
```

Ниже представлены прототипы функций библиотеки.

uint8_t Check_Slave(int port);

Параметры:

port – номер порта на котором необходимо обрабатывать запросы. Для удобства ввода данного параметра рекомендуется использовать константы COM_PORT1 и COM_PORT2 из библиотеки COM.

Функция Check_Slave предназначена для выполнения проверки поступивших запросов от ведущего устройства, обработки запроса и ответа.

Возвращаемые значения:

Возвращает GOOD, если операция выполнена успешно, данные были приняты. MODBUS_NO_DATA_RECIVED, если запрос от ведущего устройства не поступал.

Возвращает код ошибки в случае возникновения ошибки: COM_ERROR, VALUE_ERROR, POSITION_ERROR, MODBUS_FUNC_ERROR, MODBUS_DATA_ERROR, MODBUS_ADD_ERROR.

uint8_t Write_COIL(uint8_t selected_port, int position, uint8_t value);

1.10. Библиотека COM_SLAVE.h

Параметры:

selected_port – номер порта ведомого устройства. Для удобства ввода данного параметра рекомендуется использовать константы COM_PORT1 и COM_PORT2 из библиотеки COM.

position – адрес.

value – значение (0 или 1).

Данная функция предназначена для записи значения в Coils.

Возвращаемые значения:

Возвращает GOOD, если операция выполнена успешно.

Возвращает код ошибки в случае возникновения ошибки: VALUE_ERROR, POSITION_ERROR, COM_ERROR.

uint8_t Write_multi_COIL(uint8_t selected_port, uint8_t *data, uint16_t length, int position);

Параметры:

selected_port – номер порта ведомого устройства. Для удобства ввода данного параметра рекомендуется использовать константы COM_PORT1 и COM_PORT2 из библиотеки COM.

data – указатель на массив данных, подлежащих записи.

length – количество записываемых Coils.

position – начальный адрес.

Данная функция предназначена для множественной записи значений в Coils.

Возвращаемые значения:

Возвращает GOOD, если операция выполнена успешно.

Возвращает код ошибки в случае возникновения ошибки: VALUE_ERROR, COM_ERROR, POSITION_ERROR, LENGTH_ERROR.

uint8_t Read_COIL(uint8_t selected_port, int position, uint8_t* storage);

Параметры:

Описание библиотек

`selected_port` – номер порта ведомого устройства. Для удобства ввода данного параметра рекомендуется использовать константы `COM_PORT1` и `COM_PORT2` из библиотеки `COM`.

`position` – адрес.

`storage` – указатель на массив данных в котором необходимо сохранить считываемые значения.

Данная функция предназначена для считывания значения из `Coils`.

Возвращаемые значения:

Возвращает `GOOD`, если операция выполнена успешно.

Возвращает код ошибки в случае возникновения ошибки: `COM_ERROR`, `POSITION_ERROR`.

`uint8_t Write_DISCRETE_INPUT(uint8_t selected_port, int position, uint8_t value);`

Параметры:

`selected_port` – номер порта ведомого устройства. Для удобства ввода данного параметра рекомендуется использовать константы `COM_PORT1` и `COM_PORT2` из библиотеки `COM`.

`position` – адрес.

`value` – значение (0 или 1).

Данная функция предназначена для записи значения в `discrete input`.

Возвращаемые значения:

Возвращает `GOOD`, если операция выполнена успешно.

Возвращает код ошибки в случае возникновения ошибки: `VALUE_ERROR`, `COM_ERROR`, `POSITION_ERROR`.

`uint8_t Write_multi_DISCRETE_INPUT(uint8_t selected_port, uint8_t *data, uint16_t length, int position);`

Параметры:

`selected_port` – номер порта ведомого устройства. Для удобства ввода данного параметра рекомендуется использовать константы `COM_PORT1` и `COM_PORT2` из библиотеки `COM`.

`data` – указатель на массив данных, подлежащих записи.

1.10. Библиотека COM_SLAVE.h

length – количество записываемых discrete input.

position – начальный адрес.

Данная функция предназначена для множественной записи значений в discrete input.

Возвращаемые значения:

Возвращает GOOD, если операция выполнена успешно.

Возвращает код ошибки в случае возникновения ошибки: VALUE_ERROR, COM_ERROR, POSITION_ERROR, LENGTH_ERROR.

uint8_t Read_DISCRETE_INPUT(uint8_t selected_port, int position, uint8_t* storage);

Параметры:

selected_port – номер порта ведомого устройства. Для удобства ввода данного параметра рекомендуется использовать константы COM_PORT1 и COM_PORT2 из библиотеки COM.

position – адрес.

storage – указатель на хранилище данных в которое необходимо сохранить считываемые значения.

Данная функция предназначена для считывания значения из discrete input.

Возвращаемые значения:

Возвращает GOOD, если операция выполнена успешно.

Возвращает код ошибки в случае возникновения ошибки: POSITION_ERROR, COM_ERROR.

uint8_t Write_INPUT_REGISTER(uint8_t selected_port, int position, uint16_t value);

Параметры:

selected_port – номер порта ведомого устройства. Для удобства ввода данного параметра рекомендуется использовать константы COM_PORT1 и COM_PORT2 из библиотеки COM.

position – адрес.

value – значение.

Описание библиотек

Данная функция предназначена для записи значения в input register.

Возвращаемые значения:

Возвращает GOOD, если операция выполнена успешно.

Возвращает код ошибки в случае возникновения ошибки: COM_ERROR, POSITION_ERROR.

uint8_t Write_multi_INPUT_REGISTER(uint8_t selected_port, uint16_t *data, uint16_t length, int position);

Параметры:

selected_port – номер порта ведомого устройства. Для удобства ввода данного параметра рекомендуется использовать константы COM_PORT1 и COM_PORT2 из библиотеки COM.

data – указатель на массив данных, подлежащих записи.

length – количество записываемых input register.

position – начальный адрес.

Данная функция предназначена для множественной записи значений в input register.

Возвращаемые значения:

Возвращает GOOD, если операция выполнена успешно.

Возвращает код ошибки в случае возникновения ошибки: COM_ERROR, POSITION_ERROR, LENGTH_ERROR.

uint8_t Read_INPUT_REGISTER(uint8_t selected_port, int position, uint16_t* storage);

Параметры:

selected_port – номер порта ведомого устройства. Для удобства ввода данного параметра рекомендуется использовать константы COM_PORT1 и COM_PORT2 из библиотеки COM .

position – адрес.

storage – указатель на массив данных в котором необходимо сохранить считываемые значения.

Данная функция предназначена для считывания значения из input register.

Возвращаемые значения:

1.10. Библиотека COM_SLAVE.h

Возвращает GOOD, если операция выполнена успешно.

Возвращает код ошибки в случае возникновения ошибки: COM_ERROR, POSITION_ERROR.

uint8_t Write_HOLDING_REGISTER(uint8_t selected_port, int position, uint16_t value);

Параметры:

selected_port – номер порта ведомого устройства. Для удобства ввода данного параметра рекомендуется использовать константы COM_PORT1 и COM_PORT2 из библиотеки COM.

position – адрес.

value – значение.

Данная функция предназначена для значения в holding register.

Возвращаемые значения:

Возвращает GOOD, если операция выполнена успешно.

Возвращает код ошибки в случае возникновения ошибки: COM_ERROR, POSITION_ERROR.

uint8_t Write_multi_HOLDING_REGISTER(uint8_t selected_port, uint16_t *data, uint16_t length, int position);

Параметры:

selected_port – номер порта ведомого устройства. Для удобства ввода данного параметра рекомендуется использовать константы COM_PORT1 и COM_PORT2 из библиотеки COM.

data – указатель на массив данных, подлежащих записи.

length – количество записываемых holding register.

position – начальный адрес.

Данная функция предназначена для множественной записи значений в holding register.

Возвращаемые значения:

Возвращает GOOD, если операция выполнена успешно.

Описание библиотек

Возвращает код ошибки в случае возникновения ошибки: COM_ERROR, POSITION_ERROR, LENGTH_ERROR.

uint8_t Read_HOLDING_REGISTER(uint8_t selected_port, int position, uint16_t* storage);

Параметры:

selected_port – номер порта ведомого устройства. Для удобства ввода данного параметра рекомендуется использовать константы COM_PORT1 и COM_PORT2 из библиотеки COM.

position – адрес.

storage – указатель на хранилище данных в которое необходимо сохранить считываемые значения.

Данная функция предназначена для считывания значения из holding register.

Возвращаемые значения:

Возвращает GOOD, если операция выполнена успешно.

Возвращает код ошибки в случае возникновения ошибки: COM_ERROR, POSITION_ERROR.

uint8_t Resolve_Position(uint8_t port, int type_register, unsigned int position);

Параметры:

selected_port – номер порта ведомого устройства. Для удобства ввода данного параметра рекомендуется использовать константы COM_PORT1 и COM_PORT2 из библиотеки COM.

type_register – тип регистра. Для удобства ввода данного параметра рекомендуется использовать константы COIL, DISCRETE_INPUT, ANALOG_INPUT, ANALOG_OUTPUT описанные в начале раздела.

position – адрес.

Данная функция позволяет проверить доступность номера порта, типа регистра и адреса для записи/чтения.

Возвращаемые значения:

Возвращает GOOD, если операция выполнена успешно.

1.10. Библиотека COM_SLAVE.h

Возвращает код ошибки в случае возникновения ошибки: COM_ERROR, POSITION_ERROR, REG_ERROR.

ПРИМЕР COM_SLAVE_STATUS:

Данная программа обрабатывает запросы от ведущего устройства по порту COM_PORT1. Также для визуализации было разработано меню, которое отображает значение во всех регистрах выбранного порта, навигация по меню осуществляется с помощью кнопок 3 и 0.

```
#define F_CPU 14745600UL           /*Стандартная константа задающая частоту микро-
контроллера*/
#define COM_RX_BUFFER_SIZE 256   /*Размер приемного буфера 256 байт*/
#define COM_TX_BUFFER_SIZE 256   /*Размер передающего буфера 256 байт */
#define COM1_PROTOCOL MODBUS     /*Установка работы COM1-порта по протоколу
Modbus RTU*/

#define TIMER2                    /*Константа разрешающая функциям
библиотеки использовать таймер 2 для
LED.h */

#define SLAVE_COM1 ON             /*Включение порта как ведомого*/
#define SCREAN_DO 1              /*Константа определяющая номер экрана
DO(Coils)*/
#define SCREAN_DI 2              /*Константа определяющая номер экрана
DI(Discrete Input)*/
#define SCREAN_AI 3              /*Константа определяющая номер экрана
AI(Input Register)*/
#define SCREAN_AO 4              /*Константа определяющая номер экрана
AO(Holding Register)*/
#include <util/delay.h>           /*Стандартная библиотека временных
задержек*/
#include <avr/io.h>               /*Стандартная библиотека для портов вво-
да/вывода*/
#include <avr/interrupt.h>        /*Стандартная библиотека обработки
прерываний*/
#include <stdio.h>                /*Стандартная библиотека ввода-вывода, использу-
ем для доступа к функции printf*/

#include "MC8U80.h"               /*Конфигурация для контроллера */
#include "LED.h"                  /*Библиотека для работы с экраном*/

#include "COM.h"                  /*Библиотека для работы с COM-портами
контроллера по протоколам DCON и
Modbus RTU*/

#include "COM_SLAVE.h"           /*Библиотека для реализации ведомого устройства в
протоколе Modbus RTU*/
#include "Key.h"                  /*Библиотека для работы с кнопками
контроллера*/
volatile uint16_t timer_s = 0;   /*Счетчик таймера*/
ISR(TIMER0_COMP_vect) {          /*Обработчик таймера 0*/
```

Описание библиотек

```
timer_s++; /*При прерывание увеличить на 1 значение
timer_s*/
}
void Timer0_Enable(void) /*Функция для включения таймера 0*/
{
    OCR0 = F_CPU*0.001/1024+0.5; //Период прерывания 1 мс
    TMSK |=(1<<OCIE0); //Разрешить прерывание по совпадению
таймера 0
    TCCR0 =(1<<WGM01)|(1<<CS02)|(1<<CS00); /*Запуск таймера 0 в режиме
"Сброс при совпадении".*/
    sei(); //Разрешить все прерывания
}
int main(void){
    _delay_ms(1000); /*Задержка для инициализации дисплея*/
    LED_Init(); /*Инициализация дисплея*/
    Timer0_Enable(); /*Инициализация таймера 0*/

    uint8_t current_screan = 0; /*Установка текущего номера экрана для вывода
информации*/
    char message[32]; /*Временная переменная для хранения
данных выводящихся на дисплей*/
    char title[32]; /*Временная переменная для хранения
названия экрана*/
    uint8_t test_discrete_data [4] = {1,1,0,0}; /*набор тестовых данных*/
    uint16_t test_analog_data [2] = {777,877}; /*набор тестовых данных*/
    uint8_t coils[4]; /*массив для временного хранения дан-
ных получаемых из регистра дискретного
выхода*/
    uint8_t DI[4]; /*массив для временного хранения данных получа-
емых из дискретного входа*/
    uint16_t AI[2]; /*массив для временного хранения дан-
ных получаемых из регистра Input
Register*/
    uint16_t AO[2]; /*массив для временного хранения дан-
ных получаемых из регистра Holding
Register*/
    COM_Init(COM_PORT1, 9600); /*Инициализация COM1 порта*/
    Write_multi_COIL(COM_PORT1, test_discrete_data, 4,0);
/*Запись тестовых данных
в Coils*/
    Write_multi_DISCRETE_INPUT(COM_PORT1, test_discrete_data, 4,0); /*Запись
тестовых данных
в discrete Input*/
    Write_multi_INPUT_REGISTER(COM_PORT1, test_analog_data, 2,0); /*Запись
тестовых данных
в input register*/
    Write_multi_HOLDING_REGISTER(COM_PORT1, test_analog_data, 2,0);/*Запись
тестовых данных
в holding register*/
    while(1)
```

1.10. Библиотека COM_SLAVE.h

```
{
    Check_Slave(COM_PORT1);           /*Запуск проверки
                                       наличия данных в
                                       COM_PORT1*/
    switch (current_screen){          /*выбор экрана*/
        case SCREAN_DO:              /*Экран Coils*/
            if (timer_s > 250){      /*если прошло 250 мс выпол-
нить условие*/
                memset(title,0,sizeof(title)); /*очистка временных
данных*/
                memset(message,0,sizeof(message)); /*очистка
временных данных*/
                for (int i = 0; i < 4; i++){
Read_COIL(COM_PORT1, i, &coils[i]);/*Считывание 4х значений из Coils*/
                }
                sprintf(message, "%u%u%u%u",/*Формирование
сообщения*/
                    coils[0],
                    coils[1],
                    coils[2],
                    coils[3]);
                sprintf(title, "SCREAN_DO"); /*Сохранение назва-
ния экрана*/
                LED_Clear();           /*Очистка
экрана*/
                LED_SetPos(0,0);      /*Установка позиции
на дисплее*/
                LED_Write_String(title); /*Вывод
названия экрана на дисплей*/
                LED_SetPos(1,0);      /*Установка позиции
на дисплее*/
                LED_Write_String(message); /*Вывод сообщения
на дисплей*/
                timer_s = 0;
                /*Обнуление счетчика таймера*/
            }
            break;                   /*выход
(прерываем переключатель)*/
        case SCREAN_DI:              /*Экран discrete
input*/
            if (timer_s > 250){      /*если прошло 250
мс выполнить
условие*/
                memset(title,0,sizeof(title)); /*очистка
временных данных*/
                memset(message,0,sizeof(message));/*очистка временных
данных*/
                for (int i = 0; i < 2; i++){ /*если
прошло 250 мс выполнить
условие*/
```

Описание библиотек

```
Read_DISCRETE_INPUT(COM_PORT1, i
, &DI[i]);/*Считывание
2х значений из discrete input*/
}
sprintf(message, "%u%u%u%u",
/*Формирование сообщения*/
DI[0],
DI[1],
DI[2],
DI[3]);
sprintf(title, "SCREEN_DI");
/*Сохранение названия экрана*/
LED_Clear();
/*Очистка экрана*/
LED_SetPos(0,0);
/*Установка позиции на дисплее*/
LED_Write_String(title);
/*Вывод названия экрана на
дисплей*/
LED_SetPos(1,0);
/*Установка позиции на дисплее*/
LED_Write_String(message); /*Вывод
сообщения на дисплей*/
timer_s = 0;
/*обнуление таймера*/
}
break;
/*выход(прерываем
переключатель)*/
case SCREEN_AI: /*Экран
input register*/
if (timer_s > 250){ /*если
прошло 250 мс
выполнить условие*/
memset(title,0,sizeof(title)); /*очистка
временных данных*/
memset(message,0,sizeof(message));/*очистка временных данных*/
for (int i = 0; i < 2; i++){
Read_INPUT_REGISTER(COM_PORT1, i
, &AI[i]);/*Считывание
2х значений из Input Register*/
}
sprintf(message, "%u - %u",
/*Формирование сообщения*/
AI[0],
AI[1]);
sprintf(title, "SCREEN_AI");
/*Сохранение названия экрана*/
```

1.10. Библиотека COM_SLAVE.h

```

                                LED_Clear();
/*Очитска экрана*/
                                LED_SetPos(0,0);
/*Установка позиции на дисплее*/
                                LED_Write_String(title);
/*Вывод названия экрана на
                                дисплей*/
                                LED_SetPos(1,0);
/*Установка позиции на дисплее*/
                                LED_Write_String(message); /*Вывод
сообщения на дисплей*/
                                timer_s = 0;
/*обнуление таймера*/
                                }
                                break;
/*выход(прерываем
                                переключатель)*/
                                case SCREAN_AO:
                                holding register*/ /*Экран
                                if (timer_s > 250){ /*если
                                прошло 250 мс выполнить
                                условие*/
                                memset(title,0,sizeof(title)); /*очистка
временных данных*/
                                memset(message,0,sizeof(message));/*очистка временных данных*/
                                for (int i = 0; i < 2; i++){
                                /*Считывание 2х значений из Holding
                                Register*/
                                Read_HOLDING_REGISTER(COM_PORT1, i , &AO[i]);
                                }
                                sprintf(message, "%u - %u",
                                AO[0],
                                AO[1]);
                                sprintf(title, "SCREAN_AO");
/*Сохранение названия экрана*/
                                LED_Clear();
/*Очитска экрана*/
                                LED_SetPos(0,0);
/*Установка позиции на дисплее*/
                                LED_Write_String(title);
/*Вывод названия экрана на
                                дисплей*/
```

Описание библиотек

```

/*Установка позиции на дисплее*/
LED_SetPos(1,0);
сообщения на дисплей*/
LED_Write_String(message); /*Вывод
timer_s = 0;
/*обнуление таймера*/
}
break;
/*выход(прерываем
переключатель*/
default:
/*Настройка по умолчанию */
if (timer_s > 250){ /*если
прошло 250 мс выполнить
условие*/
memset(title,0,sizeof(title)); /*очистка
временных данных*/
memset(message,0,sizeof(message));/*очистка временных данных*/
sprintf(title, "Начальное ок-
но");/*Сохранение названия экрана по
умолчанию*/
sprintf(message, "Нажмите*>*"
");/*Формирование сообщения по
умолчанию*/
LED_Clear();
/*Очистка экрана*/
LED_SetPos(0,0);
/*Установка позиции на дисплее*/
LED_Write_String(title);
/*Вывод названия экрана на
дисплей*/
LED_SetPos(1,0);
/*Установка позиции на дисплее*/
LED_Write_String(message); /*Вывод
сообщения на дисплей*/
timer_s = 0;
/*обнуление таймера*/
}
}
if (Key(0) == 1){
/*Если нажата кнопка 1*/
while (Key(0) == 1){_delay_ms(10);}
/*Антидребезг ожидаем
отпускание кнопки*/

```

1.10. Библиотека COM_SLAVE.h

```
ка что экран не                                  if (current_screan == SCREAN_AO){ /* провер-
                                                последний */
не на первый экран */                          current_screan = SCREAN_DO; /* переключе-
                                                }else{
ключениe на 1 экран                            current_screan ++; /* пере-
                                                вперед */
                                                }
                                                }
/*Если нажата кнопка 1*/                      if (Key(3) == 1){
                                                while (Key(3) == 1){ _delay_ms(10);
/*Антидребезг ожидаем                        отпускание кнопки*/
ка что экран не первый */                    if (current_screan == SCREAN_DO){ /* провер-
/* переключение на последний                current_screan = SCREAN_AO;
экрaн */
/*переключение на 1 экран назад*/          }else if(current_screan > 0){
current_screan --;
}
}
}
```

Пример COM_SLAVE_NLS

В данной программе порт COM1 настроен на работу в режиме ведомого устройства, а порт COM2 в режиме ведущего устройства.

По порту COM2 осуществляются запросы ведомому устройству, значения полученные от него сохраняются в качестве данных для ведущего устройства подключённому по порту COM1 и выводятся на дисплей.

```
#define F_CPU 14745600UL /*константа задающая частоту микро-
контроллера*/
#define COM_RX_BUFFER_SIZE 256 /*Размер приемного буфера 256 байт*/
#define COM_TX_BUFFER_SIZE 256 /*Размер передающего буфера 256 байт*/
#define COM1_PROTOCOL MODBUS /*Установка работы COM1-порта по
протоколу
Modbus RTU*/
#define COM2_PROTOCOL MODBUS /*Установка работы COM2-порта по
протоколу
Modbus RTU*/
```

Описание библиотек

```
#define TIMER2 /*Константа разрешающая
функциям библиотеки
использовать таймер 2 для LED.h*/
#define SLAVE_COM1 ON /*Включение COM1 порта как ведомого*/
#include <util/delay.h> /*Стандартная библиотека
временных задержек*/
#include <avr/io.h> /*Стандартная библиотека ввода/вывода
(для портов)*/
#include <stdio.h> /*Стандартная библиотека для работы функций
ввода вывода*/
#include "MC8U80.h" /*конфигурация для контроллера */
#include "LED.h" /*Библиотека для работы с экраном*/
#include "COM.h" /*Библиотека для работы с COM-портами контрол-
лера по
протоколам DCON и Modbus RTU*/
#include "COM_SLAVE.h" /*Библиотека для реализации ведомого устройства в
протоколе Modbus RTU*/
#include "Key.h" /*Библиотека для работы с кнопками
контроллера*/
volatile uint16_t timer_s = 0; /*счетчик таймера*/
char Amount[2] = {0x00,0x02}; /*количество запрашиваемых данных - число
0x0002*/
ISR(TIMER0_COMP_vect) { /*обработчик таймера 0*/
    timer_s++; /*при прерывании увеличить на 1 значение timer_s*/
}
void Timer0_Enable(void) /*функция для включения таймера 0*/
{
    OCR0 = F_CPU*0.001/1024+0.5; //Период прерывания 1 мс
    TIMSK |= (1<<OCIE0); //Разрешить прерывание по
совпадению таймера 0
    TCCR0 = (1<<WGM01)|(1<<CS02)|(1<<CS00); /*Запуск таймера 0 в режиме
"Сброс при совпадении".*/
    sei(); //Разрешить все пре-
рывания
}
int main(void){
    _delay_ms(1000); /*задержка для инициализации
дисплея*/
    LED_Init(); /*Инициализация
дисплея*/
    Timer0_Enable(); /*включение таймера*/
    char message[32]; /*временное хранилище дан-
ных получаемых по
порту COM_PORT2(Сборка сообщения: id,
функция, кол-во)*/
    char message2[32]; /*временное хранилище дан-
ных получаемых по
порту COM_PORT2(Сборка сообщения:
запрашиваемые значения)*/
    char RX_Buf2[250]; /*Задаём приёмный буфер для
обработки*/
```

1.10. Библиотека COM_SLAVE.h

```
char* current_slave = &RX_Buf2[1];           /*Привязываем указатель для
упрощения*/
uint16_t value_reg;                          /*переменная для хранения
значения из регистра
микроконтроллера*/
uint16_t value_reg1;                         /*переменная для хранения
значения из регистра
микроконтроллера*/
COM1 порта*/
COM_Init(COM_PORT1, 9600);                   /*Инициализация
COM1 порта*/
COM2 порта*/
COM_Init(COM_PORT2, 9600);                   /*Инициализация
COM2 порта*/
while(1)
{
    if (timer_s > 500){                       /*запрос на получение данных от ведомого устрой-
ства*/

    Write_Request_Modbus(COM_PORT2,0x01,0x04,0001,Amount,4);
    Led(LD_SW);                               /*сигнал индикато-
ром*/
    timer_s = 0;                               /*обнуление тайме-
ра*/
    }
    /*получение ответа от ведомого*/
    if (Read_Modbus(COM_PORT2,RX_Buf2) == 1){
        value_reg =
((uint16_t)current_slave[3]<<8) | (current_slave[4] & 0xFF);
        /*обработка ответа*/
        value_reg1 = ((uint16_t)current_slave[5]<<8) | (current_slave[6]
& 0xFF);

        Write_INPUT_REGISTER(COM_PORT1,0,value_reg);/*сохранение ответа в
input register*/

        Write_INPUT_REGISTER(COM_PORT1,1,value_reg1);/*сохранение ответа
input register*/

        /*сборка сообщения (ответ от ведомого устройства: id, функ-
ция, кол-во*/
        sprintf(message,"%u                               %u
%u",current_slave[0],current_slave[1],current_slave[2]);
        LED_SetPos(0,0);                         /*Установка позиции на дисплее*/
        LED_Write_String(message); /*Вывод ответа от ведомого
устройства*/

        /*сборка сообщения (ответ от ведомого устройства: запраши-
ваемые значения*/
        sprintf(message2,"%u - %u",value_reg,value_reg1);
        LED_SetPos(1,0);
```

Заключительная часть

```
        /*Установка позиции на дисплее*/  
        LED_Write_String(message2); /*Вывод ответа от ведомого*/  
    }  
}
```

2. Заключительная часть

Описанные библиотеки созданы с целью упрощения использования контроллеров серии МС. Библиотеки постоянно обновляются в соответствии с пожеланиями потребителей. Все программные коды, представленные в данном документе размещены в папке Examples в виде готовых проектов.

Ждем ваших замечаний и предложений для дальнейшего улучшения библиотек и примеров их применения!