Research Laboratory
of Design Automation
НИЛ автоматизации проектирования

Тел.: (8634) 324-140, 376-157, факс: 324-139, e-mail: info@RLDA.ru, www.RLDA.ru

Программируемый логический контроллер

Программируемые логические контроллеры серии МС с дискретными входами-выходами

MC-12D4R4O, MC-12D6R, MC-12D8O, MC-8D2S, MC-8D2R

Библиотеки функций для программирования на языке Си

Версия от 6 февраля 2016 г.
Распечатано 6 февраля 2016 г.

Одной проблемой стало меньше!

Оглавление

1.1. Библиотека LED.h

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.: (8634) 477-040, 477-040,

e-mail: info@rlda.ru • http://www.rlda.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.

Введение

В комплект поставки контроллеров серии МС (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:

- MC12D4R4O.h - файл конфигурации для контроллера MC-12D4R4O;
- MC12D6R.h - файл конфигурации для контроллера MC-12D6R;
- MC128O.h - файл конфигурации для контроллера MC-12D8O;
- MC8D2S2R.h - файл конфигурации для контроллеров MC-8D2S и MC-8D2R;
- LED.h - библиотека для работы с индикатором;
- Key.h - библиотека для работы с кнопками контроллера;
- IOPort.h - библиотека для работы с дискретными входами/выходами;
- Shim.h - библиотека для управления генератором ШИМ сигналов.
- COM.h - библиотека для работы с COM-портами контроллера по протоколам DCON и Modbus RTU;

1.1. Библиотека LED.h

- 1Wire.h - библиотека для работы с шиной 1Wire и чтения температуры платы;
- RTC.h - библиотека для работы с микросхемой часов реального времени;
- Timer2.h – библиотека вспомогательных функций. Задействует аппаратный таймер-счетчик 2 для обработки событий нажатий кнопок, сдвига индикатора и формирования сигналов ШИМ.

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"
#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.

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 символом 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 (нуль-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 умещаются 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.

1.1. Библиотека LED.h

void Led(char power);

Функция позволяет включить или выключить светодиод на лицевой панели контроллера.

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 не возможно. 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 нажатии 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.

1.2. Библиотека Key.h

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:

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 микроконтролле-
ра*/

#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 "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.

```
#define F_CPU 14745600UL     /*Стандартная константа задающая частоту микроконтролле-
ра*/

#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:

```
#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:

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:

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*/
```

1.3. Библиотека IOPort.h

```
#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 ото-
бражения 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 использо-
вать 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)
{
        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 микроконтролле-
ра*/

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

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

```
                        buf[temp]=((output>>(7-temp)) & 0x01)+0x30;
                }
                LED_Write_String(buf);          //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 микроконтролле-
ра*/

#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 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 buf[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*/
                {
```

```
                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 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 па-

раметру 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 микроконтролле-
ра*/

#include "MC12D4R4O.h"       //Тип контроллера

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

#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*/
```

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.

Пример объявления констант, определяющих используемый протокол.

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

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 и выбрать используемый протокол. Если в программе не планируется изменять скорость или протокол обмена, 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 микроконтроллера. Там же в таблице указано отклонение частоты передающего сигнала. Для обеспечения безошибочной передачи информации на высоких скоростях необходимо выбирать параметры, при которых ошибка будет равна нулю.

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.

***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

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 Addres, 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.

Addres – 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.

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:

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/r

Где:

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

Ответ: !AATTCCFF/r

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 микроконтролле-
ра*/

#include "MC12D4R4O.h"        //Тип контроллера
```

1.5. Библиотека COM.h

```
#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

//---------------------------------- 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 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
                        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*/
                }
                Addres++;                   /*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) && (Addres!=0x00))           /*Выполнять поиск пока уст-
ройство 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 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*/
                        }
                        Addres++; //Увеличить адрес для поиска следующего устройства
                }
                Led(LD_ON);             //Включить светодиод
}
```

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. Ни каких 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. Ни каких принципиальных
изменений в работе портов нет. Протокол связи Modbus RTU.
*/

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

#include "MC12D4R4O.h"       //Тип контроллера

#include <avr/io.h>          //Стандартная библиотека ввода-вывода
#include <avr/eeprom.h>      //Стандартная библиотека работы с EEPROM

#define COM_RX_BUFFER_SIZE 32      //Размер приемных буферов COM портов
#define COM_TX_BUFFER_SIZE 32      //Размер передающих буферов COM портов
```

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

#define COM2_PROTOCOL MODBUS     /*Установить для порта COM2 протокол Modbus
RTU (перепрограммировать таймер-счетчик 3 не допустимо)*/

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

//---------------------------- Инициализация EEPROM -------------------------------
unsigned char EE_Address_device EEMEM=0x01;     //Адрес модуля в EEPROM
unsigned char EE_COM_Speed EEMEM=0x06;          //Скорость модуля в EEPROM
unsigned char EE_Name[] EEMEM="MC12D4R4O"; //Имя модуля в EEPROM
unsigned char EE_Version[] EEMEM="060415";      //Версия программы в EEPROM

//---------------------------- Инициализация RAM ----------------------------------
char Address_device;                              //Адрес модуля
char COM_Speed;                                   //Скорость модуля
char Name[10];                                    //Имя модуля
char Version[6];                                  //Версия программы

char Func;                                        //Код функции
int SubFunc;                                      //Код подфункции
int Data;                                         //Регистр данных

char buf[32];                          //Буфер для чтения данных по Modbus RTU
int Address_EEP;                //Переменная для хранения адреса при обращении к EEPROM

//============== Функция инициализации модуля ==============
void Init(void)
{
        int temp;                                 //Вспомогательная переменная
        long Speed;                               //Переменная для определения скорости

        Address_device=eeprom_read_byte(&EE_Address_device);      /*Копирование адре-
са устройства из EEPROM в ОЗУ*/

        COM_Speed=eeprom_read_byte(&EE_COM_Speed);  //Чтение скорости из EEPROM

        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);			//Инициализация порта COM1
	COM_Init(COM_PORT2,Speed);			//Инициализация порта COM2

	Address_EEP=(int)&EE_Name;		/*Определение адреса в EEPROM масси-
ва содержащего имя модуля*/

	for ( temp=0; temp<10; temp++)		//Копирование имени модуля из EEPROM
		Name[temp]=eeprom_read_byte(Address_EEP+(uint8_t*)temp);

	Address_EEP=(int)&EE_Version;		/*Определение адреса в EEPROM масси-
ва содержащего версию программы*/

	for (temp=0; temp<6; temp++)		//Копирование версии программы
		Version[temp]=eeprom_read_byte(Address_EEP+(uint8_t*)temp);
}

//=== Функция декодирования команд полученных по COM портам ===
void Command_decode(char Port)
{
/*
	buf[0]				- Количество прочитанных байт
	buf[1]				- Адрес устройства
	buf[2]				- Код функции
	buf[3]+buf[4]			- Код подфункции (адрес элемента)
	buf[5]+buf[6]			- Данные (для функций 0x05, 0x06 записываемые
данные, для 0x01-0x04 количество считываемых данных)
	buf[7]+buf[8]			- CRC свертка (контрольная сумма стандарта
Modbus RTU)
*/
	char temp;			//Вспомогательная переменная
	int int_temp;			//Вспомогательная переменная

	if (buf[1]==Address_device)		//Проверка адреса устройства
	{
		Func=buf[2];			//Определение кода функции
		switch(Func)			//Обработка кода функции
		{
		//Функция чтения внутренних регистров (0x03)
		case 0x03:
			SubFunc=(buf[3]<<8)|buf[4];		/*Определение кода
подфункции*/
			Data=(buf[5]<<8)|buf[6];	//Определение данных

			switch (SubFunc)		//Обработка кода подфункции
			{
//------------------------------ Чтение имени модуля -------------------------------
			case 0x00C8:
				if (Data!=4)		/*Если количество запраши-
ваемых регистров не соответствует команде*/
				{
```

```
                                        buf[0]=0x03;
        //Ответ "Ошибка"
        Write_Response_Modbus(Port,Address_device,Func | 0x80,SubFunc,buf,1);
                                        break;
                                }

                                //Перекопирование имени модуля
                                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!=3)            /*Если количество запраши-
ваемых регистров не соответствует команде*/
                                {
                                        buf[0]=0x03;
        //Ответ "Ошибка"
        Write_Response_Modbus(Port,Address_device,Func | 0x80,SubFunc,buf,1);
                                        break;
                                }

                                //Перекопирование версии программы
                                for (int_temp=0; int_temp<6; int_temp++)

                        buf[int_temp]=Version[int_temp];

        Write_Response_Modbus(Port,Address_device,Func,SubFunc,buf,3);
                                break;

//------------------------------------ Чтение адреса -------------------------------------
                        case 0x0200:
                                if (Data!=1)            /*Если количество запраши-
ваемых регистров не соответствует команде*/
                                {
                                        buf[0]=0x03;
        //Ответ "Ошибка"
        Write_Response_Modbus(Port,Address_device,Func | 0x80,SubFunc,buf,1);
                                        break;
                                }

                temp=eeprom_read_byte(&EE_Address_device);      /*Копирование адре-
        са 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 запраши-
ваемых регистров не соответствует команде*/
                                                {
                                                            buf[0]=0x03;
            //Ответ "Ошибка"
            Write_Response_Modbus(Port,Address_device,Func | 0x80,SubFunc,buf,1);
                                                            break;
                                                }

                                                //Копирование скорости связи из EEPROM в ОЗУ
                                                temp=eeprom_read_byte(&EE_COM_Speed);
                                                buf[0]=0;
                                                buf[1]=temp;

            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 подфунк-
ции*/
                                    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))       /*Если записывае-
мый адрес выходит за допустимый диапазон*/
                                                {
                                                            buf[0]=0x03;
            //Ответ "Ошибка"
            Write_Response_Modbus(Port,Address_device,Func | 0x80,SubFunc,buf,1);
                                                            break;
                                                }
                                                //Запись в EEPROM нового адреса
```

```
                                        eeprom_write_byte(&EE_Address_device,Data);

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

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

//------------------------------ Запись скорости связи -------------------------------
                                case 0x0201:
                                        if ((Data<0x03) || (Data>0x0A))  /*Если записывае-
мый код скорости выходит за допустимый диапазон*/

                                                                        {
                                                buf[0]=0x03;
        //Ответ "Ошибка"
        Write_Response_Modbus(Port,Address_device,Func | 0x80,SubFunc,buf,1);
                                                break;
                                        }
                                        //Запись в EEPROM новой скорости связи
                                        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:

                                        //Если данная подфункция не поддерживается
                                        buf[0]=0x02;
                                        Write_Response_Modbus(Port,Address_device,Func |
0x80,SubFunc,buf,1);                    //Ответ "Ошибка"
                                }
                                break;

                        case 0x10:              //Функция записи сразу нескольких регистров
(0x10)
                                SubFunc=(buf[3]<<8)|buf[4];     /*Определение кода подфунк-
ции*/
                                int_temp=(buf[5]<<8)|buf[6];    /*Определение количества
регистров*/

                                switch (SubFunc)        /*Выбор в зависимости от полученного
кода подфункции*/
                                {
//------------------------------ Запись имени модуля --------------------------------
                                case 0x00C8:
```

```
                                        if (int_temp!=4)        /*Если длинна команды не
соответствует требуемой*/
                                        {
                                                buf[0]=0x03;
        //Ответ "Ошибка"
        Write_Response_Modbus(Port,Address_device,Func | 0x80,SubFunc,buf,1);
                                                break;
                                        }
                                //Вычисление адреса в EEPROM для записи имени модуля
                                        Address_EEP=(int)&EE_Name;

                                        //Определение данных
                                        for (int_temp=0; int_temp<8; int_temp++)

                        {
                                                Name[int_temp]=buf[int_temp+8];
        //Запись в EEPROM очередного байта нового имени модуля
        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:

                                        //Если данная подфункция не поддерживается
                                        buf[0]=0x02;
                                        Write_Response_Modbus(Port,Address_device,Func |
0x80,SubFunc,buf,1);                    //Ответ "Ошибка"
                                }
                                break;

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

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

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

```
                    if (Read_Modbus(COM_PORT1,buf)==1)
Command_decode(COM_PORT1);          /*Если данные на COM порт поступили, вызвать
процедуру декодирования команды*/
                    if (Read_Modbus(COM_PORT2,buf)==1)
Command_decode(COM_PORT2);          /*Если данные на COM порт поступили, вызвать
процедуру де-кодирования команды*/
            }
}
```

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;

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 date, unsigned char month, unsigned char year);
Функция осуществляет установку даты в микросхеме (число, месяц, год).

Параметры:

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* date, unsigned char* month, unsigned char* year);
Функция осуществляет чтение даты из микросхемы (число, месяц, год).

Параметры:

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

&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 реализует обычные 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 микроконтроллера*/

#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 реального 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
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 ок-
ругление секунд произошло в большую сторону*/
                                if (min>59) {min=0; hour++;}    /*Коррекция минут и
часов, если минуты вышли за предел*/
                                if (hour>23) 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
                }
//---------------------------------------- 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 ок-
ругление 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

                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:

```
#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
#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*/
                }
        }
}
```

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!