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

Библиотеки функций для программирования серии ПЛК NLCon-1AT на языке Си

Руководство по эксплуатации

Версия от 14 апреля 2015 г.
Распечатано 14 апреля 2015 г.

Одной проблемой стало меньше!

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 или факсу:

НИЛ АП, ул. Зои Космодемьянской, 2, Таганрог, 347924,

Тел. (8634) 324-139, 376-157, факс (8634) 324-139,

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

Оглавление

1. Вводная часть

В комплект поставки контроллеров серии NLcon-1AT (далее - "контроллер") входят описанные ниже библиотеки для разработки программного обеспечения пользователя. Библиотеки предназначены для использования в среде программирования AVR Studio 6. Последняя версия среды программирования AVR Studio 6 доступна для бесплатного копирования с сайта компании ATMEL http://www.atmel.com/microsite/atmel_studio6/.

Библиотеки позволяют управлять: устройством индикации (далее-индикатором), дискретными линиями ввода/вывода, шиной 1-Wire, интерфейсом RS485 с поддержкой протоколов DCON и Modbus RTU, а также проводить опрос кнопок и формировать сигналы ШИМ. Контроллеры NLcon-1AT выпускаются в пяти разных модификациях: NLcon-1AT, NLcon-1AT-D-3, NLcon-1AT-Ex, NLcon-1AT2 и NLcon-1AT2-D-3 (для более полной информации см. руководство по эксплуатации контроллеров). Для каждого контроллера создан специальный заголовочный файл описаний, позволяющий правильно сконфигурировать библиотеки под конкретную модификацию контроллера.

Перечень библиотек комплекта:

- NLcon1AT.h - файл конфигурации для контроллера NLcon-1AT;
- NLcon1ATD3.h - файл конфигурации для контроллера NLcon-1AT-D-3;
- NLcon1ATEx.h - файл конфигурации для контроллера NLcon-1AT-Ex;
- NLcon1AT2.h - файл конфигурации для контроллера NLcon-1AT2;
- NLcon1AT2D3.h - файл конфигурации для контроллера NLcon-1AT2-D-3;
- Timer2.h – библиотека вспомогательных функций. Задействует аппаратный таймер-счетчик 2 для обработки событий нажатий кнопок и формирования сигналов ШИМ.
- Display.h - библиотека для работы с индикатором;
- Key.h - библиотека для работы с элементами ручного управления (далее-кнопки) контроллера;

2.1. Библиотека Display.h

- IOPort.h - библиотека для работы с дискретными входами/выходами;
- COM.h - библиотека для работы с COM-портами контроллера в соответствии протоколами DCON и Modbus RTU;
- 1-Wire.h - библиотека для работы с шиной 1-Wire;
- DS18B20.h - библиотека для чтения значения температуры с датчиков DS18B20;
- Shim.h - библиотека для управления генератором ШИМ сигналов.

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

2. Подробное описание библиотек

2.1. Библиотека Display.h

Функции работы с индикатором контроллера

Данная библиотека содержит функции управления индикатором и светодиодом. Функция управления светодиодом применима для всех модификаций контроллера, а функции управления индикатором только для NLcon-1AT-D-3 и NLcon-1AT2-D-3. Функции управления индикатором позволяют обеспечить вывод на индикатор данных, заданных в символьном или числовом виде. При использовании функций данной библиотеки рекомендуется использовать оптимизацию кода программы. Уровень оптимизации может быть произвольным.

В начале исходного текста программы необходимо обязательно указать тип модификации контроллера, путем подключения соответствующего заголовочного файла. В библиотеке предусмотрены следующие константы упрощающие использование функций.

```
#define LED_OFF     0     /*Константа "Выключить светодиод"*/
#define LED_ON      1     /*Константа "Включить светодиод"*/
```

Ниже представлены прототипы функций данной библиотеки.

void DisplayInit(void);

Функция предназначена для инициализации выходных линий управления индикатором. 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 DisplayStr(char *data_buf);

Функция предназначена для вывода на индикатор данных, представленных в символьном виде.

Параметры:

***data_buf -** строка символов, отображаемая на индикаторе. Строка может состоять из символов цифр, первых шести символов (как строчных, так и заглавных) латинского алфавита (A,B,C,D,E,F,a,b,c,d,e,f), символов плюс и минус (только в начале строки), символов точек или запятых. Причем точка и запятая считаются идентичными символами. В случае вхождения в строку символов отличных от указанных, они будут исключены из выводимой строки. Длина принимаемой функцией строки не ограничена, однако на индикатор будут выведены только первые допустимые символы, количество которых соответствует разрядности. Если строка содержит несколько символов "точка" или "запятая", на индикаторе будет отображен только крайний левый (см. примеры ниже). Признаком окончания строки по стандарту языка СИ является символ с ASCII кодом 0x00 (нуль-символ).

char DisplayDec(int data, int digit, unsigned int point);

Функция предназначена для преобразования целочисленных данных к строке десятичных цифр и вывода их на индикатор. При использовании данной функции, вместе с самим числом всегда отображается и его знак, даже если число положительное и знак плюс при записи не был указан.

Параметры

data — целочисленные данные. Аргумент **data** может принимать целочисленное значение от -19999 до +19999, в случае выхода значения аргумента из указанного диапазона функция завершиться ошибкой (вернет код 0xFF).

digit — количество значащих цифр на индикаторе. Данный параметр указывает минимальное количество разрядов индикатора, которое будет отображено. Если реальное количество разрядов меньше значения данного параметра, все недостающие разряды перед числом будут заполнены нулями. Если же реальное количество разрядов больше значения данного параметра, то данный параметр будет проигнорирован. Это может быть полезно при выводе на индикатор дробных чисел, значение которых меньше нуля. Если вывод дробных чисел, меньших нуля, не используется с данной функцией, данный параметр лучше оставлять равным нулю.

point — позиция точки на индикаторе. Фактически, данный параметр указывает количество знаков после запятой у дробного числа. При выводе точки на индикатор, параметр должен принимать значение от 0 до 4. При больших значениях аргумента, точка отображена не будет, это необходимо использовать в тех случаях, если вывод точки не требуется.

Возвращаемое значение:

Функция возвращает 0 в случае успешного выполнения, либо 0xFF, в случае если данные выходят за допустимый диапазон.

void DisplayHex(unsigned int data);

Функция предназначена для преобразования целочисленных данных к строке шестнадцатеричных цифр и выводу их на индикатор.

data — целочисленные данные выводимые на индикатор.

char DisplayFloat(float data);

Функция позволяет преобразовать вещественные данные к строке десятичных цифр и вывести их на индикатор. В отличии от функции отображения десятичных цифр, данная функция:

- всегда выводит разделительную точку;
- проводит округление дробной части числа, не помещающейся на индикатор, до ближайшего отображаемого значения.
- всегда максимально полно использует разряды индикатора. Т.е. если при выводе числа могут остаться не задействованные разряды индикатора, функция сдвинет число максимально влево, а дробную часть заполнит нулями.

Параметры

data — вещественные данные. Аргумент **data** может принимать вещественное значение от -19999 до +19999. Индикатор имеет 4½ разряда. Целая часть числа выводится на индикатор полностью, оставшиеся разряды отводятся под дробную часть. При этом значение, отображаемое на индикаторе, будет округлено до ближайшего значащего разряда.

Пример:

Если аргумент **data** равен 123.456, под целую часть, будет выделено 3 разряда. Оставшиеся 2 разряда будут выделены под дробную часть. Если число 123.456, по правилам математики округлить до ближайшего значения, с точностью 2 знака после запятой, получится 123.46. Это значение и будет выведено на индикатор. Однако, если первая значащая цифра будет больше единицы (к примеру число 223.456), то на индикатор будет выведено значение 233.5, т.к. цифра 2 не может быть отображена в старшем разряде индикатора. В случае выхода значения аргумента из допустимого диапазона, функция будет завершаться ошибкой (возвращает код 0xFF).

Возвращаемое значение:

Функция возвращает 0 в случае успешного выполнения, либо 0xFF, в случае если данные выходят за допустимый диапазон.

void Led(char power);

Функция позволяет включить или выключить зеленый светодиод на лицевой панели контроллера.

Параметры

power — параметр управляющий состоянием светодиода (1-включен, 0-выключен). Для удобства работы с данным параметром целесообразно применять константы, описанные выше.

Примеры вызова функций работы с индикатором:

DisplayInit(); //Инициализация светодиодного индикатора

//---------- вывод строк ----------

DisplayStr(“12345”); /*Вывести на индикатор строку «12345». */

DisplayStr(“12.345”); /*Вывести на индикатор строку «12.345» (символ точка не является самостоятельным символом и прикрепляется к символу стоящему слева от него).*/

DisplayStr(“-12.3456”); */Последний символ не поместится на индикаторе и будет отброшен, поэтому на индикатор будет выведена строка «-12.345».*/

DisplayStr(“--A--”); /*Вывести на индикатор строку «-A». Остальные знаки минус отображены не будут, т.к. учитывается только знаки + и – стоящие в начале строки.*/

DisplayStr(“+-10”); /*Вывести на индикатор строку «±10». Оба знака числа будут выведены, т.к. допускается указывать в начале строки сразу оба знака следующих друг за другом.*/

DisplayStr(“aBcD”); /*Вывести на индикатор строку символов “ABCD”. Регистр значения не имеет.*/

DisplayStr(“A.B.C.D.”); /*Вывести на индикатор строку символов “A.BCD”. Символ точки размещен будет только после буквы «A», т.к. остальные точки будут отброшены.*/

DisplayStr(“LMNO”); /*Вывести на индикатор пустую строку (символы не являются отображаемыми, поэтому индикатор будет пустым).*/

//---------- вывод десятичных чисел ----------

DisplayDec(78,0,5); /*Вывести на индикатор значение 78, без дробной части. На индикатор будет выведена строка «+ 78».*/

DisplayDec(78,0,1); /*Вывести на индикатор значение 78, дробную часть сделать равной 1 разряду. На индикатор будет выведена строка «+ 7.8».*/

DisplayDec(12345,0,3); /*Вывести на индикатор значение 12345. Под дробную часть выделить 3 разряда. На индикатор будет выведена строка «+12.345».*/

DisplayDec(10,4,3); /*Вывести на индикатор значение 10. Под все число вывести 4 разряда из них под дробную часть 3 разряда. На индикатор будет выведена строка «+0.010».*/

DisplayDec(1,2,1); /*Вывести на индикатор значение 1. Под все число вывести 2 разряда, из них под дробную часть 1 разряд. На индикатор будет выведена строка «+ 0.1».*/

DisplayDec(-20000,0,5); /*Функция завершиться ошибкой (вернет код 0xFF, выход за диапазон). На индикаторе останется прежнее значение (вывод данных осуществляться не будет).*/

DisplayDec(-19999,0,5); /*Вывести на индикатор значение -19999, без дробной части. На индикатор будет выведена строка «-19999».*/

//---------- вывод шестнадцатеричных чисел ----------

DisplayHex(-19999); /*Вывести на индикатор число представленное в шестнадцатеричном виде. На индикатор будет выведена строка «B1E1» (дополнительный код числа -19999). В данном примере аргумент задан как десятичное число.*/

DisplayHex(0x5F8A); /*Вывести на индикатор число представленное в шестнадцатеричном виде. На индикатор будет выведена строка «5F8A». В данном примере аргумент задан как шестнадцатеричное число.*/

//---------- вывод вещественных чисел ----------

DisplayFloat(-19999); /*Вывести на индикатор число -19999. На индикатор будет выведена строка «-19999.». При использовании данной функции разделительная точка выводится всегда.*/

DisplayFloat(1.5); /*Вывести на индикатор вещественное число 1.5. На индикатор будет выведена строка «+1.5000».*/

DisplayFloat(12.3456); /*Вывести на индикатор вещественное число 12.345 На индикатор будет выведена строка «+12.346» (округление изменило значение последней отображаемой цифры).*/

DisplayFloat(-20.0/3); /*Вывести на индикатор результат операции деления. На индикатор будет выведена строка «-6.667» (цифра 6 не смогла уместиться в старшем разряде, поэтому весь результат был перемещен на один разряд правее, при этом уменьшилось количество значащих цифр в дробной части числа).*/

DisplayFloat(M_PI); /*Вывести на индикатор значение числа π. На индикатор будет выведена строка «+3.142».*/

//---------- управление светодиодом ----------

Led(LED_ON); /*Включить зеленый светодиод на лицевой панели контроллера.*/

Led(LED_OFF); /*Выключить зеленый светодиод на лицевой панели контроллера.*/

Пример

```
#define F_CPU 16000000UL        /*Стандартная константа, задающая
частоту микроконтроллера*/
```

```
#include "NLCON1ATD3.h"        /*Файл конфигурации контроллера*/

#include <avr/io.h>            //Стандартная библиотека ввода-вывода
#include <util/delay.h>        //Стандартная библиотека временных
                               задержек
#include "Display.h"           //Библиотека работы с индикатором

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

        DisplayStr("A8F4");    //Результат A8F4 (это строка)
        _delay_ms(5000);       //Задержка на 5 секунд

        DisplayFloat(-20.0/3); //Результат -20/3=-6.667
        _delay_ms(5000);       //Задержка на 5 секунд

        DisplayFloat(sqrt(2)); /*Результат квадратный корень из 2
=+ 1.4142*/
        _delay_ms(5000);       //Задержка на 5 секунд

        DisplayDec(1234,0,3);  //Результат +123.4
        _delay_ms(5000);       //Задержка на 5 секунд

        DisplayHex(0xA8F4);    //Результат A8F4h
                               //(это шестнадцатеричное число)
        _delay_ms(5000);       //Задержка на 5 секунд
```

```
    Led(LED_ON);            //Включить светодиод

    while(1);               //Бесконечный цикл
}
```

Данная программа выводит на индикатор различные значения и отображает каждое из них в течении 5 секунд.

2.2. Библиотека Key.h

Функции работы с кнопками контроллера

Данная библиотека предназначена для модификаций NLcon-1AT-D-3 и NLcon-1AT2-D-3. Причем, в контроллере NLcon-1AT2-D-3, средняя кнопка схемотехнически объединена с дискретным входом-выходом IO15, поэтому одновременное использование этого входа-выхода и средней кнопки не возможно. Функции данной библиотеки позволяют определять состояния кнопок контроллера. Отдельно для каждой кнопки можно включить режим фиксации нажатия, который будет подробно описан ниже.

В начале исходного текста программы необходимо обязательно указать тип модификации контроллера, путем подключения соответствующего заголовочного файла. 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*/
#define LEFT        1    /*Константа "Левая кнопка"*/
#define MIDDLE      2    /*Константа "Средняя кнопка"*/
#define RIGHT       3    /*Константа "Правая кнопка"*/
```

При использовании режима фиксации нажатия хотя бы одной из кнопок, 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 у какой-либо кнопки 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.

char KeyInit(char key, 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 работающих в режиме фиксации. Без предварительного вызова данной функции, корректное определение состояния кнопок не возможно. Данную функцию достаточно вызвать один раз при инициализации контроллера. Функция позволяет для каждой отдельной кнопки переключать режим работы. Существуют два режима работы: 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 и считывается как статус кнопки. Данный флаг сбрасывается автоматически после каждого выполнения функции чтения статуса.

Режим с фиксацией позволяет определять нажатия кнопок, не пользуясь 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 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

key - номер кнопки. Может принимать значения от 1 до 3. Левая кнопка промаркирована знаком «-» на корпусе, средняя кнопка - треугольником, а правая кнопка - знаком «+». Для удобства указания данного параметра целесообразно применять константы, описанные выше.

sw — параметр, включающий/отключающий режим фиксации кнопки. Для удобства указания данного параметра целесообразно применять константы, описанные выше.

Возвращаемое значение:

Функция возвращает 0 в случае успешного завершения работы, либо 0xFF в случае, если указан номер несуществующей кнопки или неверно задан параметр фиксации.

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, для которой переключается режим. Может принимать значения от 1 до 3. Нумерация кнопок на контроллере идет слева направо. Кнопка 1 промаркирована знаком «-» на корпусе, кнопка 2 - треугольником, а кнопка 3 - знаком «+». Для удобства указания данного параметра целесообразно применять константы, описанные выше.

Возвращаемое значение

Функция возвращает состояние кнопки или флага фиксации, либо 0xFF в случае, если указан номер несуществующей кнопки. Для удобства работы с данным параметром целесообразно применять константы, описанные выше.

Пример 1

```
#define F_CPU 16000000UL        /*Стандартная константа, задающая
частоту микроконтроллера*/

#include "NLCON1ATD3.h"         /*Файл конфигурации контроллера*/

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

2.2. Библиотека Key.h

```
#include <util/delay.h>     //Стандартная библиотека временных задержек
#include "Display.h"        //Библиотека работы с индикатором
#include "Key.h"            //Библиотека работы с кнопками


int main(void)
{
        DisplayInit();      //Инициализация светодиодного индикатора
        KeyInit(LEFT, LATCH_OFF);       //Инициализация левой кнопки
        KeyInit(MIDDLE, LATCH_OFF); //Инициализация средней кнопки
        KeyInit(RIGHT, LATCH_OFF);    //Инициализация правой кнопки

        int Count=0;        //Счетчик отображаемый на индикаторе
        DisplayDec(Count,0,0);    //Вывод на индикатор нуля


        while (1)           //Бесконечный цикл
        {
                if (Key(RIGHT)==DOWN)          //Если нажата кнопка +
                {
                        Led(LED_ON);           //Включить светодиод
                        DisplayDec(++Count,0,5); /*Увеличить счетчик и
вывести его значение на индикатор*/
                }
                if (Key(LEFT)==DOWN) //Если нажата кнопка -
                {
                        Led(LED_ON);           //Включить светодиод
                        DisplayDec(--Count,0,5);  /*Уменьшить счетчик и
вывести его значение на индикатор*/
                }
```

```
                if (Key(MIDDLE)==DOWN) //Если нажата средняя кнопка
                {
                        Led(LED_ON);            //Включить светодиод
                        Count=0;
                        DisplayDec(Count,0,5);  /*Обнулить счетчик и
вывести его значение на индикатор*/
                }
if ((Key(LEFT)==UP) && (Key(MIDDLE)==UP) && (Key(RIGHT)==UP))
Led(LED_OFF); /*Если ни одна из кнопок не нажаты, выключить светоди-
од*/
                _delay_ms(100);         /*Задержка на 100 мс для умень-
шения скорости счета*/
        }
}
```

Данная программа при нажатии на кнопку «плюс» начинает циклически увеличивать счетчик, отображая при этом его значение на индикаторе. При нажатии на кнопку «минус» значение счетчика уменьшается. При нажатии на среднюю кнопку значение счетчика обнуляется. Также при нажатии на любую из этих кнопок загорается зеленый светодиод. Если все кнопки отпущены, светодиод гаснет.

Пример 2

```
#define F_CPU 16000000UL        /*Стандартная константа, задающая
частоту микроконтроллера*/

#include "NLCON1ATD3.h"         /*Файл конфигурации контроллера*/

#include <avr/io.h>             //Стандартная библиотека ввода-вывода
#include <util/delay.h>         //Стандартная библиотека временных задержек
```

2.2. Библиотека Key.h

```
#include "Display.h"        //Библиотека работы с индикатором

#define TIMER2 /*Подключить обработчик прерывания таймера 2 для обработки фиксации нажатия кнопок*/
#include "Key.h"            //Библиотека работы с кнопками

int main(void)
{
        KeyInit(LEFT,LATCH_ON);        /*Инициализация левой кнопки в режиме работы с фиксацией нажатия*/

        while (1)           //Бесконечный цикл
        {
                if (Key(LEFT)==DOWN) Led(LED_ON); /*Если кнопка - была нажата, зажечь светодиод*/
                else Led(LED_OFF);                //Иначе погасить светодиод
                _delay_ms(5000);          //Задержка на 5 секунд.
        }
}
```

Данная программа при нажатии на кнопку «минус» через некоторое время зажигает зеленый светодиод. Время срабатывания может составлять от 0 до 5 секунд. Светодиод погаснет по истечении некоторого времени, которое может также составлять от 0 до 5 секунд. Данная программа демонстрирует, что несмотря на то, что кнопка была отпущена до того как произошел опрос ее состояния (в программе выполнялась задержка), факт ее нажатия был зафиксирован и обработан. Аналогичным образом обрабатывалось и отпускание кнопки.

2.3. Библиотека IOPort.h

Функции управления дискретными входами-выходами контроллера

Данная библиотека предназначена для всех модификаций. Однако модификации NLcon-1AT и NLcon-1AT-D-3 имеют 6 дискретных входов-выходов, а модификации NLcon-1AT-Ex, NLcon-1AT2 и NLcon-1AT2-D-3 имеют 16 дискретных входов-выходов. Причем, в контроллере NLcon-1AT2-D-3, средняя кнопка схемотехнически объединена с дискретным входом-выходом IO15, поэтому одновременное использование этого входа-выхода и средней кнопки не возможно. Функции данной библиотеки предназначены для настройки дискретных входов-выходов контроллера и осуществления их чтения и установки. Специальной инициализации для функций данной библиотеки не требуется. Направление линии настраивается автоматически в зависимости от применяемой функции.

Модификации NLcon-1AT и NLcon-1AT-D-3 позволяют осуществить подключение дискретного входа через подтягивающий резистор к земле, питанию или оставить в состоянии высокого импеданса (подтягивающий резистор не подключен). Модификации NLcon-1AT-Ex, NLcon-1AT2 и NLcon-1AT2-D-3 позволяют осуществить подключение дискретного входа через подтягивающий резистор только к питанию или оставить в состоянии высокого импеданса (подтягивающий резистор не подключен).

В начале исходного текста программы необходимо обязательно указать тип модификации контроллера, путем подключения соответствующего заголовочного файла. Для удобства пользования функциями в данной библиотеке объявлены следующие константы:

```
#define LOW         0     /*Низкое состояние линии*/
#define HI          1     /*Высокое состояние линии*/
#define Z_STATE     2     /*Состояние высокого импеданса*/

#define IO0  0            /*Линия ввода/вывода I/O 0*/
#define IO1  1            /*Линия ввода/вывода I/O 1*/
#define IO2  2            /*Линия ввода/вывода I/O 2*/
```

```
#define IO3    3          /*Линия ввода/вывода I/O 3*/
#define IO4    4          /*Линия ввода/вывода I/O 4*/
#define IO5    5          /*Линия ввода/вывода I/O 5*/
```

Для модификаций имеющих 16 дискретных входов-выходов дополнительно объявлены константы:

```
#define IO6    6          /*Линия ввода/вывода I/O 6*/
#define IO7    7          /*Линия ввода/вывода I/O 7*/
#define IO8    8          /*Линия ввода/вывода I/O 8*/
#define IO9    9          /*Линия ввода/вывода I/O 9*/
#define IO10   10         /*Линия ввода/вывода I/O 10*/
#define IO11   11         /*Линия ввода/вывода I/O 11*/
#define IO12   12         /*Линия ввода/вывода I/O 12*/
#define IO13   13         /*Линия ввода/вывода I/O 13*/
#define IO14   14         /*Линия ввода/вывода I/O 14*/
#define IO15   15         /*Линия ввода/вывода I/O 15*/
```

Далее представлены прототипы функций для управления дискретными входами-выходами.

char InLine(char port_line, char PullUp);

Функция осуществляет настройку указанной линии на вход и определяет уровень дискретного сигнала на ней. В случае, когда на линии отсутствует внешний сигнал, состояние линии определяется подтягивающим резистором.

Параметры

port_line - номер дискретной линии (от 0 до 15). Для удобства работы с данным параметром целесообразно применять константы, описанные выше.

PullUp – параметр определяющий состояние входа, когда на нем нет внешних сигналов.

0x00 – дискретная линия притянута к земле (только для модификаций NLcon-1AT и NLcon-1AT-D-3);

0x01 – дискретная линия притянута к питанию;

0x02 – внутренняя подтяжка отключена (состояние высокого импеданса);

Возвращаемое значение

Функция возвращает 0 в случае низкого уровня сигнала на линии, 1 в случае высокого уровня сигнала или 0xFF в случае, если был указан номер несуществующей линии. Для удобства работы с данным параметром целесообразно применять константы, описанные выше.

char OutLine(char port_line, char level);

Функция осуществляет настройку указанной линии на выход и устанавливает уровень дискретного сигнала на ней.

Параметры

port_line - номер дискретной линии (от 0 до 15). Для удобства работы с данным параметром целесообразно применять константы, описанные выше.

level - уровень сигнала на выходе линии (0-низкий уровень; 1-высокий уровень). Для удобства указания данного параметра целесообразно применять константы, описанные выше.

Возвращаемое значение

Функция возвращает 0 в случае успешного завершения работы, либо 0xFF в случае, если указан номер не существующей линии дискретного ввода-вывода.

Пример 1

```
#define F_CPU 16000000UL        /*Стандартная константа задающая
частоту микроконтроллера*/
```

2.3. Библиотека IOPort.h

```
#include "NLCON1ATD3.h"        /*Файл конфигурации контроллера*/

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

#include "IOPort.h"            /*Библиотека работы с дискретными входами-выходами контроллера*/


int main(void)

{

        OutLine(IO0,HI); //Установить линию IO0 в высокое состояние

        OutLine(IO1,LOW); //Установить линию IO1 в низкое состояние

        OutLine(IO2,HI); //Установить линию IO2 в высокое состояние

        OutLine(IO3,LOW); //Установить линию IO3 в низкое состояние

        OutLine(IO4,HI); //Установить линию IO4 в высокое состояние

        OutLine(IO5,LOW); //Установить линию IO5 в низкое состояние


        while(1);          //Бесконечный цикл

}
```

Данная программа устанавливает дискретные выходы контроллера в различные состояния. Проверить выполнение программы можно, подключив вольтметр между дискретными выходами и выводом GND. На выходе, установленном в высокое состояние, должно присутствовать постоянное напряжение +5В±10%; на выходе, установленном в низкое состояние, напряжение должно быть равно нулю.

Пример 2

```
#define F_CPU 16000000UL       /*Стандартная константа задающая

частоту микроконтроллера*/
```

```
#include "NLCON1ATD3.h"        /*Файл конфигурации контроллера*/

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


int main(void)
{
        char status=0;         //Переменная для записи состояния входов
        DisplayInit();         //Инициализация светодиодного индикатора


        while(1)               //Цикл бесконечного опроса состояния входов
        {
                if (InLine(IO0,HI)==LOW) status&=0xFE; /*Если вход 0 в низком состоянии, сбросить 0 бит*/
                else status|=0x01;                     /*иначе установить 0 бит переменной*/


                if (InLine(IO1,HI)==LOW) status&=0xFD; /*Если вход 1 в низком состоянии, сбросить 1 бит*/
                else status|=0x02;                     /*иначе установить 1 бит переменной*/


                if (InLine(IO2,HI)==LOW) status&=0xFB; /*Если вход 2 в низком состоянии, сбросить 2 бит*/
                else status|=0x04;                     /*иначе установить 2 бит переменной*/
```

```
        if (InLine(IO3,HI)==LOW) status&=0xF7; /*Если вход 3 в
низком состоянии, сбросить 3 бит*/
        else status|=0x08;                     /*иначе устано-
вить 3 бит переменной*/

        if (InLine(IO4,HI)==LOW) status&=0xEF; /*Если вход 4 в
низком состоянии, сбросить 4 бит*/
        else status|=0x10;                     /*иначе устано-
вить 4 бит переменной*/

        if (InLine(IO5,HI)==LOW) status&=0xDF; /*Если вход 5 в
низком состоянии, сбросить 5 бит*/
        else status|=0x20;                     /*иначе устано-
вить 5 бит переменной*/

        DisplayHex(status);                    /*Вывести на
индикатор состояние дискретных входов*/
    }
}
```

Контроллер по данной программе осуществляет проверку состояния дискретных входов и заносит их в переменную статуса, объявленную в программе. Номера бит в переменной статуса соответствует номерам дискретных входов. Все входы подтянуты к питанию через специальные встроенные резисторы, поэтому при включении контроллера на индикатор будет выведено значение 003F. Это означает, что на всех 6 входах будет установлен высокий уровень напряжения. Если соединить отрезком провода вывод GND контроллера и какой-либо дискретный вход, информация на индикаторе изменится. К примеру, если замкнуть на вывод GND вход IO0, индикатор покажет значение 003E, т.е. на дискретный вход IO0 подан низкий уровень напряжения. Если вход IO3 замкнуть на вывод GND, индикатор покажет значение 0037.

2.4. 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. У контроллеров NLcon-1AT и NLcon-1AT-D-3 порт COM1 реализован с интерфейсом RS485, порт COM2 - с интерфейсом RS232.

В начале исходного текста программы необходимо обязательно указать тип модификации контроллера, путем подключения соответствующего заголовочного файла. Для удобства пользования функциями в данной библиотеке объявлены следующие константы:

```
#define ENABLE      1     /*Контрольная сумма в протоколе DCON
используется*/
#define DISABLE     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 16 МГц). Если этого не сделать, компилятор выдаст предупреждение и установит частоту кварца по умолчанию 16 МГц.

Пример объявления стандартной константы, указывающей частоту кварца микроконтроллера:

```
#define F_CPU 3686400UL        /*Установить частоту кварца 3,6864 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 протокола Modbus RTU в начале исходного текста (до подключения библиотеки) необходимо объявить приведенные ниже константы. Константа **COM1_MODBUS** включает в исходный текст программы обработчик прерывания таймера-счетчика 1, при этом в дальнейшем, не допускается перепрограммировать данный таймер-счетчик. Константа **COM2_MODBUS** включает в исходный текст программы обработчик прерывания таймера-счетчика 3, при этом в дальнейшем, не допускается перепрограммировать данный таймер-счетчик. Если какую-либо константу не объявить, компилятор не выдаст ошибки, но обмен информацией по соответствующему COM порту осуществляться не будет. Пример

объявления констант, включающих в исходный текст программы обработчики прерываний таймеров-счетчиков 1 и 3.

```
#define COM1_MODBUS          /*Подключить обработчик таймера 1*/
#define COM2_MODBUS          /*Подключить обработчик таймера 3*/
#include "COM.h"          //Библиотека работы с COM портами
```

Далее представлены прототипы функций данной библиотеки.

char COM_Init(int port, long COM_speed, char Protocol);

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 выше.

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.

Protocol — протокол сетевого обмена. Для удобства ввода данного параметра рекомендуется использовать константы, описанные выше.

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 выше.

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

Возвращаемое значение

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

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 выше.

***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;

0x80 – несоответствие протокола (включен протокол Modbus RTU);

0xFF – ошибка контрольной суммы.

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 выше.

***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.

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 выше.

***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;

0x80 – несоответствие протокола (включен протокол DCON);

0xFF – ошибка контрольной суммы.

void 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 сервера). 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 выше.

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).

void 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 клиента). 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 выше.

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 те 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).

Пример 1:

```
#define F_CPU 16000000UL        /*Стандартная константа задающая
частоту микроконтроллера*/

#include "NLCON1AT2D3.h"        /*Файл конфигурации контроллера*/
```

2.4. Библиотека COM.h

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

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

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

int main(void)
{
        DisplayInit();       //Инициализация светодиодного индикатора

        COM_Init(COM_PORT1,9600,DCON);     /*Инициализация COM1.
Скорость обмена 9600 бит/с. Протокол DCON.*/

        int error,temp;              //Вспомогательные переменные
        char buf[32];                //Массив для считываемых данных

        while (1)                    //Бесконечный цикл программы
        {
                Write_DCON(COM_PORT1, "$012");       /*Запись в порт
COM1 команды чтения конфигурации для модуля NL-16DI */

                _delay_ms(100);              //Ожидание ответа
                temp=Read_DCON(COM_PORT1, buf);    /*Чтение прием-
ного буфера COM1*/

                if (temp==1)                 //Если данные прочитаны
                {
                        error=0;             //Обнуление флага ошибки

                        /*По байтовая проверка полученных данных*/
                        if (buf[0]!=10) error=1;
                        if (buf[1]!='!') error=1;
                        if (buf[2]!='0') error=1;
                        if (buf[3]!='1') error=1;
                        if (buf[4]!='4') error=1;
                        if (buf[5]!='0') error=1;
```

```
                        if (buf[6]!='0') error=1;
                        if (buf[7]!='6') error=1;
                        if (buf[8]!='0') error=1;
                        if (buf[9]!='0') error=1;
                        if (buf[10]!=0x0D) error=1;

                        /*Если нет ошибки, включить светодиод, иначе
выключить его.*/
                        if (error==0) Led(1);
                        else Led(0);
                }

                DisplayDec(buf[0],0,5);    /*Вывод на индикатор количест-
ва принятых байт*/
                _delay_ms(1000);           //Задержка после ответа
        }
}
```

Данная программа отправляет команду чтения конфигурации модулю NL-16DI. 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 входного диапазона (для модуля NL-16DI всегда равен 0x40);

CC-код скорости связи (для скорости 9600 бит/с равен 0x06);

FF-формат команды;

Более подробную информацию по команде, можно найти в руководстве по эксплуатации на модуль NL-16DI.

В программе подразумевается, что модуль NL-16DI имеет адрес равный 0x01, скорость связи 9600 бит/с, формат обмена без контрольной суммы (заводские настройки модуля NL-16DI).

Программа проверяет полученный ответ и в случае если он корректный, включает зеленый светодиод. Также программа выводит на светодиодный индикатор количество принятых байт ответного сообщения. Если в процессе приема данных не возникало ошибок, количество принятых байт должно быть равно 10.

Пример 2:

```
#define F_CPU 16000000UL        /*Стандартная константа задающая
частоту микроконтроллера*/

#include "NLCON1ATD3.h"         /*Файл конфигурации контроллера*/

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

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

#define COM2_MODBUS             /*Порт COM2 настроен на работу по про-
токолу Modbus RTU, будет подключен обработчик таймера 3*/

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

int main(void)
{
        DisplayInit();          //Инициализация светодиодного индикатора
```

```
COM_Init(COM_PORT2,9600,MODBUS);          /*Инициализаци COM2. Скорость обмена 9600 бит/с. Протокол Modbus.*/
COM_Init(COM_PORT1,9600,DCON);            /*Инициализаци COM1. Скорость обмена 9600 бит/с. Протокол DCON.*/

int error,temp;                 //Вспомогательные переменные
char buf[64];                   //Массив для считываемых данных

while(1)                                //Бесконечный цикл
{
        temp=Read_Modbus(COM_PORT2, buf);  /*Чтение приемного буфера COM2*/

        if (temp==1)                    //Если данные прочитаны
        {
                error=0;                //Обнуление флага ошибки

                if (buf[0]!=8) error=1;     /*По байтовая проверка принятой посылки*/
                if (buf[1]!=0x01) error=1;
                if (buf[2]!=0x05) error=1;
                if (buf[3]!=0x00) error=1;
                if (buf[4]!=0x20) error=1;
                if (buf[5]!=0xFF) error=1;
                if (buf[6]!=0x00) error=1;
                if (buf[7]!=0x8D) error=1;
                if (buf[8]!=0xF0) error=1;
```

```
                DisplayDec(buf[0],0,5); /*Вывод на индикатор количества принятых байт*/

/*Ответное сообщение по протоколу Modbus RTU в данном примере не отправляется*/

                /*Если нет ошибки отправка по COM1 команды в DCON*/

                if (error==0) Write_DCON(COM_PORT1,"$012");


                /*Очистка массива для принятых данных*/

                for (temp=0;temp<10; temp++) buf[temp]=0;

            }

            _delay_ms(1000);        //Задержка на 1 секунду

        }

}
```

Программа реализует функции простейшего преобразователя протоколов. Данная программа использует сразу оба COM порта контроллера. Один настроен на работу по протоколу DCON, другой Modbus RTU. Если по порту COM2, принята последовательность [01][05][00][20][FF][00][8D][F0] (протокол Modbus RTU), то программа выводит на индикатор количество принятых байт, в данном случае это число восемь и во второй порт отправляет команду «$012» (протокол DCON). Ответное сообщение по протоколу Modbus RTU не отправляется.

Пример 3:

```
#define F_CPU 16000000UL        /*Стандартная константа задающая частоту микроконтроллера*/

#include "NLCON1AT2.h"          /*Файл конфигурации 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_MODBUS             /*Подключить обработчик таймера 1*/
#define COM2_MODBUS             /*Подключить обработчик таймера 3*/

#include "COM.h"                //Библиотека работы с COM портами
#include "IOPort.h"             /*Библиотека работы с дискретными вхо-
дами-выходами*/

//---------------------------- 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="NLcon-1AT2"; /*Имя модуля в
EEPROM*/
unsigned char EE_Version[] EEMEM="150214";     /*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 обраще-
нии к EEPROM*/

//============== Функция инициализации модуля ==============
void Init(void)
{
```

```
        int temp;                   //Вспомогательная переменная
        long Speed;                 //Переменная для определения скорости

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

        /*Чтение скорости из EEPROM*/
        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;
        }

        /*Инициализация порта COM1.*/
        COM_Init(COM_PORT1,Speed,MODBUS);

        /*Инициализация порта COM2.*/
        COM_Init(COM_PORT2,Speed,MODBUS);

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

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

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

/*Копирование версии программы*/
for (temp=0; temp<6; temp++)
```

```
        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 под-
функции*/
                        {
//---------------------- Чтение дискретных входов с 0 по 7 ----------------------
                        case 0x0000:
                                if (Data!=1)    /*Если количество за-
прашиваемых регистров не соответствует команде*/
                                {
                                        buf[0]=0x03;
```

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

                                temp=0;
                                for (int_temp=0; int_temp<8; int_temp++)
temp|=(InLine(int_temp,1)<<int_temp);       //Чтение дискретных входов

                                buf[0]=0;
                                buf[1]=temp;

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

//---------------------- Чтение дискретных входов с 8 по 15 ----------------------
                        case 0x0010:
                                if (Data!=1)        /*Если количество за-
прашиваемых регистров не соответствует команде*/
                                {
                                        buf[0]=0x03;
        //Ответ "Ошибка"
        Write_Response_Modbus(Port,Address_device,Func |
0x80,SubFunc,buf,1);                                                    break;
                                }

                                temp=0;
                                for (int_temp=0; int_temp<8; int_temp++)
temp|=(InLine(int_temp+8,1)<<int_temp);   //Чтение дискретных входов

                                buf[0]=0;
                                buf[1]=temp;

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

//------------------------------ Чтение имени модуля -------------------------------
                        case 0x00C8:
```

```
                                        if (Data!=5)        /*Если количество за-
прашиваемых регистров не соответствует команде*/
                                        {
                                                buf[0]=0x03;

        //Ответ "Ошибка"
        Write_Response_Modbus(Port,Address_device,Func |
0x80,SubFunc,buf,1);                                                    break;
                                        }
//Перекопировани имени модуля
for (int_temp=0; int_temp<10; int_temp++) buf[int_temp]=Name[int_temp];

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

//---------------------------- Чтение версии программы ----------------------------
                                case 0x00D4:
                                        if (Data!=3)        /*Если количество за-
прашиваемых регистров не соответствует команде*/
                                        {
                                                buf[0]=0x03;
        //Ответ "Ошибка"
        Write_Response_Modbus(Port,Address_device,Func |
0x80,SubFunc,buf,1);                                                    break;
                                        }
        //Перекопировани 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 за-
прашиваемых регистров не соответствует команде*/
                                        {
                                                buf[0]=0x03;
        //Ответ "Ошибка"
        Write_Response_Modbus(Port,Address_device,Func |
0x80,SubFunc,buf,1);
                                                break;
```

```
                                        }

        //Копирование адреса устройства из EEPROM в ОЗУ
        temp=eeprom_read_byte(&EE_Address_device);

                                        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;
        //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;

//Функция записи одиночного регистра (0x06)
                case 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 дискретных входов с 0 по 7 ------------------------
                        case 0x0000:
                                if (Data>0x00FF)            /*Если значение
выходит за допустимый диапазон*/
                                {
                                        buf[0]=0x03;
        //Ответ "Ошибка"
        Write_Response_Modbus(Port,Address_device,Func |
0x80,SubFunc,buf,1);                                                    break;
                                }

                                temp=0;
        //Установка дискретных входов
        for (int_temp=0; int_temp<8; int_temp++)
                OutLine(int_temp,(Data>>int_temp) & 0x01);

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

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

//----------------------- Запись дискретных входов с 8 по 15 ------------------------
                        case 0x0010:
                                if (Data>0x00FF)            /*Если значение
выходит за допустимый диапазон*/
                                {
                                        buf[0]=0x03;
        //Ответ "Ошибка"
```

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

                                        temp=0;
        //Установка дискретных входов
        for (int_temp=0; int_temp<8; int_temp++)
                OutLine(int_temp+8,(Data>>int_temp) & 0x01);

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

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

//------------------------------------ Запись адреса ----------------------------------------
                                case 0x0200:
                                        if ((Data==0) || (Data>0xF7))           /*Если
записываемый 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
записываемый 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 поддержи-
вается
                                        buf[0]=0x02;
        //Ответ "Ошибка"
        Write_Response_Modbus(Port,Address_device,Func |
0x80,SubFunc,buf,1);
        }
                        break;

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

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

```
                                    if (int_temp!=5)  /*Если длинна команды
не соответствует требуемой*/
                                    {
                                            buf[0]=0x03;
        //Ответ "Ошибка"
        Write_Response_Modbus(Port,Address_device,Func |
0x80,SubFunc,buf,1);                                                    break;
                                    }

                                    Address_EEP=(int)&EE_Name;
/*Вычисление адреса в EEPROM для записи имени модуля*/

                                    //Определение данных
                                    for (int_temp=0; int_temp<10; 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 поддержи-
вается
                                    buf[0]=0x02;
        //Ответ "Ошибка"
        Write_Response_Modbus(Port,Address_device,Func |
0x80,SubFunc,buf,1);
        }
                            break;

//------------------------ Не поддерживаемый код функции -------------------------
                    default:
                            buf[0]=0x01;
        //Ответ "Ошибка"
```

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

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

Данный пример представляет собой тестовую программу для контроллера NLcon-1AT2. Программа позволяет принимать команды протокола Modbus RTU по обоим интерфейсам RS485 и выполнять действия, предусмотренные в приведенной ниже таблице.

Код подфункции	**Назначение команды**	**Код функции чтения**	**Код функции записи**	**Количество регистров**	**Диапазон данных**
00h 00h	Чтение-установка дискретных входов с 0 по 7	03h	06h	01h	0000h-00FFh

00h 10h	Чтение-установка дискретных входов с 8 по 15	03h	06h	01h	0000h-00FFh
00h C8h	Чтение-запись имени модуля	03h	10h	05h	5 регистров по 2 байта (ASCII кодирование символов)
00h D4h	Чтение версии программы	03h	-	03h	3 регистра по 2 байта (ASCII кодирование в формате DDMMYY)
02h 00h	Чтение-запись адреса модуля	03h	06h	01h	0001h-00F7h
02h 01h	Чтение-запись скорости модуля	03h	06h	01h	0003h-000Ah

2.5. Библиотека 1-Wire.h

Функции работы с интерфейсом 1-Wire

Данная библиотека предназначена для модификаций NLcon-1AT и NLcon-1AT-D-3. Функции данной библиотеки предназначены для работы с устройствами, имеющими однопроводный интерфейс 1-wire. Они позволяют осуществлять поиск устройств на шине 1-Wire, проводить инициализацию шины, выполнять запись и чтение данных, а также производить расчет контрольной суммы. Для корректной работы функций данной библиотеки, в настройках компилятора обязательно должна быть включена оптимизация кода программы. Обусловлено это тем, что в данной библиотеке применяются стандартные функции временных задержек, которые при выключенной оптимизации кода не обеспечивают соблюдение требуемых величин задержек по времени. Уровень оптимизации может быть произвольным.

В начале исходного текста программы необходимо обязательно указать тип модификации контроллера, путем подключения соответствующего заголовочного файла.

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 16 МГц). Если этого не сделать, компилятор выдаст предупреждение и установит частоту кварца по умолчанию 16 МГц.

Пример объявления стандартной константы, указывающей частоту кварца микроконтроллера:

```
#define F_CPU 3686400UL    /*Установить частоту кварца 3,6864 МГц*/
```

При использовании данной библиотеки в начале исходного текста (до подключения библиотеки) необходимо объявить константу **ONE_WIRE_BUS** и присвоить ей номер дискретного входа, к которому подключено устройство с интерфейсом 1-Wire.

2.5. Библиотека 1-Wire.h

Пример объявления константы, определяющей подключение устройств с интерфейсом 1-Wire:

#define ONE_WIRE_BUS IO0 /*Назначить под устройства с интерфейсом 1-Wire дискретный вход IO0*/

Ниже приведены прототипы функций описанных в данной библиотеке.

int w1_search(void *p);

Функция предназначена для поиска устройств находящихся на шине 1-Wire и занесения их адреса в массив указанный как параметр функции. Функция возвращает в качестве результата, количество обнаруженных устройств. Для каждого устройства, которое предполагается обнаружить, требуется зарезервировать 8 байт под индивидуальный адрес устройства. Рекомендуется в качестве хранилища адресов использовать двумерные массивы, первый элемент которого указывает порядковый номер устройства, а второй непосредственно номер байта в 8-байтовом адресе устройства. Когда количество устройств находящихся на шине не известно, память необходимо выделять с запасом, т.к. функция поиска не имеет средств защиты от переполнения памяти.

Параметры:

***p –** массив для записи адресов найденных устройств.

Возвращаемое значение:

Функция возвращает количество обнаруженных устройств на шине.

char w1_init(void);

Функция предназначена для инициализации шины 1-Wire. В ходе инициализации, функция определяет наличие устройств на шине.

Возвращаемое значение:

Функция возвращает значение равное единицы, если на шине обнаружено хотя бы одно устройство с интерфейсом 1-Wire, в противном случае возвращается значение равное нулю.

char w1_read(void);

Функция предназначена для чтения одного байта данных с шины 1-Wire.

Возвращаемое значение

Функция возвращает данные прочитанные с шины 1-Wire.

void w1_write(char data);

Функция предназначена для записи одного байта данных на шину 1-Wire.

Параметры

data – значение записываемое на шину 1-Wire.

char w1_crc8(void *p, char n);

Функция предназначена для вычисления контрольной суммы указанного массива данных по специальному алгоритму, используемому в устройствах 1-Wire.

Параметры

***p –**массив, для которого требуется рассчитать контрольную сумму.

n – количество байт участвующих в вычислении контрольной суммы. Контрольная сумма может применяться в различных ситуациях (например, в адресе устройства она записывается восьмым байтом, а в блокнотной памяти - девятым), поэтому количество байт, участвующих в расчете, должно быть указано пользователем.

Возвращаемое значение

Функция возвращает вычисленное значение контрольной суммы.

Пример 1

```
#define F_CPU 16000000UL        /*Стандартная константа задающая
частоту микроконтроллера*/

#include "NLCON1ATD3.h"         /*Файл конфигурации контроллера*/

#define ONE_WIRE_BUS IO0        /*Назначить под устройства с интерфейсом 1-Wire дискретный вход IO0*/

#include <avr/io.h>             //Стандартная библиотека ввода-вывода
#include "Display.h"            //Библиотека работы с индикатором
```

```
#include "1Wire.h"          //Библиотека работы с шиной 1-Wire

int main(void)
{
        DisplayInit();      //Инициализация светодиодного индикатора

        int temperature;  /*Переменная для рассчитанного значения температуры*/
        char buf[9];        //Буфер для чтения блокнотной памяти
        int temp;           //Вспомогательная переменная

        while (1)           //Бесконечный цикл опроса датчика DS18B20
        {
                w1_init();          //Инициализация датчика
                w1_write(0xCC); //Команда "Пропуск ПЗУ"
                w1_write(0x44);  //Команда "Запуск преобразования"
                _delay_ms(750); /*Задержка 750мс на преобразование температуры датчиком*/

                w1_init();          //Инициализация датчика
                w1_write(0xCC);          //Команда "Пропуск ПЗУ"
                w1_write(0xBE);          //Команда "Чтение блокнотной памяти"

                //Чтение блокнотной памяти
                for (temp=0; temp<9; temp++) buf[temp]=w1_read();

                if (buf[8]==w1_crc8(buf,8))        /*Вычисление и проверка контрольной суммы*/
                {
                        //Вычисление температуры
                        temperature=buf[0]+(buf[1]<<8);
                DisplayFloat(temperature*0.0625);          /*Вывод значения температуры на индикатор*/
```

```
                }
        }
}
```

Данная программа осуществляет чтение данных с подключенного к контроллеру датчика DS18B20, выполняет вычисление температуры и выводит ее значение на индикатор. Данная программа рассчитана на подключение только одного датчика.

Пример 2

```
#define F_CPU 16000000UL        /*Стандартная константа задающая
частоту микроконтроллера*/

#include "NLCON1ATD3.h"         /*Файл конфигурации контроллера*/

#define ONE_WIRE_BUS IO0        /*Назначить под устройства с интерфейсом 1-Wire дискретный вход IO0*/

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

#define TIMER2 /*Подключить обработчик прерывания таймера 2 для обработки фиксации нажатия кнопок*/
#include "key.h"                //Библиотека работы с кнопками

int main(void)
{
        DisplayInit();          //Инициализация светодиодного индикатора
        KeyInit(MIDDLE,LATCH_ON); /*Инициализация средней кнопки в режиме работы с фиксацией нажатия*/
```

```
        int temperature[2];         /*Массив для рассчитанных значений температуры*/
        char buf[9];                //Буфер для чтения блокнотной памяти
        int temp,i;                 //Вспомогательные переменные
        char address_buf[2][8];     /*Массив для записи адресов найденных устройств*/
        int Sensor=0;               /*Номер датчика, показания которого выводятся на индикатор*/

        if (w1_search(address_buf)!=2)   /*Выполнить поиск устройств на шине 1-Wire*/
        {
                /*Если количество найденных датчиков не равно 2, вывести на индикатор букву «E»*/
                DisplayStr("E");
                while(1);               //Зациклить программу
        }

        while (1)           //Бесконечный цикл опроса датчиков DS18B20
        {
                if (Key(2)==DOWN)       /*Если была нажата средняя кнопка*/
                {
                        Sensor^=0x01;   /*Переключить номер датчика, показания которого выводятся на индикатор*/

                        Led(Sensor);    //Переключить светодиод
                }

                w1_init();              //Инициализация датчиков
                w1_write(0xCC);         /*Команда "Пропуск ПЗУ" (обращение сразу ко всем датчикам)*/
                w1_write(0x44);         /*Команда "Запуск преобразования"*/
```

```
                _delay_ms(750);          /*Задержка 750мс на преобразо-
вание температуры датчиками*/

                for (i=0; i<2; i++)      //Цикл для двух датчиков
                {
                        w1_init();       //Инициализация датчиков
                        w1_write(0x55);  //Команда "Выбор ПЗУ"

                //Передача адреса устройства
                for(temp=0;temp<8;temp++) w1_write(address_buf[i][temp]);

                        w1_write(0xBE); /*Команда "Чтение блокнотной
памяти"*/

                //Чтение данных с шины
                for (temp=0; temp<9; temp++) buf[temp]=w1_read();

        //Если контрольная сумма совпадает, вычисление температуры
        if (buf[8]==w1_crc8(buf,8)) temperature[i]=buf[0]+(buf[1]<<8);

/*Вывод значения температуры на индикатор*/
DisplayDec(temperature[Sensor]*0.625,2,1);
                }
        }
}
```

Данная программа осуществляет чтение данных с подключенных к контроллеру датчиков DS18B20, выполняет вычисление температуры и выводит значение, измеренное одним из них на индикатор. Нажимая среднюю кнопку, можно выбрать с какого именно датчика будет выводиться информация на индикатор. При этом, когда на индикатор выводится информация с первого обнаруженного на шине датчика, зеленый светодиод на лицевой панели контроллера выключен, а когда со второго, он включен.

2.6. Библиотека DS18B20.h

Функции работы с датчиками DS18B20

Данная библиотека предназначена для модификаций NLcon-1AT и NLcon-1AT-D-3 и спроектирована как дополнение к библиотеке 1-Wire.h. Функции данной библиотеки позволяют выполнить преобразование и чтение значения температуры, измеренное датчиками DS18B20. Для безошибочной работы функций библиотеки в настройках используемого компилятора обязательно должна быть включена оптимизация кода программы. Обусловлено это тем, что в данной библиотеке применяются стандартные функции временных задержек, которые при выключенной оптимизации кода не обеспечивают соблюдение требуемых интервалов времени. Уровень оптимизации может быть произвольным. Применение данной библиотеки при решении задач требующих выполнения в реальном масштабе времени не целесообразно из-за большого значения времени выполнения отдельных функций. Поэтому в задачах, требующих выполнения в реальном масштабе времени, рекомендуется применять только библиотеку 1-Wire.h.

В начале исходного текста программы необходимо обязательно указать тип модификации контроллера, путем подключения соответствующего заголовочного файла.

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 16 МГц). Если этого не сделать, компилятор выдаст предупреждение и установит частоту кварца по умолчанию 16 МГц.

Пример объявления стандартной константы, указывающей частоту кварца микроконтроллера:

```
#define F_CPU 3686400UL    /*Установить частоту кварца 3,6864 МГц*/
```

При использовании данной библиотеки в начале исходного текста (до подключения библиотеки) необходимо объявить константу **ONE_WIRE_BUS** и присвоить ей номер дискретного входа, к которому подключено устройство с интерфейсом 1-Wire.
Пример объявления константы, определяющей подключение устройств с интерфейсом 1-Wire:

```
#define ONE_WIRE_BUS IO0      /*Назначить под устройства с интерфейсом 1-Wire дискретный вход IO0*/
```

Ниже приведены прототипы функций описанных в данной библиотеке.

char DS18B20_Conversion(void);

Функция выполняет запуск преобразования температуры для всех датчиков DS18B20, находящихся на указанной шине 1-Wire. Время выполнения данной функции занимает более 750 мс.

Возвращаемое значение

Функция возвращает 0 в случае успешного выполнения, либо 0xFF, если на шине не обнаружено ни одного датчика.

int DS18B20_Read(void *addr);

Функция предназначена для чтения блокнотной памяти датчика с указанным адресом, а также проверки контрольной суммы. В случае безошибочности принятых данных функция возвращает первые два байта блокнотной памяти, которые содержат преобразованное значение температуры.

Параметры:

***addr –** 64-битный уникальный адрес датчика. Данный адрес может быть получен при помощи функции поиска устройств из библиотеки 1-Wire.h.

Возвращаемое значение

Функция возвращает значение температуры, прочитанное с датчика. Значение температуры будет представлено в двоичном дополнительном коде.

int DS18B20_temperature(void *addr);

Данная функция полностью аналогична предыдущей, за исключением того, что она выполняет приведение прочитанной температуры к десятичному коду. Возвращаемый функцией результат в 10 раз больше реально полученного значения температуры. Это сделано для удобства представления результата, т.к. реальное значение температуры является дробным числом. Т.е., если в результате выполнения функции получено значение 234, реальное значение температуры равно 23.4 градуса.

Возвращаемое значение

Функция возвращает значение, приведенное к десятичному коду температуры. Для удобства дальнейшего применения значение температуры, возвращаемое функцией, в 10 раз больше реального.

Пример программы

```
#define F_CPU 16000000UL        /*Стандартная константа задающая
частоту микроконтроллера*/

#include "NLCON1ATD3.h"         /*Файл конфигурации контроллера*/

#define ONE_WIRE_BUS IO0        /*Назначить под устройства с интерфейсом 1-Wire дискретный вход IO0*/

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

#define TIMER2 /*Подключить обработчик прерывания таймера 2 для обработки фиксации нажатия кнопок*/
#include "key.h"                //Библиотека работы с кнопками
#include "DS18B20.h"            //Библиотека работы с датчиками DS18B20
```

```
#include <util/delay.h>     //Стандартная библиотека временных задержек


int main(void)

{

        char address_buf[2][8];    /*Массив для записи адресов найденных датчиков на шине 1*/

        int Sensor=0;              /*Номер датчика, показания которого выводятся на индикатор*/


        DisplayInit();             //Инициализации индикаторов

        KeyInit(MIDDLE,LATCH_ON); /*Инициализация средней кнопки в режиме работы с фиксацией нажатия*/

        w1_search(address_buf);  //Поиск датчиков на шине 1-Wire


        while (1)                  /*Бесконечный цикл опроса датчиков DS18B20*/

        {

                if (Key(2)==DOWN)      /*Если была нажата средняя кнопка*/

                {

                        Sensor^=0x01; /*Переключить номер датчика, показания которого выводятся на индикатор */

                                                //

                        Led(Sensor);       //Переключить светодиод

                }


                DS18B20_Conversion(); /*Запуск преобразования температуры для датчиков DS18B20 находящихся на выбранной шине*/
```

```
/*Вывод ни индикатор значения температуры одного из найденных датчиков*/
        DisplayDec(DS18B20_temperature(address_buf[Sensor]),2,1);
        }
}
```

Данная программа полностью аналогична примеру 2 из предыдущего раздела, но написана с использованием библиотеки DS18B20.h.

2.7. Библиотека Shim.h

Функции для работы с генератором ШИМ сигналов

Данная библиотека предназначена для всех модификаций контроллеров. Функции данной библиотеки позволяют осуществлять инициализацию и запуск программного ШИМ. Для всех модификаций контроллеров в качестве выводов ШИМ используются дискретные входы-выходы IO0-IO5. Входы-выходы контроллера имеют альтернативное назначение, поэтому одновременное использование одного и того же входа-выхода и для генерации ШИМ и как дискретного входа-выхода невозможно. Каждый из каналов ШИМ может управляться отдельно. Величина периода регулирования и длительность управляющего импульса варьируется от 0 до 10 секунд, с шагом 10 мс.

При использовании данной библиотеки не допускается перепрограммировать таймер-счетчик 2 микроконтроллера.

В начале исходного текста программы необходимо обязательно указать тип модификации контроллера, путем подключения соответствующего заголовочного файла. 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 (от 0 до 5). Для удобства работы с данным параметром целесообразно применять константы, описанные в библиотеке работы с дискретными входами-выходами «IOPort.h», которая подключается автоматически в данной библиотеке.

Pulse – длительность управляющего импульса. Одна единица соответствует 10 мс. Длительность управляющего импульса не должна превышать длительности периода регулирования;

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

Возвращаемое значение

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

Пример программы

```
#define F_CPU 16000000UL        /*Стандартная константа задающая
частоту микроконтроллера*/

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

int main(void)
{
        Shim(IO0,50,100); /*Запуск генерации ШИМ
0 канал, период 1 сек, длительность импульса 500 мс*/
        while(1);               //Бесконечный цикл
}
```

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

2.8. Библиотека RTC.h

Функции для работы с часами реального времени

Данная библиотека предназначена для контроллеров NLcon-1AT, NLcon-1AT-D-3 и NLcon-1AT2-EX. Функции этой библиотеки позволяют осуществлять установку и чтение даты и времени.

В начале исходного текста программы необходимо обязательно указать тип модификации контроллера, путем подключения соответствующего заголовочного файла. Ниже приведены прототипы функций описанных в данной библиотеке.

void 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.

void 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);

void 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);

void 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);

void 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);

Пример программы:

```
#define F_CPU 16000000UL        /*Стандартная константа задающая
частоту микроконтроллера*/

#include "NLCON1ATD3.h"         /*Файл конфигурации контроллера*/
```

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

#include "Display.h"         //Библиотека работы с индикатором
#include "RTC.h"             /*Библиотека работы с микросхемой часов реаль-
ного времени*/

unsigned char hour, min, sec;

int main(void)
{
        DisplayInit();       //Инициализация светодиодного индикатора
        RTC_Init();          /*Инициализация микросхемы часов реального
времени*/
        Led(LED_ON);         //Включить светодиод

//Записать в микросхему значение часов, минут, секунд
                RTC_Set_Time(0,0,0);

        while(1)
        {
//Прочитать из микросхемы текущее значение часов, минут, секунд
                RTC_Get_Time(&hour,&min,&sec);
//Вывести на индикатор значение минут и секунд разделенных точкой
                DisplayDec(min*100+sec,0,2);
        }
}
```

Данная программа при запуске устанавливает нулевые значения часов, минут и секунд и далее осуществляет непрерывное чтение информации из микросхемы часов реального времени. При этом осуществляется вывод на индикатор значения прочитанных минут и секунд, разделенных точкой.

3. Заключительная часть

Описанная библиотека создана с целью упрощения использования серии контроллеров NLcon-1AT. Библиотека постоянно расширяется в соответствии с пожеланиями потребителей.

Поэтому ждем ваших предложений и замечаний для ее дальнейшего улучшения.