Библиотека / генератор кода AXLIB для AtmelStudio 6.x – 7.x

Содержание:

Вступление

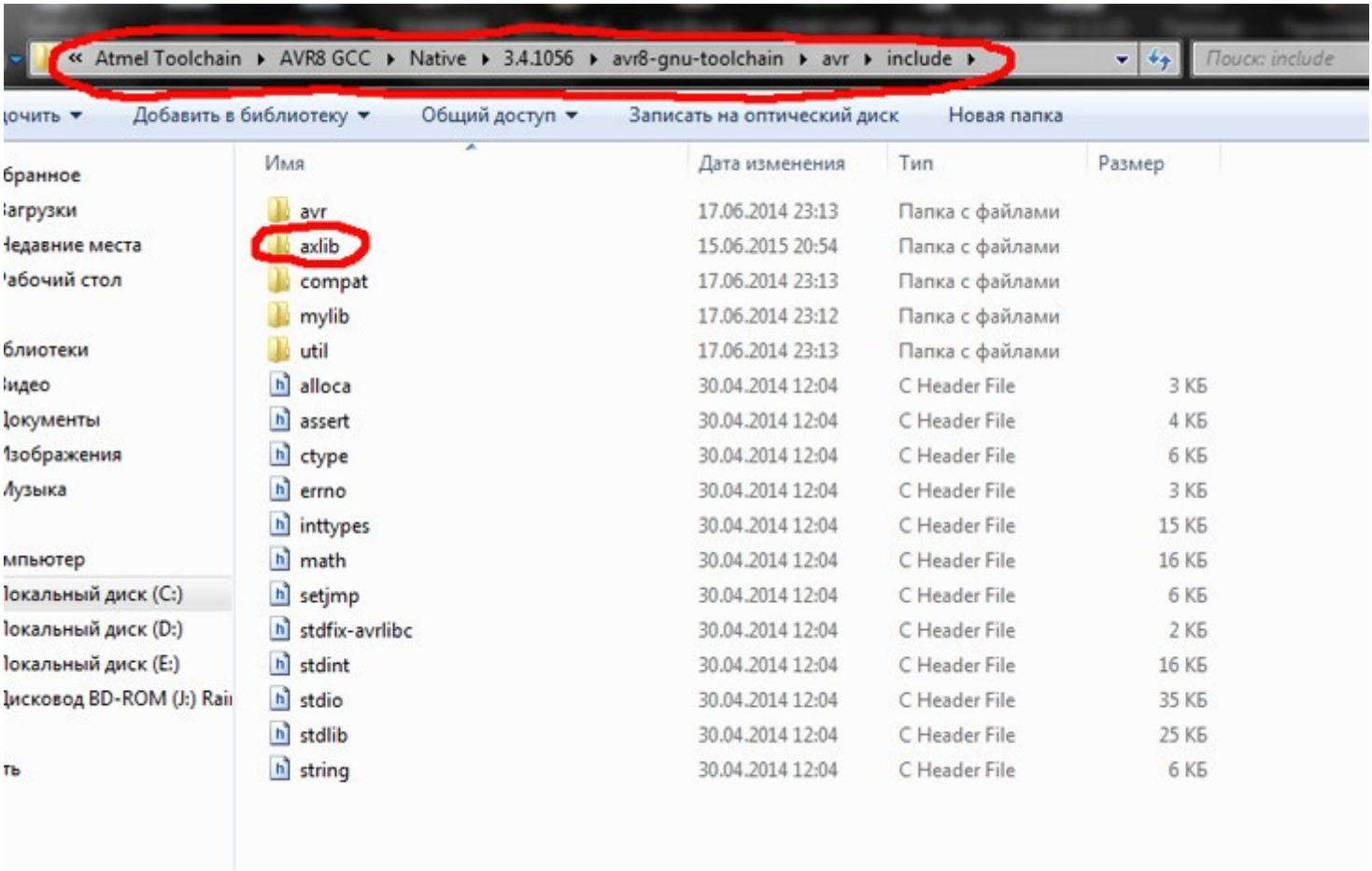
Многие новички решившие начать программирование AVR микроконтроллеров фирмы Atmel чаще всего идут по двум путям. Самый простой - это Arduino, ничего паять не надо и полно готовых прошивок. Кто по серьезней, паяет сам и использует чаще всего CodeVisionAVR, так как там много готовых библиотек для МК. Но мне часто задают один и тот же вопрос "А есть ли такие же библиотеки для AtmelStudio?". Нет. Почему-то Атмеловцы решили забить на пользователей и открестились лишь тем, что последняя версия может хватать код сгенерированный CodeVisionAVR. Вроде бы ничего, но... Но, заключается в том, что CodeVisionAVR является платной программой и стоит денег. Для бесплатного использования слишком мало ресурсов, а хочется большего. Вот и я решил выйти из этого положения и начал потихоньку писать свою библиотеку для AtmelStudio. Со временем накопилось куча разных вариантов и я собравшись с силами решил потихоньку все это систематизировать и собрать в одну кучу. И так родилась библиотека под названием AXLIB. Почему такое название, а хрен его знает. Просто так захотелось. Короче выкладываю первую версию на ваш суд. Если кому интересно можете пользоваться, она абсолютно бесплатна. Я надеюсь, что не забью на это и найду силы для ее пополнения. И так приступим.

Первое что надо сделать - скачать архив с библиотекой axlib 1.1

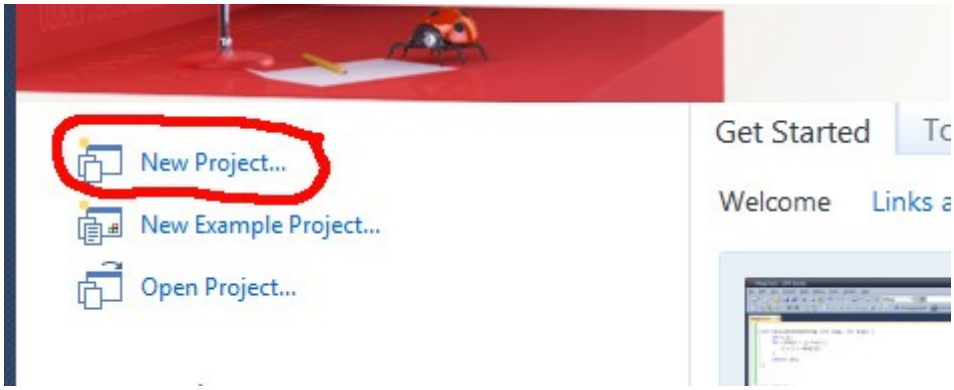
Или скачать генератор кода тут, и он сделает все за вас. После того как скачали на диск ее нужно распаковать в папку по адресу

Program Files (x86) -> Atmel -> Atmel Toolchain -> AVR8 GCC -> Native -> 3.4.1056 -> avr8-gnu-toolchain -> avr->include

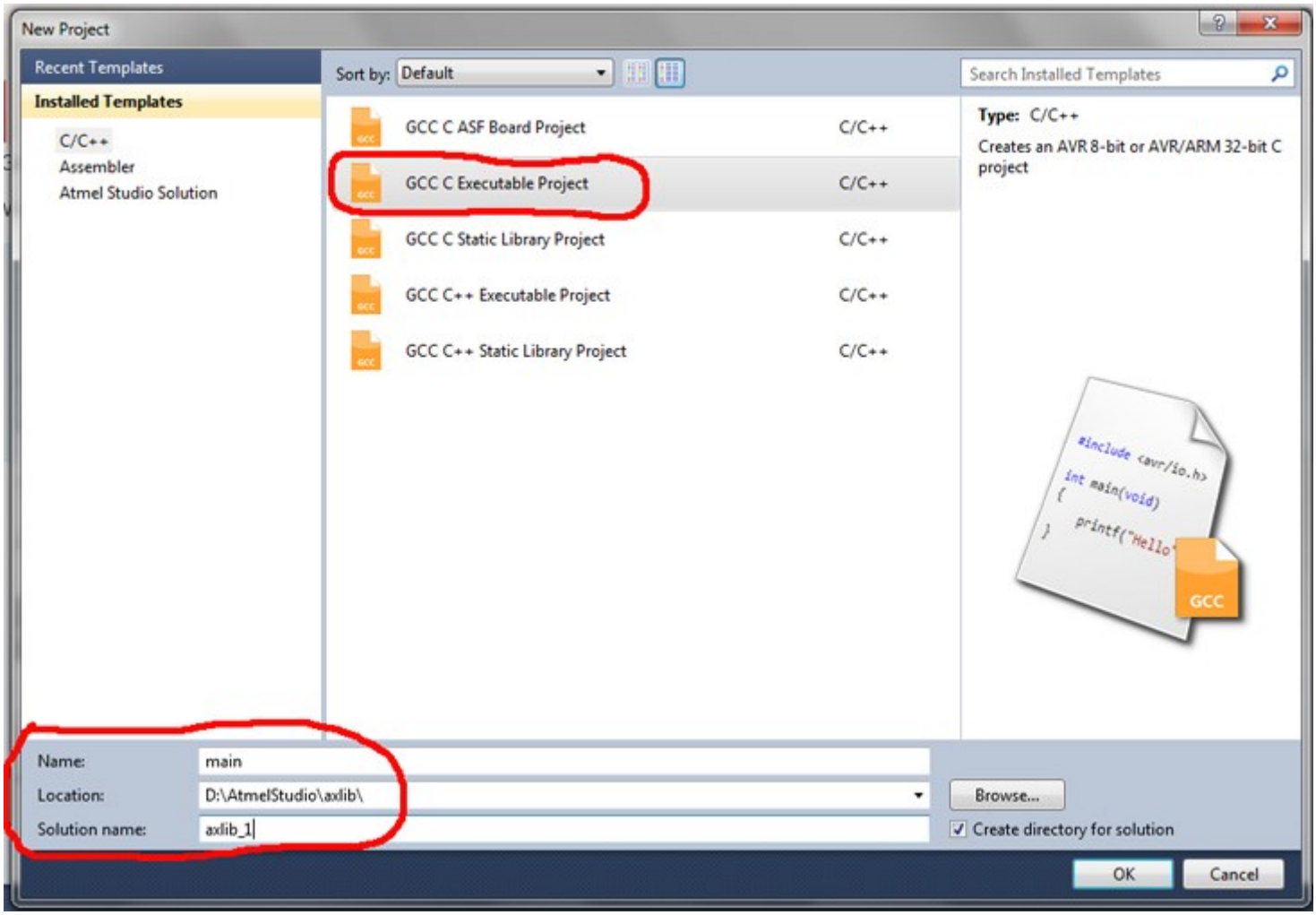
Вот это вот 3.4.1056 у вас может называться по-другому из-за версии, но суть остается та же. После переноса папки axlib можно приступать к использованию.



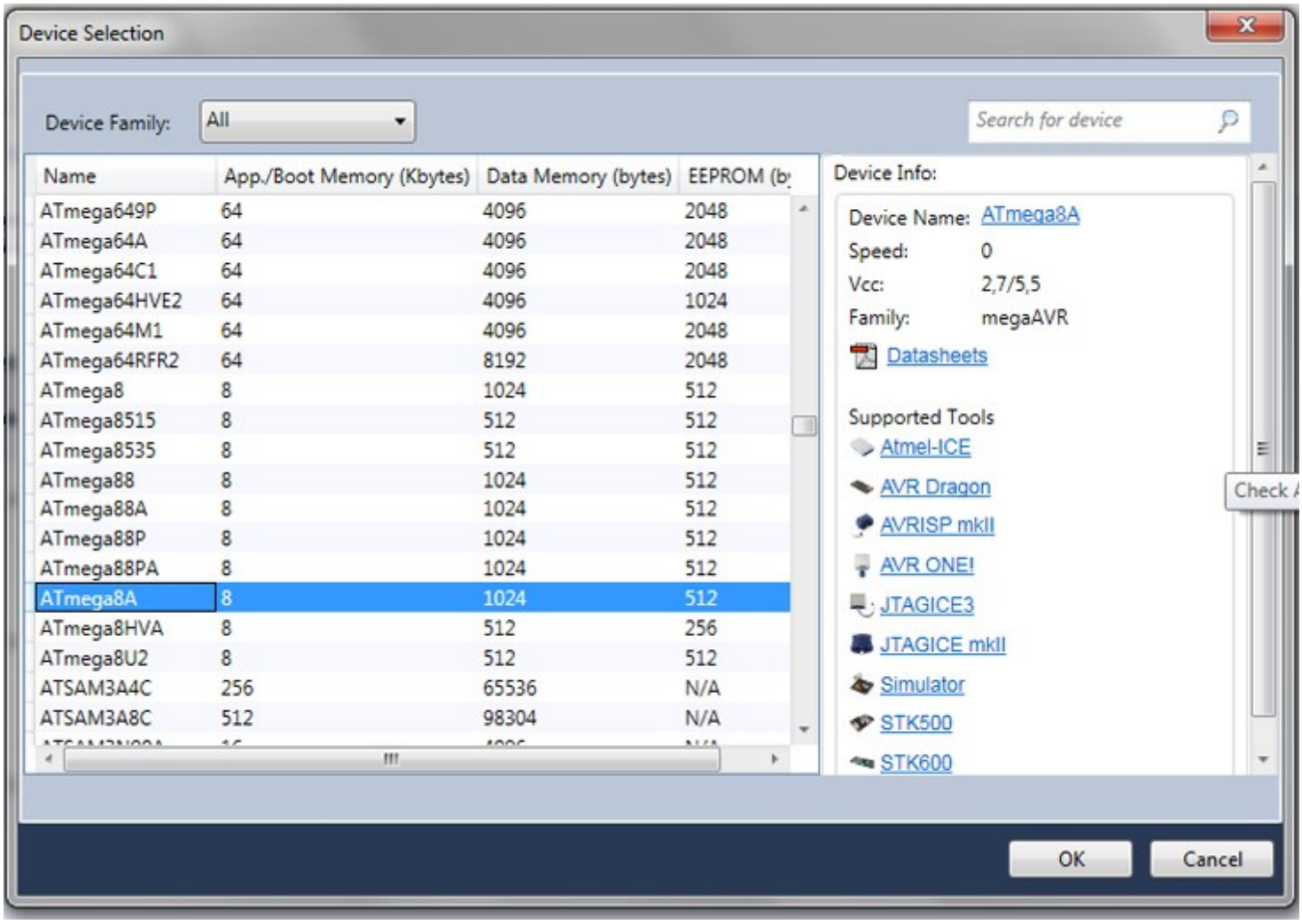
Буду расписывать с самого начала. Запускаем программу AtmelStudio 6.2 и создаем новый проект, нажав на ссылку слева вверху *New Project*...



В появившемся окне, выбираем GCC C Executable Project, а в нижних трех строчках *Name*: - это имя главного файла и имя бинарного файла который нужно загрузить в МК. *Location*: - это адрес где будет создан проект со всеми файлами. *Solution name*: - это название которое появится слева при старте программы в поле *Recent Projects*. Если по нему потом щелкнуть, то загрузится проект.



Жмем ОК и видим новое окно. В нем выбираем МК, с которым будем работать. В данный момент у меня это ATmega8A.



Жмем ОК и ждем пока не будет создан проект. Немного подумав, программа родит окно, в котором будет вот такая запись.

main.c

```c
/*
 * main.c
 *
 * Created: 15.06.2015 22:36:56
 *  Author: Алексей
 */


#include <avr/io.h>

int main(void)
{
    while(1)
    {
        //TODO:: Please write your application code
    }
}
```

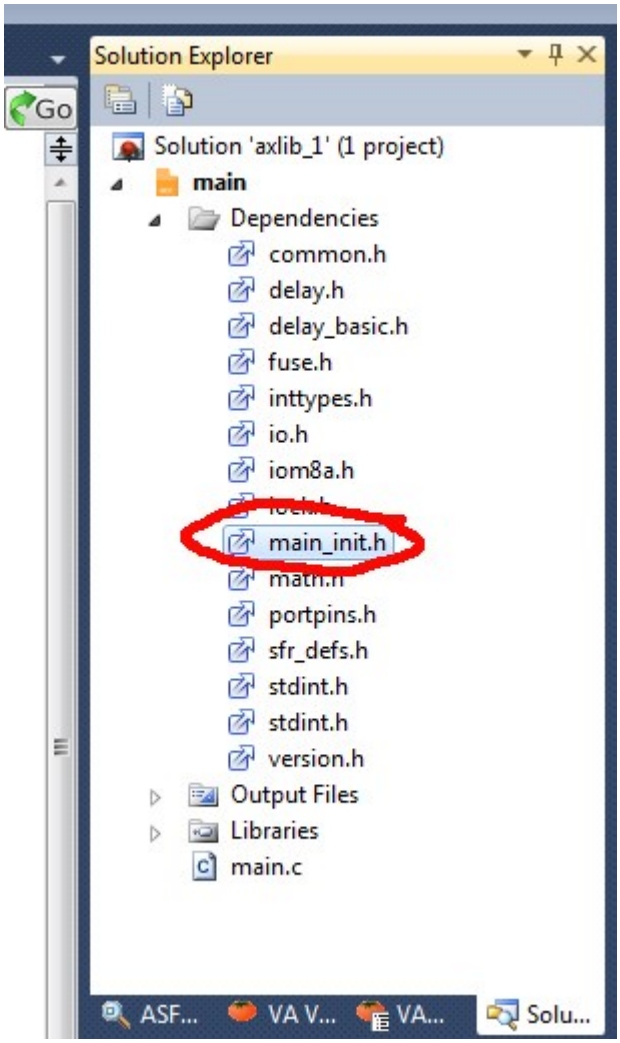
Нажимаем F7 и видим в нижнем поле процесс компиляции, а в конце сообщение, что все прошло отлично. То есть мы скомпилировали пустой проект. А вот теперь давайте начнем работать с библиотекой axlib.

Библиотека сделана таким образом, что вносить какие-либо поправки нужно только в одном файле. (Изначально данный файл находился в той же директории, что и все остальные файлы).

В связи с частым переносом проектов было принято решение данный файл переместить в корень проекта. Теперь расположения файла main_init.h должно быть там же где и основной файл программы. Для тех, кто использует старый вариант, ничего не меняется. Лишь скачайте новую версию библиотеки и при создании нового проекта перенесите файл main_init.h в корень проекта. Теперь для каждого проекта будет свой файл main_init.h) Он называется main_init.h. Как ни странно, но для подключения тех или иных функций необходимо первым делом подключить именно этот файл. Делается это так. После строки #include <avr/io.h> пишем #include "main_init.h" (Старый вариант #include <axlib/main_init.h>)

```
#include <avr/io.h>
#include "main_init.h"
int main(void)
{
  while(1)
  {
  }
}
```

Теперь снова жмем F7. После компиляции справа в окне появится наш файл. Жмем на него двойным щелчком.



Появится тело файла. Что собственно здесь написано и что здесь нужно править.

```
 8
 9  // Частота используемого кварца
10
11  #define F_CPU 7372800UL
12
13  // Подключение необходимых библиотек
14
15  #include <util/delay.h>
16
17  // Оределение типов данных
18
19  #define BYTE    char
20  #define WORD    int
21  #define DWORD   long
22  #define ADATA   float
23
24  //-----------------------------------------------------------------------
25  //  Параметры для работы с ЖК знакосинтезирующим дисплеем до
26  //  40 символов и 4 строк в 4-ч битном режиме.
27  //
28  //  Здесь необходимо указать порты
29  //  и пины подключенные к соответствующим
30  //  пинам МК и ЖК
31  //
32  //  Ножку R/W на дисплее нужно завести на GND
33  //-----------------------------------------------------------------------
34  #define DDR_RS  DDRB    // Порт на котором будет RS
35  #define DDR_E   DDRD    // Порт на котором будет E
36  #define DDR_D4  DDRD    // Порт на котором будет D4
37  #define DDR_D5  DDRD    // Порт на котором будет D5
38  #define DDR_D6  DDRD    // Порт на котором будет D6
39  #define DDR_D7  DDRD    // Порт на котором будет D7
40
41  #define PORT_RS PORTB   // Порт на котором будет RS
42  #define PORT_E  PORTD   // Порт на котором будет E
43  #define PORT_D4 PORTD   // Порт на котором будет D4
44  #define PORT_D5 PORTD   // Порт на котором будет D5
45  #define PORT_D6 PORTD   // Порт на котором будет D6
46  #define PORT_D7 PORTD   // Порт на котором будет D7
47
48  #define RS      0       // Номер пина RS
49  #define E       2       // Номер пина E
50  #define D4      4       // Номер пина D4
51  #define D5      5       // Номер пина D5
52  #define D6      6       // Номер пина D6
53  #define D7      7       // Номер пина D7
54
55  #endif /* MAIN_INIT_H_ */
```

Самым первым делом нужно указать, на какой частоте будет работать МК. Поэтому после сборки железа, первым делом вписываем эту частоту в герцах. На данный момент я тестировал библиотеку на МК с частотой 7,3728 МГц, поэтому сейчас там записана такая частота. Обязательно поменяйте ее на свою, так как это важно для расчетов некоторых значений регистров периферии. Далее идет определение типов данных. Их трогать не нужно. За ними следует настройка подключения ЖК знакосинтезирующего дисплея по 4-х битной шине. Если использование ЖК не планируется, то можно ничего не трогать. Если же ЖК будет подключатся, то необходимо прописать на каких портах и каких пинах будут подключены ножки МК и ЖК. Сложного ничего нет, просто меняем на ту букву порта, на которой подключен ЖК, а в конце прописываем номера пинов. Вот и вся настройка. (Пока)

По вопросам и замечаниям, а так же предложениям, можно обратится на форум.
Генератор кода для AtmelStudio 6.x с поддержкой библиотеки AXLIB

Порты ввода/вывода

Для тех кто любит писать так *PORTC.0 = 1;*
Мой вариант работы с портами ввода/вывода.
Подключаемый файл библиотеки называется ports.h

```
#include <avr/io.h>
#include "main_init.h"
#include <axlib/ports.h>

int main(void)
{
   while(1)
   {

   }
}
```

Принцип инициализации был взят из Arduino и немного упрощен.

Для настройки всего порта на вывод: **PORTx_OUT_ALL()**;
где x - это буква порта A,B,C,D.

Для настройки всего порта на ввод: **PORTx_IN_ALL();**
где x - это буква порта A,B,C,D.

Для настройки одного пина на вывод: **PORTx_OUT(pin);**
где x - это буква порта A,B,C,D, а pin - это номер разряда.

Для настройки одного пина на ввод: **PORTx_IN(pin);**
где x - это буква порта A,B,C,D, а pin - это номер разряда.

Если порт настроен на вывод:

Для вывода байта в порт: **PORTx_DATA_OUT(byte);**
где x - это буква порта A,B,C,D, а byte - это байт для вывода.

Если порт настроен на ввод:

Для чтения байта в порт: **PORTx_DATA_IN();**
где x - это буква порта A,B,C,D.

Если порт настроен на вывод:

Для вывода 1 на одном разряде: **PORTx_ON(pin);**
где x - это буква порта A,B,C,D, а pin - это номер разряда.

Для вывода 0 на одном разряде: **PORTx_OFF(pin);**
где x - это буква порта A,B,C,D, а pin - это номер разряда.

Если порт настроен на ввод:

Для чтения одного разряда: **PORTx_RD(pin);**
где x - это буква порта A,B,C,D, а pin - это номер разряда.

Для инверсии одного разряда: **PORTx_XOR(pin);**
где x - это буква порта A,B,C,D, а pin - это номер разряда.

Пример

```
  #include <avr/io.h>
  #include "main_init.h"
  #include <axlib/ports.h>

int main(void)
{
        PORTC_IN(1);            // Пин 1 порта C настроен на вход
        PORTC_OUT(2);           // Пин 2 порта C настроен на выход
        PORTC_OUT(3);           // Пин 3 порта C настроен на выход

        PORTC_ON(1);            // Подключение подтягивающего резистора к питанию

  while(1)
  {
    PORTC_ON(3);                // Вывесть 1 на 3 пин
        _delay_ms(500);
        PORTC_OFF(3);           // Вывести 0 на 3 пин
        _delay_ms(500);

        if(!PORTC_RD(1)) PORTC_XOR(2);        // Если на пине 1 порта C будет 0, то    инвертировать пин 2 порта C
  }
}
```

Таймеры

Очень часто многие задают вопрос, как можно выполнять разные действия с разным периодом времени и при этом не тормозить МК. Ну, очевидный ответ, это использовать прерывания таймера. К сожалению, мудреная настройка таймеров вгоняет в уныние новичков и они чаще бросают эту затею и переходят на Arduino. Давайте решим эту проблему. Файл timers.h инициализирует прерывания от таймера T2 и используя его ресурсы может создавать восемь почти независимых таймеров и обработчиков для каждого. Фактически восемь псевдопрерываний.

Для подключения файла библиотеки нужно прописать строку вида #include <axlib/timers.h>

Затем в начале тела функции main(), перед основным циклом программы, необходимо прописать функцию инициализации таймеров**. timers_init();.** Данная функция рассчитает необходимые значения и заполнит все необходимые регистры.

Для включения одного из восьми таймеров:

timer(TIMER_x, t, n);

где x - это номер таймера от 0 до 7, t - время периода в мс, n - режим ON включен, OFF выключен.

Пример: **timer(TIMER_4, 250, ON);**

Включить таймер 4 и выполнять действия через 250 мс.

Обработчик таймера.

Теперь после того как таймер включен, необходимо в теле основной программы написать обработчик таймера. Как он выглядит.

```
if(Tx_START)
{
   // Здесь нужно написать код который должен выполнятся
   Tx_STOP;
}
```

Вот и весь обработчик. x - это номер таймера который мы включили, а вместо комментария нужно вписать какой-то код, который необходимо выполнять в определенный период.

Пример.

```
#include <avr/io.h>
#include "main_init.h"
#include <axlib/ports.h>        // Подключили библиотеку
#include <axlib/timers.h>       // Подключили библиотеку

int main(void)
{
   timers_init();               // Проинициализировали тамер T2

   timer(TIMER_0, 500, ON);     // Включили таймер 0 на работу раз в пол секунды
   PORTC_OUT(0);                // Настраиваем 0 пин порта C на выход

   while(1)
   {
      if(T0_START)
      {
         PORTC_XOR(0);          // Пин 0 порта C будет инвертироваться раз в 500 мс
         T0_STOP;
      }
   }
}
```

ЖК дисплей с максимальным разрешением 4x40

Не всегда удобно индицировать работу того или иного устройства при помощи светодиодов. Иногда хочется вывести какой-либо текст или значение. Для этих целей существует родная библиотека под названием stdio.h. Данная библиотека сформирует массив для вывода, но... Куда выводить? Вот для этих целей в комплект axlib входит файл для

работы с ЖК знакосинтезирующими дисплеями с максимальным разрешением 4 строки 40 символов в каждой. Называется он lcd.h

Первым делом необходимо указать порты и пины на которых подключен дисплей. Это делается методом корректировки файла main_init.h. Файл в этом месте подробно закомментирован и в пояснении не нуждается. Для подключения файла библиотеки необходимо написать строку: #include <axlib/lcd.h>

(В библиотеке axlib добавились функции для работы со сдвиговым регистром 74HC595. Данный регистр может выступать шлюзом для подключения ЖК дисплея, что приводит к экономии двух пинов МК. Для перевода функций ЖК дисплея в режим шлюза, необходимо раскомментировать строку #define R74HC595_LCD в файле main_init.h После с функциями ЖК дисплея можно работать как будто дисплей подключен напрямую к МК. Схему подключения дисплея к сдвиговому регистру можно посмотреть в файле main_init.h)

Функции:

void lcd_init(BYTE lcd) - это функция должна быть вызвана самой первой. Данная функция инициализирует дисплей для работы в четырех байтном режиме. В качестве аргумента, функция получает одно из четырех значений.
0 - Курсор установлен в верхний левый угол. Курсор не виден.
1 - Курсор установлен в верхний левый угол. Курсор в виде мерцающего символа.
2 - Курсор установлен в верхний левый угол. Курсор в виде подчеркивания.
3 - Курсор установлен в верхний левый угол. Курсор в виде подчеркивания и мерцающего символа.

void lcd_gotoxy(BYTE x, BYTE y) - эта функция перемещает курсор в заданный адрес. Получаемые аргументы координаты адреса.
x значение положения символа в строке. Принимает значение от 0 до 39.
y значение положения символа по строке. Принимает значение от 0 до 3.

void lcd_clear(void) - эта функция очищает дисплей и переносит курсор в верхний левый угол.

void lcd_char_out(BYTE data) - эта функция выводит на дисплее, по текущему адресу расположения курсора, один символ. Принимаемый аргумент восьми битный код символа.

void lcd_str_out(BYTE *str) - эта функция выводит строку завершенную нулевым символом. Принимаемый аргумент указатель на первый элемент массива, в котором записана строка.

void lcd_simbol(BYTE simbol, BYTE *str) - эта функция предназначеня для создания своего символа. Первый принимаемый аргумент это адрес памяти CGRAM, а второй, указатель на массив с байтами строк символа.

Пример.

```
#include <avr/io.h>
#include "main_init.h"
#include <axlib/lcd.h>  // Подключили библиотеку

int main(void)
{
   lcd_init(0); // Инициализация ЖК в 4-х битном режиме. Курсор не виден

   while(1)
   {
      lcd_gotoxy(0,0);          // Установили курсор на первую строку и первый символ
      lcd_char_out('Q');        // Вывели символ Q
      lcd_gotoxy(0,1);          // Установили курсор на вторую строку и первый символ
```

```
    lcd_str_out("Test axlib"); // Вывели текст Test axlib
  }
}
```

АЦП Аналогово-цифровой преобразователь

Данный набор функций дает возможность легко настроить АЦП и получить данные от подключенного устройства. Для подключения файла нужно написать строку #include <axlib/adc.h>

Функции:

void adc_init(ADATA aref, BYTE inref) - эта функция должна вызываться самой первой. Аргументы, передаваемые функции:
aref Величина опорного напряжения АЦП. Принимает любое вещественное число, но нужно хорошо помнить, что опорное напряжение АЦП не должно превышать напряжения питания самого МК. Если МК питается от 5 вольт, то опорное напряжение не должно быть выше 5 вольт. Если от 3,3 вольт, то соответственно опорное не должно быть выше 3,3 вольта.
inref Включить или не включать внутренний ИОН. Принимает значения [ADC_IREF_ON] включен, [ADC_IREF_OFF] выключен.

WORD adc_data(BYTE ch) - эта функция возвращает значения регистра после выборки. Возвращаемое значение лежит в диапазоне от 0 до 1023. В качестве аргумента принимает BYTE ch номер канала. Номер канала можно менять в теле программы. При вызове данной функции в цикле, можно получить значения всех каналов.

ADATA adc_volt(BYTE ch) - эта функция возвращает значение в вольтах вычисленных из данных полученных от АЦП. Возвращают значение от 0 вольт до величины опорного напряжения. BYTE ch принимает номер канала. Аналогично предыдущей функции.

Пример.

```
#include <avr/io.h>

#include "main_init.h"
#include <axlib/adc.h>                    // Подключили библиотеку

WORD data = 0;
ADATA volt = 0.0;

int main(void)
{
  adc_init(1.1, ADC_IREF_ON);          // Инициализация АЦП с опорным напряжением  1.1 вольт
                                       // и включенным внутренним ИОН.

  while(1)
  {
    data = adc_data(ADC_CH_0);       // Теперь в переменной data лежит значение  АЦП канала 0
    volt = adc_volt(ADC_CH_1);       // Теперь в переменной volt лежит величина  напряжения канала 1
  }
}
```

UART

Общение с UART я решил изобразить на подобие Arduino. Уж больно мне понравилось удобства настройки. Для подключения файла библиотеки нужно написать строку: #include <axlib/usart.h>

Функции.

void usart_init(DWORD baudrate) - эта функция должна вызывается самая первая. Принимаемый аргумент, скорость передачи в БОД. Настраивает USART на передачу в асинхронном режиме, 8 бит данных, 1 стоп-бит и без контроля четности.

void usart_char_out(BYTE data) - эта функция принимает в качестве аргумента один байт и посылает его в порт.

BYTE usart_char_in(void) - эта функция возвращает полученный байт из порта.

void usart_str_out(BYTE *str, BYTE count) - эта функция посылает в порт строку. Принимаемый аргумент указатель на первый элемент массива. Второй аргумент - длина передаваемых байт.

void usart_str_rn(BYTE *str) - эта функция эквивалентна первой за исключением того, что она после отправки строки в порт отсылает в конце два служебных символа \r и \n. Удобно для работы с GSM-модулями.

BYTE usart_str_in(BYTE *str, BYTE count) - эта функция принимает строку из порта и кладет ее в массив. Принимаемые аргументы, указатель на первый элемент массива и количество принимаемых байт. После вызова данной функции, в массиве, переданном в виде аргумента, будет полученная строка, а функция вернет количество принятых байт. Так же функция имеет таймаут. Если в течении 250мс данные не приходят, то функция возвращает 0

Пример.

```
#include <avr/io.h>
#include "main_init.h"
#include <axlib/usart.h>

BYTE data = 0;
BYTE string[20];

int main(void)
{
	usart_init(9600);		// Инициализация USART скорость 9600, 8 бит, 1 стоп, без паритета

  while(1)
  {
	_delay_ms(1000);		// Секундная задержка чтобы не повесить терминал
	usart_char_out('Q');		// Посылаем в порт букву Q
	data = usart_char_in();		// Получили байт в переменную data
	usart_char_out(data);		// Посылаем в порт Значение переменной data
					// Отправили строку в порт
	usart_str_out("Передача по USART при помощи библиотеки axlib");
	usart_str_in(string, 10);	// Получили строку в 10 байт из порта в массив string[]
	usart_str_rn("Передача строки"); // В конце строки передастся в порт \r\n
  }
}
```

1WIRE

Эта часть библиотеки axlib предназначена для связи устройств связанных шиной 1wire. Для подключения библиотеки нужно написать строку: #include <axlib/1w.h>. Так же в файле main_init.h необходимо прописать порт и пин для линии данных.

Функции.

BYTE owire_init() - эта функция инициализирует шину и возвращает 1 если присутствует хотя бы одно устройство на шине. Если на шине нет устройств, то функция возвращает 0.
void owire_write(BYTE b) - эта функция передает байт на шину.
BYTE owire_read() - эта функция читает байт из шины.

Пример.

```
#include <avr/io.h>
#include "main_init.h"
#include <axlib/1w.h>

BYTE data = 0;

int main(void)
{
	owire_init();		// Инициализация 1wire

  while(1)
  {
	owire_write(0x00);	// Посылаем устройству байт 0x00
	data = owire_read();	// Получили байт от устройства в переменную data
  }
}
```

I2C

Для подключения файла библиотеки нужно написать строку: #include <axlib/i2c.h>. Так же в файле main_init.h необходимо прописать порт и пины для линий данных и стробирования.

Функции.

void i2c_init(void) - эта функция инициализирует шину.
BYTE i2c_stop(void) - эта функция выдает на линию команду СТОП. Возвращает значения: OK если на линии все в порядке, либо SCL_FAIL если ошибка линии SCL, либо SDA_FAIL если ошибка на линии SDA, либо SDA_SCL_FAIL если ошибки на обеих линиях.
void i2c_start(void) - эта функция выдает на шину команду СТАРТ.
void i2c_restart(void) - эта функция выдает на шину команду ПОВСТАРТ.
BYTE i2c_send_byte(BYTE data) - эта функция выдает на шину байт данных. Возвращает значение ACK если байт принят устройством и NACK если устройство не приняло байт.
BYTE i2c_read_byte(BYTE ask) - эта функция читает байт из шины. Передаваемый параметр либо ACK, либо NACK. Зависит от того как ответить устройству по окончанию приема байта.

Пример.

```
#include <avr/io.h>
#include "main_init.h"
#include <axlib/i2c.h>
BYTE data = 0;

int main(void)
{

  while(1)
  {
        i2c_start();                    // Команда СТАРТ
        i2c_send_byte(0xD0);            // Передача байта 0xD0
        i2c_send_byte(0x0E);            // Передача байта 0x0E
        i2c_restart();                  // Команда ПОВСТАРТ
        i2c_send_byte(0xD1);            // Передача байта 0xD1
        data = i2c_read_byte(NACK);     // Чтение байта из шины и посылка NACK
        i2c_stop();                     // Команда СТОП
  }
}
```

Датчик давления от ВАЗ-2103 на прибор

Данный датчик подключается в режиме резисторного делителя и устанавливается в верхнее плечо. Схему включения можно посмотреть в файле main_init.h. Для подключения файла библиотеки нужно написать строку: #include <axlib/psensor_vaz.h>.

Функции.

void p_sensor_vaz_init(void) - эта функция инициализирует датчик с параметрами по умолчанию. 5в опорное напряжение, сопротивление датчика при нуле 300 Ом, сопротивление нижнего плеча 22 Ом, максимальное давление 8 кг/см, внутренний ИОН отключен.

void p_sensor_vaz_init_data(WORD rs, WORD r2, ADATA pmax, ADATA u_in, BYTE inref) - эта функция инициализирует датчик с устанавливаемыми параметрами. Данную функцию необходимо вызвать взамен предыдущей, если в схеме присутствуют данные отличные от тех, что заданы по умолчанию. Например, используется датчик от другого автомобиля и его сопротивление отличается от заданного по умолчанию. Либо используется схема с питанием на 3,3в. Передаваемые аргументы функции:

WORD rs - это сопротивление датчика при нуле в Ом

WORD r2 - это величина сопротивления в Ом

ADATA pmax - это максимальное давление в кг/см

ADATA u_in - это величина опорного напряжения АЦП

BYTE inref - ключить/выключить внутренний ИОН. [ADC_IREF_ON], [ADC_IREF_OFF].

ADATA press_vaz(BYTE adc_ch) - эта функция возвращает значение давления с типом ADATA. В качестве аргумента BYTE adc_ch принимает номер канала, к которому подключен датчик.

Пример.

```
#include <avr/io.h>
#include <stdio.h>
#include "main_init.h"
#include <axlib/lcd.h>
```

```
#include <axlib/psensor_vaz.h>

int main(void)
{
	p_sensor_vaz_init();		// Инициализация датчика с параметрами по умолчанию
	lcd_init(0);
	BYTE str[10];

  while(1)
  {
		lcd_gotoxy(0,0);
		sprintf(str, "P %0.2f", press_vaz(ADC_CH_0));
		lcd_str_out(str);		// После получения данных на ЖК дисплее будет	видно давление.
  }
}
```

Датчик температуры DS1820

Перед тем как подключать файл библиотеки ds1820.h, необходимо сначала подключить файл библиотеки 1w.h. Для подключения файла библиотеки нужно написать строку: #include <axlib/ds1820.h>.

Функции.

BYTE ds1820_rom_code(BYTE *str) - эта функция вернет ROM код датчика при условии что он является единственным устройством на шине.

WORD ds1820_all(void) - эта функция вернет температуру умноженную на 10 при условии что на шине установлен единственный датчик. Так же после вызова данной функции в глобальной переменной ZNAK запишется знак температуры, либо MINUS, либо PLUS.

WORD ds1820_read_t(BYTE *rom) - эта функция вернет температуру умноженную на 10 датчика, чей ROM код совпадет с передаваемым. Передаваемый аргумент, ROM код датчика. Применяется при нахождении на шине более одного устройства. Так же после вызова данной функции в глобальной переменной ZNAK запишется знак температуры, либо MINUS, либо PLUS.

Пример.

```
#include <avr/io.h>
#include <stdio.h>
#include "main_init.h"
#include <axlib/1w.h>
#include <axlib/ds1820.h>
#include <axlib/lcd.h>

int main(void)
{
	owire_init();		// Инициализация шины 1wire
	lcd_init(0);

	BYTE str[10];
```

```
  while(1)
  {
                lcd_gotoxy(0,0);
                sprintf(str, "T  %0.1f", (ADATA)(ds1820_all()/10));
        if(ZNAK == MINUS)               // Если температура отрицательная
        {
          str[2] = '-';
        }
        else
        {
          str[2] = ' ';
        }
                lcd_str_out(str);       // После получения данных на ЖК дисплее будет     видно температуру.
  }
}
```

SPI

Для подключения файла библиотеки нужно написать строку: #include <axlib/spi.h>.

Функции:

void SPI_init(SPI_InitTypeDef *spi); Данная функция принимает в качестве аргумента указатель на структуру которую необходимо заполнить перед вызовом данной функции с требуемыми параметрами.

UBYTE SPI_M_byte_io(BYTE data); Данная функция предназначена для передачи и приема байта по шине SPI. Получаемый аргумент, есть передаваемый байт. Возвращаемое значение, есть полученный байт. Данная функция работает только в режиме Master.

UBYTE SPI_S_byte_io(BYTE data, BYTE timeout); Данная функция предназначена для передачи и приема байта по шине SPI. Получаемые аргументы функции это передаваемый байт мастеру и время действия функции в миллисекундах (таймаут). Если данные не пришли до истечения заданного времени, то функция возвращает значение FULL. В случае удачного получения/передачи данных, функция возвращает полученный байт. Данная функция работает только в режиме Slave.

Значение структуры

```
SPI_InitTypeDef typedef struct
{
        BYTE SPI_set;           // Включить/выключить SPI
        BYTE SPI_Mode;          // Выбор Мастер или Слейв
        BYTE SPI_Direct;        // Выбор направления передачи байта
        BYTE SPI_Prescaler;     // Выбор предделителя
        BYTE SPI_Polaric;       // Выбор полярности тактового сигнала
        BYTE SPI_Phase;         // Выбор фазы тактового сигнала

}SPI_InitTypeDef;
```

Пример для Master

```
#include <avr/io.h>
#include "main_init.h";
#include <axlib/spi.h>
BYTE data_in = 0x00;
BYTE data_out = 0x00;

int main(void)
{

// Инициализация SPI
SPI_InitTypeDef SPI_InitType;
  // Включить SPI
  SPI_InitType.SPI_set = SPI_ON;
  // Выбрать режим Master
  SPI_InitType.SPI_Mode = SPI_MASTER;
  // Выбрать претделитель (F_CPU/2)
  SPI_InitType.SPI_Prescaler = SPI_PRESCALER_2;
  // Задание полярности тактового сигнала
  SPI_InitType.SPI_Polaric = SPI_CPOL_LOW;
  // Задание фазы тактового сигнала
  SPI_InitType.SPI_Phase = SPI_CPHA_1EDGE;
  // Выбор направления передачи данных
  SPI_InitType.SPI_Direct = SPI_DIRECT_MSB;
  // Инициализация SPI
 SPI_init(&SPI_InitType);

  while(1)
  {
    data_in = SPI_M_byte_io(data_out); // Значение data отправлено в порт
    data_out = data_in;
  }
}
```

Пример для Slave

```
#include <avr/io.h>
#include <stdio.h>
#include "main_init.h"
#include <axlib/spi.h>
#include <axlib/lcd.h>
BYTE data = 0;
BYTE data_old = 0;
BYTE str[20];

int main(void)
{

// Инициализация SPI
SPI_InitTypeDef SPI_InitType;
  // Включить SPI
  SPI_InitType.SPI_set = SPI_ON;
```

```
  // Выбрать режим Slave
  SPI_InitType.SPI_Mode = SPI_SLAVE;
  // Претделитель не нужен
  SPI_InitType.SPI_Prescaler = SPI_PRESCALER_NO;
  // Задание полярности тактового сигнала
  SPI_InitType.SPI_Polaric = SPI_CPOL_LOW;
  // Задание фазы тактового сигнала
  SPI_InitType.SPI_Phase = SPI_CPHA_1EDGE;
  // Выбор направления передачи данных
  SPI_InitType.SPI_Direct = SPI_DIRECT_MSB;
  SPI_init(&SPI_InitType); // Инициализация SPI
  lcd_init(0); // Инициализация LCD для вывода данных

  while(1)
  {
    data = SPI_S_byte_io(0xFF, 10); // Получение байта из порта

    // Проверка на получение новых данных
    if((data != data_old) && (data != 0xFF))
     {
          data_old = data;          // Обновить старые данные новыми
          lcd_gotoxy(0, 0);         // Установить курсор
          sprintf(str, "0x%02X", data); // Подготовить данные для вывода
          lcd_str_out(str);         // Вывести полученные данные на LCD
          data++;                   // Увеличить полученные данные на единицу
          SPI_S_byte_io(data, 10); // Отправить увеличенные данные мастеру
     }
  }
}
```

Данная программа получает от мастера байт, проверяет, не изменился ли он по отношению предыдущего полученного байта и если разница есть, то записывает текущее значение как старые, выводит на дисплей, увеличивает на единицу и отправляет мастеру. Мастер, получив байт от слейва, тут же возвращает его назад. Если данные пример залить в МК то на дисплее появится нулевое значение, а затем каждую секунду оно будет увеличиваться на единицу.

Цветной TFT дисплей с управляющим контроллером ST7735

Эта часть библиотеки предназначена для вывода графической и буквенной информации на цветной TFT дисплей, управляемый контроллером ST7735: #include <axlib/st7735.h>.

Функции:

void lcd_st7735_init(void); Данная функция производит инициализацию дисплея и настраивает шину SPI для работы с дисплеем. Перед использованием данной функции, необходимо внести значения вспомогательных выводов RES, A0 и CS в файле main_init.h.

```
//-----------------------------------------------------------------------
//  Параметры необходимые для работы с TFT дисплеем ST7735 160x128
//
```

```
// Даннае значения нужны для аппаратной части работы с дисплеем
// CS необходим для включения дисплея
// A0 необходим для выбора данные/команда
// RES необходим для аппаратного сброса дисплея
//-----------------------------------------------------------------------

#define DDR_CS          DDRD   // Порт на котором будет CS
#define DDR_A0          DDRD   // Порт на котором будет A0
#define DDR_RES         DDRD   // Порт на котором будет RES

#define PORT_CS         PORTD  // Порт на котором будет CS
#define PORT_A0         PORTD  // Порт на котором будет A0
#define PORT_RES        PORTD  // Порт на котором будет RES

#define CS              0      // Номер пина для CS
#define A0              1      // Номер пина для A0
#define RES             2      // Номер пина для RES
```

---

void lcd_st7735_screen(UWORD color); Данная функция заливает весь дисплей одним цветом.

void lcd_st7735_full_rect(BYTE startX, BYTE startY, BYTE stopX, BYTE stopY, UWORD color); Данная функция отрисовывает залитый прямоугольник по координатам.

void lcd_st7735_put_pix(BYTE x, BYTE y, UWORD color); Данная функция включает пиксель по координатам с заданным цветом.

void lcd_st7735_rect(BYTE x1, BYTE y1, BYTE x2, BYTE y2, WORD color); Данная функция отрисовывает пустой прямоугольник по координатам.

void lcd_st7735_line(WORD x0, WORD y0, WORD x1, WORD y1, UWORD color); Данная функция отрисовывает прямую линию по координатам.

void lcd_st7735_putchar(BYTE x, BYTE y, BYTE chr, UWORD charColor, UWORD bkgColor); Данная функция выводит символ по координатам с заданным цветом символа и фона. (Для вывода русских символов, необходимо установить на компьютер с Atmel Studio 6 русскоязычную Visual Studio 10 или выше. После установки запустить Atmel studio 6. Выбрать Русский в меню *Tools -> Options -> Environment -> International settings.* Перезагрузить Atmel Studio 6).

void lcd_st7735_putstr(BYTE x, BYTE y, const BYTE str[], UWORD charColor, UWORD bkgColor); Данная функция выводит строку по координатам с заданным цветом символа и фона.

void lcd_st7735_putstr_xy(BYTE x, BYTE y, const BYTE str[], UWORD charColor, UWORD bkgColor); Данная функция выводит строку по координатам курсора с заданным цветом символа и фона. Нулевая строка начинается снизу экрана. Всего в экран умещается 10 строк. Соответственно от 0 до 9. Количество возможных символов для вывода равно 20 шт. Координата x, указывает положение курсора от 0 до 19, а координата y, номер строки.

Пример:

```
#include <avr/io.h>
#include "main_init.h"
#include <axlib/spi.h>
```

```
#include <axlib/st7735.h>

int main(void)
{
	// Инициализация дисплея
	lcd_st7735_init();
	// Заливка экрана белым цветом
	lcd_st7735_screen(RGB16(0xFFFFFF));
	// Отрисовка закрашенного прямоугольника по координатам  зеленым цветом
	lcd_st7735_full_rect(20, 20, 139, 107, RGB16(0x00FF00));
	// Отрисовка контура прямоугольника красным цветом
	lcd_st7735_rect(20, 20, 139, 107, RGB16(0xFF0000));
	// Отрисовка диоганалей синим цветом
	lcd_st7735_line(20, 20, 139, 107, RGB16(0x0000FF));
	lcd_st7735_line(20, 107, 139, 20, RGB16(0x0000FF));
	// Вывод буквы по координатам пикселей
	lcd_st7735_putchar(1, 112, 'Я', RGB16(0xB906FC), RGB16(0xFFFFFF));
	// Вывод строки по координатам пикселей
	lcd_st7735_putstr(15, 112, "и библиотека axlib", RGB16(0xFCA206), RGB16(0xFFFFFF));
	// Вывод строки по координатам курсора
	lcd_st7735_putstr_xy(0, 0, "Нижняя строка", RGB16(0xEEFC06), RGB16(0x0000FF));

  while(1)
  {

  }
}
```

Функции для работы с часами DS1307

Для начала общения с микросхемой необходимо в файле main_init.h прописать порт и пины для шины i2c.

```
//-----------------------------------------------------------------------
//	Для работы с шиной I2C необходимо выбрать порт и пины на данном порту//
//-----------------------------------------------------------------------

#define I2C_DDR		DDRC		// Порт для выбора направления
#define I2C_PORT	PORTC		// Порт для вывода данных
#define I2C_PIN		PINC		// Порт для ввода данных
#define SCL		0		// Пин порта SCL
#define SDA		1		// Пин порта SDA
```

После настройки аппаратной части шины i2c можно приступать непосредственно к работе с микросхемой. Для подключения функций необходимо подключит файл ds1307.h

<u>Описание функций:</u>

void ds1307_init(void); Данная функция вызывается один раз для инициализации часов. При ее вызове данные в микросхеме не меняются. Если были установлены время, дата и режим работы вывода SQW/OUT, то после вызова данной функции они не изменятся.

void ds1307_write_time(BYTE h1224, BYTE hours, BYTE minutes, BYTE seconds); Данная функция записывает время.

Принимает аргументы:

BYTE h1224 Режим формата часов.

DS1307_24 24 часовой формат

DS1307_AM 12 часовой формат. Режим AM

DS1307_PM 12 часовой формат. Режим PM

BYTE hours Запись часов. При выборе 12 часового формата, функция автоматически переведет значение часов, если ей передали значение больше 12.

BYTE minutes Запись минут.

BYTE seconds Запись секунд.

BYTE ds1307_read_time(BYTE *str); Данная функция читает время.

Принимает аргумент:

BYTE *str Указатель на первый элемент массива. Массив должен иметь 3 элемента.

В первый элемент массива функция запишет часы.

Во второй элемент массива функция запишет минуты.

В третий элемент массива функция запишет секунды.

Возвращаемое значение:

DS1307_GET_AM Если время AM

DS1307_GET_PM Если время PM

При 24 часовом формате возвращает всегда 0.

void ds1307_write_data(BYTE data, BYTE day, BYTE month, BYTE year); Данная функция записывает дату.

Принимает аргументы:

BYTE data День недели (1-7).

BYTE day Число (1-31).

BYTE month Месяц (1-12).

BYTE year Год (00-99).

void ds1307_read_data(BYTE *str); Данная функция читает дату.

Принимает аргумент:

BYTE *str Указатель на первый элемент массива. Массив должен иметь 4 элемента.

В первый элемент массива функция запишет день недели.

Во второй элемент массива функция запишет число.

В третий элемент массива функция запишет месяц.

В четвертый элемент массива функция запишет год.

void ds1307_sqw_on(BYTE rs); Данная функция настраивает частоту меандра на выводе SQW/OUT.

Принимает аргумент:

BYTE rs Частота на выводе SQW/OUT

DS1307_SQW_OUT_1 1 Гц

DS1307_SQW_OUT_4 4.096 кГц

DS1307_SQW_OUT_8 8.193 кГц

DS1307_SQW_OUT_32 32.768 кГц

void ds1307_sqw_off(BYTE out); Данная функция задает уровень на выводе SQW/OUT

Принимает аргумент:

BYTE out Выбор уровня на выводе SQW/OUT

DS1307_OUT_HIGHT Высокий (1)

DS1307_OUT_LOW Низкий (0)

Пример:

```
#include <avr/io.h>
#include <stdio.h>
#include "main_init.h"
#include <axlib/lcd.h>
#include <axlib/ds1307.h>

int main(void)
{
	BYTE time[3] = {0};
	BYTE data[4] = {0};
	BYTE string[20] = {0};
	BYTE hour = 0;

	lcd_init(0);

	// Инициализация часов
	ds1307_init();
	// Установка времени
	ds1307_write_time(DS1307_AM, 11 59, 55);
	// Установка даты
	ds1307_write_data(7, 27, 12, 15);
	// Включить вывод SQW/OUT с частотой 1Гц
	ds1307_sqw_on(DS1307_SQW_OUT_1);
	// Выключить вывод SQW/OUT и вывести на выводе 0
	ds1307_sqw_off(DS1307_OUT_LOW);

  while(1)
  {
		// Прочитать время
    hour = ds1307_read_time(time);

		// Вывести время в зависимотсти от AM/PM
		if(hour == DS1307_GET_PM)
		{
			sprintf(string, "pm %02i:%02i:%02i", time[0], time[1], time[2]);
		}
		else
		{
			sprintf(string, "am %02i:%02i:%02i", time[0], time[1], time[2]);
		}

		lcd_gotoxy(0,0);
		lcd_str_out(string);

		// Прочитать дату
```

```
            ds1307_read_data(data);

            // Вывести дату
            sprintf(string, "%i %i.%i.20%02i", data[0], data[1], data[2], data[3]);

            lcd_gotoxy(0,1);
            lcd_str_out(string);
  }
}
```

Функции для работы с часами DS3231

Для начала общения с микросхемой необходимо в файле main_init.h прописать порт и пины для шины i2c.

```
//-----------------------------------------------------------------------
//      Для работы с шиной I2C необходимо выбрать порт и пины на данном порту//
//-----------------------------------------------------------------------

#define I2C_DDR         DDRC        // Порт для выбора направления
#define I2C_PORT         PORTC      // Порт для вывода данных
#define I2C_PIN         PINC        // Порт для ввода данных
#define SCL             0           // Пин порта SCL
#define SDA             1           // Пин порта SDA
```

После настройки аппаратной части шины i2c можно приступать непосредственно к работе с микросхемой. Для подключения функций необходимо подключит файл ds3231.h

Описание функций:

void ds3231_init(void); Данную функцию необходимо вызвать единожды для инициализации часов.

void ds3231_write_time(BYTE h1224, BYTE hours, BYTE minutes, BYTE seconds); Данная функция записывает время в микросхему.
Принимает аргументы:
BYTE h1224 Тип счета часов. DS3231_24, DS3231_AM, DS3231_PM
BYTE hours Часы
BYTE minutes Минуты
BYTE seconds Секунды

BYTE ds3231_read_time(BYTE *str); Данная функция возвращает текущее время вычитанное из микросхемы.
Принимаемый аргумент:
BYTE *str Указатель на первый элемент массива. После вызова данной функции в массиве из трех элементов будут находиться часы, минуты и секунды.
Возвращаемое значение:
Тип счета часов. DS3231_GET_24, DS3231_GET_AM, DS3231_GET_PM

void ds3231_write_data(BYTE data, BYTE day, BYTE month, BYTE year); Данная функция записывает дату в микросхему.

Принимаемые аргументы:
BYTE data День недели
BYTE day Число
BYTE month Месяц
BYTE year Год

void ds3231_read_data(BYTE *str); Данная функция читает текущую дату из микросхемы.
Принимаемый аргумент:
BYTE *str Указатель на первый элемент массива. После вызова данной функции в массиве из четырех элементов будут находится день недели, число, месяц и год.

void ds3231_sqw_on(BYTE rs); Данная функция включает вывод микросхемы INT/SQW на вывод импульсов с заданной частотой.
Принимаемый аргумент:
BYTE rs Требуемая частота.
DS3231_SQW_OUT_1HZ 1Гц
DS3231_SQW_OUT_1KHZ 1 кГц
DS3231_SQW_OUT_4KHZ 4.096 кГц
DS3231_SQW_OUT_8KHZ 8.193 кГц

void ds3231_en32khz(BYTE level); Данная функция включает или выключает вывод микросхемы EN32KHZ
Принимаемый аргумент:
BYTE level Состояние вывода.
DS3231_ON Включить
DS3231_OFF Выключить

SBYTE ds3231_read_temp(void); Данная функция возвращает измеренную температуры.
Возвращаемое значение:
Однобайтовое число со знаком.

void ds3231_set_alarm(BYTE alarm, BYTE h1224, BYTE hours, BYTE minutes); Данная функция настраивает внешнее прерывание на выводе INT/SQW, включает разрешение работы будильника и записывает время в формате часы, минуты.
Принимаемые аргументы:
BYTE alarm Номер будильника. DS3231_ALARM_1, DS3231_ALARM_2
BYTE h1224 Режим счтета часов. DS3231_24, DS3231_AM, DS3231_PM
BYTE hours Часы
BYTE minutes Минуты
Пока будильники взведены, на ножке INT/SQW присутствует лог 1 (Данная ножка обязательно должна быть подтянута резистором к плюсу питания!), как только сработает один из будильников, на данной ножке появится лог 0.

void ds3231_start_alarm(BYTE alarm); Данная функция нужна для старта будильника если его остановили. Использовать ее можно только после выполнения функции **void ds3231_set_alarm(BYTE alarm, BYTE h1224, BYTE hours, BYTE minutes);**. Так же эту функцию не обязательно вызывать после установки будильника.
Принимает аргумент:
BYTE alarm Запустить будильник. DS3231_ALARM_1, DS3231_ALARM_2

void ds3231_stop_alarm(BYTE alarm); Данная функция нужна для приостановки будильника если он работал. Использовать ее можно только после выполнения функции **void ds3231_set_alarm(BYTE alarm, BYTE h1224, BYTE hours, BYTE minutes);**.
Принимает аргумент:

BYTE alarm Приостановить будильник. DS3231_ALARM_1, DS3231_ALARM_2

BYTE ds3231_get_alarm(void); Данная функция возвращает состояние будильников. После ее вызова, если хоть один будильник сработал, то она его взведет снова.
Возвращаемый параметр:
DS3231_ALARM_OFF Ни один из будильников не сработал
DS3231_ALARM_1_ON Сработал первый будильник
DS3231_ALARM_2_ON Сработал второй будильник
DS3231_ALARM_ALL_ON Сработали оба будильника

Пример:

```
#include <avr/io.h>
#include <stdio.h>
#include "main_init.h"
#include <axlib/lcd.h>
#include <axlib/ds3231.h>

int main(void)
{
	BYTE time[3] = {0};
	BYTE data[4] = {0};
	BYTE string[20] = {0};
	SBYTE temperatura = 0;

	lcd_init(0);

	// Инициализация часов
	ds3231_init();
	// Установка время
	ds3231_write_time(DS3231_24, 12, 0, 0);
	// Установка даты
	ds3231_write_data(1, 1, 1, 16);
	// Включение вывода INT/SQW с частотой 1 Гц
	ds3231_sqw_on(DS3231_SQW_OUT_1HZ);
	// Выключение вывода частоты на ножке TN32KHZ
	ds3231_en32khz(DS3231_OFF);
	// Установка первого будильника
	ds3231_set_alarm(DS3231_ALARM_1, DS3231_24, 7, 0);

	while(1)
	{
		// Прочитать время
		ds3231_read_time(time);
		// Вывести время
		sprintf(string, "%02i:%02i:%02i", time[0], time[1], time[2]);

		lcd_gotoxy(0,0);
		lcd_str_out(string);

		// Прочитать дату
		ds3231_read_data(data);
```

```
// Вывести дату
sprintf(string, "%i.%i.20%02i", data[1], data[2], data[3]);

lcd_gotoxy(0,1);
lcd_str_out(string);

// Получит температуру
temperatura = ds3231_read_temp();

// Вывести температуру
sprintf(string, "%3i", temperatura);

lcd_gotoxy(13,0);
lcd_str_out(string);

switch(data[0])
{
        case 1: // ПН
        string[0] = 0xA8;
        string[1] = 0x48;
        string[2] = 0x00;
        break;
        case 2: // ВТ
        string[0] = 0x42;
        string[1] = 0x54;
        string[2] = 0x00;
        break;
        case 3: // СР
        string[0] = 0x43;
        string[1] = 0x50;
        string[2] = 0x00;
        break;
        case 4: // ЧТ
        string[0] = 0xAB;
        string[1] = 0x54;
        string[2] = 0x00;
        break;
        case 5: // ПТ
        string[0] = 0xA8;
        string[1] = 0x54;
        string[2] = 0x00;
        break;
        case 6: // СБ
        string[0] = 0x43;
        string[1] = 0xA0;
        string[2] = 0x00;
        break;
        case 7: // ВС
        string[0] = 0x43;
        string[1] = 0x43;
        string[2] = 0x00;
```

```
                break;
            }

            // Вывести день недели
            lcd_gotoxy(14,1);
            lcd_str_out(string);
        }
}
```

Функции для работы со сдвиговым регистром 74HC595

Для начала работы с микросхемой необходимо в файле main_init.h прописать порт и пины для связи. Так же при использовании данного сдвигового регистра, им можно управлять ЖК-дисплеем с максимальным разрешением 40 символов и 4 строки. Для этого необходимо активировать изменения, в функциях ЖК-дисплея сняв комментарий со строчки #define R74HC595_LCD.

```
//-----------------------------------------------------------------------
//  Параметры необходимые для работы со сдвиговым регистром 74HC595
//
//  Если требуется подключить ЖК дисплей до 40 символов и 4 строк
//      в 4-х битном режиме через данный регистр, то необходимо подключить
//      дисплей к регистру по данной схеме:
//
//      74HC595 |   LCD
//      ------------------
//      |  Q2   |       RS   |
//  ------------------
//      |  Q3   |       E    |
//  ------------------
//      |  Q4   |       D4   |
//  ------------------
//      |  Q5   |       D5   |
//  ------------------
//      |  Q6   |       D6   |
//  ------------------
//      |  Q7   |       D7   |
//  ------------------
//
//  Ножку R/W на дисплее нужно завести на GND
//
//  Для активации режима работы с диисплеем, необходимо
//  раскомментировать ниже записанный #define R74HC595_LCD
//-----------------------------------------------------------------------
// #define R74HC595_LCD // Раскомментировать для включения поддержки ЖК дисплея
//-----------------------------------------------------------------------

#define R74HC595_PORT_RESET         PORTC  // Порт на который подключается RESET
#define R74HC595_PORT_CLK           PORTD  // Порт на который подключается CLK
#define R74HC595_PORT_SHIFT         PORTB  // Порт на который подключается SHIFT
#define R74HC595_PORT_DATA          PORTD  // Порт на который подключается DATA
```

```
#define R74HC595_DDR_RESET        DDRC   // Порт на который подключается RESET
#define R74HC595_DDR_CLK          DDRD   // Порт на который подключается CLK
#define R74HC595_DDR_SHIFT        DDRB   // Порт на который подключается SHIFT
#define R74HC595_DDR_DATA         DDRD   // Порт на который подключается DATA

#define R74HC595_RESET            0      // Разряд сброса регистра
#define R74HC595_CLK              0      // Разряд стробирования
#define R74HC595_SHIFT            5      // Разряд защелки
#define R74HC595_DATA             1      // Разряд данных
```

После настройки аппаратной части можно приступать непосредственно к работе с микросхемой. Для подключения функций необходимо подключит файл 74hc595.h Если микросхема используется для работы с ЖК дисплеем, то данный файл подключать не надо. Нужно лишь настроить порты, пины и раскомментировать строку #define R74HC595_LCD.

Описание функций:

void reg_74hc595_init(void); Данную функцию необходимо вызвать один раз для настройки портов и инициализации микросхемы. После вызова данной функции можно работать с микросхемой.

void reg_74hc595_reset(void); Данная функция сбрасывает оба регистра микросхемы. В памяти будет записано 0x00 и на выходе будет тоже 0x00.

void reg_74hc595_byte(BYTE data); Данная функция записывает в регистр байт и выводит его. В качестве аргумента принимает выводимый байт.

Пример:

```
#include <avr/io.h>
#include "main_init.h"
#include <axlib/74hc595.h>

int main(void)
{
        reg_74hc595_init();             // Инициализация микросхемы
        reg_74hc595_reset();            // Очистка регистра

  while(1)
  {
        reg_74hc595_byte(0x3B);         // Вывод числа 0x3B через регистр
  }
}
```

Функции для чтения/записи микросхемы EEPROM серии AT24Cx

Для начала работы с микросхемой памяти необходимо в файле main_init.h прописать порт и пины для шины i2c, так как по этому интерфейсу общается микросхема с МК.

```
//-----------------------------------------------------------------------
//        Для работы с шиной I2C необходимо выбрать порт и пины на данном порту//
//-----------------------------------------------------------------------

#define I2C_DDR         DDRC        // Порт для выбора направления
#define I2C_PORT        PORTC       // Порт для вывода данных
#define I2C_PIN         PINC        // Порт для ввода данных
#define SCL             0           // Пин порта SCL
#define SDA             1           // Пин порта SDA
```

После настройки аппаратной части шины i2c можно приступать непосредственно к работе с микросхемой. Для подключения функций необходимо подключит файл at24c.h

Описание функций:

BYTE at24c_init(BYTE add); Данная функция производит инициализацию микросхемы памяти. В качестве аргумента функция принимает адрес микросхемы в диапазоне от 0x50 до 0x57. После вызова функция вернет ACK при удачной инициализации, либо NACK при неудачной.

BYTE at24c_write_byte(UBYTE add, UWORD addbyte, BYTE data); Данная функция записывает байт в микросхему памяти.
Принимаемые аргументы:
UBYTE add Адрес микросхемы от 0x50 до 0x57.
UWORD addbyte Адрес байта для записи в микросхему
BYTE data Байт который необходимо записать в память.
Функция возвращает ACK при удачной записи и NACK при неудачной.

BYTE at24c_write_page(UBYTE add, UWORD addpage, BYTE *data, UWORD count); Данная функция записывает страницу байт в микросхему памяти. После записи страницы байта, функция вернет ACK при удачной записи, либо NACK при неудачной.
Принимаемые аргументы:
UBYTE add Адрес микросхемы от 0x50 до 0x57.
UWORD addpage Номер страницы для записи в микросхему от 0 до 255
BYTE *data Указатель на массив данных для записи. Массив желательно передавать равным длине страницы.
UBYTE count Количество байт в странице. Зависит от типа микросхемы.
Функция возвращает ACK при удачной записи и NACK при неудачной.

BYTE at24c_write_str(UBYTE add, UWORD addbyte, BYTE *data, UBYTE count); Данная функция записывает массив байт в микросхему памяти.
Принимаемые аргументы:
UBYTE add Адрес микросхемы от 0x50 до 0x57.
UWORD addbyte Адрес начального байта для записи в микросхемы
BYTE *data Указатель на массив данных для записи.
UBYTE count Количество передаваемых байт. Значение должно быть больше нуля.
Функция возвращает ACK при удачной записи и NACK при неудачной.

BYTE at24c_read_byte(UBYTE add, UWORD addbyte); Данная функция читает один байт из микросхемы памяти.
Принимаемые аргументы:
UBYTE add Адрес микросхемы от 0x50 до 0x57.

UWORD addbyte Адрес байта для чтения из микросхемы
Функция возвращает значение байта при удачном чтении и NACK при неудачном.

BYTE at24c_read_str(UBYTE add, UWORD addbyte, BYTE *data, UBYTE count); Данная функция читает заданное количество байт из микросхемы в массив.
Принимаемые аргументы:
UBYTE add Адрес микросхемы от 0x50 до 0x57.
UWORD addbyte Адрес начального байта для чтения из микросхему
BYTE *data Указатель на массив, в который нужно записать данные из микросхемы.
UBYTE count Количество байт для чтения. Значение должно быть больше нуля.
Функция возвращает ACK при удачном чтении и NACK при неудачном.

Пример работы с микросхемой

```
#define EEPROM_ADD  0x56 // Адрес микросхемы
#include <avr/io.h>
#include "main_init.h"
#include <axlib/at24c.h>

int main(void)
{
    // Массив для записи
        BYTE data[] = {0x31, 0x32, 0x33, 0x34, 0x35, 0x36, 0x37, 0x38, 0x39, 0x00};
    // Массив для чтения
        BYTE read[10] = {0};
        // Инициализация микросхемы
        at24c_init(EEPROM_ADD);
        // Запись массива в микросхему с 0 ячейки 10 байт
        at24c_write(EEPROM_ADD, 0, data, 10);
        // Чтение 5 байт с 0 ячейки в массив из микросхемы
        at24c_read(EEPROM_ADD, 0, read, 5);
}
```

Функции для управления микросхемой MAX7219

Для начала работы с микросхемой необходимо в файле main_init.h прописать порты и пины.

```
//-----------------------------------------------------------------------
//      Настройка портов и пинов для управления микросхемой MAX7219
//-----------------------------------------------------------------------

// Выводы для управление микросхемой MAX7219
#define MAX7219_DIN_DDR     DDRC
#define MAX7219_CS_DDR      DDRC
#define MAX7219_CLK_DDR     DDRC

#define MAX7219_DIN_PORT    PORTC
#define MAX7219_CS_PORT     PORTC
#define MAX7219_CLK_PORT    PORTC
```

```
#define MAX7219_DIN_PIN      0
#define MAX7219_CS_PIN       1
#define MAX7219_CLK_PIN      2
```

После настройки аппаратной части можно приступать непосредственно к работе с микросхемой. Для подключения функций необходимо подключит файл max7219.h.

Описание функций:

void max7219_init(BYTE test, BYTE decode, BYTE diglimit); Данную функцию необходимо вызвать один раз для настройки портов и инициализации микросхемы. После вызова данной функции можно работать с микросхемой. Принимаемые аргументы функцией:

BYTE test - Включает/Выключает тест режим при инициализации
- MAX7219_TEST_ON - Включить тест
- MAX7219_TEST_OFF - Выключить тест
- Режим тест включает все сегменты на 1 секунду, а потом их гасит.

BYTE decode - Кодировать/не кодировать символы
- MAX7219_DECODE - Кодировать
- MAX7219_UNDECODE - Не кодировать

BYTE diglimit - Количество знаков от 1 до 8

void max7219_data_out(BYTE add, BYTE dig, BYTE point); Данная функция выводит заданный символ по заданному разряду. Если включен режим кодирования, то функция так же получает состояние точки знака. Если режим кодирования отключен, то состояние точки нужно отключить и можно управлять любым сегментом. Принимаемые аргументы функцией:

BYTE add - Адрес разряда от 1 до 8

BYTE dig - Цифра от 0 до 15

BYTE point - Включает/Выключает точку текущего разряда
- MAX7219_POINT_ON - Включить точку
- MAX7219_POINT_OFF - Выключить точку
- MAX7219_POINT_NO - Не обрабатывать разряд D7

void max7219_clear(); Данная функция очищает дисплей.

void max7219_light(BYTE light); Данная функция задает яркость свечения сегментов в диапазоне от 0 до 15. 0 самое тусклое свечение, 15 самое яркое. В качестве аргумента принимает значение свечения от 0 до 15.

Пример:

```
#include <avr/io.h>
#include "main_init.h"
#include <axlib/max7219.h>

int main(void)
{
	// Инициализация MAX7219
	max7219_init(MAX7219_TEST_ON, MAX7219_DECODE, 8);
	// Настройка ¤ркости сегментов
	max7219_light(5);
	// Очистка дисплея
	max7219_clear();
```

```
        // Вывод цифры 5 в первый разряд без дробной точки
        max7219_data_out(1, 5, MAX7219_POINT_OFF);

  while(1)
  {

  }
}
```

Функции для реализации протокола MH-BUS

Эта часть библиотеки предназначена для передачи данных между несколькими устройствами с одним ведущим (Мастер) и несколькими (От 1 до 50) ведущими (Слейв). Функции, включенные в данный раздел берут на себя обязанности формирования пакетов, проверки целостности и вспомогательный характер. Перед началом работы с протоколом необходимо произвести небольшие настройки в файле main_init.h

---

```
//-----------------------------------------------------------------------
//  Протокол MH-BUS для передачи данных между устройствами
//        Для работы с шиной RS-485 необходимо выбрать порт и пин на данном порту
//        для управления передачей и скорость в БОД.
//
//  Также необходимо задать уникальный ID на шине в диапазоне от 1 до 65535
//-----------------------------------------------------------------------

// Раскомментируйте данную строку, если требуется автоматическое  управление потоком данных драйвера RS-485
// #define MH_BUS_RS485_ON                          // Включить поддержку RS-485

// Раскоментируйта данную строку, если необходимы функции режима Мастера
//#define MH_BUS_MASTER                                    // Включить в режиме Мастер

// Раскоментируйта данною строку если необходимы функции режима Слейва
//#define MH_BUS_SLAVE                                     // Включить в режиме Слейв

// Текущие данные необходимы только в режиме Слейв
#if defined(MH_BUS_SLAVE)
        #define MH_BUS_SLAVE_ID      65530U// Идентификационный номер устройства
#endif

#define MH_BUS_USART_BOD    38400 // Скорость обмена

// Текущие данные необходимы только при использовании автоматическим
// управлением потоком дпнных драйвера RS-485
#if defined(MH_BUS_RS485_ON)
        #define RS_485_DDR    DDRC    // Порт на котором будет выбран пин упраления
        #define RS_485_PORT   PORTC  // Порт на котором будет выбран пин упраления
        #define RS_485_PIN    0        // Пин управления передачей данных
#endif

#define MH_BUS_RS485_ON  //Необходимо раскомментировать если требуется использовать микросхему драйвер RS-
// 485. Данная строка включает вспомогательные макросы в функциях отправки и чтения пакетов. Настройка ножки
// вывода для управления драйвером осуществляется в трех строках ниже.
```

```
#define RS_485_DDR    DDRC
#define RS_485_PORT   PORTC
#define RS_485_PIN    0

// Первая и вторая строка указывают на порт, а последняя на разряд этого порта. К этой ножке нужно подключить
// выводы микросхемы драйвера RS-485 для управления направлением данных.

// Раскомментровав данную строку, устройство будет работать в режиме Мастер. Все  функции, относящиеся к
// режиму Слейв, будут не доступны. (ВАЖНО!!! Нельзя одновременно использовать Мастер и  Слейв режим. Это
// приведет в возникновению большого количества ошибок.)
#define MH_BUS_MASTER

// Раскомментровав данную строку, устройство будет работать в режиме Слейв. Все функции относящиеся к режиму
// Мастер будут не доступны. (ВАЖНО!!! Нельзя одновременно использовать Мастер и Слейв режим. Это приведет в
// возникновению большого количества ошибок.)
#define MH_BUS_ SLAVE

#define MH_BUS_SLAVE_ID      65530U // Данная строка задает ID Слейв устройства. В режиме Мастер, данные этой
//строки игнорируются.

#define MH_BUS_USART_BOD 38400  // В данной строке задается скорость в бодах.
```

Функции для работы с протоколом MHBUS

Функции для режима Мастер

BYTE mhbus_read(WORD timeout); Функция проверяет наличие полученного пакета.
Принимаемые аргументы:
WORD timeout Таймаут в миллисекундах.
Возвращаемые значения:
FALSE - данные не пришли.
MH_BUS_OK - Пакет получен. Целостность пакета не нарушена. Команда в разрешенном диапазоне.
MH_BUS_START_FAIL - Получен пакет, но при проверке обнаружен битый СТАРТ-БАЙТ.
MH_BUS_STOP_FAIL - Получен пакет, но при проверке обнаружен битый СТОП-БАЙТ.
MH_BUS_CON_SUM_FAIL - Получен пакет, но при проверке обнаружено несовпадение контрольной суммы.
MH_BUS_COM_FAIL - Получен пакет, но при проверке обнаружено значение команды вне разрешенного диапазона.

void mhbus_write(BYTE add, BYTE com, BYTE reg, BYTE *str); Функция передачи пакета Слейву.
Принимаемые аргументы:
BYTE add - Адрес Слейва которому адресуется пакет.
BYTE com - Команда адресуемая Слейву
BYTE reg - Регистр адресуемый Слейву
BYTE *str - Указатель на начало массива с четырьмя байтами для передачи Слейву

BYTE mhbus_id(UWORD id) ; Функция производит проверку на совпадения ID устройства после запроса Мастером на изменение адреса Слейва.
Возвращаемые значения:
TRUE - ID адрес, переданный в качестве аргумента совпадает с ID переданным в ответном пакете Слейвом.
FALSE - ID адрес, переданный в качестве аргумента не совпадает с ID переданным в ответном пакете Слейвом.

Функции для режима Слейв

BYTE mhbus_read(BYTE test_add, BYTE timeout); Функция проверяет наличие полученного пакета.
Принимаемые аргументы:
WORD timeout Таймаут в миллисекундах.
BYTE test_add Адрес слейв устройства, которому адресован пакет.
Возвращаемые значения:
FALSE - данные не пришли.
MH_BUS_OK - Пакет получен. Целостность пакета не нарушена. Команда в разрешенном диапазоне.
MH_BUS_START_FAIL - Получен пакет, но при проверке обнаружен битый СТАРТ-БАЙТ.
MH_BUS_STOP_FAIL - Получен пакет, но при проверке обнаружен битый СТОП-БАЙТ.
MH_BUS_CON_SUM_FAIL - Получен пакет, но при проверке обнаружено несовпадение контрольной суммы.
MH_BUS_COM_FAIL - Получен пакет, но при проверке обнаружено значение команды вне разрешенного диапазона.

void mhbus_write(BYTE com, BYTE reg, BYTE *str); Функция передачи пакета Мастеру. Адрес слейва автоматическа подключается к пакету из EEPROM памяти Слейва.
Передаваемые аргументы:
BYTE com - Команда запрошенная Мастером.
BYTE reg - Либо регистр, если Слейв выполнил условия Мастера, либо код ошибки по причине, которой Слейв не выполнил условие Мастера.
BYTE *str - Либо данные по запросу Мастера, либо набор хаотичных байт, если Мастеру данные не нужны, либо описание ошибки не входящие в состав стандартных ошибок.

BYTE mhbus_wr_add(BYTE add); Функция установки адреса Слейва. Запись адреса производится в EEPROM. Адрес сохраняется даже при снятии питания с МК.
Передаваемый аргумент:
BYTE add - Адрес для записи.

BYTE mhbus_rd_add(void); Функция чтения адреса Слейв из EEPROM.
Возвращаемое значение:
Текущий адрес Слейва.

BYTE mhbus_id(void); Функция расчета первых двух байт и проверки на совпадение с ID слейва.
Возвращаемое значение:
TRUE - Если ID совпал.
FALSE - Если ID не совпал.

Статья с примером работы слейв устройства.