МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ **«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. В. Г. ШУХОВА»**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

ЛАБОРАТОРНАЯ РАБОТА №3
Дисциплина: ЭВМ и периферийные устройства
Тема: Изучение принципов организации обмена данными по последовательному интерфейсу I2C на примере управления блоком светодиодов и программного опроса клавиатуры

Выполнил: ст. группы ВТ-31
Подкопаев Антон Валерьевич
Проверил: доцент кафедры ПО и ВТАС
Шамраев Анатолий Анатольевич

Белгород 2020

Цель работы: изучить принципы программного управления двунаправленным обменом данных по последовательному интерфейсу I2C.

Указания по организации самостоятельной работы

Перед работой необходимо проработать теоретический материал по литературе и конспекту лекций, ознакомиться с основными возможностями и принципами функционирования последовательного интерфейса I2C, принципами программного управления двунаправленным обменом данных по последовательному интерфейсу I2C.

Порядок проведения работы и указания по ее выполнению

Перед началом выполнения практической части лабораторной работы проводится экспресс-контроль знаний по принципам функционирования модулей USART в режиме I2C, входящих в состав микроконтроллера MSP430, а также по протоколу обмена данными по интерфейсу I2C. При подготовке к лабораторной работе необходимо составить предварительный вариант листинга программы, в соответствие с индивидуальным заданием.

Задание. Разработать в среде программирования IAR Embedded Workbench программу на языке С, которая выполняет опрос клавиатуры лабораторного стенда и выводит информацию о нажатых клавишах с помощью блока светодиодов.

Вариант 3 (13)

Разработать программу, фиксирующую нажатия клавиш 9, 0 и 1 матричной клавиатуры включением светодиодов 1, 2 и 3 соответственно. Выход из цикла опроса осуществляется при нажатии клавиши #. Частота тактовых импульсов на линии SCL – 100 кГц.

Порядок выполнения задания:

– включить лабораторный макет.

– запустить компилятор IAR Embedded Workbench

– создать пустой проект.

– создать файл ресурса для кода программы и подключить его к проекту.

– ввести код исходного модуля программы обмена данными между микроконтроллером MSP430F1611 с регистрами PCA9538 по интерфейсу I2C соответствие с индивидуальным заданием, приведенным в таблице 4.3.8.

– выполнить компиляцию исходного модуля программы и устранить ошибки, полученные на данном этапе.

– настроить параметры программатора.

– создать загрузочный модуль программы и выполнить программирование микроконтроллера.

Теоретические сведения

Функционирование модуля I2C

Модуль I2C поддерживает любые ведущие и ведомые устройства, совместимые с I2C. На рисунке 4.3.2 показан пример шины I2C. Каждое устройство обладает уникальным адресом и может работать и как передатчик и как приемник. Устройство, подключенное к шине I2C, во время передачи данных может рассматриваться как ведущее или ведомое. Ведущий инициирует передачу данных и генерирует тактовый сигнал SCL. Любое устройство, адресованное ведущим, рассматривается как ведомое.

Условия «СТАРТ» и «СТОП» I2C

Условия «СТАРТ» и «СТОП», показанные на рисунке 4.3.5, генерируются ведущим. Условие «СТАРТ» возникает при переходе с высокого уровня на низкий на линии SDA, когда SCL имеет высокий уровень. Условие «СТОП» появляется при переходе с низкого уровня на высокий на линии SDA при высоком уровне на SCL. Бит занятости I2CBB устанавливается после условия «СТАРТ» и сбрасывается после «СТОП».

Режимы адресации I2C

Модуль I2C поддерживает 7-разрядный и 10-разрядный режимы адресации.

7-разрядная адресация

В 7-разрядном формате адресации (рисунок 4.3.6), первый байт – это 7разрядный адрес ведомого и бит R/W. Бит ACK посылается приемником после каждого байта

10-разрядная адресация

В 10-разрядном адресном формате (рисунок 4.3.7), первый байт содержит 11110b плюс два старших бита 10-разрядного адреса ведомого и бит R/W. Бит ACK посылается приемником после каждого байта. Следующий байт содержит оставшиеся 8 бит 10-разрядного адреса ведомого, завершающиеся битом ACK, за которыми следуют байты данных и бит ACK.

Режимы работы модуля I2C

Модуль I2C работает в режимах «ведущий передатчик», «ведущий приемник», «ведомый передатчик» или «ведомый приемник».

Режим ведущего

В режиме ведущего выполнение передачи и приема управляется с помощью битов I2CRM, I2CSTT и I2CSTP, как описано в таблице 4.3.1. Режим ведущего приемника вводится установкой I2CTRX=0 после передачи адресного байта ведомого и установленного бита R/W. SCL удерживается в низком состоянии, когда необходимо вмешательство ЦПУ после передачи байта.

Режим ведомого

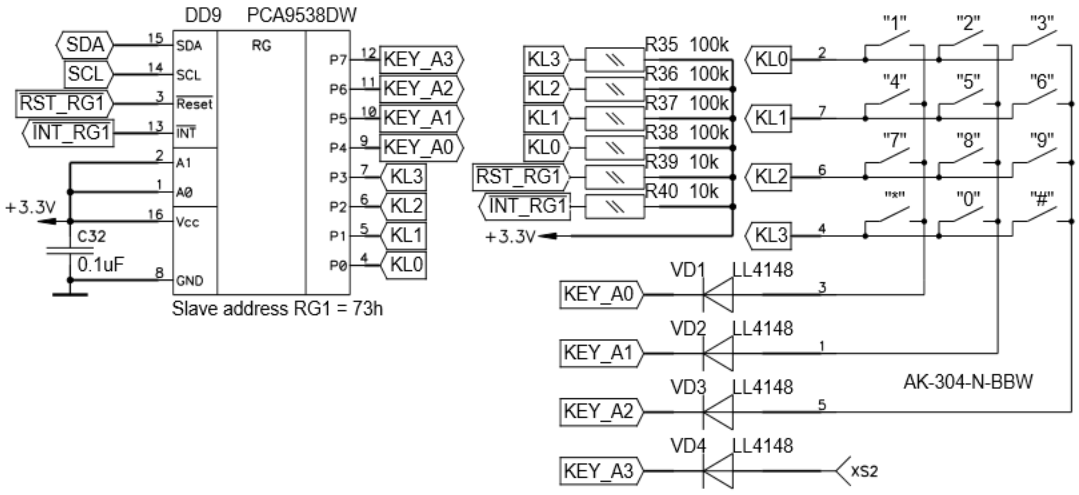
В режиме ведомого операции передачи и приема управляются автоматически модулем I2C. В режиме ведомого приемника биты данных принимаются на SDA и сдвигаются по тактовым импульсам, генерируемым ведущим устройством. Ведомое устройство не генерирует тактовый сигнал, но может удерживать линию SCL в состоянии низкого уровня, если после приема байта необходимо вмешательство ЦПУ. В режим ведомого передатчика можно войти только из режима ведомого приемника. Вход в режим ведомого передатчика происходит, если байт адреса ведомого, переданный ведущим, является таким же адресом, как и его собственный и был послан установленный бит R/W, указывая на запрос отправки данных ведущему. Ведомый передатчик сдвигает последовательные данные из устройства на SDA по импульсам тактирования, генерируемым ведущим устройством. Ведомое устройство не генерирует тактовых сигналов, но может удерживать линию SCL в состоянии низкого уровня, если после передачи байта необходимо вмешательство ЦПУ.

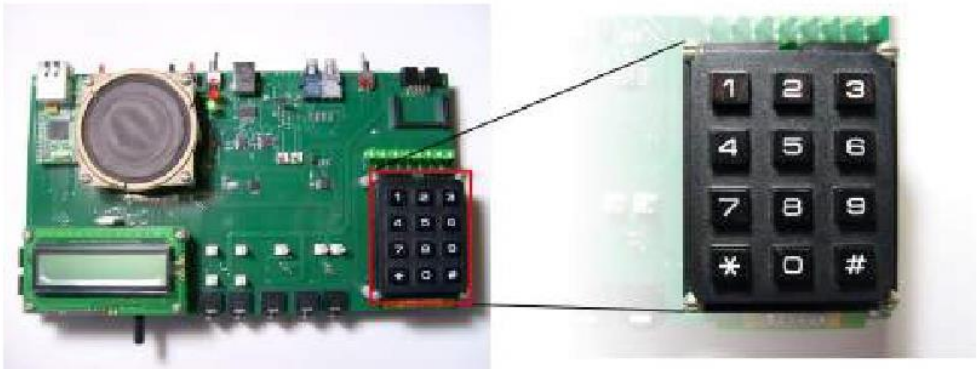
Подключение клавиатуры

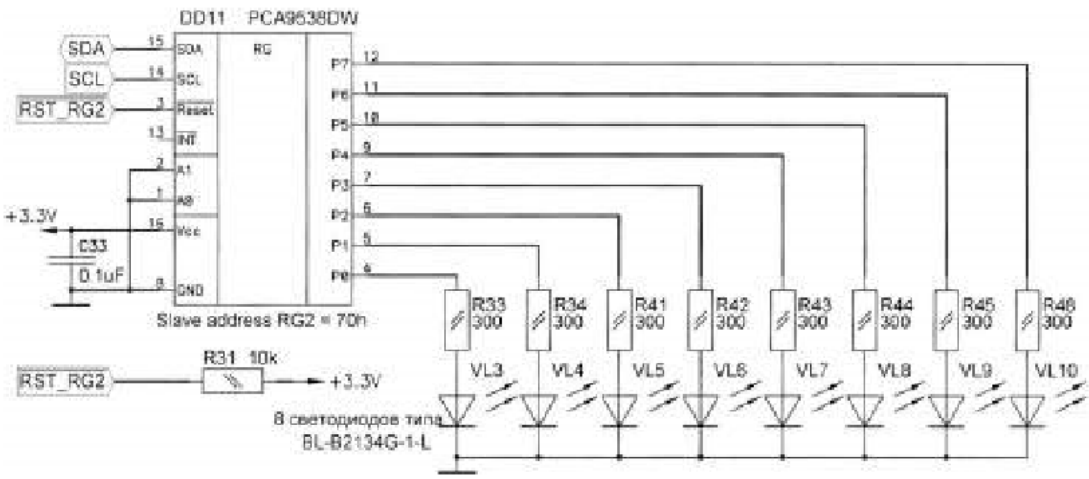
Для обеспечения возможности ручного управления микроконтроллерным стендом в его состав включена клавиатура 3x4 (AK-304-N-BBW), которая подключена через регистр PCA9538 к шине I2C. PCA9538 – это 8-разрядный регистр ввода/вывода для последовательной двухпроводной шины данных I2C, с рабочим диапазоном напряжением питания от 2,3В до 5,5В. Он обеспечивает общие функции ввода/вывода для большинства типов микроконтроллеров через интерфейс I2C (сигнал тактирования SCL и сигнал данных SDA). Адрес регистра на шине I2C определяется пятью фиксированными старшими разрядами – 11100, и двумя адресными линиями A0 и A1, которые для управляющего клавиатурой регистра подключены к шине питания, поэтому адрес регистра на шине I2C – 73h (1110011). Формат обмена данными между ЦП и последовательным регистром по протоколу I2C изображен на рисунке 4.3.11. Условия «СТАРТ» и «СТОП» генерируются ведущим устройством (ЦП). Под условием «СТАРТ» понимается переход с высокого уровня линии SDA на низкий при высоком уровне на SCL. Первый после условия «СТАРТ» байт состоит из 7 разрядного адреса устройства (73h) и бита W R / . Когда W R / =0, ЦП передает данные регистру, а когда W R / =1, то ЦП принимает данные от регистра. Бит ACK посылается приемником после каждого байта на 9-ом такте SCL. Два младших разряда второго байта (B1, B0) указывают номер внутреннего регистра датчика, к которому идет обращение. Во третьем байте передаются непосредственно данные. В конце обмена данными ЦП генерирует условие «СТОП» – переход с низкого уровня на линии SDA на высокий при высоком уровне на SCL.

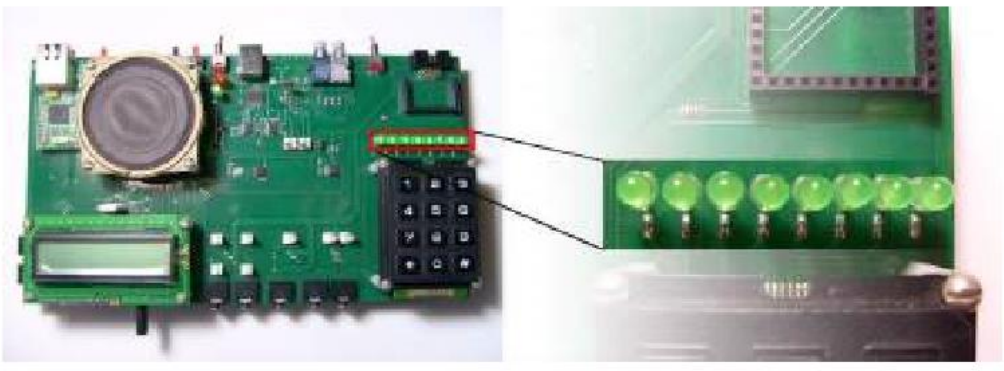
Подключение светодиодного индикатора

Индикатор из 8 светодиодов подключен через последовательный регистр PCA9538 к шине I2C. Адресные линии регистра PCA9538, управляющего светодиодами, A0 и A1 подключены к общей шине, поэтому адрес регистра при обращении к нему по шине I2C – 70h.


DD9 PCA9538DW
SDA
SCL
RST_RG1
INT_RG1
+3.3V
C32
0.1uF
RG
P7 KEY_A3
P6 KEY_A2
P5 KEY_A1
P4 KEY_A0
P3 KL3
P2 KL2
P1 KL1
P0 KL0
Slave address RG1 = 73h
R35 100k
R36 100k
R37 100k
R38 100k
R39 10k
R40 10k
"1" "2" "3"
"4" "5" "6"
"7" "8" "9"
"*" "0" "#"
VD1 LL4148
VD2 LL4148
VD3 LL4148
VD4 LL4148
AK-304-N-BBW
XS2


1 2 3
4 5 6
7 8 9
* 0 #


DD11 PCA9538DW
SDA
SCL
RST_RG2
+3.3V
C33
0.1uF
RG
Slave address RG2 = 70h
R31 10k
R33 300
R34 300
R41 300
R42 300
R43 300
R44 300
R45 300
R46 300
VL3 VL4 VL5 VL6 VL7 VL8 VL9 VL10
8 светодиодов типа
BL-B2134G-1-L



Main.c

```
#include <msp430.h>
#include "system_define.h"
#include "system_variable.h"
#include "function_prototype.h"
#include "main.h"

void main(void)
{
    _enable_interrupt();
    WDTCTL = WDTPW + WDTHOLD;
    Init_System_Clock();
    Init_System();
    Init_I2C();
    int f = 0;
    char k_stop = '0', KEYS_last;
    while(f!=1)
    {
        KEYS_last = KEYS_scannow();
        if ((KEYS_last == '*') && (k_stop == '#')) f = 1;
        if ((KEYS_last == '#' )&& (k_stop == '*') )f = 1;
        k_stop = KEYS_last;
        if (KEYS_last == '9') LED_set(1);
        if (KEYS_last == '0') LED_set(2);
        if (KEYS_last == '1') LED_set(3);
        wait_1ms(50);
        LED_reset(1);
        LED_reset(2);
        LED_reset(3);
    }
    LED_reset(1);
    LED_reset(2);
    LED_reset(3);
}
```

i2c.c

```
/*****************************************************************************/
#include "function_prototype.h"
#include "system_define.h"
#include "I2C.h"
/*****************************************************************************/

//===========================================================================
// Инициализация модуля UART0 для работы в режиме I2C
void Init_I2C()
{
  P3SEL |= 0x0A;            // Выбор альтернативной функции для линий порта P3
                            // в режиме I2C SDA->P3.1, SCL->P3.3
  U0CTL |= I2C + SYNC;      // Выбрать режим I2C для USART0
  U0CTL &= ~I2CEN;          // Выключить модуль I2C
// Конфигурация модуля I2C
  I2CTCTL=I2CSSEL_2;       // SMCLK
  I2CSCLH = 0x26;          // High period of SCL
  I2CSCLL = 0x26;          // Low period of SCL

  U0CTL |= I2CEN;          // Включить модуль I2C

  // формирование строба сброса I2C-регистров PCA9538 - RST_RG1->P3.1 и RST_RG2->P3.2
  P3DIR |= 0x05;             // переключаем эти ножки порта на вывод,
  P3SEL &= ~0x05;            //   выбираем функцию ввода-вывода для них
  P3OUT &= ~0x05;            //   и формируем строб сброса на 1 мс
  wait_1ms(1);
  P3OUT |= 0x05;
}
//===========================================================================

//===========================================================================
// отправка данных по протоколу I2C
void Send_I2C(unsigned char* buffer,unsigned int num, unsigned char address)
{
   while (I2CBUSY & I2CDCTL);            // проверка готовности модуля I2C
   BufTptr=buffer;
   I2CSA = address;                      // установка адреса приемнмка
   I2CNDAT =num;                         // количество передаваемых байт
   I2CIE = TXRDYIE+ALIE;                 // разрешение прерываний по окончанию передачи байта и
по потере арбитража
   U0CTL |= MST;                         // режим Master
   I2CTCTL |= I2CSTT + I2CSTP + I2CTRX;  // инициализировать передачу

   while ((I2CTCTL & I2CSTP) == 0x02);   // ожидание условия СТОП
}
//===========================================================================

//===========================================================================
// прием данных по протоколу I2C
void Receive_I2C(unsigned char* buffer,unsigned int num, unsigned char address)
{
   while (I2CBUSY & I2CDCTL);            // проверка готовности модуля I2C
   BufRptr=buffer;
   I2CSA=address;
   I2CTCTL&=~I2CTRX;                     // режим приема
   I2CNDAT=num;
   I2CIE=RXRDYIE;                        // резрешение прерывания по окончанию приема байта
   U0CTL |= MST;
   I2CTCTL |= I2CSTT + I2CSTP;           // инициализировать прием

   while ((I2CTCTL & I2CSTP) == 0x02);   // ожидание условия СТОП
}
```

```
//=============================================================================

//=============================================================================
// отправка байта устройству на шине I2C
void I2C_SendByte(char data, char i2c_addr)
{
  Tx_Data[0] = data;                   // отправляемый байт
  Send_I2C(&Tx_Data[0], 1, i2c_addr); // вывод по I2C на устройство
}
//=============================================================================


//=============================================================================
// запись байта в регистр устройства на шине I2C
void I2C_WriteByte(char reg, char data, char i2c_addr)
{
  Tx_Data[0] = reg;                    // выбираем регистр
  Tx_Data[1] = data;                   // записываемые данные
  Send_I2C(&Tx_Data[0], 2, i2c_addr);  // вывод по I2C на устройство
}
//=============================================================================


//=============================================================================
// чтение байта из регистра устройства на шине I2C
byte I2C_ReadByte(char reg, char i2c_addr)
{
  Tx_Data[0] = reg;                        // выбираем регистр
  Send_I2C(&Tx_Data[0], 1, i2c_addr);
  Receive_I2C(&Rx_Data[0], 1, i2c_addr);  // получаем значение из регистра
  return Rx_Data[0];
}
//=============================================================================


//=============================================================================
// чтение слова (2 байта) из регистра устройства на шине I2C
int I2C_ReadWord(char reg, char i2c_addr)
{
  Tx_Data[0] = reg;                        // выбираем регистр
  Send_I2C(&Tx_Data[0], 1, i2c_addr);
  Receive_I2C(&Rx_Data[0], 2, i2c_addr);  // получаем 2 байта значение из регистра
  return Rx_Data[0] + (Rx_Data[1] * 256);
}
//=============================================================================



//=============================================================================
//Обработка прерывания от модуля USART0, работающего в режиме I2C

// вектор прерываний для модуля I2C
#pragma vector=USART0TX_VECTOR
__interrupt void I2C_ISR()
{
    switch(I2CIV)
     {
       case  0: break;                          // нет прерывания
       case  2: break;                          // потеря арбитража
       case  4: break;                          // нет подтверждения
       case  6: break;                          // прерывание собственного адреса
       case  8: break;                          // регистр доступен для чтения
       case 10:                                 // окончание приема байта
        *BufRptr++=I2CDRB;
            break;
```

```
       case 12:                                  // окончание передачи байта
        I2CDRB=*BufTptr++;
            break;
       case 14: break;                           // общий вызов
       case 16: break;                           // обнаружено условие СТАРТ
     default : break;
     }
}//=============================================================================


//-----------------------------------------------------------------------------
```

keys.c

```
#include "function_prototype.h"
#include "sysfunc.h"
#include "keys.h"

byte keycol, keyline, KEYS_last=0;
char table_keys[12] = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '*', '0', '#'};

// Проверка нажатия клавиши в текущий момент, результат:
//  0 - клавиша не нажата
//  ASCII-код клавиши
char KEYS_scannow()
{
  keyline=0;
  // выбираем регистр конфигурации направления (0x03)
  //  и конфигурируем P4-P7 на вывод - для строба столбцов,
  //  а P0-P3 на ввод - для опроса строк (1-ввод, 0-вывод)
  I2C_WriteByte(0x03, 0x0F, KEYS_i2c_addr);

  for (keycol=0; keycol<3; keycol++) {
  // последовательно подаем сигнал низкого уровня на столбцы (P4-P7)
    I2C_WriteByte(0x01, ~(1<<keycol<<4) & 0xf0, KEYS_i2c_addr);
    wait_1ms(1);
    // и опрашиваем строки (P0-P3) на наличие нуля
    keyline = ~(I2C_ReadByte(0x00, KEYS_i2c_addr)) & 0x0f;
    if (keyline) break;
  }
  if (!keyline) return 0;       // если не была нажата никакая клавиша - возвращаем 0
  if (keyline == 4) keyline = 3;  // переводим номера разрядов в номер строки
  if (keyline == 8) keyline = 4;
  KEYS_last = table_keys[--keyline*3+keycol]; // получаем код нажатой клавиши из таблицы
  return KEYS_last;
}


// Возвращает код последней нажатой клавиши, результат:
//  0 - не нажималась никакая клавиша
//  ASCII-код клавиши
char KEYS_lastkey()
{
  KEYS_scannow();
  return KEYS_last;
}


// Очистка последней нажатой клавиши
void KEYS_clear()
{
  KEYS_last = 0;
  wait_1ms(200);
}
```

```
// Ожидание нажатия клавиши, результат - ASCII-код нажатой клавиши
char KEYS_waitkey()
{
  KEYS_clear();           // очистка последней нажатой клавиши
  while (!KEYS_scannow()) // пока не нажата никакая клавиша,
    wait_1ms(1);          //    сделать паузу
  return KEYS_last;       // вернуть код нажатой клавиши
}


// пауза с циклическим опросом клавиатуры, прерывается если нажата клавиша
void KEYS_pause(byte cnt)
{
  byte i;
  KEYS_clear();           // очистка последней нажатой клавиши
  for (i=0; i<cnt; i++)
    if (KEYS_scannow())
      break;
}
```

leds.c

```
// LED-indicator functions

#include "function_prototype.h"
#include "sysfunc.h"
#include "leds.h"

char LED_config=0;                  // хранится конфигурация светодиодов (вкл/выкл)

void LED_out(char leds)
{
    // регистр конфигурации направления 0x03 конфигурируем на вывод информации (1-ввод, 0-
вывод)
  I2C_WriteByte(0x03, 0x00, LED_i2c_addr);
  I2C_WriteByte(0x01, leds, LED_i2c_addr);    // выводим данные в регистр OUTPUT (0x01)
  LED_config=leds;                // сохраняем новую конфигурацию
}


// Преобразование номера светодиода в бит, с которым нужно проводить операцию
// 1 = 10000000, 2 = 01000000 ... 8 = 00000001
char LED_convert(char led)
{
  if(led<1)
    led=1;
  if(led>8)
    led=8;
  led=9-led;
  return (1<<(led-1));
}


// Выключить все светодиоды
void LED_clear()
{
  LED_out(0x00);
}


// Инвертировать все светодиоды
void LED_invert()
{
  LED_out(LED_config ^ 0xff);
}
```

```
// Включить светодиод с номером от 1 до 8 (слева направо)
void LED_set(char led)
{
  led=LED_convert(led);
  LED_config |= led;      // устанавливаем соответствующий разряд
  LED_out(LED_config);    // выводим в регистр
}


// Выключить светодиод с номером от 1 до 8 (слева направо)
void LED_reset(char led)
{
  led=LED_convert(led);
  LED_config &= ~(led);   // сбрасываем соответствующий разряд
  LED_out(LED_config);    // выводим в регистр
}


// Сменить состояние светодиода с номером от 1 до 8 (слева направо)
void LED_change(byte led)
{
  led=LED_convert(led);
  LED_config ^= led;      // меняем состояние соответствующего разряда (XOR)
  LED_out(LED_config);    // выводим в регистр
}


void LED_fx1(int n)
{
  LED_clear();
  LED_set(1);
  wait_1ms(n);
  LED_set(3);
  wait_1ms(n);
  LED_set(5);
  wait_1ms(n);
  LED_set(7);
  wait_1ms(n);
  LED_reset(1);
  wait_1ms(n);
  LED_reset(3);
  wait_1ms(n);
  LED_reset(5);
  wait_1ms(n);
  LED_reset(7);
  wait_1ms(n);
  LED_clear();
}


void LED_fx2(int n)
{
  LED_clear();
  LED_out(0x81);
  wait_1ms(n);
  LED_out(0x42);
  wait_1ms(n*8);
  LED_out(0x24);
  wait_1ms(n*5);
  LED_out(0x18);
  wait_1ms(n*3);
```

```
  LED_out(0x24);
  wait_1ms(n*5);
  LED_out(0x42);
  wait_1ms(n*8);
  LED_clear();
}


void LED_fx3(int n)
{
  LED_clear();
  LED_out(0x01);
  wait_1ms(n*6);
  LED_out(0x02);
  wait_1ms(n*4);
  LED_out(0x04);
  wait_1ms(n*2);
  LED_out(0x08);
  wait_1ms(n*1);
  LED_out(0x10);
  wait_1ms(n*1);
  LED_out(0x20);
  wait_1ms(n*2);
  LED_out(0x40);
  wait_1ms(n*4);
  LED_out(0x80);
  wait_1ms(n*6);

  LED_out(0x40);
  wait_1ms(n*4);
  LED_out(0x20);
  wait_1ms(n*2);
  LED_out(0x10);
  wait_1ms(n*1);
  LED_out(0x08);
  wait_1ms(n*1);
  LED_out(0x04);
  wait_1ms(n*2);
  LED_out(0x02);
  wait_1ms(n*4);
  LED_clear();
}
```