МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра Систем автоматического управления

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №2
ПО ДИСЦИПЛИНЕ «МИКРОПРОЦЕССОРНАЯ ТЕХНИКА В МЕХАТРОНИКЕ И РОБОТОТЕХНИКЕ»
ТЕМА: «РАБОТА С ТРЁХЦВЕТНЫМ СВЕТОДИОДОМ (ШИМ)»

Студенты гр. 1492 _____ Старцев Н.А.

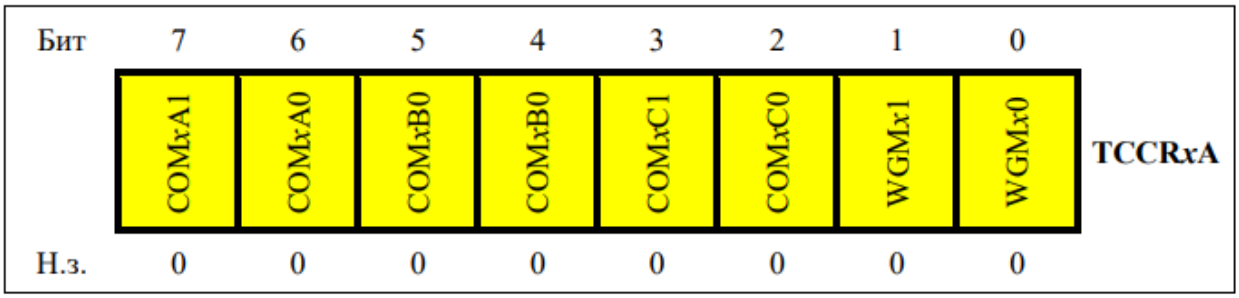
Преподаватель _____ Илатовская Е.В.

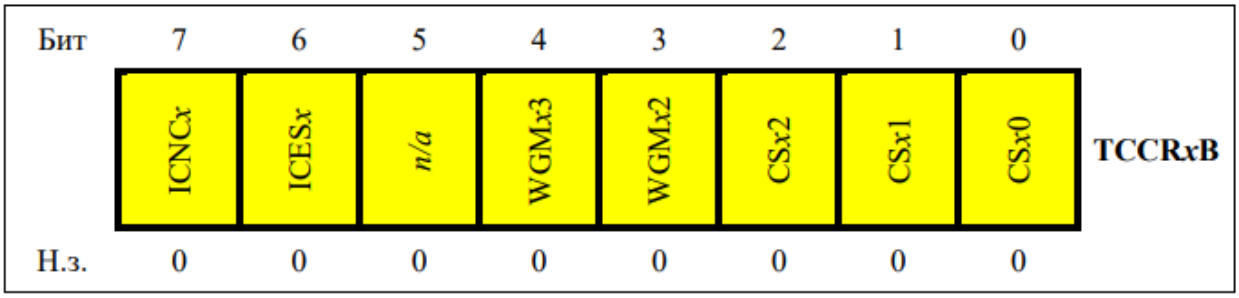
Санкт-Петербург
2024

Цель работы: получение навыков работы с таймер счетчиками, использование таймеров для шим управления яркостью и цветом трехцветного светодиода

Основные сведения

Управление 16-битными таймерами/счётчиками. 16-битные таймеры Atmega имеют 3 независимых канала ШИМ – *A*, *B* и *C*. Для установки режима работы таймера используются регистры TCCR*x*A и TCCR*x*B (*x* – 1 или 3).





Биты CS*xi* – это биты предделителя. Если необходимо замедлить таймер, пользователь может использовать предделитель. В таблице ниже дано описание битов CS*xi*. Согласно ей, если все биты CS*xi* равны нулю, таймер остановлен, это означает, что если пользователю нужно запустить таймер, биты CS*xi* должны быть отредактированы.

CSx2	*CSx1*	*CSx0*	*Описание*
0	0	0	Таймер остановлен **(!)**
0	0	1	Предделитель на 1
0	1	0	Предделитель на 8
0	1	1	Предделитель на 64
1	0	0	Предделитель на 256
1	0	1	Предделитель на 1024
1	1	0	Внешний задатчик на ножке Tn. Счёт по срезу
1	1	1	Внешний задатчик на ножке Tn. Счёт по фронту

Биты WGM*xi* определяют режим генерации сигнала таймера/счетчика и задают верхний предел счёта (TOP). В таблице ниже показаны режимы работы 16-ти битных таймеров/счётчиков.

WGMx3	*WGMx2*	*WGMx1*	*WGMx0*	*Режим таймера*	*Предел*
0	0	0	0	Нормальный	0xFFFF
0	0	0	1	ШИМ с ФК, 8 бит	0x00FF
0	0	1	0	ШИМ с ФК, 9 бит	0x01FF
0	0	1	1	ШИМ с ФК, 10 бит	0x03FF
0	1	0	0	Сброс по совпадению (СТС)	OCR*x*A
0	1	0	1	Быстрая ШИМ, 8 бит	0x00FF
0	1	1	0	Быстрая ШИМ, 9 бит	0x01FF
0	1	1	1	Быстрая ШИМ, 10 бит	0x03FF
1	0	0	0	ШИМ ФЧК	ICR*x*
1	0	0	1	ШИМ ФЧК	OCR*x*A
1	0	1	0	ШИМ ФК	ICR*x*
1	0	1	1	ШИМ ФК	OCR*x*A
1	1	0	0	СТС	ICR*x*
1	1	0	1	Резерв	-
1	1	1	0	Быстрая ШИМ	ICR*x*
1	1	1	1	Быстрая ШИМ	OCR*x*A

Биты COM*xNi* (где *N* – имя канала, *i* – 0…1) определяют вид выходного сигнала (OC*xN*). Если один или оба бита COM*x*A1: 0 выставлены в единицу, выход OC*x*A запрещает нормальное функционирование ножки порта вывода/ввода, к которому он подключен. То же самое для каналов *B* и *C*. Бит регистра направления данных (DDR), соответствующий контакту OC*x*A, OC*x*B или OC*x*C, должен быть выставлен в 1 для посылки генерируемого ШИМ сигнала во внешнее устройство. См. Таблицы ниже.

Режим вывода при сравнении, не ШИМ

COMxN1	*COMxN0*	*Описание*
0	0	Нормальный режим, OC*xN* отключен
0	1	Изменение состояния OC*xN* при совпадении
1	0	Сброс OC*xN* при совпадении
1	1	Установка OC*xN* при совпадении

Режим вывода при сравнении, быстрая ШИМ

COMxN1	*COMxN0*	*Описание*
0	0	Нормальный режим, OC*xN* отключен
0	1	Зарезервировано
1	0	Сброс OC*xN* при совпадении
1	1	Установка OC*xN* при совпадении

Режим вывода при сравнении, ШИМ с фазовой коррекцией

COMxN1	*COMxN0*	*Описание*
0	0	Нормальный режим, OC*xN* отключен
0	1	Зарезервировано
1	0	Сброс OC*xN* во время прямого счёта, установка во время обратного счёта
1	1	Установка OC*xN* во время обратного счёта, сброс во время прямого счёта

В МК сигнал ШИМ формируется при сравнении значения регистра TCNT с заранее заданным значением из регистра OCR (см. рис. 4.2.1).

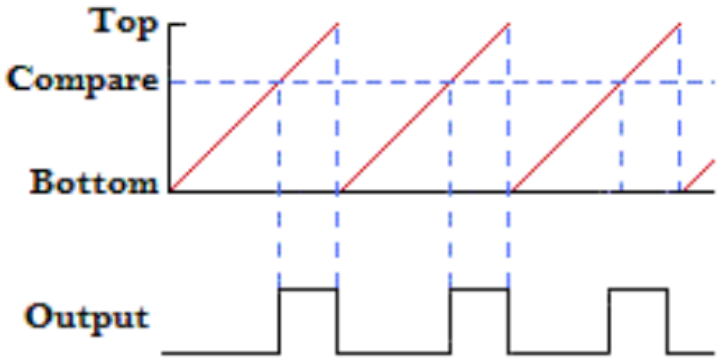
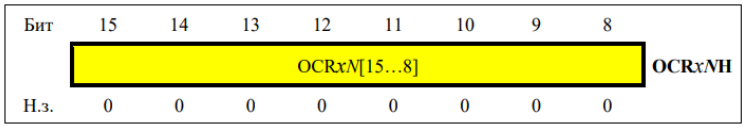
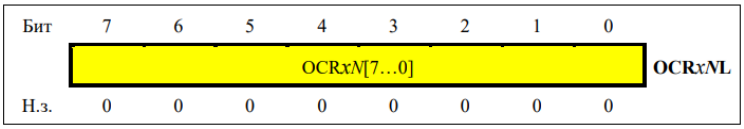


Рис. 4.2.1. ШИМ сигнал

Сравниваемое значение хранится в регистрах OCR*x*NH и OCR*x*NL (где *N* – имя канала).

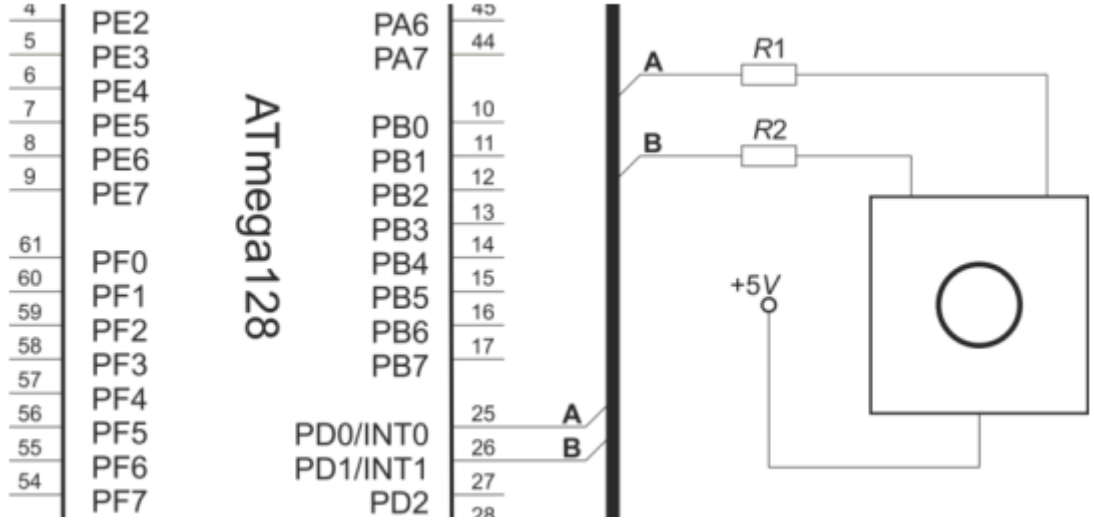


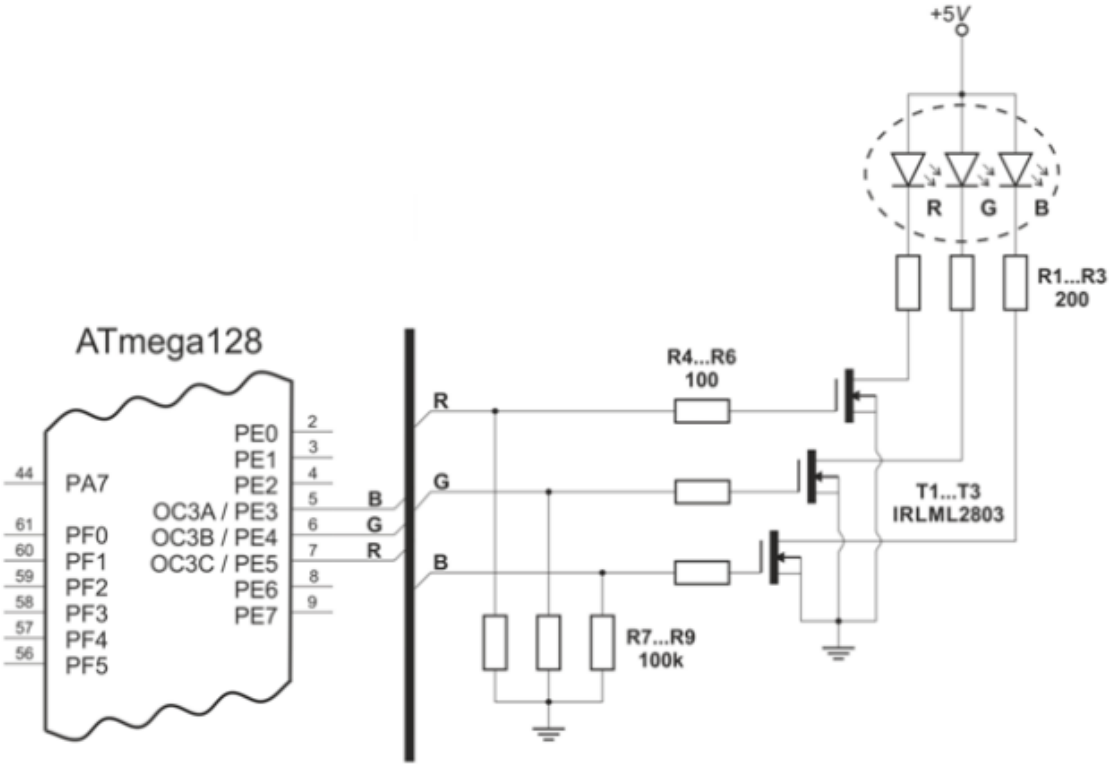


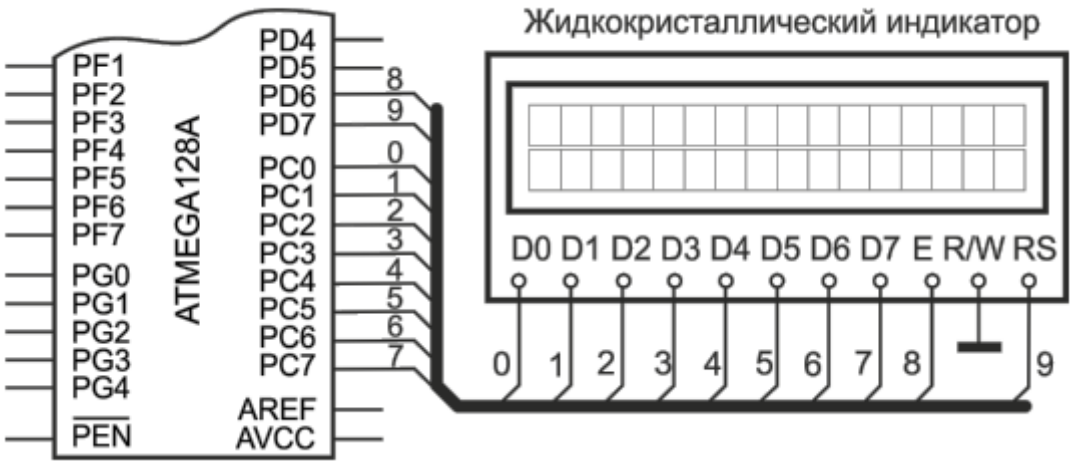
Список всех управляющих комбинаций сигналов RS, R/W и разрядов регистра IR, а также выполняемые по ним команды приведены в таблице.

RS	R/W	D7	D6	D5	D4	D3	D2	D1	D0	Назначение команды
0	0	0	0	0	0	0	0	0	1	Очистить дисплей и установить курсор в нулевую позицию (адрес 0)
0	0	0	0	0	0	0	0	1	*	Установить курсор в нулевую позицию (адрес 0). Установить дисплей относительно буфера DDRAM в начальную позицию.
0	0	0	0	0	0	0	1	I/D	S	Установить направление сдвига курсора влево или вправо (I/D = 0/1) при записи очередного кода в DDRAM. Разрешить (S = 1) сдвиг окна вместе со сдвигом курсора
0	0	0	0	0	0	1	D	C	B	Включить-выключить (D = 1/0) индикатор. Зажечь-погасить (C = 1/0) курсор. Сделать курсор в виде мигающего знакоместа (B = 1)
0	0	0	0	0	1	S/C	R/L	*	*	Переместить курсор (S/C = 0) или сдвинуть дисплей (S/C = 1) вправо (R/L = 1) или влево (R/L = 0)
0	0	0	0	1	DL	N	F	*	*	Установить разрядность шины данных 4 или 8 бит (DL = 0/1), количество строк – одна или две (N = 0/1), шрифт – 5 × 7 или 5 × 10 точек (F = 0/1)
0	0	0	1	AG	AG	AG	AG	AG	AG	Установка адреса CGRAM.
0	0	1	AD	AD	AD	AD	AD	AD	AD	Установка адреса DDRAM.

Схема подключения







Результаты работы

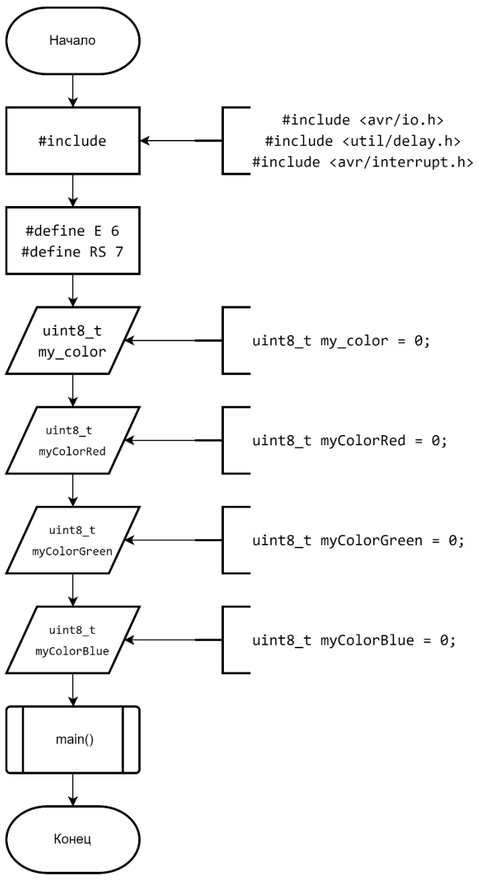
написать программу управления цветом и яркостью светодиода с помощью механического энкодера. В таблице заданий 0....255 означает, что при вращении энкодера по часовой стрелке яркость данного цвета увеличивается, а против часовой стрелки – уменьшается. На ЖКИ выводятся значения яркости (0…255) для каждого из трёх каналов RGB диода в виде: R: xxx G: xxx B: xxx

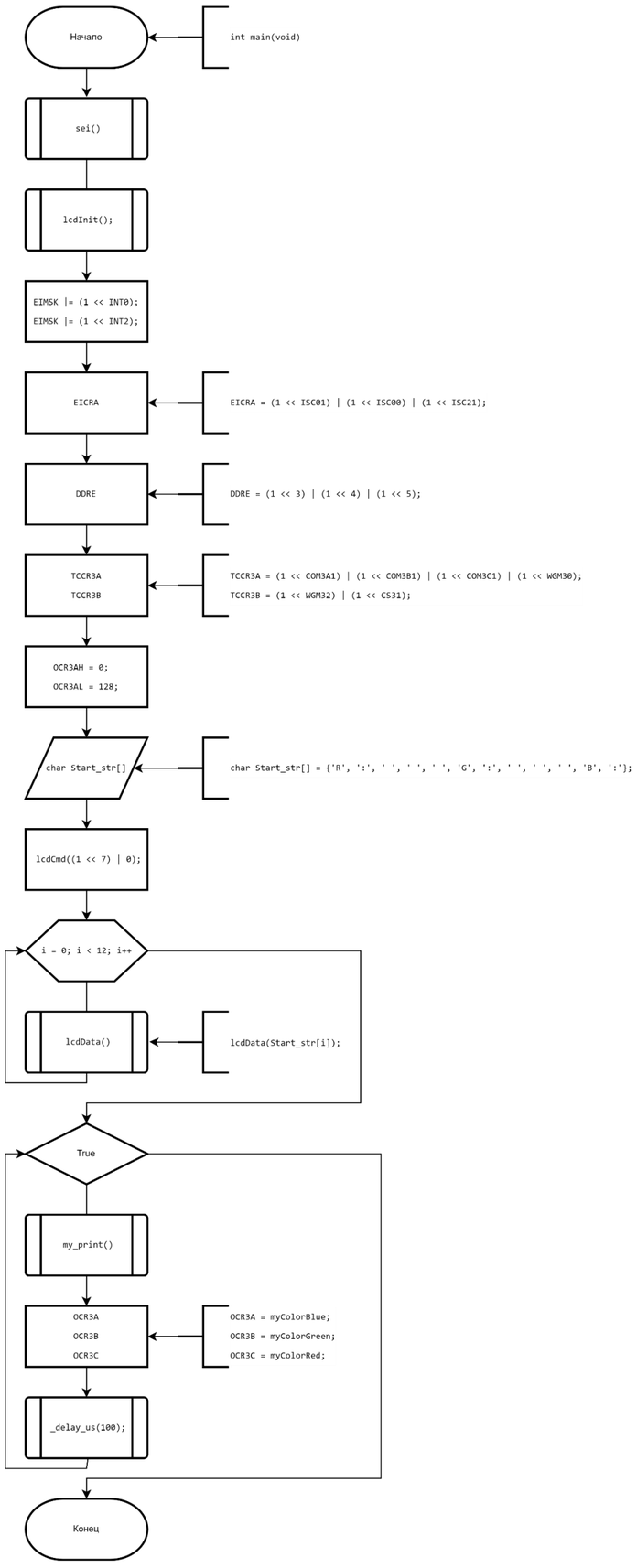
Код программы:

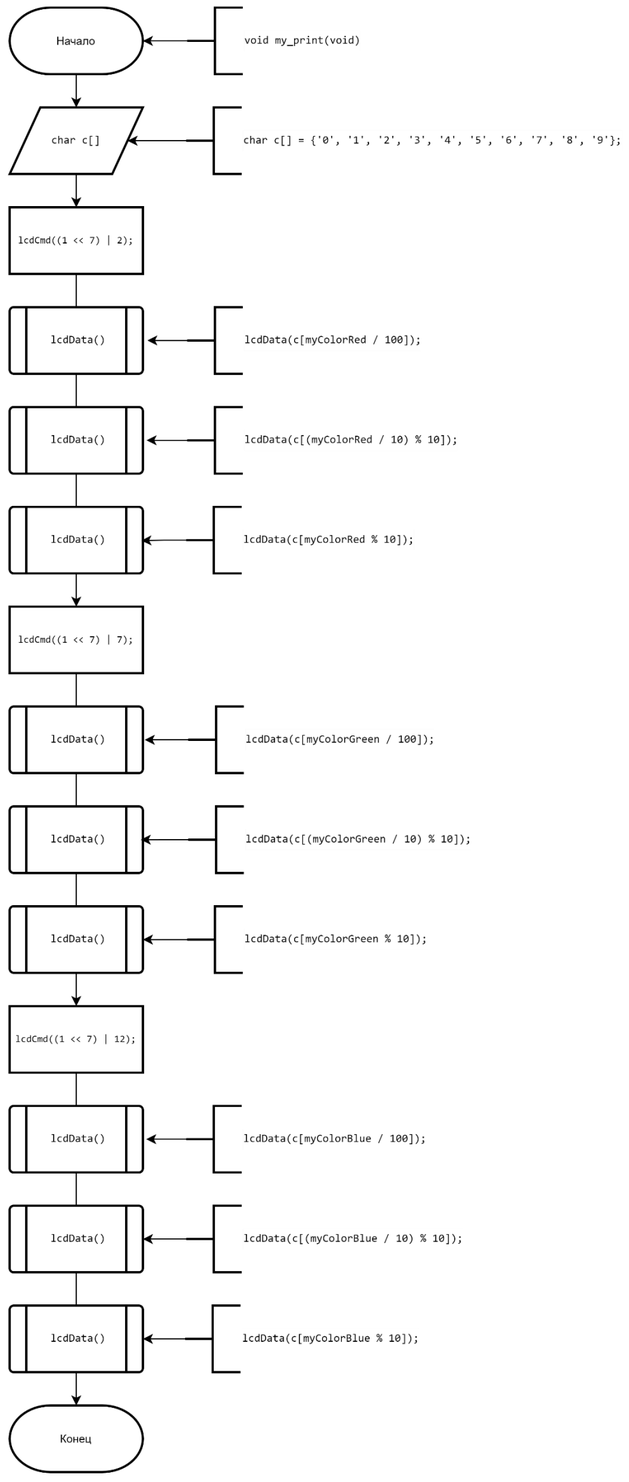
```
#include <avr/io.h>
// #define F_CPU 11059200
#include <util/delay.h> //для использования пауз

#include <avr/interrupt.h>

#define E 6
#define RS 7

uint8_t my_color = 0;
uint8_t myColorRed = 0;
uint8_t myColorGreen = 0;
uint8_t myColorBlue = 0;

void lcdCmd(uint8_t cmd)
{                                   // посыл команды на экран
    DDRC = 0xFF;                    // все разряды PORTC на выход
    DDRD |= ((1 << E) | (1 << RS)); // разряды PORTD на выход
    PORTD &= ~(1 << RS);            // выбор регистра команд RS=0
    PORTC = cmd;                    // записать команду в порт PORTC
    PORTD |= (1 << E);              // \ сформировать на
    _delay_us(5);                   // | выводе E строб 1-0
    PORTD &= ~(1 << E);             // / передачи команды
    _delay_ms(10);                  // задержка для завершения записи
}
void lcdInit(void)
{                                   // инициализация (ВКЛ) экрана
    DDRC = 0xFF;                    // все разряды PORTC на выход
    DDRD |= ((1 << E) | (1 << RS)); // разряды PORTD на выход
    _delay_ms(100);                 // задержка для установления питания
    lcdCmd(0x30);                   // \ вывод
    lcdCmd(0x30);                   // | трех
    lcdCmd(0x30);                   // / команд 0x30
    lcdCmd(0x38);                   // 8 разр.шина, 2 строки, 5 × 7 точек
    lcdCmd(0x0C);                   // включить ЖКИ
    lcdCmd(0x06);                   // инкремент курсора, без сдвига экрана
    lcdCmd(0x01);                   // очистить экран, курсор в начало
}
void lcdData(uint8_t data)
{ // посыл данных на экран
    DDRC = 0xFF;
    DDRD |= ((1 << E) | (1 << RS));
    PORTD |= (1 << RS);
    PORTC = data;
    PORTD |= (1 << E);
    _delay_us(5);
    PORTD &= ~(1 << E);
    _delay_ms(1);
```

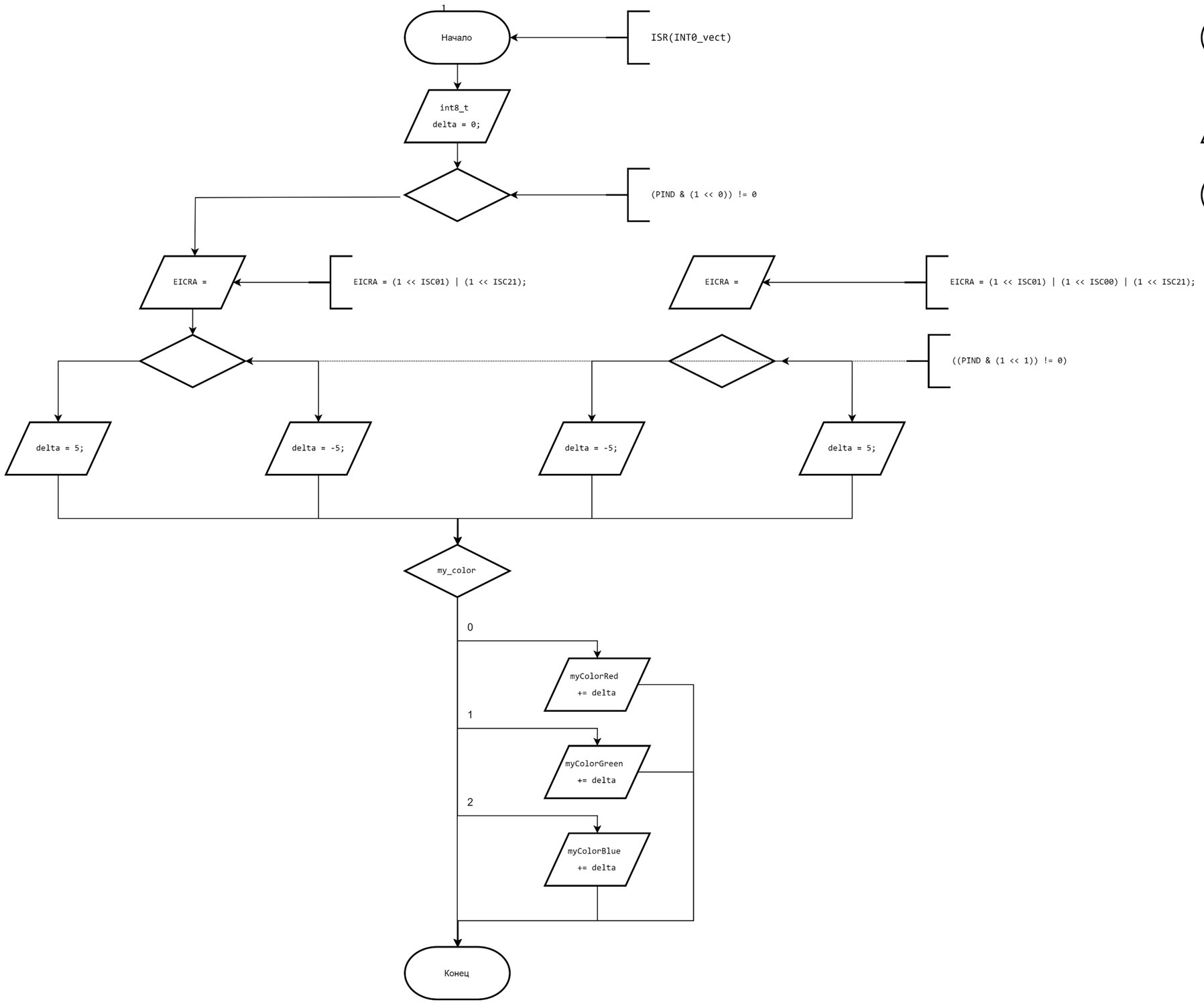
```
}

ISR(INT0_vect) // прерывания по инкодеру
{
    int8_t delta = 0;
    if ((PIND & (1 << 0)) != 0)
    {
        EICRA = (1 << ISC01) | (1 << ISC21);
        if ((PIND & (1 << 1)) != 0)
            delta = 5;
        else
            delta = -5;
    }
    else
    {
        EICRA = (1 << ISC01) | (1 << ISC00) | (1 << ISC21);
        if ((PIND & (1 << 1)) != 0)
            delta = -5;
        else
            delta = 5;
    }
    switch (my_color)
    {
    case 0:
        myColorRed += delta;
        break;
    case 1:
        myColorGreen += delta;
        break;
    case 2:
        myColorBlue += delta;
        break;

    default:
        break;
    }
}
ISR(INT2_vect) { my_color = (my_color + 1) % 3; }

uint8_t TabCon[] = {0x41, 0xA0, 0x42, 0xA1, 0xE0, 0x45, 0xA3, 0xA4,
                    0xA5, 0xA6, 0x4B, 0xA7, 0x4D, 0x48, 0x4F, 0xA8, 0x50, 0x43, 0x54,
0xA9,
                    0xAA, 0x58, 0xE1, 0xAB, 0xAC, 0xE2, 0xAD, 0xAE, 0x62, 0xAF, 0xB0,
0xB1,
                    0x61, 0xB2, 0xB3, 0xB4, 0xE3, 0x65, 0xB6, 0xB7, 0xB8, 0xB9, 0xBA,
0xBB,
                    0xBC, 0xBD, 0x6F, 0xBE, 0x70, 0x63, 0xBF, 0x79, 0x5C, 0x78, 0xE5,
0xC0,
                    0xC1, 0xE6, 0xC2, 0xC3, 0xC4, 0xC5, 0xC6, 0xC7};
uint8_t Code(uint8_t symb)
{
    //[]-----------------------------------------------[]
    //| Назначение: перекодировка символов кириллицы |
    //| Входные параметры: symb – символ ASCII |
    //| Функция возвращает код отображения символа |
    //[]-----------------------------------------------[]
    uint8_t a = symb >= 192 ? TabCon[symb - 192] : symb;
    return a;
}

void my_print(void)
{
```

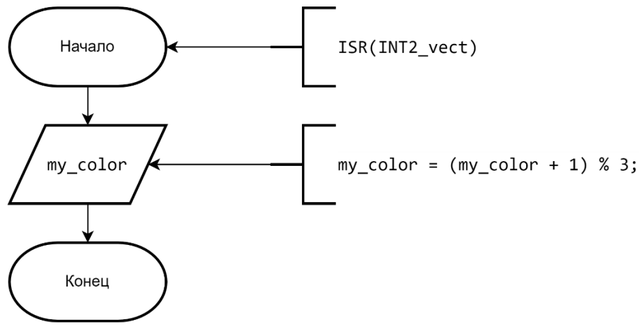
```
    char c[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};
    lcdCmd((1 << 7) | 2);
    lcdData(c[myColorRed / 100]);
    lcdData(c[(myColorRed / 10) % 10]);
    lcdData(c[myColorRed % 10]);
    lcdCmd((1 << 7) | 7);
    lcdData(c[myColorGreen / 100]);
    lcdData(c[(myColorGreen / 10) % 10]);
    lcdData(c[myColorGreen % 10]);
    lcdCmd((1 << 7) | 12);
    lcdData(c[myColorBlue / 100]);
    lcdData(c[(myColorBlue / 10) % 10]);
    lcdData(c[myColorBlue % 10]);
}

int main(void)
{
    sei();
    lcdInit();

    EIMSK |= (1 << INT0);
    EIMSK |= (1 << INT2);
    EICRA = (1 << ISC01) | (1 << ISC00) | (1 << ISC21);

    DDRE = (1 << 3) | (1 << 4) | (1 << 5);
    /* Инициализация таймера №3. 8-ми битная быстрая
    ШИМ, преддедитель на 8 */
    TCCR3A = (1 << COM3A1) | (1 << COM3B1) | (1 << COM3C1) | (1 << WGM30);
    TCCR3B = (1 << WGM32) | (1 << CS31);
    OCR3AH = 0;
    OCR3AL = 128;

    char Start_str[] = {'R', ':', ' ', ' ', ' ', 'G', ':', ' ', ' ', ' ', 'B', ':'};
    lcdCmd((1 << 7) | 0);
    for (uint8_t i = 0; i < 12; i++)
    {
        lcdData(Start_str[i]);
    }

    while (1)
    {
        my_print();
        OCR3A = myColorBlue;
        OCR3B = myColorGreen;
        OCR3C = myColorRed;
        _delay_us(100);
    }
}
```


Начало
#include
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#define E 6
#define RS 7
uint8_t my_color
uint8_t my_color = 0;
uint8_t myColorRed
uint8_t myColorRed = 0;
uint8_t myColorGreen
uint8_t myColorGreen = 0;
uint8_t myColorBlue
uint8_t myColorBlue = 0;
main()
Конец


Начало
int main(void)
sei()
lcdInit();
EIMSK |= (1 << INT0);
EIMSK |= (1 << INT2);
EICRA
EICRA = (1 << ISC01) | (1 << ISC00) | (1 << ISC21);
DDRE
DDRE = (1 << 3) | (1 << 4) | (1 << 5);
TCCR3A
TCCR3B
TCCR3A = (1 << COM3A1) | (1 << COM3B1) | (1 << COM3C1) | (1 << WGM30);
TCCR3B = (1 << WGM32) | (1 << CS31);
OCR3AH = 0;
OCR3AL = 128;
char Start_str[]
char Start_str[] = {'R', ':', ' ', ' ', ' ', 'G', ':', ' ', ' ', ' ', 'B', ':'};
lcdCmd((1 << 7) | 0);
i = 0; i < 12; i++
lcdData()
lcdData(Start_str[i]);
True
my_print()
OCR3A
OCR3B
OCR3C
OCR3A = myColorBlue;
OCR3B = myColorGreen;
OCR3C = myColorRed;
_delay_us(100);
Конец


Начало
void my_print(void)
char c[]
char c[] = {'0', '1', '2', '3', '4', '5', '6', '7', '8', '9'};
lcdCmd((1 << 7) | 2);
lcdData()
lcdData(c[myColorRed / 100]);
lcdData(c[(myColorRed / 10) % 10]);
lcdData(c[myColorRed % 10]);
lcdCmd((1 << 7) | 7);
lcdData(c[myColorGreen / 100]);
lcdData(c[(myColorGreen / 10) % 10]);
lcdData(c[myColorGreen % 10]);
lcdCmd((1 << 7) | 12);
lcdData(c[myColorBlue / 100]);
lcdData(c[(myColorBlue / 10) % 10]);
lcdData(c[myColorBlue % 10]);
Конец


Начало
ISR(INT0_vect)
int8_t
delta = 0;
(PIND & (1 << 0)) != 0
EICRA =
EICRA = (1 << ISC01) | (1 << ISC21);
EICRA =
EICRA = (1 << ISC01) | (1 << ISC00) | (1 << ISC21);
((PIND & (1 << 1)) != 0)
delta = 5;
delta = -5;
delta = -5;
delta = 5;
my_color
0
myColorRed
+= delta
1
myColorGreen
+= delta
2
myColorBlue
+= delta
Конец


Начало
ISR(INT2_vect)
my_color
my_color = (my_color + 1) % 3;
Конец

Вывод:

В ходе лабораторной работы были освоены навыки взаимодействия с внешними прерываниями, обработка прерываний энкодера и кнопки. Научились использовать шим для регулирования яркости свечения светодиода. И использовали символьный дисплей для отображения данных о шим