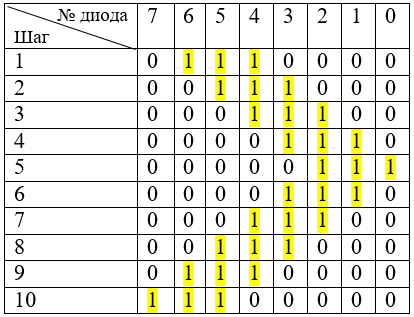
«Змейка». Таблица состояний:



#include <avr/io.h>

#include <avr/interrupt.h>

#define INITIAL\_TIMER\_VALUE 60000

*uint8\_t* pattern = 0b11111000;

*uint8\_t* direction = 0; // Направление движения змейки

*uint8\_t* step = 0;

ISR(TIMER1\_OVF\_vect)

{

PORTC = pattern;

if (direction == 0) {

pattern = (pattern << 1) | (pattern >> 7); // Сдвиг вправо

} else {

pattern = (pattern >> 1) | (pattern << 7); // Сдвиг влево

}

step++;

if(step > 4) {

direction = !direction;

step = 0;

}

TCNT1 = INITIAL\_TIMER\_VALUE;

}

int main(void) {

DDRC=0xFF;

TCCR1B|=(1<<CS12)|(1<<CS10);

TIMSK|=(1<<TOIE1);

TCNT1 = INITIAL\_TIMER\_VALUE;

sei();

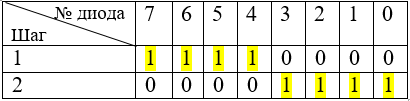
while(1)

{

}

}

«Полицейская мигалка». Таблица состояний:



#include <avr/io.h>

#include <avr/interrupt.h>

#define INITIAL\_TIMER\_VALUE 60000

*uint8\_t* pattern = 0b11110000;

ISR(TIMER1\_OVF\_vect)

{

PORTC = pattern;

pattern = pattern^0b11111111;

TCNT1 = INITIAL\_TIMER\_VALUE;

}

int main(void) {

DDRC=0xFF;

TCCR1B|=(1<<CS12)|(1<<CS10);

TIMSK|=(1<<TOIE1);

TCNT1=63000;

sei();

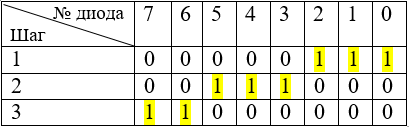
while(1)

{

}

}

«Светофор». Таблица состояний:



#include <avr/io.h>

#include <avr/interrupt.h>

#define INITIAL\_TIMER\_VALUE 60000

*uint8\_t* svetofor = 0;

*uint8\_t* flag = 1;

void update() {

if (flag) {

svetofor++;

} else {

svetofor--;

}

}

ISR(TIMER1\_OVF\_vect) {

switch (svetofor) {

case 0: flag = 1; PORTC = 0b11111000; update(); break;

case 1: PORTC = 0b11000111; update(); break;

case 2: flag = 0; PORTC = 0b00111111; update(); break;

}

TCNT1 = INITIAL\_TIMER\_VALUE;

}

int main(void) {

DDRC=0xFF;

TCCR1B|=(1<<CS12)|(1<<CS10);

TIMSK|=(1<<TOIE1);

TCNT1=INITIAL\_TIMER\_VALUE;

sei();

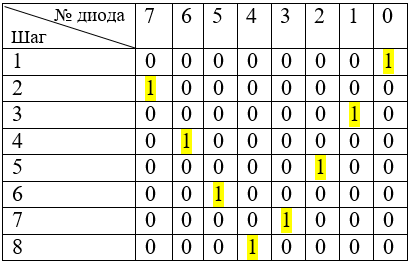
while(1)

{

}

}

«Край-край с смещением к центру». Таблица состояний:



#include <avr/io.h>

#include <avr/interrupt.h>

#define INITIAL\_TIMER\_VALUE 60000

*uint8\_t* triangle = 0b11111110;

*uint8\_t* step = 7;

ISR(TIMER1\_OVF\_vect)

{

PORTC = triangle;

if (step % 2 == 1){

triangle = (triangle<<step)|(triangle>>(8-step));

} else {

triangle=(triangle>>step)|(triangle<<(8-step));

}

if (step == 0){

step = 8;

triangle = 0b11111110;

}

step--;

TCNT1 = INITIAL\_TIMER\_VALUE;

}

int main(void) {

DDRC=0xFF;

TCCR1B|=(1<<CS12)|(1<<CS10);

TIMSK|=(1<<TOIE1);

TCNT1=INITIAL\_TIMER\_VALUE;

sei();

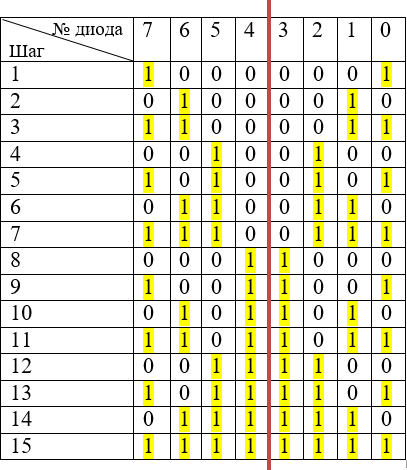
while(1)

{

}

}

«Вывод кодов чисел 1-15 в младшую тетраду и их зеркальное представление в старшей тетраде». Таблица состояний:



#include <avr/io.h>

#include <avr/interrupt.h>

#define INITIAL\_TIMER\_COUNT 63000

#define MAX\_NUM 16

*uint8\_t* num = 1;

unsigned char get\_low\_nibble(unsigned char num) {

return num & 0x0F;

}

unsigned char reverse\_nibble(unsigned char nibble) {

unsigned char reversed\_nibble = 0;

for (int i = 0; i < 4; i++) {

unsigned char bit = (nibble >> i) & 1;

reversed\_nibble |= bit << (3 - i);

}

return reversed\_nibble;

}

unsigned char combine\_nibbles(unsigned char low\_nibble, unsigned char reversed\_nibble) {

return (reversed\_nibble << 4) | low\_nibble;

}

ISR(TIMER1\_OVF\_vect)

{

if (num == MAX\_NUM) {

num = 0;

} else {

*uint8\_t* low\_nibble = get\_low\_nibble(num);

PORTC = combine\_nibbles(low\_nibble, reverse\_nibble(low\_nibble)) ^ 0xFF;

}

num++;

TCNT1 = INITIAL\_TIMER\_COUNT;

}

int main(void) {

DDRC = 0xFF;

TCCR1B |= (1 << CS12) | (1 << CS10);

TIMSK |= (1 << TOIE1);

TCNT1 = INITIAL\_TIMER\_COUNT;

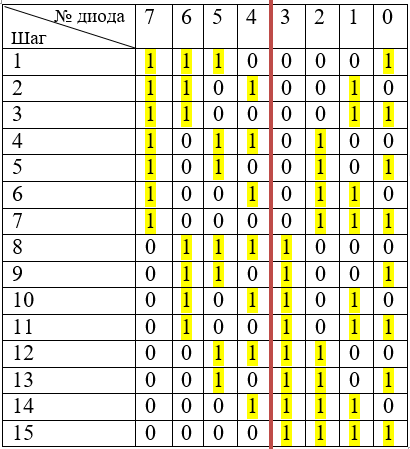
sei();

while(1) {

}

}

«Вывод кодов чисел 1-15 в младшую тетраду и их побитовая инверсия в старшей тетраде». Таблица состояний:



#include <avr/io.h>

#include <avr/interrupt.h>

#define INITIAL\_TIMER\_VALUE 60000

*uint8\_t* get\_low\_nibble(*uint8\_t* num) {

return num & 0x0F;

}

*uint8\_t* reverse\_nibble(*uint8\_t* nibble) {

return nibble ^ 0b1111;

}

*uint8\_t* combine\_nibbles(*uint8\_t* low\_nibble, *uint8\_t* reversed\_nibble) {

return (reversed\_nibble << 4) | low\_nibble;

}

volatile *uint8\_t* num = 1;

ISR(TIMER1\_OVF\_vect)

{

if (num == 16) num = 0;

else {

*uint8\_t* low\_nibble = get\_low\_nibble(num);

PORTC = combine\_nibbles(low\_nibble, reverse\_nibble(low\_nibble)) ^ 0xFF;

}

num++;

TCNT1 = INITIAL\_TIMER\_VALUE;

}

int main(void) {

DDRC = 0xFF;

TCCR1B |= (1 << CS12) | (1 << CS10);

TIMSK |= (1 << TOIE1);

TCNT1 = INITIAL\_TIMER\_VALUE;

sei();

while(1) {

}

}

**Вопросы по лабораторной работе №1**

4.1 Задача предделителя тактовой частоты в микроконтроллерах.

Предделитель тактовой частоты в микроконтроллерах - это функциональный блок, который позволяет делить входную тактовую частоту на более низкую выходную частоту. Это может быть полезно, например, для уменьшения частоты тактового сигнала, чтобы уменьшить энергопотребление или синхронизировать работу различных компонентов микроконтроллера.

Предделитель может быть настроен на определенное деление, обычно путем программирования специальных регистров микроконтроллера. Например, если у нас есть микроконтроллер с тактовой частотой 16 МГц, и нам нужно установить частоту 1 МГц для работы с периферийными устройствами, мы можем использовать предделитель, чтобы поделить исходную частоту на 16 и получить нужную частоту.

4.2 Как задается режим работы портов микроконтроллера?

Режим работы портов микроконтроллера задается через специальные регистры управления портами (Port Control Registers). Эти регистры позволяют настраивать порты микроконтроллера на вход или выход.

Вот пример того, как можно настроить порт в качестве входа или выхода на микроконтроллере AVR

// Настройка порта D на вход

DDRD &= ~(1 << PD2); // Сброс бита PD2

// Настройка порта D на выход

DDRD |= (1 << PD3); // Установка бита PD3

4.3 Как изменится значение переменной, если необходимо вызывать прерывание каждые 2 секунды?

Для настройки прерывания каждые 2 секунды на микроконтроллере ATmega16, необходимо использовать таймер. Вот пример, как можно настроить таймер на ATmega16 для вызова прерывания каждые 2 секунды:

#include <avr/io.h>

#include <avr/interrupt.h>

volatile *uint16\_t* counter = 0;

void timer1\_init() {

TCCR1B |= (1 << WGM12); // Выбираем режим CTC

OCR1A = 31249; // Устанавливаем значение для достижения интервала 2 секунды при тактовой частоте 16 МГц

TIMSK |= (1 << OCIE1A); // Разрешаем прерывание при совпадении с OCR1A

TCCR1B |= (1 << CS12) | (1 << CS10); // Устанавливаем предделитель на 1024

}

ISR(TIMER1\_COMPA\_vect) {

counter++; // Увеличиваем значение переменной каждый раз при вызове прерывания

}

int main() {

timer1\_init();

sei(); // Разрешаем прерывания

while(1) { }

return 0;

}