МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ПОВОЛЖСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ»

Факультет информатики и
вычислительной техники

Кафедра ИиСП

Отчет
по лабораторной работе № 8

по дисциплине «Машинно-зависимые языки программирования»

Выполнил: студент группы ПС-11
Щеглов Г.С
Проверил: Баев А.А.

г. Йошкар-Ола
2024

Цель работы: научиться работать с ЖКИ WH1602

Задания на лабораторную работу:

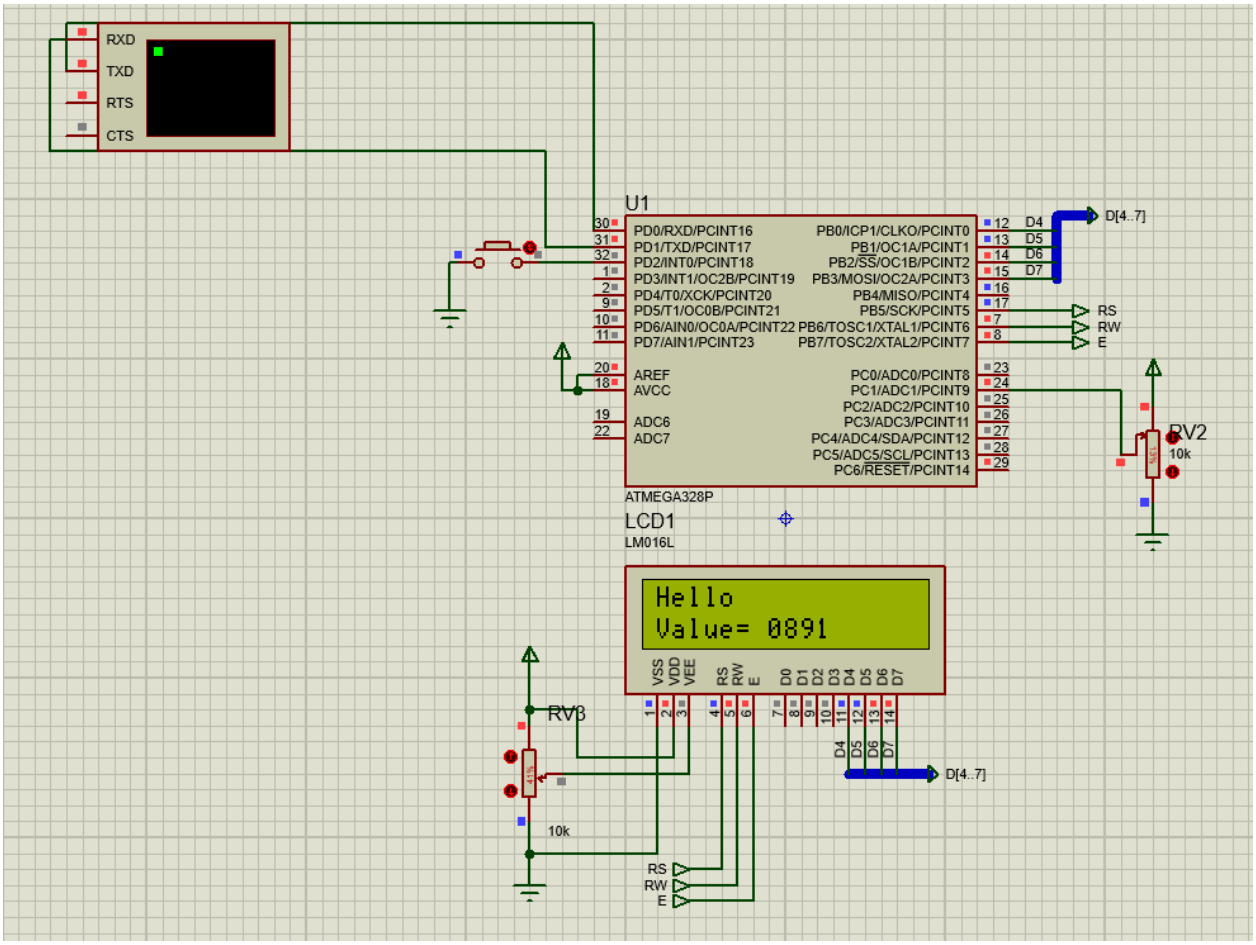
1. Реализовать схему с ЖКИ WH1602
2. Оптимизация кода
3. Дополнительное задание – строка загрузки

1. Теоретические сведения

Учебное пособие - ПРИМЕНЕНИЕ МИКРОКОНТРОЛЛЕРОВ В РАДИОТЕХНИЧЕСКИХ И БИОМЕДИЦИНСКИХ СИСТЕМАХ

2. Практическая часть

Схема с ЖКИ WH1602:



Код на C:

```
#define F_CPU 8000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#define D4 PB0
#define D5 PB1
#define D6 PB2
#define D7 PB3
#define RS PB5
#define RW PB6
#define E PB7
#define CMD 0
#define DATA 1
void InitPorts(void);
void InitTimer1(void);
void Bin2Dec(uint16_t data);
```

```
void DisplayData (uint16_t data);
void InitADC(void);
void InitUSART(void);
void SendChar(char symbol);
void SendString(char * buffer);
void InitLCD(void);
void LCD_Write(uint8_t type, char data);
char LCD_Read(void);
volatile uint8_t bcd_buffer[] = {0,0,0,0};
volatile uint16_t ADC_val, temperature = 0;

int main(void)
{
  InitPorts();
  InitTimer1();
  EIMSK |= (1 << INT0);
  EICRA |= (1 << ISC01);
  InitADC();
  InitUSART();
  InitLCD();

  sei();
  DisplayData(0);
  SendString("Hello\r\n");

  LCD_Write(DATA,'H');
  LCD_Write(DATA,'e');
  LCD_Write(DATA,'l');
  LCD_Write(DATA,'l');
  LCD_Write(DATA,'o');
  LCD_Write(CMD,0x40|0x80); // переход на вторую строку
  LCD_Write(DATA,'V');
  LCD_Write(DATA,'a');
  LCD_Write(DATA,'l');
  LCD_Write(DATA,'u');
  LCD_Write(DATA,'e');
  LCD_Write(DATA,'=');
  LCD_Write(DATA,0x20); // вывод пробела

  while (1)
  {
    Bin2Dec(ADC_val);
    LCD_Write(CMD, 0x47 | 0x80);
    LCD_Write(DATA,0x30+bcd_buffer[3]);
    LCD_Write(DATA,0x30+bcd_buffer[2]);
    LCD_Write(DATA,0x30+bcd_buffer[1]);
    LCD_Write(DATA,0x30+bcd_buffer[0]);
  }
}
```

```
ISR(TIMER1_COMPB_vect){
  //DisplayData(0x1E61);
}

ISR(INT0_vect){
  Bin2Dec(ADC_val);
  SendString("Value = ");
  SendChar(0x30 + bcd_buffer[3]);
  SendChar(0x30 + bcd_buffer[2]);
  SendChar(0x30 + bcd_buffer[1]);
  SendChar(0x30 + bcd_buffer[0]);
  SendString("\r\n");
}

ISR(ADC_vect){
  ADC_val  = ADC;
}

ISR(USART_RX_vect){
  if(UDR0 == 0x20){
    SendString("Roger that\r\n");
  }
}

void InitPorts(void){
  DDRB=0xFF;
  PORTB=0;
}

void InitTimer1( void){
  TCCR1A = 0;
  TCCR1B = (1<<CS11 | 1<<CS10 | 1<<WGM12);
  TCNT1 = 0;
  TIMSK1 |= (1<<OCIE1B);
  OCR1A = 12500;
  OCR1B = 12500;
}

void Bin2Dec(uint16_t data){
  bcd_buffer[3] = (uint8_t)(data/1000);
  data = data % 1000;
  bcd_buffer[2] = (uint8_t)(data/100);
  data = data % 100;
  bcd_buffer[1] = (uint8_t)(data/10);
  data = data % 10;
  bcd_buffer[0] = (uint8_t)(data);
}

void DisplayData (uint16_t data){
  Bin2Dec(data);
```

```
}

void InitADC( void){
  ADMUX = (1<<MUX0);
  //Align left, ADC1
  ADCSRB = (1<<ADTS2 | 1<<ADTS0); //Start on Timer1 COMPB
  //Enable, auto update, IRQ enable
  ADCSRA = (1<<ADEN | 1<<ADATE | 1<<ADIE);
}

void InitUSART(){
  UCSR0B = (1<<RXEN0 | 1<<TXEN0 | 1<<RXCIE0);
  UCSR0C = (1<<UCSZ01 | 1<<UCSZ00);
  UBRR0H = 0;
  UBRR0L = 0x67;
}

void SendChar(char symbol){
  while (!(UCSR0A & (1<<UDRE0)));
  UDR0 = symbol;
}

void SendString(char * buffer){
  while(*buffer != 0){
    SendChar(*buffer++);
  }
}

void InitLCD(void){
  uint8_t BF = 0x80;  //объявление переменной-флага BF
  _delay_ms(40);      //ожидание согласно алгоритму
  PORTB &= ~(1<<RS);  //выставление на шину данных
  PORTB = (0x30>>4);  //старшей тетрады команды 0x30
  PORTB |= (1<<E);    // выставление сигнала E
  asm("nop");        // ожидание в течение 3-х тактов
  asm("nop");
  asm("nop");
  PORTB &= ~(1<<E);
  PORTB = 0;
  _delay_ms(5);       // ожидание согласно алгоритму
  PORTB &= ~(1<<RS);
  PORTB = (0x30>>4);  // вторая отправка команды 0x30
  PORTB |= (1<<E);
  asm("nop");
  asm("nop");
  asm("nop");
  PORTB &= ~(1<<E);
  PORTB = 0;
  _delay_us(150);     // ожидание согласно алгоритму
  PORTB &= ~(1<<RS);
```

```
  PORTB = (0x30>>4);  // третья отправка команды 0x30
  PORTB |= (1<<E);
  asm("nop");
  asm("nop");
  asm("nop");
  PORTB &= ~(1<<E);
  PORTB = 0;
  _delay_ms(5);
  do{                  // ожидание до тех пор, пока флаг BF
    // не освободится
    BF = (0x80 & LCD_Read());
  }
  while(BF == 0x80);
  PORTB &= ~(1<<RS);
  PORTB = (0x20 >> 4); //установка ширины шины данных
  //в 4 бит
  PORTB |= (1<<E);
  asm("nop");
  asm("nop");
  asm("nop");
  PORTB &= ~(1<<E);
  PORTB = 0;
  do{
    BF = (0x80 & LCD_Read());
  }
  while(BF == 0x80);
  // здесь обмен начинает работать по 4 проводной шине
  LCD_Write(CMD,0x28); // установка режима 2 строки,
  // знакоместо 5*8
  LCD_Write(CMD,0x0C); // включение отображения
  LCD_Write(CMD,0x06); // автоинкремент счетчика
}

void LCD_Write(uint8_t type,char data){
  uint8_t BF = 0x80;
  do{
    // ожидание завершения предыдущей операции
    BF = 0x80 & LCD_Read();
  }
  while(BF == 0x80);
  PORTB |= (type<<RS); // установка RS в зависимости от
  // типа данных
  PORTB |= (1<<E);
  PORTB &= ~(0x0F);
  PORTB |= (0x0F & (data>>4)); //передача старшей тетрады
  PORTB &= ~(1<<E);
  asm("nop");
  asm("nop");
  asm("nop");
  PORTB |= (1<<E);
```

```
  PORTB &= ~(0x0F);
  PORTB |= (0x0F & data);       // передача младшей тетрады
  PORTB &= ~(1<<E);
  PORTB = 0;
}

char LCD_Read(void){
  char retval = 0;
  PORTB &= ~(1<<RS);
  PORTB |= (1<<RW);
  DDRB &= ~(1<<D4|1<<D5|1<<D6|1<<D7);
  PORTB |= (1<<E);
  asm("nop");
  asm("nop");
  retval = (PINB & 0x0F)<<4;    //чтение старшей тетрады байта
  PORTB&=~(1<<E);
  asm("nop");
  asm("nop");
  asm("nop");
  PORTB |= (1<<E);
  asm("nop");
  asm("nop");
  retval |= (PINB & 0x0F);      // чтение младшей тетрады
  PORTB&=~(1<<E);
  DDRB |= (1<<D4|1<<D5|1<<D6|1<<D7); //настройка выводов
  // на выход
  PORTB = 0;
  return retval;
}
```

Оптимизированный код:

```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>

#define D4 PB0
#define D5 PB1
#define D6 PB2
#define D7 PB3
#define RS PB5
#define RW PB6
#define E PB7

#define CMD 0
#define DATA 1
volatile uint8_t bcd_buffer[4] = {0};
volatile uint16_t ADC_val = 0;
```

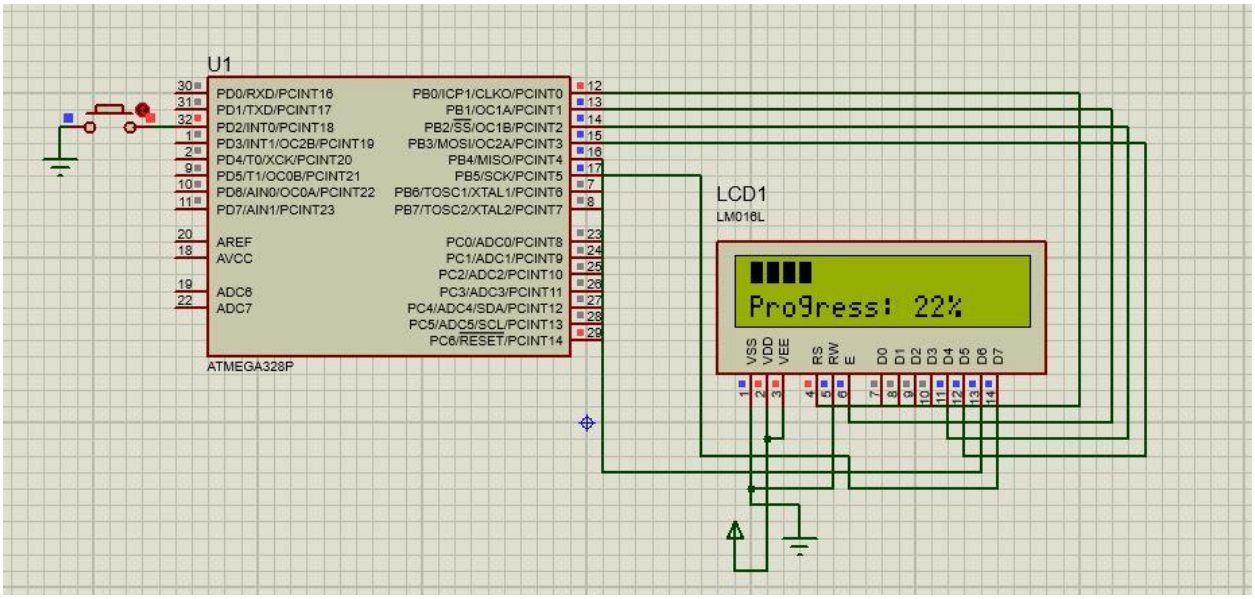
```
void InitPorts(void);
void InitTimer1(void);
void Bin2Dec(uint16_t data);
void InitADC(void);
void InitUSART(void);
void SendChar(char symbol);
void SendString(const char *buffer);
void InitLCD(void);
void LCD_Write(uint8_t type, uint8_t data);
uint8_t LCD_Read(void);

int main(void)
{
  InitPorts();
  InitTimer1();
  InitADC();
  InitUSART();
  InitLCD();

  EIMSK |= (1 << INT0);
  EICRA |= (1 << ISC01);
  sei();

  const char hello_msg[] = "Hello";
  const char value_msg[] = "Value= ";
  for(uint8_t i = 0; hello_msg[i]; i++) LCD_Write(DATA, hello_msg[i]);
  for(uint8_t i = 0; value_msg[i]; i++) LCD_Write(DATA, value_msg[i]);

  SendString("Hello\r\n");

  while (1)
  {
    Bin2Dec(ADC_val);
    LCD_Write(CMD, 0xC7);
    for(uint8_t i = 0; i < 4; i++) {
      LCD_Write(DATA, '0' + bcd_buffer[3-i]);
    }
  }
}


ISR(TIMER1_COMPB_vect) {}
ISR(INT0_vect) {
  Bin2Dec(ADC_val);
  SendString("Value = ");
  for(uint8_t i = 0; i < 4; i++) {
    SendChar('0' + bcd_buffer[3-i]);
  }
  SendString("\r\n");
}
```

```
ISR(ADC_vect) { ADC_val = ADC; }
ISR(USART_RX_vect) { if(UDR0 == 0x20) SendString("Roger that\r\n"); }

void InitPorts(void) {
  DDRB = 0xFF;
  PORTB = 0;
}

void InitTimer1(void) {
  TCCR1A = 0;
  TCCR1B = (1<<CS11) | (1<<CS10) | (1<<WGM12);
  TCNT1 = 0;
  TIMSK1 |= (1<<OCIE1B);
  OCR1A = 12500;
  OCR1B = 12500;
}

void InitADC(void) {
  ADMUX = (1<<MUX0);
  ADCSRB = (1<<ADTS2) | (1<<ADTS0);
  ADCSRA = (1<<ADEN) | (1<<ADATE) | (1<<ADIE) | (1<<ADPS2) |
(1<<ADPS1);
}

void InitUSART(void) {
  UCSR0B = (1<<RXEN0) | (1<<TXEN0) | (1<<RXCIE0);
  UCSR0C = (1<<UCSZ01) | (1<<UCSZ00);
  UBRR0H = 0;
  UBRR0L = 103; // 4800 бод при 8 МГц
}

void Bin2Dec(uint16_t data) {
  bcd_buffer[3] = (data / 1000) % 10;
  bcd_buffer[2] = (data / 100) % 10;
  bcd_buffer[1] = (data / 10) % 10;
  bcd_buffer[0] = data % 10;
}


void SendChar(char symbol) {
  while (!(UCSR0A & (1<<UDRE0)));
  UDR0 = symbol;
}

void SendString(const char *buffer) {
  while (*buffer) SendChar(*buffer++);
}
void InitLCD(void) {
  for(uint8_t i = 0; i < 3; i++) {
    PORTB = (0x30>>4) | (1<<E);
```

```
    _delay_us(1);
    PORTB = 0;
    if(i == 0) _delay_ms(5);
    else if(i == 1) _delay_us(150);
    else _delay_ms(5);
  }

  while((0x80 & LCD_Read()) == 0x80);

  PORTB &= ~(1<<RS);
  PORTB = (0x20>>4);
  PORTB |= (1<<E);
  _delay_us(1);
  PORTB &= ~(1<<E);
  PORTB = 0;

  while((0x80 & LCD_Read()) == 0x80);

  LCD_Write(CMD,0x28);
  LCD_Write(CMD,0x0C);
  LCD_Write(CMD,0x06);
}

void LCD_Write(uint8_t type, uint8_t data) {
  while((0x80 & LCD_Read()) == 0x80);

  PORTB = (type<<RS);
  PORTB |= (1<<E);
  PORTB |= (data>>4);
  PORTB &= ~(1<<E);

  PORTB = (type<<RS);
  PORTB |= (1<<E);
  PORTB |= (data & 0x0F);
  PORTB &= ~(1<<E);

  PORTB = 0;
}

uint8_t LCD_Read(void) {
  uint8_t retval = 0;
  PORTB &= ~(1<<RS);
  PORTB |= (1<<RW);
  DDRB &= ~(1<<D4|1<<D5|1<<D6|1<<D7);

  PORTB |= (1<<E);
  _delay_us(1);
  retval = (PINB & 0x0F)<<4;
  PORTB &= ~(1<<E);
```

```
  PORTB |= (1<<E);
  _delay_us(1);
  retval |= (PINB & 0x0F);
  PORTB &= ~(1<<E);

  DDRB |= (1<<D4|1<<D5|1<<D6|1<<D7);
  PORTB = 0;
  return retval;
    }
```

Дополнительное задание – строка загрузки:




```
#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <stdlib.h>

typedef enum {
  STATE_LOADING,
  STATE_COMPLETE,
  STATE_WAITING
} SystemState;

typedef enum {
  LCD_RS = PB0,
  LCD_EN = PB1,
  LCD_D4 = PB2,
  LCD_D5 = PB3,
  LCD_D6 = PB4,
```

```
  LCD_D7 = PB5
} LcdPins;

#define BUTTON_PIN PD2

const uint8_t PROGRESS_BAR_CHARS[8][8] = {
  {0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00},
  {0x10,0x10,0x10,0x10,0x10,0x10,0x10,0x10},
  {0x18,0x18,0x18,0x18,0x18,0x18,0x18,0x18},
  {0x1C,0x1C,0x1C,0x1C,0x1C,0x1C,0x1C,0x1C},
  {0x1E,0x1E,0x1E,0x1E,0x1E,0x1E,0x1E,0x1E},
  {0x1F,0x1F,0x1F,0x1F,0x1F,0x1F,0x1F,0x1F},
  {0x1F,0x1F,0x1F,0x1F,0x1F,0x1F,0x1F,0x1F},
  {0x1F,0x1F,0x1F,0x1F,0x1F,0x1F,0x1F,0x1F}
};

void lcd_send_nibble(uint8_t data) {
  PORTB = (PORTB & 0xC3) | ((data & 0x0F) << 2);
  PORTB |= (1 << LCD_EN);
  _delay_us(1);
  PORTB &= ~(1 << LCD_EN);
  _delay_us(100);
}

void lcd_send_command(uint8_t cmd) {
  PORTB &= ~(1 << LCD_RS);
  lcd_send_nibble(cmd >> 4);
  lcd_send_nibble(cmd);
  _delay_ms(2);
}

void lcd_send_data(uint8_t data) {
  PORTB |= (1 << LCD_RS);
  lcd_send_nibble(data >> 4);
  lcd_send_nibble(data);
  _delay_us(100);
}

void lcd_set_position(uint8_t row, uint8_t col) {
  uint8_t address = (row == 0) ? 0x00 : 0x40;
  lcd_send_command(0x80 | (address + col));
}

void lcd_print_string(const char *str) {
  while (*str) {
    lcd_send_data(*str++);
  }
}

void lcd_create_custom_char(uint8_t char_id, const uint8_t *pattern) {
```

```
  lcd_send_command(0x40 | (char_id << 3));
  for (uint8_t i = 0; i < 8; i++) {
    lcd_send_data(pattern[i]);
  }
}

void clear_lcd_line(uint8_t line) {
  lcd_set_position(line, 0);
  for (uint8_t i = 0; i < 16; i++) {
    lcd_send_data(' ');
  }
}

uint8_t is_button_pressed() {
  if (!(PIND & (1 << BUTTON_PIN))) {
    _delay_ms(50);
    return !(PIND & (1 << BUTTON_PIN));
  }
  return 0;
}

void lcd_initialize() {
  DDRB |= (1 << LCD_RS) | (1 << LCD_EN) | (1 << LCD_D4) | (1 <<
LCD_D5) | (1 << LCD_D6) | (1 << LCD_D7);
  _delay_ms(50);
  lcd_send_nibble(0x03);
  _delay_ms(5);
  lcd_send_nibble(0x03);
  _delay_us(100);
  lcd_send_nibble(0x03);
  lcd_send_nibble(0x02);
  lcd_send_command(0x28);
  _delay_us(100);
  lcd_send_command(0x0C);
  _delay_us(100);
  lcd_send_command(0x01);
  _delay_ms(2);
  lcd_send_command(0x06);
  _delay_us(100);
  for (uint8_t i = 0; i < 8; i++) {
    lcd_create_custom_char(i, PROGRESS_BAR_CHARS[i]);
  }
}

void update_progress_bar(uint8_t percent) {
  if (percent > 100) percent = 100;
  uint8_t full_cells = (percent * 16) / 100;
  uint8_t partial_fill = (percent * 16 * 8) / 100 % 8;
  lcd_set_position(0, 0);
  for (uint8_t i = 0; i < full_cells; i++) {
```

```
    lcd_send_data(5);
  }
  if (full_cells < 16) {
    lcd_send_data(partial_fill);
  }
  for (uint8_t i = full_cells; i < 16; i++) {
    lcd_send_data(0);
  }
  lcd_set_position(1, 0);
  lcd_print_string("Progress: ");
  lcd_send_data('0' + percent/10);
  lcd_send_data('0' + percent%10);
  lcd_print_string("%");
}

void run_loading_sequence(uint8_t* progress) {
  update_progress_bar(*progress);
  uint16_t delay_time = (rand() % 300) + 100;
  for (uint16_t i = 0; i < delay_time; i++) {
    _delay_ms(1);
    if (is_button_pressed()) {
      *progress = 0;
      clear_lcd_line(1);
      return;
    }
  }
  (*progress)++;
}

void initialize_system() {
  lcd_initialize();
  DDRD &= ~(1 << BUTTON_PIN);
  PORTD |= (1 << BUTTON_PIN);
}

int main(void) {
  initialize_system();
  SystemState state = STATE_LOADING;
  uint8_t progress = 0;
  while (1) {
    switch (state) {
      case STATE_LOADING:
      if (progress <= 100) {
        run_loading_sequence(&progress);
        } else {
        lcd_set_position(1, 0);
        lcd_print_string("Complete!      ");
        _delay_ms(1000);
        state = STATE_COMPLETE;
      }
```

```
        break;
        case STATE_COMPLETE:
        if (is_button_pressed()) {
          progress = 0;
          clear_lcd_line(1);
          state = STATE_LOADING;
        }
        _delay_ms(100);
        break;
        default:
        progress = 0;
        state = STATE_LOADING;
        break;
      }
    }
}
```

Выводы: мы научились работать с ЖКИ WH1602