ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ТЕЛЕКОММУНИКАЦИЙ
И ИНФОРМАТИКИ»

Кафедра вычислительных систем

КУРСОВОЙ ПРОЕКТ

по дисциплине
«Микропроцессорные системы»
на тему

Разработка системы управления любым элементом с компьютера (посредством UART)

Выполнил студент Рудцких Владислав Евгеньевич
Ф.И.О.

Группы ИВ-222

Работу принял _____ Ст. преп. А. С. Коновалов
подпись

Защищена _____ Оценка _____

Новосибирск – 2025

Содержание

ЗАДАНИЕ

Разработка и реализация программно-аппаратного комплекса на базе микроконтроллера ATmega328P для управления внешним устройством (светодиодом) по командам, поступающим через последовательный интерфейс UART, с отображением текущего состояния на LCD-дисплее с интерфейсом I2C.

1. Написание кода на языке C
2. Реализация драйвера UART с кольцевым буфером и обработкой прерываний
3. Разработка драйвера I2C для взаимодействия с LCD-дисплеем
4. Создание парсера команд для интерпретации пользовательского ввода

ОПИСАНИЕ КОМПОНЕНТОВ

Для реализации потребуется:

- Плата Arduino Uno
  Высокопроизводительный 8-битный микроконтроллер AVR® с низким энергопотреблением
  - Усовершенствованная RISC-архитектура
  - 131 мощная инструкция – большинство выполняется за один такт
  - 32 × 8 универсальных регистров
  - Полностью статическая работа
  - Пропускная способность до 16 MIPS при 16 МГц
  - Встроенный 2-тактовый умножитель
  - Высоконадежные энергонезависимые сегменты памяти
  - 32 Кбайт встроенной самопрограммируемой флэш-памяти
  - 1 Кбайт EEPROM
  - 2 Кбайт внутренней SRAM
  - Циклы записи/стирания: 10 000 во флэш-памяти/100 000 в EEPROM
  - Дополнительный раздел загрузочного кода с независимыми битами блокировки
  - Встроенное программирование с помощью встроенной загрузочной программы
  - Истинное чтение во время записи
  - Блокировка программирования для программной безопасности
  - Периферийные устройства Характеристики
  - Два 8-битных таймера/счетчика с отдельным делителем частоты и режимом сравнения
  - Один 16-битный таймер/счетчик с отдельным делителем частоты, режимом сравнения и режимом захвата
  - Счетчик реального времени с отдельным генератором
  - Шесть каналов ШИМ
  - 8-канальный 10-битный АЦП в корпусах TQFP и QFN/MLF
  - Измерение температуры
  - Программируемый последовательный интерфейс USART
  - Последовательный интерфейс SPI "ведущий/ведомый"
  - Байт-ориентированный 2-проводной последовательный интерфейс
  - Программируемый сторожевой таймер с отдельным встроенным генератором
  - Встроенный аналоговый компаратор
  - Прерывание и пробуждение при изменении состояния вывода

- Дисплей lcd1602

  - Тип дисплея: Символьный, на основе жидкокристаллической матрицы (STN)
  - Формат: 2 строки по 16 символов (всего 32 символа)
  - Напряжение питания: 5V постоянного тока
  - Рабочий ток: До 2 мА (без учета подсветки). Полное потребление с подсветкой может достигать 100 мА

  - Контроллер: Стандартный HD44780
  - Подсветка: Светодиодная
  - Знакогенератор: Стандартный набор ASCII
- Светодиод

СХЕМА

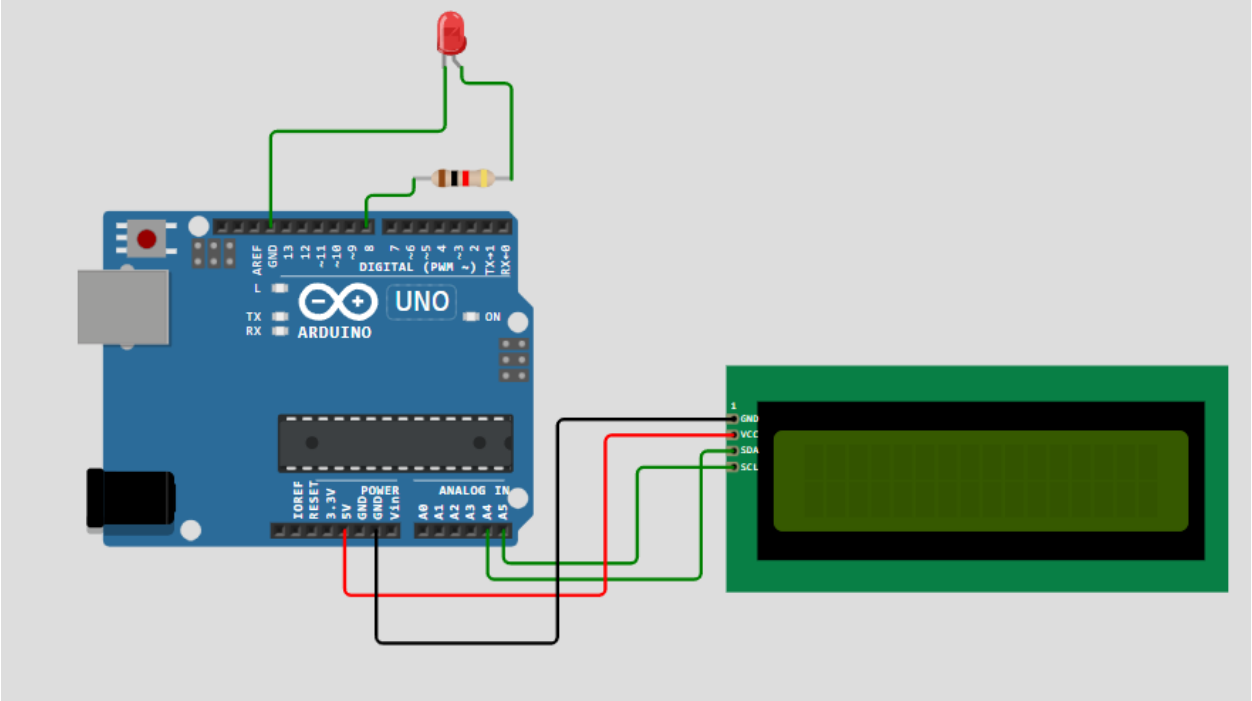

Рисунок. 1 Схема в Wokwi

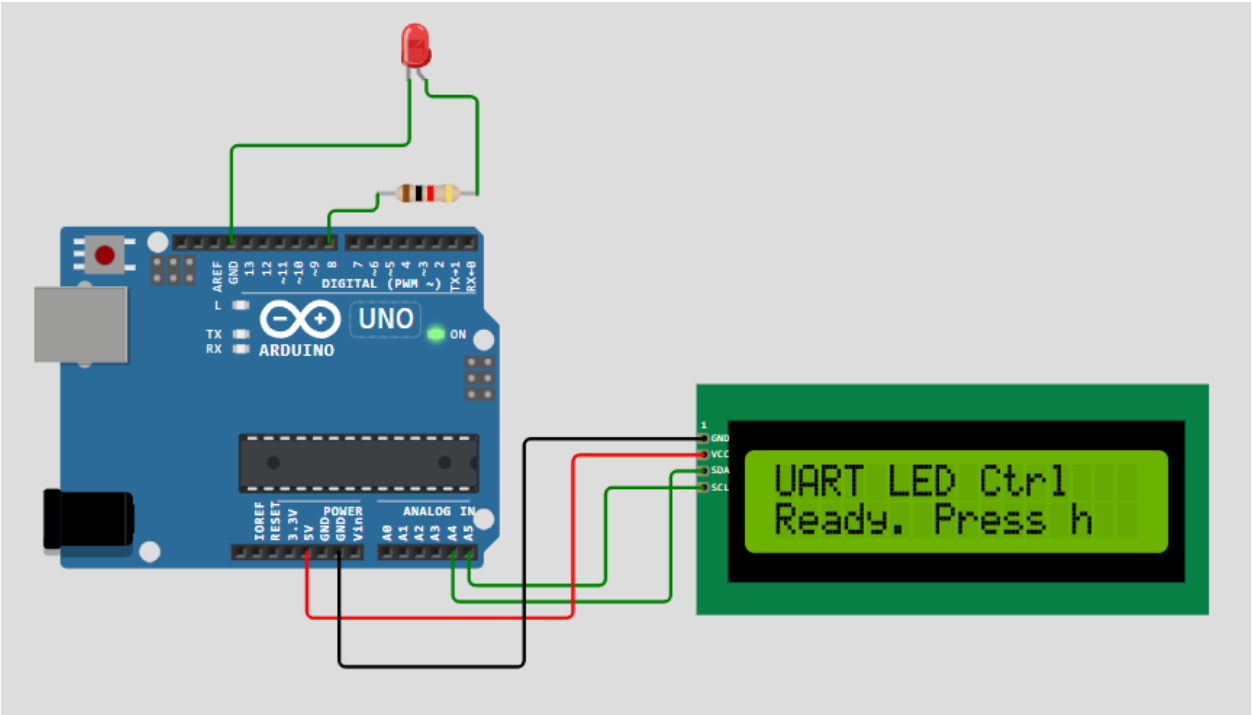

Рисунок. 2 Схема в начальном состоянии

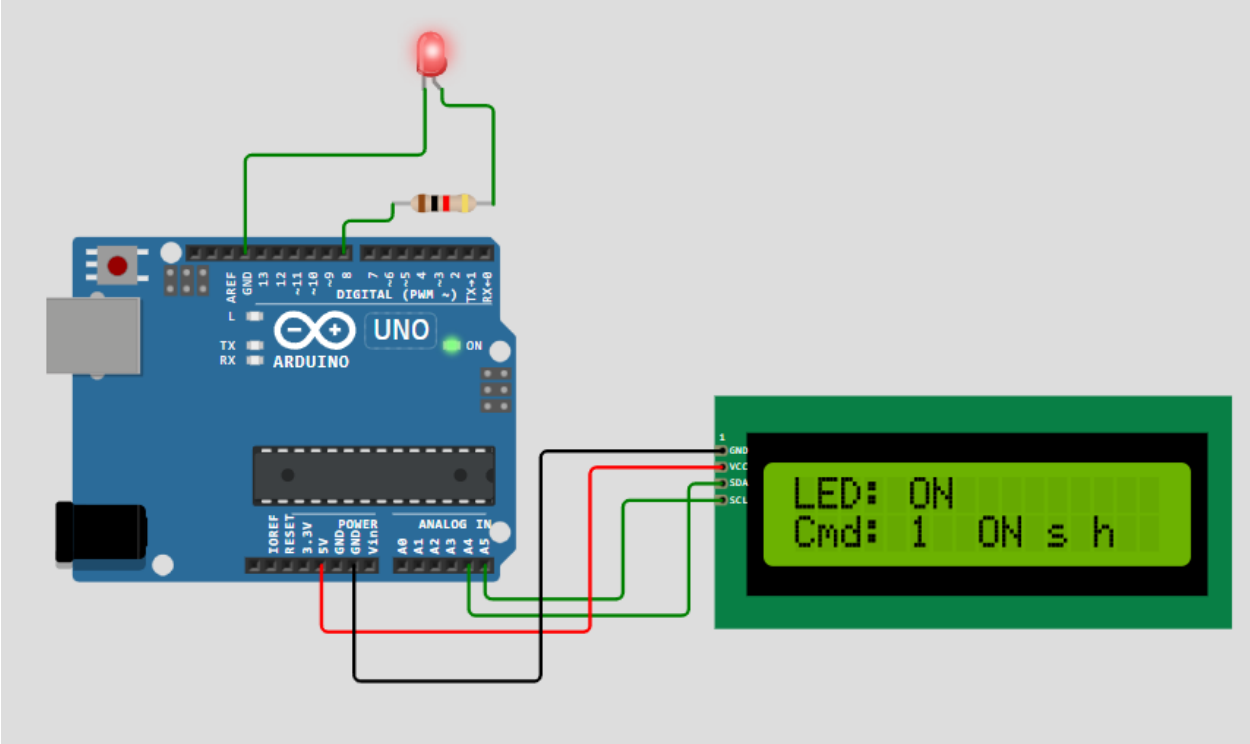

Рисунок. 3 Схема после подачи сигнала включения светодиода

Описание схемы

На рисунках изображена схема подключения Arduino Uno к дисплею по I2C. Порт выхода PC4 (SDA) и PC5 (SDL) подключаются в соответсвующие порты на LCD1602 дисплее. Дисплей запитывается через порт VCC который подключен к подаче 5V на плате. Светодиод подключен к плате через порт PB0 через резистор.

ОПИСАНИЕ АЛГОРИТМА РАБОТЫ

Начальная настройка

Устанавливаем Пин 13 (PB5) как выход (встроенный LED) и Пин 8 (PB0) как выход для внешнего светодиода. UART на 9600 бод и инициализируем i2c

I2C:

I2C – это последовательный асимметричный полудуплексный протокол связи между микросхемами, использующий две линии: SDA (линия данных) и SCL (линия тактового сигнала). Каждое устройство имеет уникальный 7-битный адрес и может быть ведущим (Master) или ведомым (Slave). Линии подключаются к питанию через резисторы, так как выходы работают по принципу «открытого коллектора». Стандартная скорость передачи данных – до 100 кбит/с. Для начала передачи Master переводит SDA из высокого состояния в низкое при высоком уровне SCL (старт-условие), затем отправляет адрес устройства и бит направления (чтение или запись). Если ведомое устройство готово к обмену, оно подтверждает это, устанавливая «0» на SDA. После каждого переданного байта следует бит подтверждения от Slave. Завершается передача стоп-условием, когда Master переводит SDA из низкого состояния в высокое при высоком уровне SCL. Протокол поддерживает многомастерный режим: при одновременной передаче происходит арбитраж, и одно из устройств уступает линию. Если ведомое устройство не успевает обработать данные, оно может удерживать SCL в низком состоянии, заставляя Master ждать. При возникновении ошибки Master прерывает передачу, формирует стоп-условие и повторяет попытку позже или обращается к другому устройству. Если на старт-условие никто не отвечает (нет подтверждения), Master также завершает передачу. Коды состояния позволяют определить причину ошибки.

Начальное состояние

Выключаем внешний LED (set_external_led(0)). На LCD выводится приветствие:
Строка 0: "UART LED Ctrl"; Строка 1: "Ready. Press h"
В UART отправляется ASCII-графика с рамкой и информацией о системе

Обработка UART

Прерывание USART_RX_vect: ISR(USART_RX_vect)
Сохраняем символ в кольцевой буфер.
Игнорируем \r и \n.
Защита от переполнения
Каждый полученный байт (кроме символов перевода строки) сохраняется в буфер

uart_rx_buffer. Используется кольцевой буфер с индексами head/tail и счетчиком uart_rx_count.

Основной цикл обработки

Проверяем на наличие данных:

Если есть, то извлекаем команду из буфера, обновляем индексы и счетчик, мигаем встроенным LED (индикация приема), обрабатываем команду и отправляем новое приглашение "> "

Команды управления

'1' - Включить LED:

Вызывает set_external_led(1);

На LCD строке 0: "LED: ON";

В UART отправляется "[ON]";

На LCD строке 1, позиция 8 обновляется статус "ON";

'0' - Выключить LED:

Вызывает set_external_led(0);

На LCD строке 0: "LED: OFF";

В UART отправляется "[OFF]";

На LCD строке 1, позиция 8 обновляется статус "OFF";

's' или 'S' - Показать статус:

Проверяет external_led_state;

На LCD строке 0, позиция 8 показывает "ON" или "OFF";

В UART отправляет "[Status: ON/OFF]";

't' или 'T' - Переключить LED:

Инвертирует состояние: set_external_led(!external_led_state);

Обновляет LCD;

В UART отправляет "[Toggled]";

'h', 'H' или '?' - Помощь:

Отправляет в UART таблицу с командами:;

Очищает LCD и показывает краткую подсказку;

ТЕСТИРОВАНИЕ СХЕМЫ

Тестирование схемы в Wokwi

Функциональное тестирование:

- Отправка команды 1 для включения светодиода и проверка его состояния на виртуальной плате
- Отправка команды 0 для выключения светодиода
- Отправка команды s для запроса статуса и сверка с фактическим состоянием LED
- Отправка команды t для переключения состояния и проверка изменения

Тестирование UART-интерфейса:

- Подключение к монитору порта в Wokwi и проверка вывода приветственного сообщения
- Проверка корректного отображения приглашения > после каждой команды
- Отправка команд в разных регистрах (заглавные/строчные буквы)

Тестирование LCD-дисплея:

- Проверка инициализации дисплея и вывода стартового сообщения
- Корректное обновление информации при включении/выключении LED
- Отображение команды на второй строке (Cmd: X)
- Очистка дисплея и вывод подсказки при вызове помощи

Стресс-тестирование:

- Многократная отправка команд подряд (проверка буфера UART)
- Быстрое переключение LED (1 0 1 0 с минимальными задержками)
- Отправка некорректных символов (цифры кроме 0/1, буквы кроме s/t/h)
- Отправка команд во время обновления LCD

Тестирование на граничных условиях:

- Заполнение буфера UART большим количеством символов
- Отправка команд без символов новой строки
- Длительная работа программы (проверка на переполнение счетчиков)
- Сброс питания в симуляторе и проверка сохранения начального состояния

Визуальная проверка:

- Мигание встроенного LED при приеме каждой команды
- Корректное отображение рамки и ASCII-графики в мониторе порта
- Работа подсветки LCD (имитация в Wokwi)

ПРИЛОЖЕНИЕ

Листинг main.c

```
#define LCD_ADDR 0x27
#define BAUD_RATE 9600

#define BUILTIN_LED PB5
#define EXTERNAL_LED_PORT PORTB
#define EXTERNAL_LED_DDR DDRB
#define EXTERNAL_LED_PIN PB0

#define UART_BUFFER_SIZE 64
volatile uint8_t uart_rx_buffer[UART_BUFFER_SIZE];
volatile uint8_t uart_rx_head = 0;
volatile uint8_t uart_rx_tail = 0;
volatile uint8_t uart_rx_count = 0;

volatile uint8_t external_led_state = 0;

volatile uint8_t command_processed = 1;

void uart_init(unsigned long baud);
void uart_transmit(uint8_t data);
void uart_send_string(const char* str);
void uart_send_number(uint8_t num);

void i2c_init(void);
void i2c_start(void);
void i2c_stop(void);
void i2c_write(uint8_t data);

void lcd_send_nibble(uint8_t nibble, uint8_t rs);
void lcd_send_byte(uint8_t byte, uint8_t rs);
void lcd_init(void);
void lcd_set_cursor(uint8_t row, uint8_t col);
void lcd_print(const char* str);
void lcd_clear(void);

void set_external_led(uint8_t state);
void process_command(uint8_t cmd);
void uart_send_prompt(void);

void i2c_init(void) {
    TWBR = 72;  // 100 кГц при 16 МГц
    TWSR = 0;
}
```

```c
void i2c_start(void) {
    TWCR = (1 << TWINT) | (1 << TWSTA) | (1 << TWEN);
    while (!(TWCR & (1 << TWINT)));
}

void i2c_stop(void) {
    TWCR = (1 << TWINT) | (1 << TWSTO) | (1 << TWEN);
}

void i2c_write(uint8_t data) {
    TWDR = data;
    TWCR = (1 << TWINT) | (1 << TWEN);
    while (!(TWCR & (1 << TWINT)));
}

void lcd_send_nibble(uint8_t nibble, uint8_t rs) {
    uint8_t data = (nibble << 4) | (rs ? 0x01 : 0x00) | 0x08;
    i2c_start();
    i2c_write(LCD_ADDR << 1);
    i2c_write(data | 0x04);  // EN = 1
    _delay_us(1);
    i2c_write(data & ~0x04);  // EN = 0
    i2c_stop();
    _delay_us(50);
}

void lcd_send_byte(uint8_t byte, uint8_t rs) {
    lcd_send_nibble(byte >> 4, rs);
    lcd_send_nibble(byte & 0x0F, rs);
}

void lcd_init(void) {
    _delay_ms(50);

    lcd_send_nibble(0x03, 0);
    _delay_ms(5);
    lcd_send_nibble(0x03, 0);
    _delay_us(150);
    lcd_send_nibble(0x03, 0);
    _delay_us(150);
    lcd_send_nibble(0x02, 0);  // Переход в 4-бит режим

    lcd_send_byte(0x28, 0);  // 4 бита, 2 строки
    lcd_send_byte(0x0C, 0);  // Включить дисплей, без курсора
    lcd_send_byte(0x06, 0);  // Автоинкремент адреса
    lcd_send_byte(0x01, 0);  // Очистка
    _delay_ms(2);
}
```

```
void lcd_set_cursor(uint8_t row, uint8_t col) {
    uint8_t addr = (row == 0) ? (0x00 + col) : (0x40 + col);
    lcd_send_byte(0x80 | addr, 0);
}

void lcd_print(const char* str) {
    while (*str) {
        lcd_send_byte(*str++, 1);
    }
}

void lcd_clear(void) {
    lcd_send_byte(0x01, 0);
    _delay_ms(2);
}

void set_external_led(uint8_t state) {
    if (state) {
        EXTERNAL_LED_PORT |= (1 << EXTERNAL_LED_PIN);    // Включить
    } else {
        EXTERNAL_LED_PORT &= ~(1 << EXTERNAL_LED_PIN);   // Выключить
    }
    external_led_state = state;

    lcd_set_cursor(1, 8);  // Вторая строка, позиция 8
    if (state) {
        lcd_print("ON ");
    } else {
        lcd_print("OFF");
    }
}

void uart_init(unsigned long baud) {
    unsigned int ubrr = F_CPU / 16 / baud - 1;
    UBRR0H = (unsigned char)(ubrr >> 8);
    UBRR0L = (unsigned char)ubrr;
    UCSR0B = (1 << RXEN0) | (1 << TXEN0) | (1 << RXCIE0);
    UCSR0C = (1 << UCSZ01) | (1 << UCSZ00);
}

void uart_transmit(uint8_t data) {
    while (!(UCSR0A & (1 << UDRE0)));
    UDR0 = data;
}

void uart_send_string(const char* str) {
    while (*str) {
```

```
        uart_transmit(*str++);
    }
}

void uart_send_prompt(void) {
    uart_send_string("\r\n> ");
}

ISR(USART_RX_vect) {
    uint8_t data = UDR0;
    uint8_t next_head = (uart_rx_head + 1) % UART_BUFFER_SIZE;

    if (data == '\r' || data == '\n') {
        return;  // Не сохраняем их в буфер
    }

    if (next_head != uart_rx_tail) {
        uart_rx_buffer[uart_rx_head] = data;
        uart_rx_head = next_head;
        uart_rx_count++;
    }
}

void process_command(uint8_t cmd) {
    lcd_set_cursor(1, 0);
    lcd_print("Cmd: ");
    lcd_send_byte(cmd, 1);
    lcd_print("   ");

    uart_transmit(cmd);

    switch(cmd) {
        case '1':
            set_external_led(1);
            lcd_set_cursor(0, 0);
            lcd_print("LED: ON         ");
            uart_send_string(" [ON]");
            break;

        case '0':
            set_external_led(0);
            lcd_set_cursor(0, 0);
            lcd_print("LED: OFF        ");
            uart_send_string(" [OFF]");
            break;

        case 's':
```

```
        case 'S':
            lcd_set_cursor(0, 0);
            lcd_print("Status:         ");
            lcd_set_cursor(0, 8);
            if (external_led_state) {
                lcd_print("ON ");
                uart_send_string(" [Status: ON]");
            } else {
                lcd_print("OFF");
                uart_send_string(" [Status: OFF]");
            }
            break;

        case 't':
        case 'T':
            set_external_led(!external_led_state);
            lcd_set_cursor(0, 0);
            if (external_led_state) {
                lcd_print("LED: ON         ");
            } else {
                lcd_print("LED: OFF        ");
            }
            uart_send_string(" [Toggled]");
            break;

        case 'h':
        case 'H':
        case '?':
            uart_send_string("\r\n");
            uart_send_string("╔═══════════════════════════╗\r\n");
            uart_send_string("║        HELP MENU          ║\r\n");
            uart_send_string("╠═══════════════════════════╣\r\n");
            uart_send_string("║ 1 - LED ON                ║\r\n");
            uart_send_string("║ 0 - LED OFF               ║\r\n");
            uart_send_string("║ s - Show Status           ║\r\n");
            uart_send_string("║ t - Toggle LED            ║\r\n");
            uart_send_string("║ h/? - This help           ║\r\n");
            uart_send_string("╚═══════════════════════════╝\r\n");

            lcd_clear();
            lcd_set_cursor(0, 0);
            lcd_print("HELP: 1/0/s/t");
            lcd_set_cursor(1, 0);
            lcd_print("h for menu");
            break;

        default:
            // Игнорируем все остальные символы (они уже отфильтрованы в
```

```
прерывании)
                break;
            }
        }

        int main(void) {
            DDRB |= (1 << BUILTIN_LED);      // Встроенный LED как выход
            EXTERNAL_LED_DDR |= (1 << EXTERNAL_LED_PIN);  // Внешний LED как выход

            uart_init(9600);
            i2c_init();
            lcd_init();

            set_external_led(0);

            sei();

            lcd_clear();
            lcd_set_cursor(0, 0);
            lcd_print("UART LED Ctrl");
            lcd_set_cursor(1, 0);
            lcd_print("Ready. Press h");

            uart_send_string("\r\n");
            uart_send_string("╔═══════════════════════╗\r\n");
            uart_send_string("║  UART LED Control     ║\r\n");
            uart_send_string("║  External LED on pin 8║\r\n");
            uart_send_string("║  LCD via I2C          ║\r\n");
            uart_send_string("╚═══════════════════════╝\r\n");
            uart_send_prompt();

            while (1) {
                if (uart_rx_count > 0) {
                    uint8_t cmd = uart_rx_buffer[uart_rx_tail];
                    uart_rx_tail = (uart_rx_tail + 1) % UART_BUFFER_SIZE;

                    cli();
                    uart_rx_count--;
                    sei();

                    PORTB |= (1 << BUILTIN_LED);
                    _delay_ms(30);
                    PORTB &= ~(1 << BUILTIN_LED);

                    process_command(cmd);

                    uart_send_prompt();
```

```
            }

            _delay_ms(10);
        }

        return 0;
    }
```

ИНСТРУКЦИЯ ПО СБОРКЕ

1. Установить плату Arduino Uno
2. Подключить I2C устройства к портам A5 и A4 к портам устройств SCL и SDA соответственно
3. Подключить светодиод к порту 8
4. Скомпилировать и загрузить на плату main.c