Практическое занятие №12
Тема: «Программирование интерфейса с помощью энкодера»

Цель работы: приобрести практические навыки по подключению и программированию энкодеров и ЖК-дисплеев на платформе Arduino.

Последовательность выполнения работы:

– Собрать схемы на макетной плате, иначе при отсутствии набора Arduino в web-приложениях (https://wokwi.com/projects/new/arduino-uno или https://www.tinkercad.com/) для приведенных примеров.

– Запрограммировать микроконтроллер согласно заданию в примере.

– Выполнить задание для самостоятельной работы.

Содержание отчета:

– Название практического занятия, его цель.

– Фото или скриншоты собранной схемы.

– Написанный программный код вставить текстом, Courier New, 12 кегль, одинарный отступ без абзацев.

– Вывод о проделанной работе.

– Файл Fritzing с принципиальной и монтажной схемой.

ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

Энкодер (энкодер поворота, rotary encoder) – это электромеханическое устройство, преобразующее механическое вращение в цифровые сигналы. Это аналог потенциометра, но с важными отличиями.

Основные типы энкодеров

1. Инкрементальный энкодер (Incremental Encoder)

Определяет только изменение положения, имеет два выхода (A и B) с квадратурными сигналами, не запоминает абсолютное положение после отключения питания.

2. Абсолютный энкодер (Absolute Encoder)

Определяет абсолютное положение, имеет уникальный код для каждого положения, запоминает положение после отключения питаниядороже и сложнее в подключении,

Как работает энкодер:

При вращении вала диск с прорезями вращается, ИК-светодиод светит через прорези, фототранзисторы улавливают прерывистый свет и тем самым генерируются импульсы на выходах S1 и S2

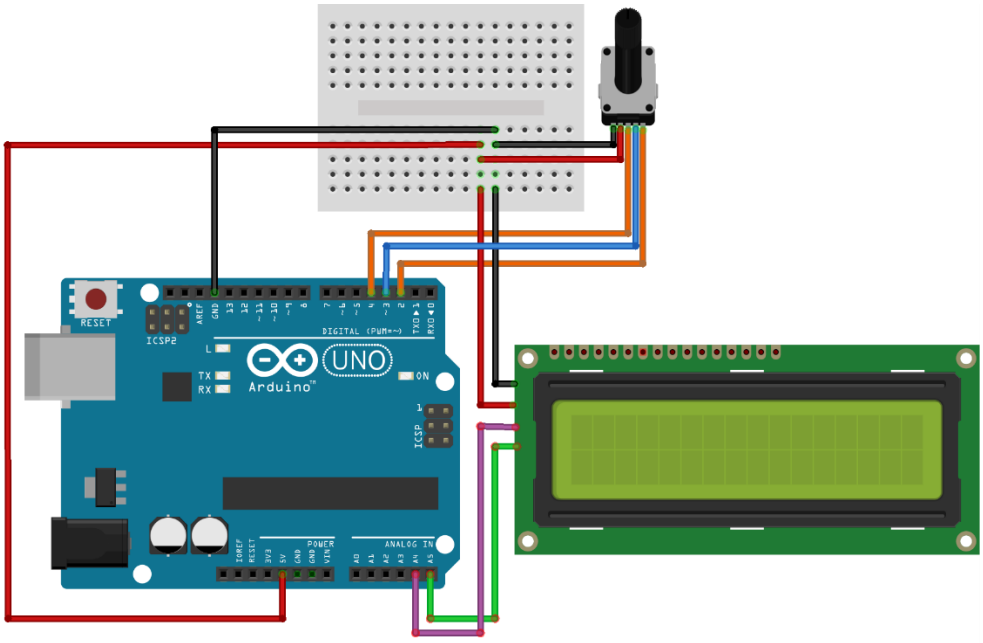
Рисунок 1 – Внешний вид Энкодеров

Назначение выводов:

Вывод	Назначение	Описание
5V	питание	напряжение питания 3.3V-5V
GND	земля	общий провод
S1	канал A (CLK)	первый квадратурный сигнал
S2	канал B (DT)	второй квадратурный сигнал
KEY	кнопка (SW)	тактовая кнопка

ЗАДАНИЯ

Собрать схему:



Написать программный код:

```
// ========== ПОДКЛЮЧЕНИЕ БИБЛИОТЕК ==========
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// ========== НАСТРОЙКИ ПИНОВ ==========
#define ENCODER_CLK 2      // CLK энкодера (прерывание 0)
#define ENCODER_DT 3       // DT энкодера
#define ENCODER_SW 4       // Кнопка энкодера
#define LED_PIN 11         // Светодиод индикации

// ========== НАСТРОЙКА LCD ==========
LiquidCrystal_I2C lcd(0x27, 16, 2); // Адрес 0x27, 16x2 дисплей

// ========== ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ ==========
// Режимы работы системы
enum SystemMode {
  MODE_MAIN,       // Главный экран
  MODE_MENU,       // Меню настроек
  MODE_SETTINGS    // Настройка параметра
};
SystemMode currentMode = MODE_MAIN;

// Параметры настроек
struct Settings {
  int brightness;    // Яркость (0-100)
  int speed;         // Скорость (1-10)
  int temperature;   // Температура (0-100)
  int timeout;       // Таймаут (1-60 сек)
  int beep;          // Звук (0-1)
};

Settings settings = {50, 5, 25, 30, 1}; // Значения по умолчанию

// Меню
const char* menuItems[] = {
  "Brightness",
  "Speed",
  "Temperature",
  "Timeout",
  "Sound"
```

```
};
const int MENU_COUNT = 5;
int currentMenuItem = 0;
int currentSetting = 0;

// Переменные энкодера
volatile int lastCLK = HIGH;
volatile int encoderChange = 0; // -1 вниз, 0 нет, 1 вверх
int lastEncoderChange = 0;

// Переменные кнопки энкодера
bool lastButtonState = HIGH;
bool buttonState = HIGH;
unsigned long lastDebounceTime = 0;
unsigned long buttonPressTime = 0;
const unsigned long debounceDelay = 50;
const unsigned long longPressDelay = 1000; // Длинное
нажатие 1 сек

// Флаги
bool needDisplayUpdate = true;
bool ledState = false;
unsigned long lastBlinkTime = 0;
unsigned long lastActivityTime = 0;
const unsigned long timeoutDelay = 30000; // 30 сек
автовыход

// ========== НАСТРОЙКА ==========
void setup() {
  Serial.begin(9600);

  // Настройка пинов энкодера
  pinMode(ENCODER_CLK, INPUT_PULLUP);
  pinMode(ENCODER_DT, INPUT_PULLUP);
  pinMode(ENCODER_SW, INPUT_PULLUP);
  pinMode(LED_PIN, OUTPUT);

  // Инициализация LCD
  lcd.init();
  lcd.backlight();
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Encoder Menu");
  lcd.setCursor(0, 1);
```

```
  lcd.print("v1.0");

  delay(1500);

  // Настройка прерывания для энкодера
  lastCLK = digitalRead(ENCODER_CLK);
  attachInterrupt(digitalPinToInterrupt(ENCODER_CLK),
encoderISR, CHANGE);

  lcd.clear();
  needDisplayUpdate = true;
  lastActivityTime = millis();
}

// ========== ГЛАВНЫЙ ЦИКЛ ==========
void loop() {
  unsigned long currentMillis = millis();

  // Проверка таймаута бездействия
  if (currentMode != MODE_MAIN &&
      currentMillis - lastActivityTime > timeoutDelay) {
    currentMode = MODE_MAIN;
    needDisplayUpdate = true;
    digitalWrite(LED_PIN, LOW);
  }

  // Обработка энкодера (вращение)
  handleEncoder();

  // Обработка кнопки энкодера
  handleEncoderButton();

  // Обновление дисплея если нужно
  if (needDisplayUpdate) {
    updateDisplay();
    needDisplayUpdate = false;
  }

  // Мигание светодиодом в главном режиме
  if (currentMode == MODE_MAIN) {
    if (currentMillis - lastBlinkTime >= 500) {
      ledState = !ledState;
      digitalWrite(LED_PIN, ledState);
      lastBlinkTime = currentMillis;
```

```
    }
  } else {
    // В других режимах светодиод горит постоянно
    digitalWrite(LED_PIN, HIGH);
  }

  delay(10); // Небольшая задержка для стабильности
}

// ========== ОБРАБОТКА ЭНКОДЕРА (ВРАЩЕНИЕ) ==========
void encoderISR() {
  int clkState = digitalRead(ENCODER_CLK);
  int dtState = digitalRead(ENCODER_DT);

  if (clkState != lastCLK) {
    // Определяем направление вращения
    if (dtState != clkState) {
      encoderChange = 1; // По часовой стрелке
    } else {
      encoderChange = -1; // Против часовой стрелки
    }
    lastCLK = clkState;
    lastActivityTime = millis(); // Активность пользователя
  }
}

void handleEncoder() {
  if (encoderChange != lastEncoderChange) {
    if (currentMode == MODE_MAIN) {
      // В главном режиме - переключение между режимами
работы
      currentSetting += encoderChange;
      if (currentSetting < 0) currentSetting = MENU_COUNT -
1;
      if (currentSetting >= MENU_COUNT) currentSetting = 0;
      needDisplayUpdate = true;
    }
    else if (currentMode == MODE_MENU) {
      // В режиме меню - переключение пунктов
      currentMenuItem += encoderChange;
      if (currentMenuItem < 0) currentMenuItem = MENU_COUNT
- 1;
      if (currentMenuItem >= MENU_COUNT) currentMenuItem =
0;
```

```
      needDisplayUpdate = true;
    }
    else if (currentMode == MODE_SETTINGS) {
      // В режиме настройки - изменение значения
      adjustSetting(currentMenuItem, encoderChange);
      needDisplayUpdate = true;
    }

    lastEncoderChange = encoderChange;
  }
  encoderChange = 0; // Сбрасываем флаг
}

// ========== ОБРАБОТКА КНОПКИ ЭНКОДЕРА ==========
void handleEncoderButton() {
  // Чтение состояния кнопки с защитой от дребезга
  int reading = digitalRead(ENCODER_SW);

  if (reading != lastButtonState) {
    lastDebounceTime = millis();
  }

  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != buttonState) {
      buttonState = reading;

      if (buttonState == LOW) {
        // Кнопка нажата - запоминаем время начала нажатия
        buttonPressTime = millis();
      } else {
        // Кнопка отпущена - обрабатываем как короткое
нажатие
        if (buttonPressTime > 0 && (millis() -
buttonPressTime) < longPressDelay) {
          processShortPress();
          buttonPressTime = 0;
        }
      }
    }
  }

  lastButtonState = reading;

  // Проверка на длинное нажатие (кнопка все еще нажата)
```

```
  if (buttonPressTime > 0 && (millis() - buttonPressTime) >=
longPressDelay) {
    // Длинное нажатие - выход в главное меню
    currentMode = MODE_MAIN;
    needDisplayUpdate = true;
    buttonPressTime = 0; // Сбрасываем время нажатия
    lastActivityTime = millis();

    // Небольшая задержка для предотвращения повторной
обработки
    delay(300);
  }
}

void processShortPress() {
  switch(currentMode) {
    case MODE_MAIN:
      // Из главного в меню
      currentMode = MODE_MENU;
      currentMenuItem = 0;
      break;

    case MODE_MENU:
      // Из меню в настройки
      currentMode = MODE_SETTINGS;
      break;

    case MODE_SETTINGS:
      // Из настроек обратно в меню
      currentMode = MODE_MENU;
      break;
  }
  needDisplayUpdate = true;
  lastActivityTime = millis();
}

// ========== НАСТРОЙКА ПАРАМЕТРОВ ==========
void adjustSetting(int settingIndex, int direction) {
  switch(settingIndex) {
    case 0: // Яркость
      settings.brightness += direction;
      if (settings.brightness < 0) settings.brightness = 0;
      if (settings.brightness > 100) settings.brightness =
100;
```

```
      break;

    case 1: // Скорость
      settings.speed += direction;
      if (settings.speed < 1) settings.speed = 1;
      if (settings.speed > 10) settings.speed = 10;
      break;

    case 2: // Температура
      settings.temperature += direction;
      if (settings.temperature < 0) settings.temperature =
0;
      if (settings.temperature > 100) settings.temperature =
100;
      break;

    case 3: // Таймаут
      settings.timeout += direction;
      if (settings.timeout < 1) settings.timeout = 1;
      if (settings.timeout > 60) settings.timeout = 60;
      break;

    case 4: // Звук
      settings.beep = !settings.beep;
      break;
  }
  lastActivityTime = millis();
}

String getSettingValue(int settingIndex) {
  switch(settingIndex) {
    case 0: return String(settings.brightness) + "%";
    case 1: return String(settings.speed) + "/10";
    case 2: return String(settings.temperature) + "C";
    case 3: return String(settings.timeout) + "s";
    case 4: return settings.beep ? "On" : "Off";
    default: return "---";
  }
}

String getSettingValueForMain(int settingIndex) {
  switch(settingIndex) {
    case 0: return String(settings.brightness) + "%";
    case 1: return String(settings.speed);
```

```
    case 2: return String(settings.temperature) + "C";
    case 3: return String(settings.timeout) + "s";
    case 4: return settings.beep ? "Yes" : "No";
    default: return "---";
  }
}

// ========== ОБНОВЛЕНИЕ ДИСПЛЕЯ ==========
void updateDisplay() {
  lcd.clear();

  switch(currentMode) {
    case MODE_MAIN:
      displayMainScreen();
      break;

    case MODE_MENU:
      displayMenu();
      break;

    case MODE_SETTINGS:
      displaySettings();
      break;
  }
}

void displayMainScreen() {
  lcd.setCursor(0, 0);
  lcd.print(">> ");
  lcd.print(menuItems[currentSetting]);

  // Заполняем оставшееся пространство пробелами
  int spaces = 13 - strlen(menuItems[currentSetting]);
  for (int i = 0; i < spaces; i++) {
    lcd.print(" ");
  }

  // Индикация режима
  lcd.setCursor(14, 0);
  lcd.print("M");

  lcd.setCursor(0, 1);
  lcd.print("Value: ");
  lcd.print(getSettingValueForMain(currentSetting));
```

```
  // Подсказка в правом нижнем углу
  lcd.setCursor(12, 1);
  lcd.print("OK>");
}

void displayMenu() {
  lcd.setCursor(0, 0);
  lcd.print("Settings Menu:");

  lcd.setCursor(0, 1);
  lcd.print(">");
  lcd.print(menuItems[currentMenuItem]);

  // Заполняем оставшееся пространство
  int spaces = 15 - strlen(menuItems[currentMenuItem]);
  for (int i = 0; i < spaces; i++) {
    lcd.print(" ");
  }

  // Подсказка в правом верхнем углу
  lcd.setCursor(14, 0);
  lcd.print("->");
}

void displaySettings() {
  lcd.setCursor(0, 0);
  lcd.print("Setting:");
  lcd.setCursor(9, 0);
  lcd.print(menuItems[currentMenuItem]);

  lcd.setCursor(0, 1);

  // Отображение значения с индикацией
  String value = getSettingValue(currentMenuItem);

  // Для логических значений (On/Off) показываем по-другому
  if (currentMenuItem == 4) {
    lcd.print("State: ");
    lcd.print(value);
  } else {
    // Для числовых значений - шкала прогресса
    lcd.print("[");
```

```
    int maxWidth = 10; // Длина шкалы
    int valueToShow;

    switch(currentMenuItem) {
      case 0: // Яркость 0-100
        valueToShow = map(settings.brightness, 0, 100, 0,
maxWidth);
        break;
      case 1: // Скорость 1-10
        valueToShow = map(settings.speed, 1, 10, 0,
maxWidth);
        break;
      case 2: // Температура 0-100
        valueToShow = map(settings.temperature, 0, 100, 0,
maxWidth);
        break;
      case 3: // Таймаут 1-60
        valueToShow = map(settings.timeout, 1, 60, 0,
maxWidth);
        break;
      default:
        valueToShow = 0;
    }

    // Рисуем шкалу
    for (int i = 0; i < maxWidth; i++) {
      if (i < valueToShow) {
        lcd.print("=");
      } else {
        lcd.print(" ");
      }
    }

    lcd.print("] ");
    lcd.print(value);
  }
}
```

```
// ========== МОНИТОР ==========
void printStatus() {
  Serial.println("\n=== System Status ===");
  Serial.print("Mode: ");
  switch(currentMode) {
    case MODE_MAIN: Serial.println("Main"); break;
    case MODE_MENU: Serial.println("Menu"); break;
    case MODE_SETTINGS: Serial.println("Settings"); break;
  }

  Serial.print("Current item: ");
  Serial.println(menuItems[currentMenuItem]);

  Serial.println("Parameters:");
  Serial.print("  Brightness: ");
Serial.print(settings.brightness); Serial.println("%");
  Serial.print("  Speed: "); Serial.print(settings.speed);
Serial.println("/10");
  Serial.print("  Temperature: ");
Serial.print(settings.temperature); Serial.println("C");
  Serial.print("  Timeout: ");
Serial.print(settings.timeout); Serial.println("sec");
  Serial.print("  Sound: "); Serial.println(settings.beep ?
"On" : "Off");
  Serial.print("  Inactivity time: ");
  Serial.print((millis() - lastActivityTime) / 1000);
  Serial.println("sec");
  Serial.println("====================\n");
}
```