Практическое занятие №
Тема: «Программирование интерфейсов с помощью энкодера»

Цель работы: приобрести практические навыки по подключению и программированию энкодеров 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

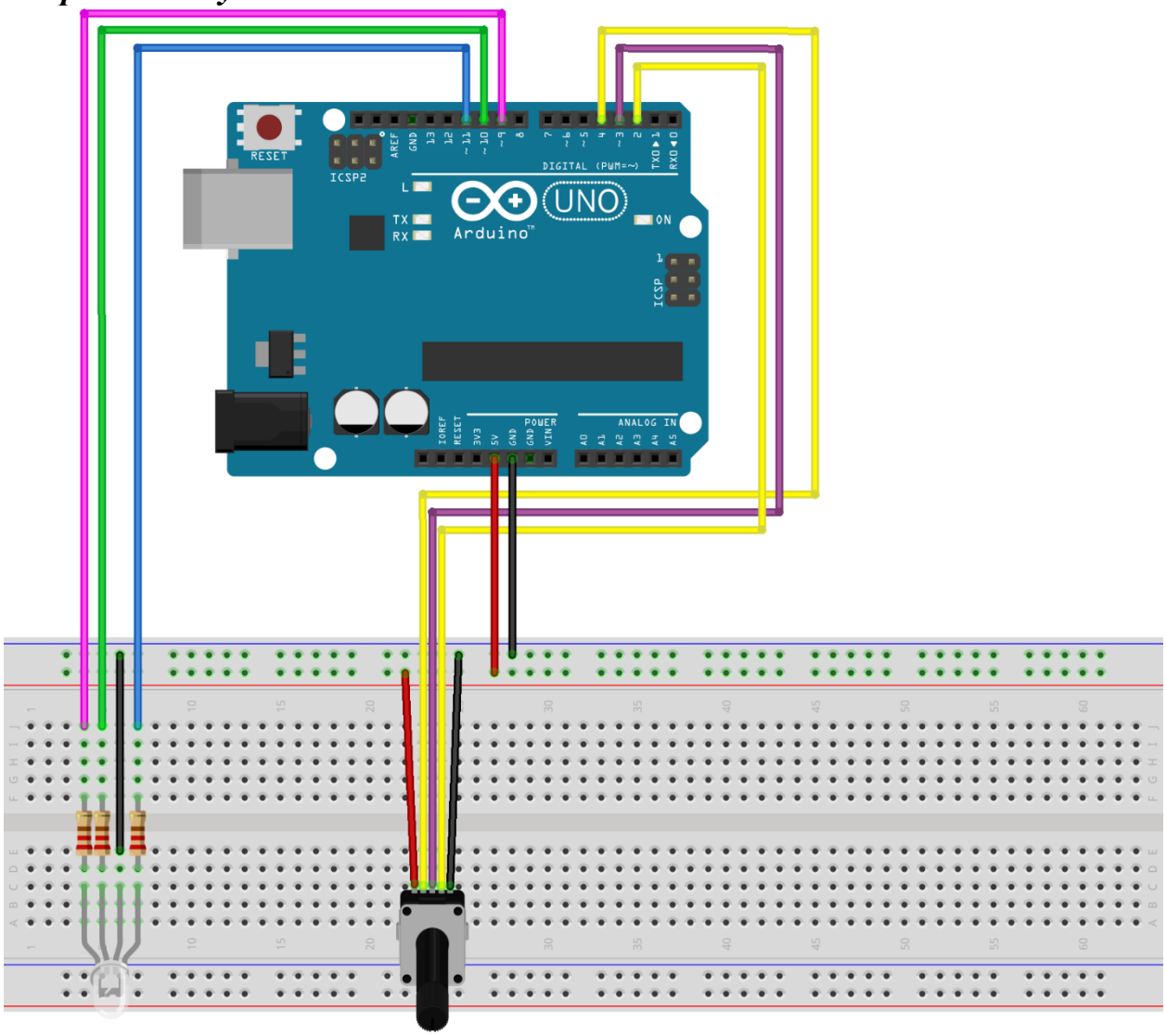
Рисунок 1 – Внешний вид Энкодеров

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:

В данной программе энкодер используется в качестве инструмента, который регулирует ШИМ-сигнал (0-255) для настройки яркости RGB-светодиода, переключаясь между цветами «вперёд» и «назад» коротким и длинным нажатием кнопки энкодера. Двойное нажатие кнопки энкодера переключит между режимами «статический» и «анимация».

```
// ========== НАСТРОЙКИ ПИНОВ ==========
#define ENCODER_CLK 2       // S1 энкодера
#define ENCODER_DT 3        // S2 энкодера
#define ENCODER_SW 4        // KEY энкодера

#define RED_PIN 9           // PWM пин для красного
#define GREEN_PIN 10        // PWM пин для зеленого
#define BLUE_PIN 11         // PWM пин для синего

// ========== ГЛОБАЛЬНЫЕ ПЕРЕМЕННЫЕ ==========
// Переменные энкодера
volatile int encoderValue = 0;
volatile int lastCLK = HIGH;
int lastEncoderValue = -1;

// Переменные RGB
int redValue = 0;
int greenValue = 0;
int blueValue = 0;

// Режимы работы
enum ColorMode {
  MODE_RED,
  MODE_GREEN,
  MODE_BLUE
};
ColorMode currentMode = MODE_RED;

// Состояние анимации
bool animationMode = false;
unsigned long lastAnimationTime = 0;
const unsigned long animationDelay = 50;
int animationPhase = 0;

// Переменные кнопки
bool lastButtonState = HIGH;
bool buttonState = HIGH;
unsigned long lastDebounceTime = 0;
unsigned long buttonPressTime = 0;
const unsigned long debounceDelay = 50;
const unsigned long longPressDelay = 1000;
unsigned long lastClickTime = 0;
const unsigned long doubleClickDelay = 300;
int clickCount = 0;
```

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

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

  // Настройка пинов RGB
  pinMode(RED_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(BLUE_PIN, OUTPUT);

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

  // Начальные значения
  updateRGB();
}

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

  // Обработка кнопки
  handleButton();

  // Обработка энкодера
  handleEncoder();

  // Анимация переливания
  if (animationMode) {
    if (currentMillis - lastAnimationTime >= animationDelay)
{
      rainbowAnimation();
      lastAnimationTime = currentMillis;
    }
  }

  printRGB();
```

```
  delay(10);
}

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

  if (clkState != lastCLK) {
    if (dtState != clkState) {
      encoderValue += 5; // Увеличиваем шаг для быстрой
регулировки
    } else {
      encoderValue -= 5;
    }
    lastCLK = clkState;
  }
}

void handleEncoder() {
  if (encoderValue != lastEncoderValue && !animationMode) {
    int change = encoderValue - lastEncoderValue;

    // Обновляем значение текущего цвета
    switch(currentMode) {
      case MODE_RED:
        redValue = constrain(redValue + change, 0, 255);
        break;
      case MODE_GREEN:
        greenValue = constrain(greenValue + change, 0, 255);
        break;
      case MODE_BLUE:
        blueValue = constrain(blueValue + change, 0, 255);
        break;
    }

    updateRGB();
    lastEncoderValue = encoderValue;
  }
}

// ========== ОБРАБОТКА КНОПКИ ==========
void handleButton() {
  int reading = digitalRead(ENCODER_SW);
```

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

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

      if (buttonState == LOW) {
        buttonPressTime = millis();
        if (millis() - lastClickTime > doubleClickDelay) {
          clickCount = 0;
        }
      } else {
        if (buttonPressTime > 0) {
          if ((millis() - buttonPressTime) < longPressDelay)
{
            clickCount++;
            lastClickTime = millis();

            // Обработка двойного нажатия
            if (clickCount == 2) {
              doubleClick();
              clickCount = 0;
            }
          }
          buttonPressTime = 0;
        }
      }
    }
  }

  // Проверка длинного нажатия
  if (buttonPressTime > 0 && (millis() - buttonPressTime) >=
longPressDelay) {
    longPress();
    buttonPressTime = 0;
    clickCount = 0;
    delay(300);
  }

  // Обработка одиночного клика (если не было двойного)
```

```
  if (clickCount == 1 && (millis() - lastClickTime) >
doubleClickDelay) {
    singleClick();
    clickCount = 0;
  }

  lastButtonState = reading;
}

void singleClick() {
  // Переход к следующему цвету
  if (!animationMode) {
    currentMode = (ColorMode)((currentMode + 1) % 3);
  }
}

void doubleClick() {
  // Переключение режима анимации
  animationMode = !animationMode;
}

void longPress() {
  // Переход к предыдущему цвету
  if (!animationMode) {
    currentMode = (ColorMode)((currentMode - 1 + 3) % 3);
  }
}

// ========== RGB ФУНКЦИИ ==========
void updateRGB() {
  analogWrite(RED_PIN, redValue);
  analogWrite(GREEN_PIN, greenValue);
  analogWrite(BLUE_PIN, blueValue);
}

void rainbowAnimation() {
  // Анимация радуги
  animationPhase = (animationPhase + 1) % 1536;

  if (animationPhase < 256) {
    // Красный -> Желтый
    redValue = 255;
    greenValue = animationPhase;
    blueValue = 0;
```

```
  } else if (animationPhase < 512) {
    // Желтый -> Зеленый
    redValue = 511 - animationPhase;
    greenValue = 255;
    blueValue = 0;
  } else if (animationPhase < 768) {
    // Зеленый -> Голубой
    redValue = 0;
    greenValue = 255;
    blueValue = animationPhase - 512;
  } else if (animationPhase < 1024) {
    // Голубой -> Синий
    redValue = 0;
    greenValue = 1023 - animationPhase;
    blueValue = 255;
  } else if (animationPhase < 1280) {
    // Синий -> Пурпурный
    redValue = animationPhase - 1024;
    greenValue = 0;
    blueValue = 255;
  } else {
    // Пурпурный -> Красный
    redValue = 255;
    greenValue = 0;
    blueValue = 1535 - animationPhase;
  }

  updateRGB();
}

// ========== СЕРИАЛЬНЫЙ ВЫВОД ==========
void printRGB() {
  Serial.print("RGB: (");
  Serial.print(redValue);
  Serial.print(", ");
  Serial.print(greenValue);
  Serial.print(", ");
  Serial.print(blueValue);
  Serial.print(") | Mode: ");
  switch(currentMode) {
    case MODE_RED: Serial.print("RED"); break;
    case MODE_GREEN: Serial.print("GREEN"); break;
    case MODE_BLUE: Serial.print("BLUE"); break;
  }
```

```
  Serial.print(" | Animation: ");
  Serial.println(animationMode ? "ON" : "OFF");
}
```