Практическое занятие №
Тема: «Программирование системы “Стиральная машина”»

Цель работы: приобрести практические навыки по подключению и программированию энкодеров, индикации, формируя законченную систему 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

4-разрядный 7-сегментный индикатор

4-разрядный 7-сегментный индикатор — это электронное устройство, представляющее собой дисплей с четырьмя отдельными 7-сегментными индикаторами. Каждый индикатор может отображать цифры от 0 до 9, а также некоторые буквы.

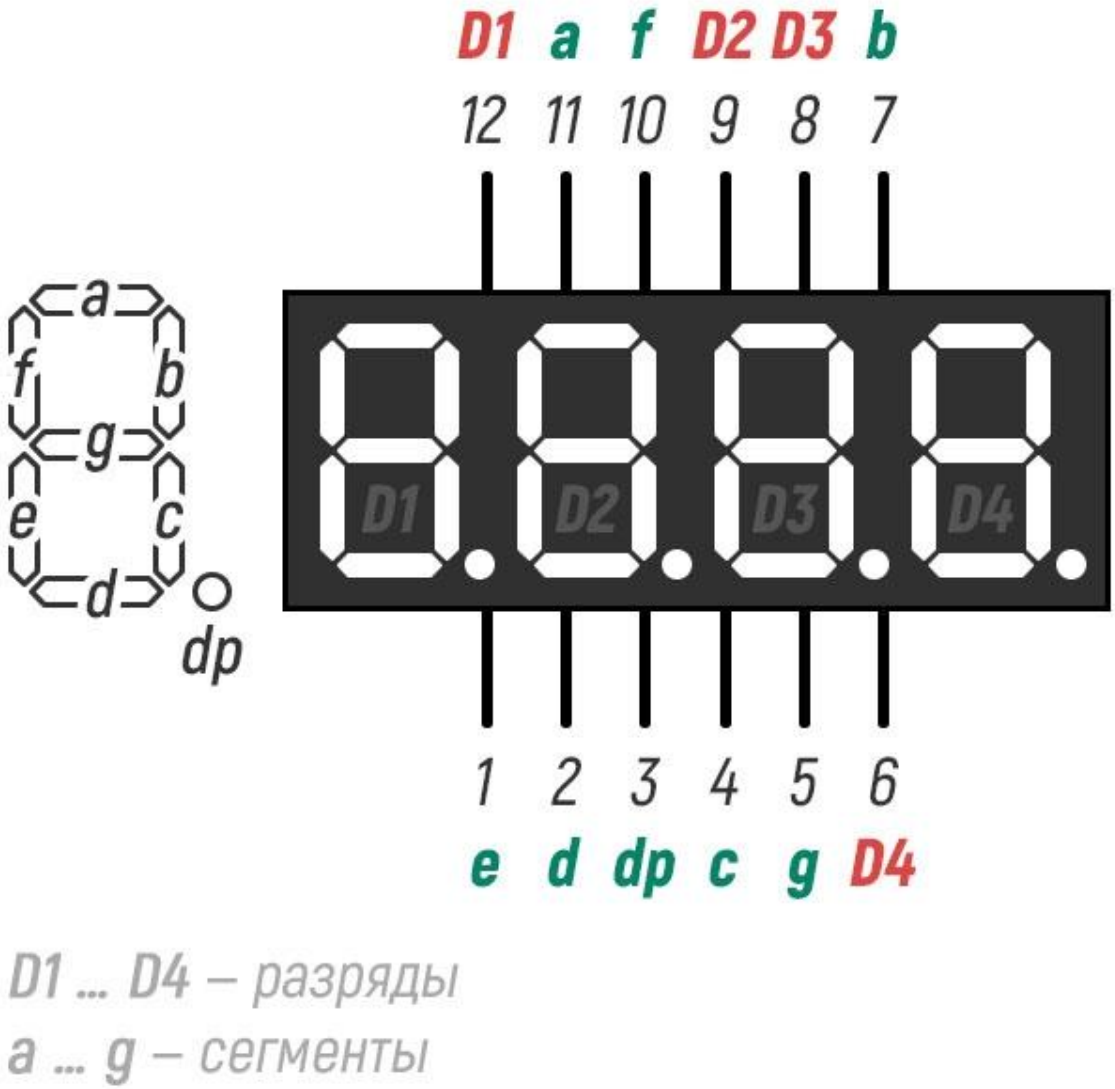


Рисунок 1 – 4 разрядный 7-сегментный цифровой LED индикатор

Восьмиразрядный сдвиговый регистр 74HC595N

74HC595N – восьмиразрядный сдвиговый регистр с последовательным вводом, последовательным или параллельным выводом информации, с триггером-защелкой и тремя состояниями на выходе. Самое распространенное применение данного регистра – экономия выходов микроконтроллера. Данный сдвиговый регистр позволяет управлять напряжением на своих восьми выходах, заняв всего три выхода микроконтроллера. Таким образом количество рабочих выводов увеличивается на пять.

Кроме того, регистры 74HC595 можно подключать каскадом один за другим (через пин **Q7'**), и таким образом из всё тех же 3 входящих линий получать 16, 24, 32 и т.д. цифровых выходов.

74HC595N имеет следующие входы:

– **[10] MR** — сброс регистра, при подаче логического нуля на MR и единицы на STCP переводит все выходы в состояние логического нуля;

– **[11] SH_CP** — вход для тактовых импульсов;

– **[12] ST_CP** — линия прерываний;

– **[13] OE** — вход, переводящий выходы из высокоимпедансного состояния в рабочее;

– **[14] DS** — вход данных;

– **[8] GND** — Ground. Земля

– **[16] VCC** — Питание +5 В.

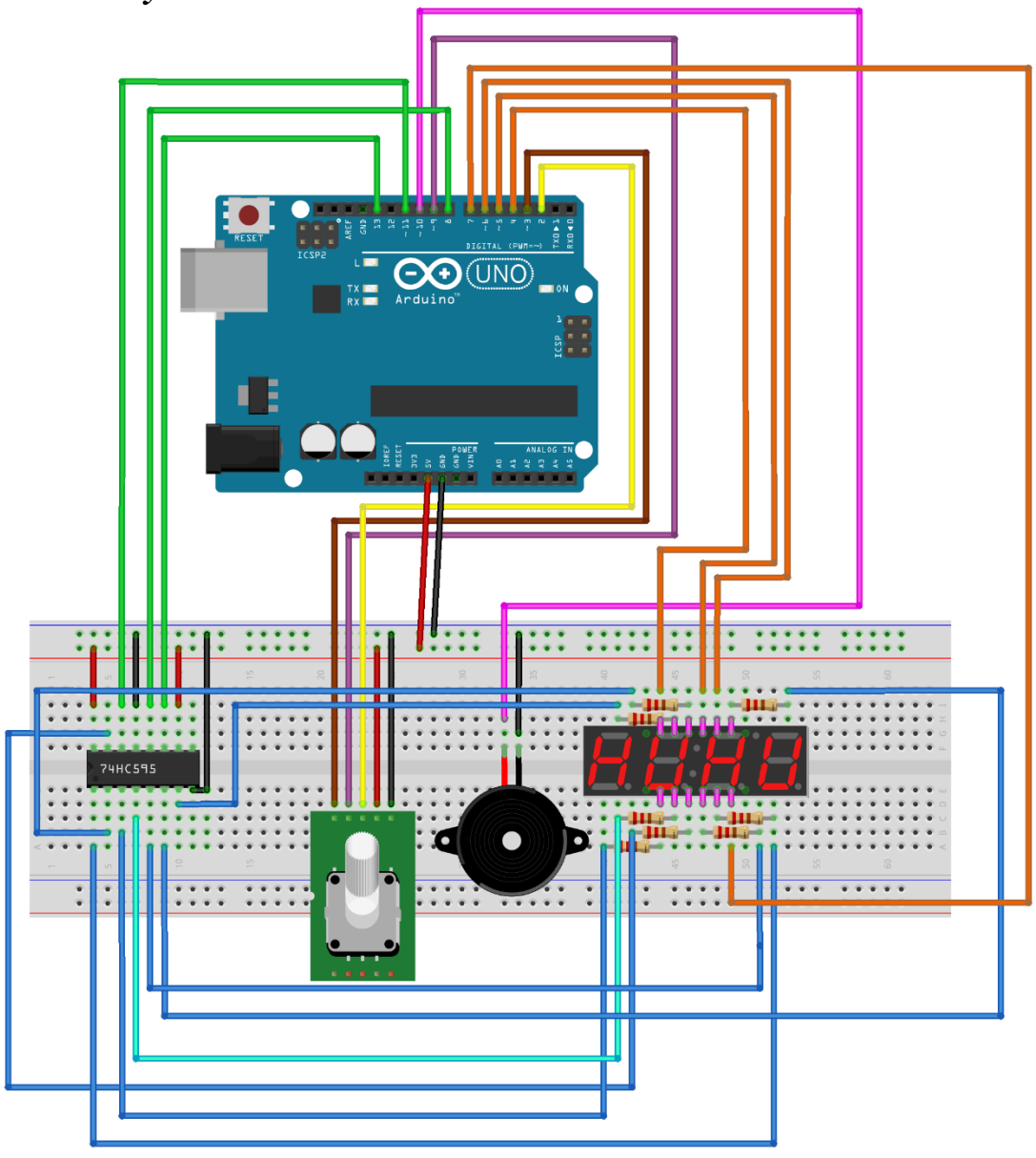
Рисунок 3 – Восьмиразрядный сдвиговый регистр

ПРЕИМУЩЕСТВА И НЕДОСТАТКИ

Преимущества:
Очень низкая стоимость.
Простота подключения и управления.
Четкий, яркий луч, видимый на больших расстояниях.
Наличие встроенной защиты (резистор, транзистор).
Недостатки:
Опасность для зрения. Главный и критический недостаток.
Активный низкий уровень (может быть неочевидно для новичков).
Невозможность фокусировки или изменения формы луча без дополнительных оптических элементов.
Мощность может варьироваться от модуля к модулю.

ЗАДАНИЯ

Собрать схему:



Написать программный код:

В данной программе модуль ky-008 используется в качестве постонного источника воздействия на фоторезистор, в случае если луч будет прерван сработает сигнализация. Отключить её можно при нажатии на кнопку.

```
#include <SPI.h>

// Пин для latch (SS)
const int pin_spi_ss = 8;

// Пины энкодера
const int pin_encoder_CLK = 2;    // S1 (CLK)
const int pin_encoder_DT = 3;     // S2 (DT)
const int pin_encoder_SW = 9;     // Кнопка (Key)
const int pin_buzzer = 10;        // Пин для зуммера

// Пины для управления разрядами индикатора (катоды)
int pindigits[4] = {4, 5, 6, 7};

// Таблица символов для 7-сегментного индикатора (с точкой)
// ABCDEFGDp (точка - старший бит, 1 - сегмент выключен, 0 - включен)
byte symbols[16] = {
  B00000011, // 0 - a,b,c,d,e,f
  B10011111, // 1 - b,c
  B00100101, // 2 - a,b,d,e,g
  B00001101, // 3 - a,b,c,d,g
  B10011001, // 4 - b,c,f,g
  B01001001, // 5 - a,c,d,f,g
  B01000001, // 6 - a,c,d,e,f,g
  B00011111, // 7 - a,b,c
  B00000001, // 8 - a,b,c,d,e,f,g
  B00001001, // 9 - a,b,c,d,f,g
  B01100011, // c (Цельсий) - a,d,e,f
  B11111111, // Пусто - все сегменты выключены
  B11000101, // o (маленькая) - c,d,e,g
  B01110001, // f (маленькая) - a,e,f,g
  B11010101  // n (для "0n") - b,c,e,g
};
```

```
// Определение индексов символов
#define SYM_0 0
#define SYM_1 1
#define SYM_2 2
#define SYM_3 3
#define SYM_4 4
#define SYM_5 5
#define SYM_6 6
#define SYM_7 7
#define SYM_8 8
#define SYM_9 9
#define SYM_C 10
#define SYM_EMPTY 11
#define SYM_o 12
#define SYM_f 13
#define SYM_n 14

// Режимы стиральной машины
String modes[6] = {"Хлопок", "Синтетика", "Шерсть",
"Деликатная", "Интенсивная", "Быстрая"};
int modeTimes[6] = {5, 2, 3, 6, 10, 1}; // Время в минуты

// Температуры
int temperatures[6] = {10, 20, 30, 40, 50, 60};

// Скорости отжима
int spinSpeeds[6] = {0, 300, 500, 800, 1000, 1200};

// Состояния машины
enum MachineState {
  STATE_INIT,
  STATE_MODE_SELECT,
  STATE_TEMP_SELECT,
  STATE_SPIN_SELECT,
  STATE_WASHING,
  STATE_FINISHED,
  STATE_IDLE
};

// Глобальные переменные
volatile int encoderPos = 0;
int lastEncoderPos = 0;
int lastEncoded = 0;
unsigned long lastEncoderTime = 0;
```

```
const int encoderDebounce = 5;

int currentMode = 0;
int currentTemp = 0;  // По умолчанию 10 градусов
int currentSpin = 2;  // По умолчанию 500 об/мин
MachineState currentState = STATE_INIT;
unsigned long buttonPressTime = 0;
bool buttonPressed = false;
bool buttonLongPressed = false;
unsigned long washingStartTime = 0;
unsigned long washingDuration = 0;
unsigned long lastDisplayUpdate = 0;
unsigned long lastBeep = 0;
unsigned long finishTime = 0;
bool finishNotification = false;
unsigned long lastFinishNotification = 0;
const unsigned long FINISH_NOTIFICATION_INTERVAL = 60000; //
1 минута

// Для динамической индикации
unsigned long lastDigitUpdate = 0;
int currentDigit = 0;

// Для анимации и мигания
unsigned long lastBlinkTime = 0;
bool blinkState = false;
int blinkCounter = 0;
bool showDot = false;
unsigned long lastDotBlink = 0;

// Флаги анимации включения
bool initAnimation = false;
int initStep = 0;
unsigned long initStepTime = 0;

// Для точки на дисплее
bool dotEnabled = true; // Точка постоянно включена при
стирке

void setup() {
  Serial.begin(9600);
  SPI.begin();

  // Настройка пина latch как выхода
```

```
  pinMode(pin_spi_ss, OUTPUT);
  digitalWrite(pin_spi_ss, HIGH);

  // Настройка пинов разрядов как выходов (катодный
индикатор)
  for (int i = 0; i < 4; i++) {
    pinMode(pindigits[i], OUTPUT);
    digitalWrite(pindigits[i], LOW); // Разряды выключены
  }

  // Настройка пинов энкодера
  pinMode(pin_encoder_CLK, INPUT_PULLUP);
  pinMode(pin_encoder_DT, INPUT_PULLUP);
  pinMode(pin_encoder_SW, INPUT_PULLUP);

  // Настройка пина зуммера
  pinMode(pin_buzzer, OUTPUT);
  digitalWrite(pin_buzzer, LOW);

  // Настройка прерываний для энкодера
  attachInterrupt(digitalPinToInterrupt(pin_encoder_CLK),
updateEncoder, CHANGE);
  attachInterrupt(digitalPinToInterrupt(pin_encoder_DT),
updateEncoder, CHANGE);

  Serial.println("====================================");
  Serial.println("     Стиральная машина v2.0");
  Serial.println("====================================");

  // Запуск анимации включения
  currentState = STATE_INIT;
  initAnimation = true;
  initStep = 0;
  initStepTime = millis();
  beep(100, 800);
}

void loop() {
  // Обработка кнопки энкодера
  handleButton();

  // Обработка вращения энкодера
  handleEncoder();
```

```
  // Обработка анимации включения
  if (initAnimation) {
    handleInitAnimation();
    updateDisplay(); // Обновляем дисплей во время анимации
    return; // Пока идет анимация, не выполняем основную
логику
  }

  // Основная логика состояний
  switch(currentState) {
    case STATE_INIT:
      // После анимации переходим к выбору режима
      currentState = STATE_MODE_SELECT;
      break;

    case STATE_MODE_SELECT:
      handleModeSelection();
      break;

    case STATE_TEMP_SELECT:
      handleTempSelection();
      break;

    case STATE_SPIN_SELECT:
      handleSpinSelection();
      break;

    case STATE_WASHING:
      handleWashing();
      break;

    case STATE_FINISHED:
      handleFinished();
      break;

    case STATE_IDLE:
      handleIdle();
      break;
  }

  // Обновление дисплея
  updateDisplay();

  // Проверка долгого нажатия для запуска стирки
```

```
  checkLongPress();
}

// Обработчик прерывания для энкодера
void updateEncoder() {
  int MSB = digitalRead(pin_encoder_CLK);
  int LSB = digitalRead(pin_encoder_DT);

  int encoded = (MSB << 1) | LSB;
  int sum = (lastEncoded << 2) | encoded;

  if(sum == 0b1101 || sum == 0b0100 || sum == 0b0010 || sum
== 0b1011) {
    encoderPos++;
  }
  if(sum == 0b1110 || sum == 0b0111 || sum == 0b0001 || sum
== 0b1000) {
    encoderPos--;
  }

  lastEncoded = encoded;
}

// Обработка вращения энкодера
void handleEncoder() {
  if (encoderPos != lastEncoderPos) {
    if (millis() - lastEncoderTime > encoderDebounce) {
      int direction = encoderPos > lastEncoderPos ? 1 : -1;

      switch(currentState) {
        case STATE_MODE_SELECT:
          currentMode += direction;
          if (currentMode < 0) currentMode = 5;
          if (currentMode > 5) currentMode = 0;
          beep(50, 300 + abs(direction) * 100);
          Serial.print("Режим: ");
          Serial.println(modes[currentMode]);
          break;

        case STATE_TEMP_SELECT:
          currentTemp += direction;
          if (currentTemp < 0) currentTemp = 5;
          if (currentTemp > 5) currentTemp = 0;
          beep(50, 300 + abs(direction) * 100);
```

```
          Serial.print("Температура: ");
          Serial.print(temperatures[currentTemp]);
          Serial.println("°C");
          break;

        case STATE_SPIN_SELECT:
          currentSpin += direction;
          if (currentSpin < 0) currentSpin = 5;
          if (currentSpin > 5) currentSpin = 0;
          beep(50, 300 + abs(direction) * 100);
          Serial.print("Отжим: ");
          Serial.print(spinSpeeds[currentSpin]);
          Serial.println(" об/мин");
          break;
      }

      lastEncoderPos = encoderPos;
      lastEncoderTime = millis();
    }
  }
}

// Обработка кнопки
void handleButton() {
  int buttonState = digitalRead(pin_encoder_SW);

  if (buttonState == LOW && !buttonPressed) {
    buttonPressTime = millis();
    buttonPressed = true;
    buttonLongPressed = false;
  }

  if (buttonState == HIGH && buttonPressed) {
    buttonPressed = false;
    unsigned long pressDuration = millis() -
buttonPressTime;

    if (pressDuration < 3000 && !buttonLongPressed) {
      // Короткое нажатие
      beep(100, 800);

      switch(currentState) {
        case STATE_MODE_SELECT:
          currentState = STATE_TEMP_SELECT;
```

```
            Serial.println("-> Выбор температуры");
            break;

          case STATE_TEMP_SELECT:
            currentState = STATE_SPIN_SELECT;
            Serial.println("-> Выбор отжима");
            break;

          case STATE_SPIN_SELECT:
            currentState = STATE_MODE_SELECT;
            Serial.println("-> Возврат к выбору режима");
            break;

          case STATE_FINISHED:
            // Нажатие после завершения стирки
            currentState = STATE_MODE_SELECT;
            finishNotification = false;
            Serial.println("-> Возврат к выбору режима");
            break;

          case STATE_IDLE:
            // Пробуждение из режима ожидания
            currentState = STATE_MODE_SELECT;
            Serial.println("-> Включение");
            break;
        }
      }
    }
  }

  // Проверка долгого нажатия
  void checkLongPress() {
    if (buttonPressed && !buttonLongPressed) {
      if (millis() - buttonPressTime > 3000) {
        buttonLongPressed = true;

        if (currentState == STATE_MODE_SELECT ||
            currentState == STATE_TEMP_SELECT ||
            currentState == STATE_SPIN_SELECT) {

          startWashing();
        }
      }
    }
```

```
}

// Запуск стирки
void startWashing() {
  currentState = STATE_WASHING;
  washingStartTime = millis();
  washingDuration = modeTimes[currentMode] * 60000UL; //
Конвертируем минуты в миллисекунды

  Serial.println("====================================");
  Serial.println("       ЗАПУСК СТИРКИ!");
  Serial.println("====================================");
  Serial.print("Режим: ");
  Serial.println(modes[currentMode]);
  Serial.print("Температура: ");
  Serial.print(temperatures[currentTemp]);
  Serial.println("°C");
  Serial.print("Отжим: ");
  Serial.print(spinSpeeds[currentSpin]);
  Serial.println(" об/мин");
  Serial.print("Время стирки: ");
  Serial.print(modeTimes[currentMode]);
  Serial.println(" минут");
  Serial.println("====================================");

  // Мелодия запуска
  playStartMelody();
}

// Обработка стирки
void handleWashing() {
  unsigned long elapsed = millis() - washingStartTime;
  unsigned long remaining = washingDuration - elapsed;

  if (remaining <= 0) {
    currentState = STATE_FINISHED;
    finishTime = millis();
    playFinishMelody();
    Serial.println("====================================");
    Serial.println("     СТИРКА ЗАВЕРШЕНА!");
    Serial.println("====================================");
    return;
  }
}
```

```
// Обработка завершения
void handleFinished() {
  // Если прошла минута без нажатия кнопки - подаем сигнал
  if (!finishNotification && millis() - finishTime >
FINISH_NOTIFICATION_INTERVAL) {
    beep(200, 1500);
    finishNotification = true;
    lastFinishNotification = millis();
  }

  // Если сигнал уже был и прошла еще минута - повторяем
  if (finishNotification && millis() -
lastFinishNotification > FINISH_NOTIFICATION_INTERVAL) {
    beep(200, 1500);
    lastFinishNotification = millis();
  }

  // Мигание "_0ff" каждые 500мс
  if (millis() - lastBlinkTime > 500) {
    blinkState = !blinkState;
    lastBlinkTime = millis();
  }
}

// Обработка режима ожидания
void handleIdle() {
  // Мигание "_0ff" каждые 500мс
  if (millis() - lastBlinkTime > 500) {
    blinkState = !blinkState;
    lastBlinkTime = millis();
  }
}

// Выбор режима
void handleModeSelection() {
  // Ничего не делаем, всё обрабатывается в handleEncoder()
}

// Выбор температуры
void handleTempSelection() {
  // Ничего не делаем, всё обрабатывается в handleEncoder()
}
```

```
// Выбор отжима
void handleSpinSelection() {
  // Ничего не делаем, всё обрабатывается в handleEncoder()
}

// Обработка анимации включения
void handleInitAnimation() {
  unsigned long currentTime = millis();

  switch(initStep) {
    case 0:
      // Показываем "__0n" и моргаем 3 раза
      if (currentTime - initStepTime > 500) {
        initStep++;
        initStepTime = currentTime;
        blinkState = true;
        blinkCounter = 0;
      }
      break;

    case 1:
      if (currentTime - initStepTime > 500) {
        blinkState = !blinkState;
        initStepTime = currentTime;
        blinkCounter++;

        if (blinkCounter >= 6) { // 3 полных цикла мигания (вкл-выкл)
          initStep++;
          initStepTime = currentTime;
          beep(100, 600);
        }
      }
      break;

    case 2:
      // Завершение анимации
      if (currentTime - initStepTime > 500) {
        initAnimation = false;
        currentState = STATE_MODE_SELECT;
        Serial.println("-> Выбор режима");
      }
      break;
  }
```

```
}

// Обновление дисплея
void updateDisplay() {
  static unsigned long lastTime = 0;
  if (millis() - lastTime < 3) return; // Задержка между
обновлениями разрядов
  lastTime = millis();

  // Выключаем все разряды
  for (int i = 0; i < 4; i++) {
    digitalWrite(pindigits[i], LOW);
  }

  // Получаем символ для текущего разряда
  byte symbolToShow = getSymbolForDigit(currentDigit);

  // Включаем текущий разряд
  digitalWrite(pindigits[currentDigit], HIGH);

  // Выводим символ на SPI
  digitalWrite(pin_spi_ss, LOW);
  SPI.transfer(symbolToShow);
  digitalWrite(pin_spi_ss, HIGH);

  // Переходим к следующему разряду
  currentDigit = (currentDigit + 1) % 4;
}

// Получение символа для разряда
byte getSymbolForDigit(int digit) {
  // Анимация включения
  if (initAnimation) {
    if (!blinkState && initStep == 1) {
      return symbols[SYM_EMPTY];
    }

    // Показываем "__0n" (0 на 3-м разряде, n на 4-м
разряде)
    if (digit == 0 || digit == 1) return symbols[SYM_EMPTY];
// Первые два разряда пустые
    if (digit == 2) return symbols[SYM_0]; // 0 на третьем
разряде
```

```
    if (digit == 3) return symbols[SYM_n]; // n на четвертом
разряде
  }

  switch(currentState) {
    case STATE_MODE_SELECT:
      // Формат: "__01" (пробелы, 0, номер режима)
      if (digit == 0 || digit == 1) return
symbols[SYM_EMPTY]; // Первые два разряда пустые
      if (digit == 2) return symbols[SYM_0]; // Всегда 0 на
третьем разряде
      if (digit == 3) return symbols[currentMode + 1]; //
Номер режима (1-6) на четвертом разряде
      break;

    case STATE_TEMP_SELECT:
      {
        int temp = temperatures[currentTemp];
        // Формат: "_10C" (пробел, 1, 0, C)
        if (temp < 100) { // Температуры до 99 градусов
          if (digit == 0) return symbols[SYM_EMPTY]; //
Пробел
          if (digit == 1) return symbols[temp / 10]; //
Десятки
          if (digit == 2) return symbols[temp % 10]; //
Единицы
          if (digit == 3) return symbols[SYM_C]; // Символ C
        }
      }
      break;

    case STATE_SPIN_SELECT:
      {
        int speed = spinSpeeds[currentSpin];
        // Отжим от 0 до 1200 - занимаем все 4 разряда
        if (digit == 0) {
          // Тысячи (для 1000-1200)
          return symbols[speed / 1000];
        }
        if (digit == 1) {
          // Сотни
          return symbols[(speed % 1000) / 100];
        }
        if (digit == 2) {
```

```
            // Десятки
            return symbols[(speed % 100) / 10];
          }
          if (digit == 3) {
            // Единицы
            return symbols[speed % 10];
          }
        }
        break;

      case STATE_WASHING:
        {
          unsigned long elapsed = millis() - washingStartTime;
          unsigned long remaining = washingDuration - elapsed;

          int totalSeconds = remaining / 1000;
          int minutes = totalSeconds / 60;
          int seconds = totalSeconds % 60;

          // Формат: "MM.SS" (минуты.секунды)
          if (digit == 0) return symbols[minutes / 10]; //
Десятки минут
          if (digit == 1) {
            // Единицы минут с точкой (точка постоянно
включена)
            byte symbol = symbols[minutes % 10];
            symbol &= ~B10000000; // Включаем точку
(сбрасываем старший бит)
            return symbol;
          }
          if (digit == 2) return symbols[seconds / 10]; //
Десятки секунд
          if (digit == 3) return symbols[seconds % 10]; //
Единицы секунд
        }
        break;

      case STATE_FINISHED:
        if (!blinkState) return symbols[SYM_EMPTY];

        // Формат: "_0ff" (пробел, 0, f, f)
        if (digit == 0) return symbols[SYM_EMPTY]; // Пробел
        if (digit == 1) return symbols[SYM_0];     // 0
        if (digit == 2) return symbols[SYM_f];     // f
```

```
      if (digit == 3) return symbols[SYM_f];     // f
      break;

    case STATE_IDLE:
      if (!blinkState) return symbols[SYM_EMPTY];

      // Формат: "_0ff" (пробел, 0, f, f)
      if (digit == 0) return symbols[SYM_EMPTY]; // Пробел
      if (digit == 1) return symbols[SYM_0];     // 0
      if (digit == 2) return symbols[SYM_f];     // f
      if (digit == 3) return symbols[SYM_f];     // f
      break;
  }

  return symbols[SYM_EMPTY];
}

// Звуковой сигнал
void beep(int duration, int frequency) {
  if (frequency <= 0) return;

  tone(pin_buzzer, frequency, duration);
}

// Мелодия запуска
void playStartMelody() {
  beep(200, 523);  // C5
  delay(100);
  beep(200, 659);  // E5
  delay(100);
  beep(300, 784);  // G5
  delay(200);
  beep(500, 1046); // C6
  delay(100);
  noTone(pin_buzzer);
}

// Мелодия завершения
void playFinishMelody() {
  beep(300, 1046); // C6
  delay(100);
  beep(300, 784);  // G5
  delay(100);
  beep(300, 659);  // E5
```

```
  delay(100);
  beep(500, 523);  // C5
  delay(200);
  beep(200, 659);  // E5
  delay(50);
  beep(200, 784);  // G5
  delay(50);
  beep(400, 1046); // C6
  delay(100);
  noTone(pin_buzzer);
}
```