Практическое занятие №
Тема: «Создание игры с индикационным дисплеем и кнопочной матрицей»

Цель работы: приобрести практические навыки по подключению и программированию 7-сегментного индикатора и сдвигового регистра 74HC595N, кнопочной матрицы 4 на 4 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

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.

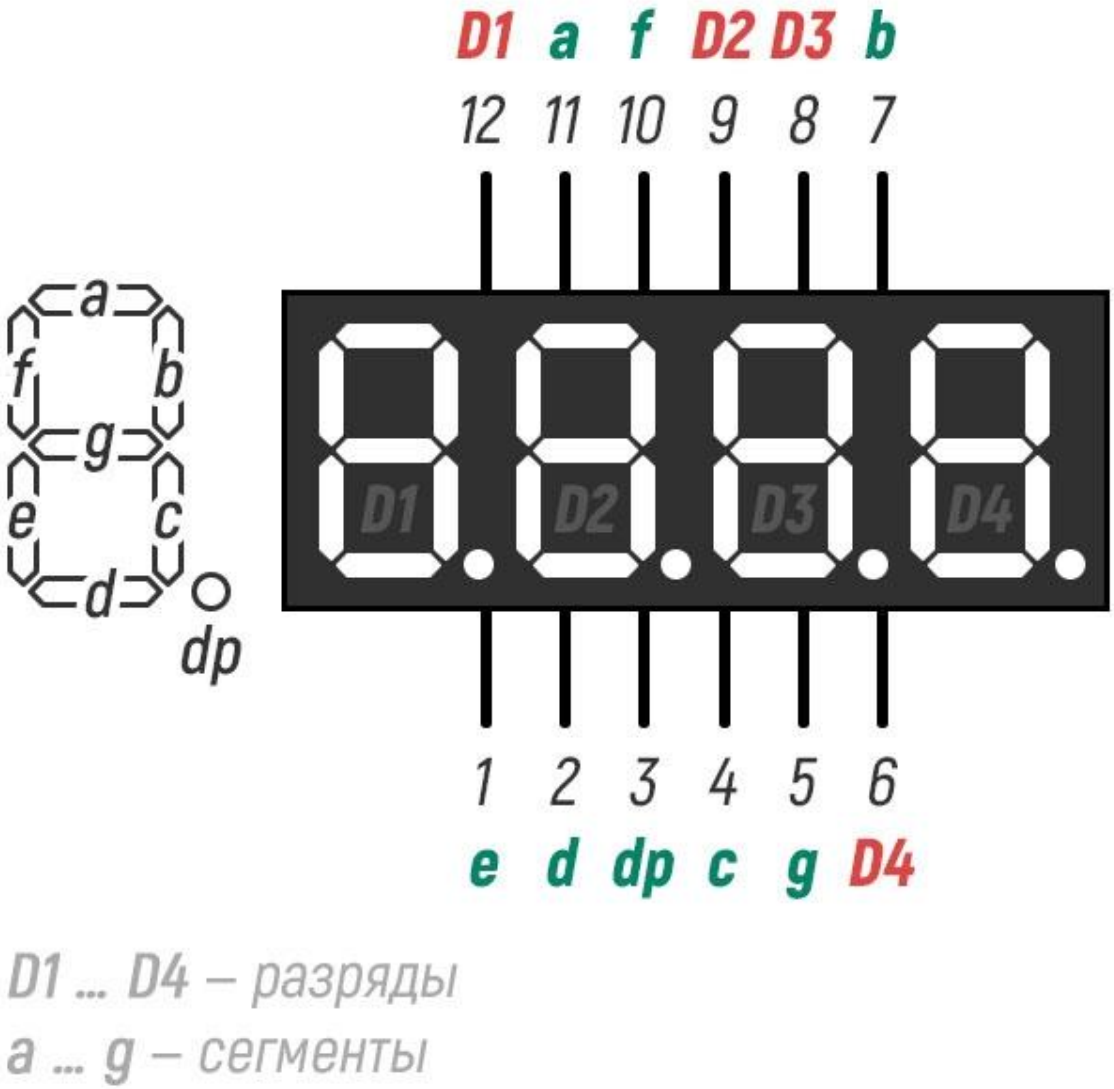


Рисунок 1 – 4 разрядный 7-сегментный цифровой LED индикатор

Восьмиразрядный сдвиговый регистр

74HC595N – 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;

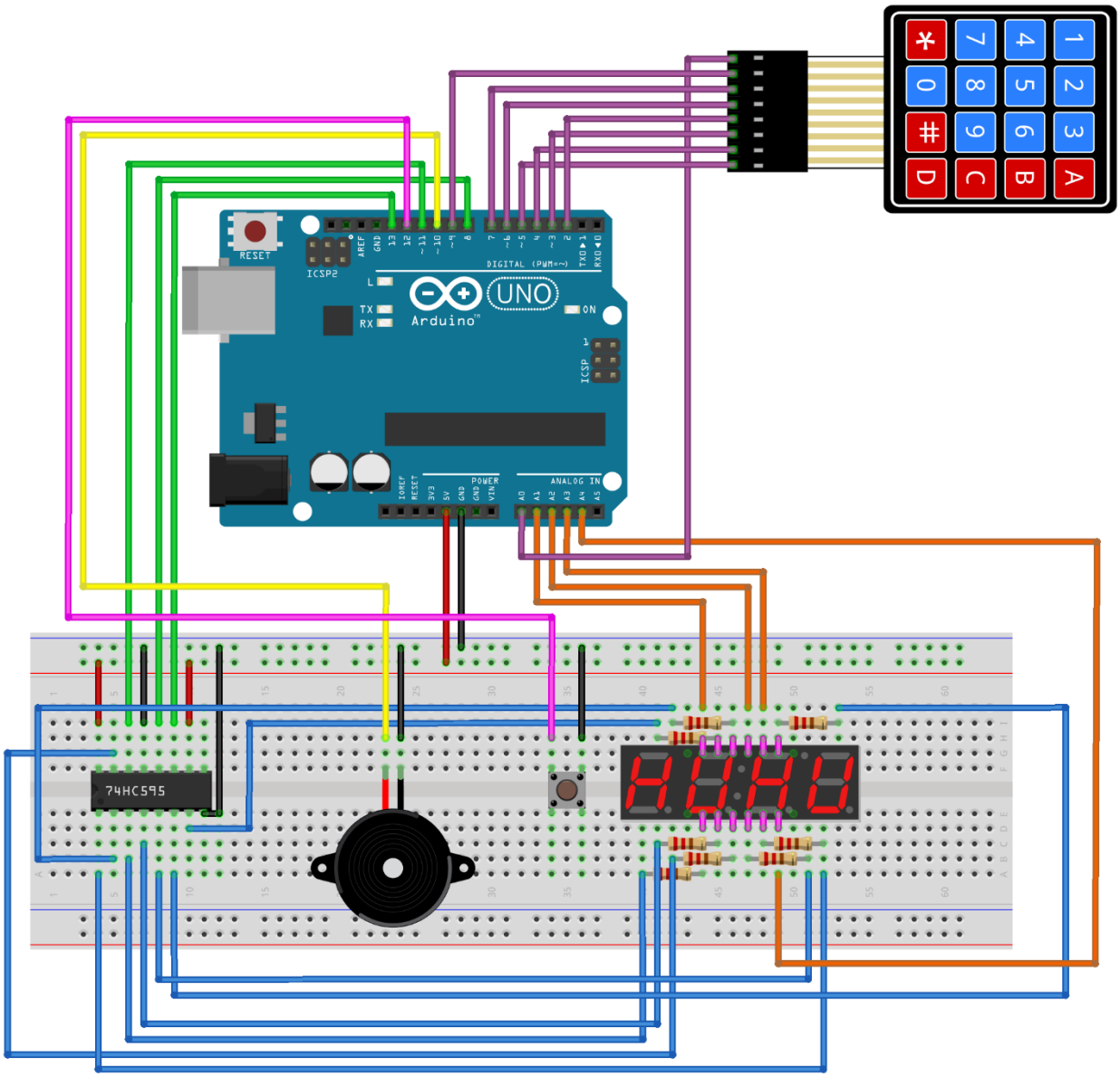
– **[8] GND** — Ground. Земля

– **[16] VCC** — Питание +5 В.

Рисунок 2 – Восьмиразрядный сдвиговый регистр

ЗАДАНИЯ

Построить схему



Написать программный код

```
#include <SPI.h>
#include <Keypad.h>

// Настройка пинов
const int BUZZER_PIN = 10;          // Пин зуммера
const int ENTER_BUTTON_PIN = 12;    // Пин кнопки ввода
const int LATCH_PIN = 8;            // Пин latch для 74HC595

// Пины для разрядов индикатора (катоды) - ОБЩИЙ АНОД!
```

```
int digitPins[4] = {A1, A2, A3, A4}; // A1-A4 соответствуют
пинам 15-18

// Пины для матричной клавиатуры
byte rowPins[4] = {2, 3, 4, 5};       // Подключение строк
byte colPins[4] = {6, 7, 9, A0};      // Подключение столбцов
(A0 - пин 14)

// Матрица символов клавиатуры
char keys[4][4] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};

// Инициализация клавиатуры
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins,
4, 4);

// Коды символов для 7-сегментного индикатора (общий анод)
// Формат: DP GFEDCBA (DP - точка, 0=вкл, 1=выкл для общего
анода)
byte digitPatterns[11] = {
  B11000000, // 0
  B11111001, // 1
  B10100100, // 2
  B10110000, // 3
  B10011001, // 4
  B10010010, // 5
  B10000010, // 6
  B11111000, // 7
  B10000000, // 8
  B10010000, // 9
  B11111111  // Все сегменты выключены)
};

// Паттерны с точкой (DP=0)
byte digitPatternsWithDot[10] = {
  B01000000, // 0 - A,B,C,D,E,F с точкой
  B01111001, // 1 - B,C с точкой
  B00100100, // 2 - A,B,D,E,G с точкой
  B00110000, // 3 - A,B,C,D,G с точкой
  B00011001, // 4 - B,C,F,G с точкой
```

```
  B00010010, // 5 - A,C,D,F,G с точкой
  B00000010, // 6 - A,C,D,E,F,G с точкой
  B01111000, // 7 - A,B,C с точкой
  B00000000, // 8 - A,B,C,D,E,F,G с точкой
  B00010000, // 9 - A,B,C,D,F,G с точкой
};

// Переменные состояния
String enteredCode = "";          // Введенный код
String correctCode = "1234";      // Правильный код по
умолчанию
bool isArmed = false;             // Флаг активации таймера
bool isExploded = false;          // Флаг завершения игры
bool gameCompleted = false;       // Флаг завершения игры
(победа)
unsigned long startTime = 0;      // Время начала отсчета
const long TOTAL_TIME = 120000; // Общее время (2 минуты в
миллисекундах)
long remainingTime = TOTAL_TIME;// Оставшееся время

// Переменные для отображения
int displayNumber = 0;            // Число для отображения
bool showDot = false;             // Флаг отображения точки
bool dotState = false;            // Состояние мигания точки
unsigned long lastDotBlink = 0; // Время последнего мигания
точки
const int DOT_BLINK_INTERVAL = 500; // Интервал мигания
точки (мс)

// Переменные для управления звуком
unsigned long lastBeepTime = 0;
int beepInterval = 1000;          // Начальный интервал звука
bool buzzerState = false;         // Состояние зуммера

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

  // Настройка пинов
  pinMode(LATCH_PIN, OUTPUT);
  digitalWrite(LATCH_PIN, HIGH);

  pinMode(ENTER_BUTTON_PIN, INPUT_PULLUP);
  pinMode(BUZZER_PIN, OUTPUT);
```

```
  digitalWrite(BUZZER_PIN, LOW); // Выключаем зуммер

  // Настройка пинов разрядов (ОБЩИЙ АНОД - HIGH=выкл, LOW=вкл)
  for(int i = 0; i < 4; i++) {
    pinMode(digitPins[i], OUTPUT);
    digitalWrite(digitPins[i], LOW); // Изначально ВКЛЮЧАЕМ все разряды
  }

  // Первоначальное отображение 00.00
  displayNumber = 0;
  showDot = true;
  dotState = true;

  Serial.println("Система инициализирована");
  Serial.println("Введите 4-значный код и нажмите Enter для запуска");
  Serial.println("Пароль по умолчанию: 1234");
}

void loop() {
  if(!isArmed && !isExploded && !gameCompleted) {
    // Режим ввода кода до активации
    handleCodeInput();
  }
  else if(isArmed && !isExploded && !gameCompleted) {
    // Режим отсчета времени
    handleCountdown();
  }
  else if (isExploded || gameCompleted) {
    // Режим завершения игры - ждем сброса
    digitalWrite(BUZZER_PIN, LOW); // Гарантируем выключение зуммера
  }

  // Обновление мигания точки (только во время отсчета)
  updateDotBlink();

  // Обновление индикатора (динамическая индикация)
  updateDisplay();

  // Обработка кнопки ввода
  handleEnterButton();
```

```
}

// Обновление мигания точки - ТОЛЬКО во время отсчета
void updateDotBlink() {
  // Точка мигает только во время активного отсчета
  if(isArmed && showDot) {
    if(millis() - lastDotBlink >= DOT_BLINK_INTERVAL) {
      dotState = !dotState;
      lastDotBlink = millis();
    }
  } else {
    // Вне отсчета точка всегда горит
    dotState = true;
  }
}

// Обработка ввода кода до активации
void handleCodeInput() {
  char key = keypad.getKey();

  if(key) {
    // Игнорируем служебные клавиши A,B,C,D,*,# при вводе
кода
    if((key >= '0' && key <= '9') && enteredCode.length() <
4) {
      // Добавление цифры к введенному коду
      enteredCode += key;
      Serial.println("Введен символ: " + String(key));
      Serial.println("Текущий ввод: " + enteredCode);

      // Обновление отображения введенного кода
      updateDisplayWithCode();
    }
    else if (key == '*') {
      // Сброс ввода
      enteredCode = "";
      Serial.println("Ввод сброшен");
      displayNumber = 0;
      showDot = true;
    }
  }
}

// Обновление отображения введенного кода
```

```
void updateDisplayWithCode() {
  if(enteredCode.length() == 4) {
    displayNumber = enteredCode.toInt();
    showDot = false;
  } else {
    // Если введено меньше 4 цифр, показываем с нулями слева
    String displayStr = enteredCode;
    while(displayStr.length() < 4) {
      displayStr = "0" + displayStr;
    }
    displayNumber = displayStr.toInt();
    showDot = false;
  }
}

// Обработка отсчета времени
void handleCountdown() {
  remainingTime = TOTAL_TIME - (millis() - startTime);

  // Проверка окончания времени
  if(remainingTime <= 0) {
    gameOver(false);
    return;
  }

  // Обновление отображения времени в формате MM.CC
  int minutes = remainingTime / 60000;
  int seconds = (remainingTime % 60000) / 1000;
  displayNumber = minutes * 100 + seconds;
  showDot = true;

  // Обновление ввода во время отсчета
  char key = keypad.getKey();
  if(key && (key >= '0' && key <= '9') &&
enteredCode.length() < 4) {
    enteredCode += key;
    Serial.println("Введен символ: " + String(key));
  }

  // Обновление звукового сигнала
  updateBuzzer();
}

// Обновление зуммера с ускорением к концу
```

```
void updateBuzzer() {
  unsigned long currentTime = millis();

  if(remainingTime <= 2000) {
    // Непрерывный сигнал последние 2 секунды
    digitalWrite(BUZZER_PIN, HIGH);
  }
  else {
    // Прерывистый сигнал с ускорением по мере уменьшения
времени
    if(currentTime - lastBeepTime >= beepInterval) {
      buzzerState = !buzzerState;
      digitalWrite(BUZZER_PIN, buzzerState);
      lastBeepTime = currentTime;

      // Ускорение зуммера по мере уменьшения времени
      // От 2 минут до 2 секунд: интервал уменьшается от
1000 мс до 100 мс
      beepInterval = map(remainingTime, 2000, TOTAL_TIME,
100, 1000);
      beepInterval = constrain(beepInterval, 100, 1000);
    }
  }
}

// Обработка кнопки ввода
void handleEnterButton() {
  static unsigned long lastPressTime = 0;
  static bool lastButtonState = HIGH;

  bool currentButtonState = digitalRead(ENTER_BUTTON_PIN);

  // Обнаружение нажатия (HIGH->LOW)
  if(lastButtonState == HIGH && currentButtonState == LOW) {
    // Защита от дребезга
    if(millis() - lastPressTime > 200) {
      if(!isArmed && !isExploded && !gameCompleted) {
        // Запуск таймера
        startCountdown();
      }
      else if(isArmed && !isExploded && !gameCompleted) {
        // Проверка кода во время отсчета
        checkCode();
      }
```

```
      else if (isExploded || gameCompleted) {
        // Сброс игры после завершения
        resetGame();
      }
      lastPressTime = millis();
    }
  }
  lastButtonState = currentButtonState;
}

// Запуск отсчета
void startCountdown() {
  // Если код не введен, используем код по умолчанию
  if (enteredCode.length() == 0) {
    enteredCode = correctCode;
  } else if (enteredCode.length() < 4) {
    Serial.println("Введите 4 цифры перед запуском!");
    return;
  }

  Serial.println("Запуск таймера на 2 минуты...");
  Serial.println("Пароль для обезвреживания: " +
enteredCode);
  delay(1000); // Задержка перед запуском

  isArmed = true;
  startTime = millis();
  remainingTime = TOTAL_TIME;
  correctCode = enteredCode; // Устанавливаем введенный код
как правильный
  enteredCode = ""; // Сбрасываем для ввода во время игры

  // Сбрасываем состояние мигания точки для начала отсчета
  dotState = true;
  lastDotBlink = millis();

  Serial.println("Таймер запущен! Введите код для
обезвреживания!");
}

// Проверка введенного кода
void checkCode() {
  if(enteredCode.length() == 4) {
    if(enteredCode == correctCode) {
```

```
      // Правильный код
      gameOver(true);
    } else {
      // Неправильный код - штраф 3 секунды
      startTime -= 3000;
      remainingTime = TOTAL_TIME - (millis() - startTime);
      if (remainingTime < 0) remainingTime = 0;

      Serial.println("Неверный код! -3 секунды! Осталось: "
+ String(remainingTime/1000) + " сек");

      // Сигнал ошибки
      for(int i = 0; i < 3; i++) {
        digitalWrite(BUZZER_PIN, HIGH);
        delay(100);
        digitalWrite(BUZZER_PIN, LOW);
        delay(100);
      }

      enteredCode = "";
    }
  } else {
    Serial.println("Введите 4 цифры для проверки кода!");
  }
}

// Сброс игры
void resetGame() {
  isArmed = false;
  isExploded = false;
  gameCompleted = false;
  enteredCode = "";
  correctCode = "1234"; // Возвращаем код по умолчанию
  remainingTime = TOTAL_TIME;
  digitalWrite(BUZZER_PIN, LOW);
  displayNumber = 0;
  showDot = true;
  dotState = true; // Точка горит постоянно вне отсчета

  Serial.println("Игра сброшена. Введите новый код или
нажмите Enter для использования 1234");
}

// Завершение игры
```

```
void gameOver(bool success) {
  isArmed = false;
  digitalWrite(BUZZER_PIN, LOW);

  if(success) {
    gameCompleted = true;
    Serial.println("Бомба обезврежена!");
    playWinTune();
  } else {
    isExploded = true;
    Serial.println("Бомба взорвалась!");
    playLoseTune();
  }

  // Сброс ввода и отображение 00.00
  enteredCode = "";
  displayNumber = 0;
  showDot = true;
  dotState = true; // Точка горит постоянно после завершения
игры
}

// Мелодия победы
void playWinTune() {
  // Двойной приятный пик
  for(int j = 0; j < 2; j++) {
    for(int i = 1000; i < 2000; i += 50) {
      tone(BUZZER_PIN, i, 10);
      delay(1);
    }
    delay(100);
  }
  noTone(BUZZER_PIN);

  // Победная мелодия
  int melody[] = {523, 659, 784, 1047}; // До, Ми, Соль, До
  int duration = 200;

  for(int i = 0; i < 4; i++) {
    tone(BUZZER_PIN, melody[i], duration);
    delay(duration + 50);
  }
  noTone(BUZZER_PIN);
}
```

```
// Мелодия проигрыша
void playLoseTune() {
  // Длинный низкий звук
  tone(BUZZER_PIN, 200, 1000);
  delay(1000);

  // Проигрышная мелодия
  int melody[] = {392, 349, 330, 294, 262}; // Соль, Фа, Ми,
Ре, До
  int duration = 300;

  for(int i = 0; i < 5; i++) {
    tone(BUZZER_PIN, melody[i], duration);
    delay(duration + 50);
  }
  noTone(BUZZER_PIN);
}

// Обновление дисплея (динамическая индикация)
void updateDisplay() {
  static unsigned long lastDisplayTime = 0;
  static int currentDigit = 0;

  // Обновляем каждые 5 мс для плавной индикации
  if(millis() - lastDisplayTime >= 5) {
    lastDisplayTime = millis();

    // Разделяем число на цифры
    int number1 = displayNumber;
    int digits[4];
    for(int i = 0; i < 4; i++) {
      digits[3-i] = number1 % 10; // digits[0]-старший
разряд, digits[3]-младший
      number1 = number1 / 10;
    }

    // Выключаем все разряды перед обновлением
    for(int j = 0; j < 4; j++) {
      digitalWrite(digitPins[j], LOW); // LOW = ВКЛ для
общего анода
    }

    // Включаем текущий разряд
```

```
    digitalWrite(digitPins[currentDigit], HIGH); // HIGH =
ВЫКЛ для общего анода

    // Подготавливаем паттерн для отображения
    byte pattern;
    if(showDot && dotState && currentDigit == 1) {
      // Для разряда 1 (второй слева) добавляем точку
      pattern = digitPatternsWithDot[digits[currentDigit]];
    } else {
      pattern = digitPatterns[digits[currentDigit]];
    }

    // Выводим цифру на индикатор
    digitalWrite(LATCH_PIN, LOW);
    SPI.transfer(pattern);
    digitalWrite(LATCH_PIN, HIGH);

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