Практическое занятие №
Тема: «Разработка интерфейса для ЖК-дисплея»

Цель работы: приобрести практические навыки по подключению и программированию ЖК-дисплея LCD1602 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

ЖК-ДИСПЛЕЙ СЕРИИ LCD1602

LCD1602 — это один из самых распространенных и популярных алфавитно-цифровых жидкокристаллических дисплеев, используемых в электронных проектах, особенно с платами типа Arduino, ESP32 и другими микроконтроллерами.

Расшифровка названия: LCD — Liquid Crystal Display Жидкокристаллический дисплей), **1602** — 16 символов в строке, 2 строки.

Основные характеристики:
Дисплей: Монохромный, с подсветкой (синяя или зеленая подсветка с темными символами).
Разрешение: 16 x 2 символов. Каждый символ имеет размер 5x8 пикселей.
Контроллер: Наиболее часто используется чип HD44780 или его клоны.

Интерфейсs связи:

Параллельный (8-битный или 4-битный): Стандартный режим работы. Требует от 6 до 10 цифровых выводов микроконтроллера.
I2C (через переходной модуль): Очень популярный способ подключения. Специальный адаптер устанавливается на заднюю часть дисплея и позволяет управлять им всего по 2 проводам (SDA, SCL) + питание. Это экономит выводы микроконтроллера.

Напряжение питания: Обычно +5В.
Возможности: Отображение букв латинского и кириллического алфавитов (зависит от прошивки), цифр, специальных символов. Также можно создавать собственные символы (до 8 штук), загружая их в память контроллера.

Преимущества:

- Низкая стоимость.
- Множество готовых библиотек и примеров в интернете.
- Простота в использовании.
- Низкое энергопотребление**.**

Недостатки:

- Ограниченность информации (всего 32 символа).
- Низкое разрешение, не подходит для графики (только символы).
- Без модуля I2C требует много выводов микроконтроллера.

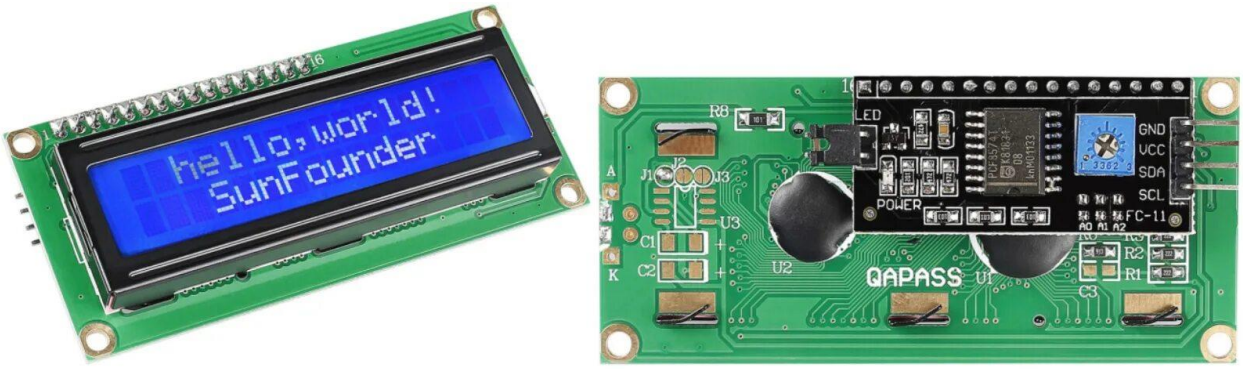

Рисунок 1 – LCD1602

ЗАДАНИЯ

Собрать схему

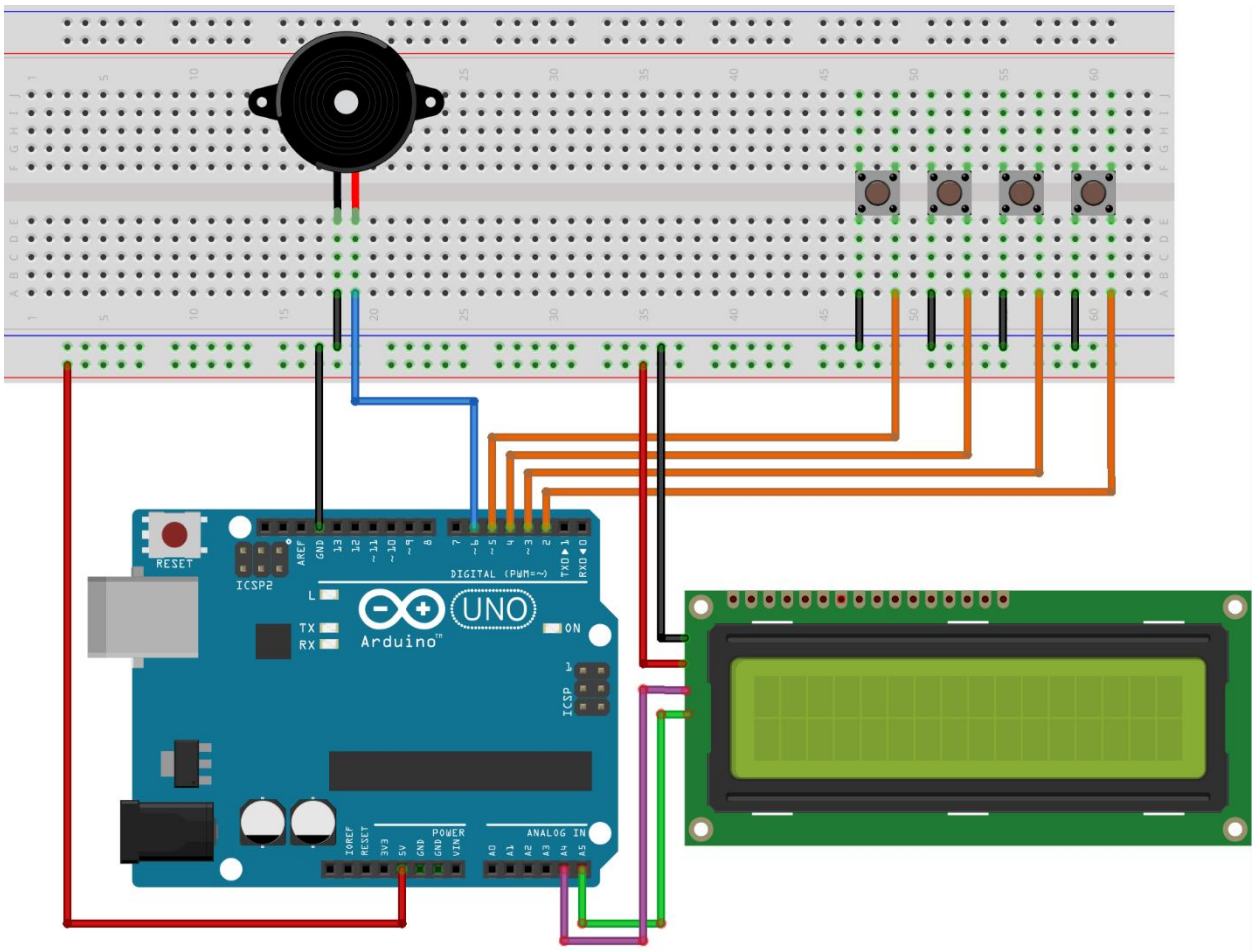

Рисунок 2 – Схема подключения

Программный код:

```
#include <LiquidCrystal_I2C.h>
#include <EEPROM.h>

LiquidCrystal_I2C lcd(0x27, 16, 2); // Укажите свой адрес I2C

// Пины для кнопок
#define BTN_UP 2
#define BTN_DOWN 3
#define BTN_JUMP 8
#define BTN_OK 5
#define BUZZER 6
```

```
// Игровые переменные
bool gameRunning = false;
int score = 0;
int highScore = 0;
int level = 1;
int baseGameSpeed = 500;
int currentGameSpeed = 500;
unsigned long lastUpdate = 0;
unsigned long jumpStartTime = 0;
unsigned long lastButtonPress = 0;
unsigned long lastCactusTime = 0;
int cactusInterval = 2000;

// Позиции элементов
int dinoPos = 1;
int cactusPositions[3] = {-1, -1, -1}; // Позиции кактусов
(-1 = нет кактуса)
bool isJumping = false;
bool isLongJump = false;
int jumpHeight = 0;
const int shortJumpDuration = 750;
const int longJumpDuration = 1500;

// Переменные меню
int menuSelection = 0; // 0: Start Game, 1: Level
bool inLevelMenu = false;

// Прототипы функций
void drawGame();
void handleInput();
void updateGame();
void showMenu();
void showResults();
void jump(bool longJump);
void endJump();
void playTone(int frequency, int duration);
void saveHighScore();
void loadHighScore();
bool buttonPressed(int pin);
void generateCactus();
void updateCacti();
```

```
void setup() {
  lcd.init();
  lcd.backlight();

  // Настройка пинов
  pinMode(BTN_UP, INPUT_PULLUP);
  pinMode(BTN_DOWN, INPUT_PULLUP);
  pinMode(BTN_JUMP, INPUT_PULLUP);
  pinMode(BTN_OK, INPUT_PULLUP);
  pinMode(BUZZER, OUTPUT);

  // Загрузка рекорда из EEPROM
  loadHighScore();

  showMenu();
}

void loop() {
  if (gameRunning) {
    handleInput();
    updateGame();
  }
}

void showMenu() {
  lcd.clear();

  while (!gameRunning) {
    if (!inLevelMenu) {
      // Главное меню
      lcd.setCursor(0, 0);
      if (menuSelection == 0) {
        lcd.print(">Start Game");
      } else {
        lcd.print(" Start Game");
      }

      lcd.setCursor(0, 1);
      if (menuSelection == 1) {
        lcd.print(">Level: ");
      } else {
        lcd.print(" Level: ");
      }
      lcd.print(level);
```

```
      // Обработка ввода в главном меню
      if (buttonPressed(BTN_UP)) {
        menuSelection = 0;
      }
      if (buttonPressed(BTN_DOWN)) {
        menuSelection = 1;
      }
      if (buttonPressed(BTN_OK)) {
        if (menuSelection == 0) {
          gameRunning = true;
          startGame();
        } else {
          inLevelMenu = true;
          lcd.clear();
        }
      }
    } else {
      // Меню выбора уровня
      lcd.setCursor(0, 0);
      lcd.print("Select Level:");
      lcd.setCursor(0, 1);
      lcd.print(">");
      lcd.print(level);

      // Обработка ввода в меню уровня
      if (buttonPressed(BTN_UP)) {
        level = min(5, level + 1);
      }
      if (buttonPressed(BTN_DOWN)) {
        level = max(1, level - 1);
      }
      if (buttonPressed(BTN_OK)) {
        inLevelMenu = false;
        lcd.clear();
      }
    }

    delay(100);
  }
}
```

```
bool buttonPressed(int pin) {
  if (!digitalRead(pin)) {
    if (millis() - lastButtonPress > 300) { // Защита от
дребезга
      lastButtonPress = millis();
      return true;
    }
  }
  return false;
}

void startGame() {
  score = 0;
  // Очищаем кактусы
  for (int i = 0; i < 3; i++) {
    cactusPositions[i] = -1;
  }
  dinoPos = 1;
  isJumping = false;
  isLongJump = false;
  jumpHeight = 0;

  // Устанавливаем скорость в зависимости от уровня
  switch (level) {
    case 1: baseGameSpeed = 500; break;
    case 2: baseGameSpeed = 400; break;
    case 3: baseGameSpeed = 300; break;
    case 4: baseGameSpeed = 200; break;
    case 5: baseGameSpeed = 150; break;
  }
  currentGameSpeed = baseGameSpeed;

  cactusInterval = 2000; // Интервал между кактусами

  lcd.clear();
}

void updateGame() {
  if (millis() - lastUpdate > currentGameSpeed) {
    // Обновляем позиции кактусов
    updateCacti();

    // Генерируем новые кактусы
    if (millis() - lastCactusTime > cactusInterval) {
```

```
      generateCactus();
      lastCactusTime = millis();

      // Случайный интервал между кактусами
      cactusInterval = random(1500, 3000);
    }

    // Обновляем прыжок
    if (isJumping) {
      unsigned long jumpTime = millis() - jumpStartTime;
      int jumpDuration = isLongJump ? longJumpDuration :
shortJumpDuration;

      if (jumpTime < jumpDuration / 2) {
        // Взлет
        jumpHeight = 1;
      } else if (jumpTime < jumpDuration) {
        // Полет
        jumpHeight = 1;
      } else {
        // Приземление
        endJump();
      }
    }

    // Проверка столкновений
    for (int i = 0; i < 3; i++) {
      if (cactusPositions[i] == dinoPos && jumpHeight == 0)
{
        gameOver();
        return;
      }
    }

    drawGame();
    lastUpdate = millis();
  }
}
```

```
void updateCacti() {
  for (int i = 0; i < 3; i++) {
    if (cactusPositions[i] >= 0) {
      cactusPositions[i]--;

      // Если кактус ушел за экран, удаляем его
      if (cactusPositions[i] < 0) {
        cactusPositions[i] = -1;
        score++;

        // Ускоряем игру после каждого прыжка (кактуса)
        if (currentGameSpeed > 50) {
          currentGameSpeed -= 2;
        }
      }
    }
  }
}

void generateCactus() {
  // Ищем свободное место для кактуса
  for (int i = 0; i < 3; i++) {
    if (cactusPositions[i] < 0) {
      // Случайно решаем, сколько кактусов генерировать (1-
2)
      int numCacti = random(1, 3);

      for (int j = 0; j < numCacti && i < 3; j++) {
        cactusPositions[i] = 15 + j; // Ставим кактусы
подряд
        i++;
      }
      break;
    }
  }
}
```

```
void drawGame() {
  lcd.clear();

  // Отрисовка земли
  lcd.setCursor(0, 1);
  for (int i = 0; i < 16; i++) {
    lcd.print("-");
  }

  // Отрисовка динозавра
  lcd.setCursor(dinoPos, 1 - jumpHeight);
  lcd.print("D");

  // Отрисовка кактусов
  for (int i = 0; i < 3; i++) {
    if (cactusPositions[i] >= 0 && cactusPositions[i] < 16)
{
      lcd.setCursor(cactusPositions[i], 1);
      lcd.print("X");
    }
  }
}

void handleInput() {
  // Проверяем кнопку прыжка
  if (!digitalRead(BTN_JUMP) && !isJumping) {
    // Определяем тип прыжка по времени удержания
    unsigned long pressStart = millis();

    // Ждем отпускания кнопки или максимального времени для
длинного прыжка
    while (!digitalRead(BTN_JUMP) && (millis() - pressStart)
< 1000) {
      delay(10);
    }

    bool longJump = (millis() - pressStart) > 500;
    jump(longJump);
  }
}
```

```
void jump(bool longJump) {
  if (!isJumping) {
    isJumping = true;
    isLongJump = longJump;
    jumpStartTime = millis();

    // Звук прыжка
    if (longJump) {
      playTone(523, 150); // Высокий звук для длинного
прыжка
    } else {
      playTone(392, 100); // Низкий звук для короткого
прыжка
    }
  }
}

void endJump() {
  isJumping = false;
  isLongJump = false;
  jumpHeight = 0;
}

void playTone(int frequency, int duration) {
  tone(BUZZER, frequency, duration);
}

void gameOver() {
  gameRunning = false;

  // Проигрышный звук
  playTone(262, 200); // C4
  delay(250);
  playTone(196, 400); // G3

  if (score > highScore) {
    highScore = score;
    saveHighScore();
  }

  delay(1000);
  showResults();
}
```

```
void showResults() {
  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Score: ");
  lcd.print(score);
  lcd.setCursor(0, 1);
  lcd.print("Best: ");
  lcd.print(highScore);

  delay(2000);

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Press OK to");
  lcd.setCursor(0, 1);
  lcd.print("continue");

  // Ожидание нажатия OK
  while (!buttonPressed(BTN_OK)) {
    delay(100);
  }

  // Сброс состояния меню
  menuSelection = 0;
  inLevelMenu = false;
  showMenu();
}

void saveHighScore() {
  EEPROM.put(0, highScore);
}

void loadHighScore() {
  EEPROM.get(0, highScore);
}
```