ESP8266 Game Console

Radu Ștefănescu

January 2024

1 Introduction

This document presents the microcontroller game console, a simple yet functional DIY project that integrates 2 ESP8266 microcontrollers with two OLED screens. The ESP8266 is chosen for its WiFi capabilities and adequate processing power, suitable for basic gaming needs. The dual OLED screen setup is a key feature of this console, providing clear and sharp displays.

2 Components

The ESP8266 game console is built using a selection of electronic components that are both accessible and versatile. The core components of this project are as follows:

- **2 ESP8266 Microcontrollers:** These are the central units of the console, known for their WiFi capabilities and sufficient processing power for basic gaming applications.
- **1 SPI OLED Screen:** This screen connects via the Serial Peripheral Interface (SPI). It is known for its fast data transfer rate, which is essential for dynamic screen updates during gameplay. (Used for Singleplayer and Multiplayer)
- **1 I2C OLED Screen:** Connected through the Inter-Integrated Circuit (I2C) interface, (Used as Player 2's screen in multiplayer)
- **2 LEDs:** These Light Emitting Diodes are used as visual indicators.
- **2 Buttons:** These are used to select the game mode (Singleplayer / Multiplayer) or to reset the game console.
- **1 Buzzer:** A simple yet effective way to add audio feedback.

Each component plays a vital role in the functionality of the console, combining to create an engaging and interactive gaming device.

3 Code and Functionality

The software architecture of the ESP8266 game console is designed to maximize the capabilities of the hardware components while ensuring smooth and responsive gameplay. The key aspects of the code and its functionality are outlined below:

- **Use of Adafruit SSD1306 Library:** The console leverages the Adafruit SSD1306 library for controlling the OLED displays. This library is widely recognized for its ease of use and efficiency in managing OLED screens, particularly in graphic rendering and text display.
- **Button Debouncing:** To ensure reliable input without false triggers, the console implements debouncing algorithms for the buttons. This technique is crucial for enhancing the user experience, as it filters out spurious signals caused by mechanical and electrical noise in the button press actions.
- **First ESP8266 Microcontroller:** This microcontroller is tasked with multiple roles. It controls the buzzer and the two LEDs, providing audio-visual feedback to the user. Moreover, it creates a WiFi access point, allowing players to connect with their smartphones and interact with the game. This feature not only adds a modern twist to the console but also expands the possibilities for gameplay and user interaction.
- **Game Control via WiFi:** Players use their smartphones to connect to the console's WiFi access point, from which they can send control commands for the game. This approach modernizes the gaming experience and makes the console more interactive and accessible.
- **Inter-ESP Communication:** The control commands received from players are transmitted to the second ESP8266 microcontroller via RX/TX (serial) communication. This separation of responsibilities allows the first ESP to focus on network handling and user input, while the second ESP is dedicated to controlling the displays.
- **Second ESP8266 Microcontroller:** This microcontroller is responsible for managing the SPI and I2C OLED screens. By receiving commands from the first ESP, it updates the display based on the game's progress and player interactions.

This architecture not only allows for efficient utilization of each ESP8266 microcontroller's strengths but also provides a scalable and flexible platform for further development and enhancement of the game console's capabilities.

4 Circuit

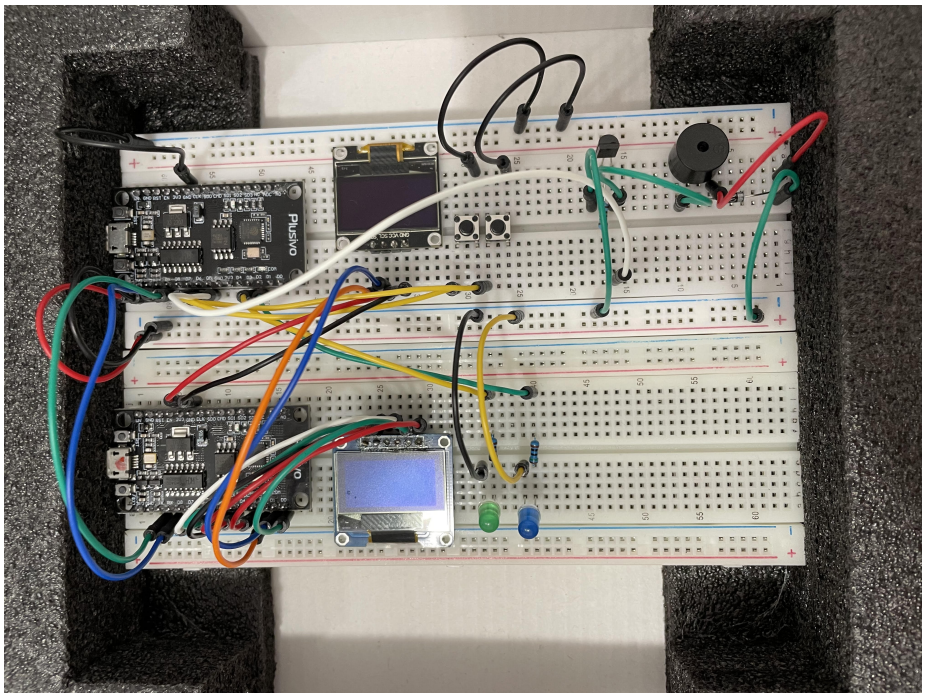

Figure 1: The Console

5 Code Highlights

Below are some examples of code from the 2 files ESP_SERVER and ESP_GAME.

Listing 1: PONG game coordinate tracking algorithm

```
  if (Serial.available() > 0) {
    String command = Serial.readStringUntil('\n');
    if (command.startsWith("P1")) {
      int t = command.substring(3).toInt();
      m1 = (abs(t - m1) > 2) ? t : m1;
    } else if (command.startsWith("P2")) {
      int t = SCREEN_WIDTH - command.substring(3).toInt();
      m2 = (abs(t - m2) > 2) ? t : m2;
    }
  ...
  if (paddle1X != m1) {
    paddle1X += (paddle1X > m1) ? min(-1, (m1 - paddle1X) /
        2) : max(1, (m1 - paddle1X) / 2);
  }

  if (paddle2X != m2) {
    paddle2X += (paddle2X > m2) ? min(-1, (m2 - paddle2X) /
        2) : max(1, (m2 - paddle2X) / 2);
  }
```

Listing 2: PONG game display logic

```
void drawPongGame() {
  displaySPI.clearDisplay();
  displayI2C.clearDisplay();

  displaySPI.fillRect(paddle1X, SCREEN_HEIGHT - paddleHeight
      , paddleWidth, paddleHeight, SSD1306_WHITE);
  displayI2C.fillRect(SCREEN_WIDTH - paddle2X - paddleWidth,
       SCREEN_HEIGHT - paddleHeight, paddleWidth,
      paddleHeight, SSD1306_WHITE);

  if (onFirstScreen) {
    displaySPI.fillRect(ballX, ballY, ballSize, ballSize,
        SSD1306_WHITE);
  } else {
    displayI2C.fillRect(ballX, ballY, ballSize, ballSize,
        SSD1306_WHITE);
  }

  displaySPI.display();
  displayI2C.display();
}
```

Listing 3: PONG game ball movement logic

```
void updatePongGame() {
  ballX += ballVelocityX;
  ballY += ballVelocityY;

  if (onFirstScreen) {
    if (ballY >= SCREEN_HEIGHT - ballSize && ballX >=
        paddle1X && ballX <= paddle1X + paddleWidth) {
      ballVelocityY = -ballVelocityY;
    } else if (ballY <= 0) {
      // Move to I2C screen
      onFirstScreen = false;
      ballVelocityY = -ballVelocityY;
      ballVelocityX = -ballVelocityX;
      ballY = 1; // Start from top of I2C screen
      ballX = SCREEN_WIDTH - ballX;
    } else if (ballX >= SCREEN_WIDTH || ballX <= 0) {
      ballVelocityX = -ballVelocityX;
    } else if (ballY >= SCREEN_HEIGHT - 1) {
      pongOver = true;
      pongGameOver(2);
    }
  } else {
    if (ballY >= SCREEN_HEIGHT - ballSize && SCREEN_WIDTH -
        ballX >= paddle2X && SCREEN_WIDTH - ballX <= paddle2X
         + paddleWidth) {
      ballVelocityY = -ballVelocityY;
    } else if (ballY < 0) {
      // Move to SPI screen
      onFirstScreen = true;
      ballVelocityY = -ballVelocityY;
      ballVelocityX = -ballVelocityX;
      ballX = SCREEN_WIDTH - ballX;
      ballY = 1; // Start from bottom of SPI screen
    } else if (ballX >= SCREEN_WIDTH || ballX <= 0) {
      ballVelocityX = -ballVelocityX;
    } else if (ballY >= SCREEN_HEIGHT - 1) {
      pongOver = true;
      pongGameOver(1);
    }
  }
}
```

Listing 4: Server initial setup

```
void setupServer() {
  server.on("/", htmlIndex);
  server.on("/buttonPress", handleButtonPress);
  server.on("/sliderMove", handleSliderMove);
  server.begin();
  Serial.println("HTTP server started");
}

void createNetwork() {
  Serial.println("Setting up as an Access Point");
  WiFi.mode(WIFI_AP);
  WiFi.softAP(ssid, password);

  IPAddress myIP = WiFi.softAPIP();
  Serial.print("AP IP address: ");
  Serial.println(myIP);
}
```

Listing 5: Server player connection logic

```
void htmlIndex() {
    IPAddress clientIP = server.client().remoteIP();

    if (!player1Connected || clientIP == player1IP) {
      player1IP = clientIP;
      player1Connected = true;
    } else if (!player2Connected || clientIP == player2IP) {
      player2IP = clientIP;
      player2Connected = true;
    } else {
      server.send(403, "text/plain", "Game is already in
          progress.");
      return;
    }

    String player = (clientIP == player1IP) ? "Player 1" : "
        Player 2";
    String sliderPlayer = ((clientIP == player1IP) ? "P1" :
        "P2");

    ...
```