Workshop V

Serial Library & Internal Pull-Up Resistors

SECTION I

Serial Library

Serial.begin

- `Serial.begin(int baud_rate)`
  - `Serial` is an object that facilitates communication between the ESP32 board and other devices over UART protocol
    - Sets communication speed between devices
  - Pass `115200` bits per second as the argument for this member function
    - This is the default baud rate

```
void setup()
{
    pinMode(LED, OUTPUT);
    pinMode(RECEIVER, INPUT);
    Serial.begin(115200);
}
```

Serial.write

- `Serial.write` is used to transmit raw binary data from the UART in the ESP32 board (different from `Serial.print`!)

Common Variations

- `Serial.write(val)` - Sends `val`, a single byte of data
- `Serial.write(str)` - Sends `str`, a series of bytes represented by a string

```
Serial.write(62); // Sends 62
Serial.write("Hello"); // Sends “Hello” over multiple frames
```

Serial.available

- `Serial.available` returns true if the ESP32's UART has received any data
- The incoming data is stored in a sort of "mailbox" called a **receive buffer**
  - The buffer is essentially an array that stores the data
  - If the buffer is full, then the incoming data is dropped and lost forever

```
if (Serial.available())
{
    int incomingByte = Serial.read(); // Stores the incoming byte
}
```

Serial.read

- `Serial.read` returns the first byte of incoming serial data available (or -1 if no data is available
  - The data is returned as an `int`
  - When this function is called, the byte returned by it is removed from the receive buffer
  - Reads **one character 1 byte**

```
int incomingByte = Serial.read(); // Stores the incoming byte
```

Serial.parseInt

- `Serial.parseInt` reads/parses a series of digits from the serial buffer, converting and returning them as one value
  - The data is returned as an `int`
  - If no valid digits were read after a certain period, 0 is returned
  - Reads a **whole number**

```
int number = Serial.parseInt(); // Stores the numeric input
```

ESP32 Echo Communication

I/A

Write a program that satisfies the following requirements:

- You must write a program in which the **ESP32 board “echoes” back any byte (0-255) sent to it over UART.**
- When the user enters a byte value (0-255) into the Arduino IDE Serial Monitor from their computer, the value is sent to the ESP32 board. The board must send the same value back to the computer.
- The ESP32 will **only “echo” back a value sent to it once**. It should not resend the value multiple times.
- The program must **run in an infinite loop**.

ESP32 Echo Communication Code

```
void setup() {
  // Initialize serial communication at 115200 baud rate
  Serial.begin(115200);
}

void loop() {
  // Check if data is available to read from the Serial buffer
  if (Serial.available() > 0) {
    // Read the incoming byte as an int
    int receivedByte = Serial.read();

    // Echo the byte back to the sender
    if (receivedByte >= 0 && receivedByte <= 255) {
      Serial.write(receivedByte);
    }
  }
}
```

ESP32 Servo Controller

I/A

Write a program that satisfies the following requirements:

- You must write a program in which the **ESP32 board receives a value between 0-180 and writes that value to a servo motor.**
- When the user enters a valid servo angle (0-180) into the Arduino IDE Serial Monitor from their computer, the value is sent to the servo motor. The servo then moves to the specified angle and stays there until another input.
- Use the `Serial.parseInt()` function instead of `Serial.read()` for this program.
- The program must **run in an infinite loop**.

ESP32 Servo Controller Code

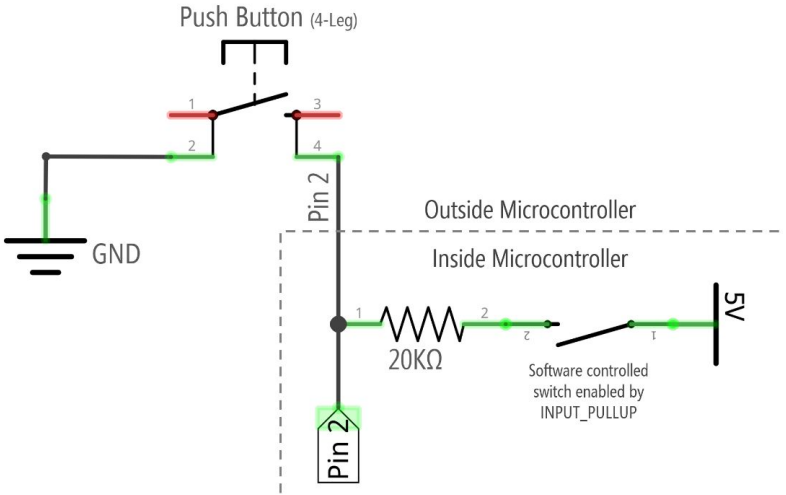
```
#include <ESP32Servo.h>

// Assign variable to pin number of Servo's PWM pin
int servoPin = 3;

// Initialize servo object with name myServo
Servo myServo;

void setup() {
    // Attach object myServo to the physical servo
    myServo.attach(servoPin);
  Serial.begin(115200);
}

void loop() {
if (Serial.available()) {
  int incomingByte = Serial.parseInt(); // Stores the incoming byte
    if (incomingByte <= 180 && incomingByte > 0) {
        myServo.write(incomingByte);
      Serial.print(incomingByte);
    }
  }
  delay(100);
}
```

SECTION II

Internal Pull-Up Resistors

What are Internal Pull-Up Resistors?

- Similar to external pull-ups, **internal pull-up resistors** also pull pins to a HIGH voltage by default
  - This is to prevent a floating signal where the pin's state is indeterminate
- The difference is that internal pull-ups are integrated into the ESP32 and are accessed through the code



Internal vs. External

- External
  - Flexibility with choosing resistance value
  - Requires more physical space
  - Increases design complexity
- Internal
  - Fixed resistance value
  - Built into the microcontroller
  - Simplifies hardware design

INPUT_PULLUP

- To use the internal pull-up resistor, specify **INPUT_PULLUP** as the mode in the **pinMode** `(pin, mode)` function
- This makes it so that the specified pin defaults to **HIGH** unless a connection is made with GND (just like an external pull-up resistor!)

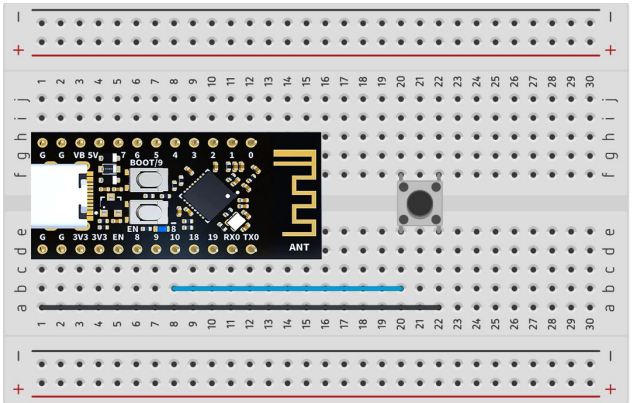
```
const int LED = 4;

void setup()
{
   pinMode(LED,
INPUT_PULLUP);
   Serial.begin(115200);
}
```

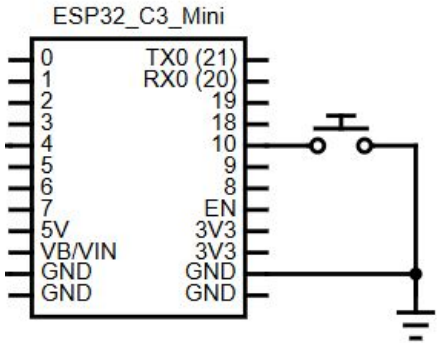
ESP32 Internal Pull-Up Circuit

I/A

Now we're going to make an internal pull-up resistor circuit!

- Build the following circuit on your breadboard (image shown below)
- Write Arduino code that sets the input pin to **INPUT_PULLUP** mode
  - Your code should also verify the **HIGH** or **LOW** state of the input pin and print the current state to the serial monitor





FAIR USE DISCLAIMER

Copyright Disclaimer under section 107 of the Copyright Act 1976, allowance is made for “fair use” for purposes such as criticism, comment, news reporting, teaching, scholarship, education and research.

Fair use is a use permitted by copyright statute that might otherwise be infringing.

Non-profit, educational or personal use tips the balance in favor of fair use.

CC BY-NC-SA 4.0