## 🛱 perplexity

# WEEK 9. CHECK MY CIRCUIT AND TELL WHETHER ITS CORRECT OR NOT

The schematic provided appears to include an ESP32 microcontroller and supporting circuitry. Below is an analysis of the circuit:

## **Observations:**

#### 1. ESP32 Microcontroller:

- The ESP32 is correctly connected with power supply pins (PWR\_3V3 and PWR\_GND).
- Decoupling capacitors are present near the power pins, which is good for stable operation.

#### 2. Voltage Regulator:

- The voltage regulator circuit is designed to provide 3.3V output.
- Ensure that the input voltage matches the regulator's requirements and that the capacitors are of appropriate values.

#### 3. Rotary Encoder:

• The rotary encoder connections seem correct, with pull-up resistors on the signal lines (A, B, and SW). Verify resistor values for proper operation.

#### 4. Boot and Enable Switches:

- The boot and enable switches are correctly wired with pull-up resistors.
- Ensure proper debounce circuitry if required.

#### 5. Programmable LED:

• The LED is connected via a resistor, which limits current flow. Confirm that the resistor value provides adequate brightness without overloading the GPIO pin.

#### 6. RX/TX Connections:

• The RX/TX pins are connected to a header for external communication. Ensure proper voltage levels for interfacing with other devices.

## 7. Decoupling Capacitors:

• Decoupling capacitors are placed near power lines, which is essential for filtering noise.

#### 8. Pin Headers:

• Pin headers are provided for GPIO access, making the design modular and flexible.

## **Recommendations:**

- Verify resistor values in all parts of the circuit (e.g., pull-ups, LED current-limiting resistors) to ensure they match the intended functionality.
- Check capacitor values in the voltage regulator section to confirm they meet datasheet specifications.
- Test the circuit in simulation or on a breadboard before finalizing the PCB design.
- Ensure proper grounding and trace widths for power lines in PCB layout for stability.

Overall, the schematic looks well-organized and functional, but testing and verification are essential before proceeding further.

\*\*

# how to add esp32 library in my arduino ide

To add the ESP32 library to your Arduino IDE, follow these steps:

## Step-by-Step Installation

#### 1. Install Arduino IDE:

• Ensure you have the latest version of Arduino IDE installed on your computer. You can download it from the official Arduino website.

#### 2. Open Preferences:

• Launch Arduino IDE and go to **File > Preferences**.

#### 3. Add ESP32 Board URL:

• In the Additional Board Manager URLs field, enter the following URL:

https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package\_esp32\_

• If you already have other URLs (e.g., for ESP8266), separate them with a comma.

#### 4. Open Boards Manager:

• Navigate to Tools > Board > Boards Manager.

#### 5. Install ESP32 by Espressif Systems:

- In the Boards Manager search bar, type ESP32.
- Locate the entry labeled "ESP32 by Espressif Systems" and click the **Install** button.

#### 6. Select Your ESP32 Board:

• After installation is complete, go to **Tools > Board**, and select your specific ESP32 model (e.g., DOIT ESP32 DEVKIT V1).

## 7. Test the Installation:

- Connect your ESP32 board to your computer via USB.
- Select the appropriate COM port under **Tools > Port**.

- Load an example sketch (e.g., File > Examples > WiFi (ESP32) > WiFiScan) and upload it to your board.
- Open the Serial Monitor at 115200 baud rate to verify functionality.

This process ensures that your Arduino IDE is configured correctly for programming ESP32 boards<sup>[1]</sup> <sup>[2]</sup> <sup>[3]</sup>.

#### \*\*

## GIVE A CODE FOR FUNCTIONING OF ROTARY ENCODER IN MY ARDUINO IDE .BASED ON MY CIRCUIT THAT I UPLOAD . SELECT THE PINS ACCORDINGLY.

Based on your circuit, the rotary encoder is connected to the ESP32 with the following pin assignments:

- CLK: GPI016
- **DT**: GPIO17
- SW: GPI018

Below is the Arduino IDE code for reading and handling the rotary encoder functionality:

```
// Pin definitions for rotary encoder
#define CLK 16 // Connected to GPI016
#define DT 17 // Connected to GPI017
#define SW 18 // Connected to GPI018
// Variables for encoder state
int lastStateCLK;
int currentStateCLK;
int counter = 0; // To track encoder position
void setup() {
  // Set pin modes
  pinMode(CLK, INPUT);
  pinMode(DT, INPUT);
  pinMode(SW, INPUT_PULLUP); // SW has a pull-up resistor
  // Initialize serial communication
  Serial.begin(115200);
  // Read initial state of CLK
  lastStateCLK = digitalRead(CLK);
  Serial.println("Rotary Encoder Initialized");
}
void loop() {
  // Read current state of CLK
```

```
currentStateCLK = digitalRead(CLK);
  // Check if there is a change in state
  if (currentStateCLK != lastStateCLK && currentStateCLK == HIGH) {
    // Read DT to determine direction
    if (digitalRead(DT) != currentStateCLK) {
      counter++;
      Serial.println("Rotated Clockwise");
    } else {
      counter--;
      Serial.println("Rotated Counterclockwise");
    }
    // Print the counter value
    Serial.print("Counter Value: ");
    Serial.println(counter);
  }
  // Update lastStateCLK
  lastStateCLK = currentStateCLK;
  // Check if the switch is pressed
  if (digitalRead(SW) == LOW) {
    Serial.println("Switch Pressed!");
    delay(200); // Debounce delay
 }
}
```

#### Explanation of Code:

#### 1. Pin Definitions:

 The pins are defined according to your schematic: CLK on GPIO16, DT on GPIO17, and SW on GPIO18.

#### 2. Setup Function:

• Configures the pins and initializes serial communication for debugging.

#### 3. Loop Function:

- Continuously checks for changes in the CLK pin state.
- Determines rotation direction by comparing DT and CLK.
- Updates a counter variable to track position.
- Detects button presses on the rotary encoder switch (SW) and prints a message.

#### 4. **Debouncing**:

• A small delay is added when detecting button presses to avoid multiple triggers.

This code should work seamlessly with your circuit. Upload it to your ESP32 using Arduino IDE and monitor the output via the Serial Monitor at a baud rate of 115200.

## THGE VALUE IS INCRIMENTED AUTOMATICALLY. I WANT TO CHANGE THE VALUE WHEN I ROTATE THE KNOB. CHANGE THE PROGRAM ACCORDINGLY

The issue of the value incrementing automatically might be caused by noise or improper handling of the rotary encoder signals. Below is an updated program that includes **debouncing** and ensures the value only changes when the rotary encoder is rotated.

## **Updated Code for Rotary Encoder**

```
// Pin definitions for rotary encoder
#define CLK 16 // Connected to GPI016
#define DT 17 // Connected to GPI017
#define SW 18 // Connected to GPI018
// Variables for encoder state
int lastStateCLK;
int currentStateCLK;
int counter = 0; // To track encoder position
unsigned long lastDebounceTime = 0; // For debouncing
const unsigned long debounceDelay = 5; // Debounce delay in milliseconds
void setup() {
  // Set pin modes
  pinMode(CLK, INPUT);
  pinMode(DT, INPUT);
  pinMode(SW, INPUT_PULLUP); // SW has a pull-up resistor
  // Initialize serial communication
  Serial.begin(115200);
  // Read initial state of CLK
  lastStateCLK = digitalRead(CLK);
  Serial.println("Rotary Encoder Initialized");
}
void loop() {
  // Read current state of CLK
  currentStateCLK = digitalRead(CLK);
  // Check if there is a change in state (with debouncing)
  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (currentStateCLK != lastStateCLK) {
      lastDebounceTime = millis(); // Reset debounce timer
      // Read DT to determine direction
      if (digitalRead(DT) != currentStateCLK) {
        counter++;
        Serial.println("Rotated Clockwise");
      } else {
```

```
counter--;
       Serial.println("Rotated Counterclockwise");
     }
     // Print the counter value
     Serial.print("Counter Value: ");
     Serial.println(counter);
   }
   lastStateCLK = currentStateCLK;
  }
  // Check if the switch is pressed (with simple debounce)
  if (digitalRead(SW) == LOW) {
   delay(200); // Debounce delay for switch press
   Serial.println("Switch Pressed!");
   while (digitalRead(SW) == LOW); // Wait until the switch is released
 }
}
```

#### Key Changes in the Program:

#### 1. **Debouncing**:

- Added a debounce mechanism using millis() to prevent false triggers caused by noise on the rotary encoder lines.
- Ensures that only valid state changes are processed.

#### 2. Counter Updates:

• The counter only increments or decrements when a valid rotation is detected.

#### 3. Switch Handling:

• Added a simple debounce mechanism for the push button (SW) to avoid multiple triggers when pressed.

#### 4. Serial Output:

 Prints messages to indicate clockwise or counterclockwise rotation and displays the updated counter value.

#### **Testing:**

- Upload this code to your ESP32.
- Open the Serial Monitor at 115200 baud rate.
- Rotate the rotary encoder and observe the counter value change only when you rotate it.
- Press the switch and confirm that it registers correctly without affecting the counter.

This should resolve your issue with automatic increments and ensure smooth operation of your rotary encoder.

# **CLK IS 18 DT 19 SW 5**

Based on your updated schematic, where the rotary encoder pins are connected as follows:

- CLK: GPI018
- **DT**: GPIO19
- **SW**: GPIO5

Here is the updated Arduino code for your rotary encoder functionality:

```
// Pin definitions for rotary encoder
#define CLK 18 // Connected to GPI018
#define DT 19 // Connected to GPI019
#define SW 5 // Connected to GPI05
// Variables for encoder state
int lastStateCLK;
int currentStateCLK;
int counter = 0; // To track encoder position
unsigned long lastDebounceTime = 0; // For debouncing
const unsigned long debounceDelay = 5; // Debounce delay in milliseconds
void setup() {
  // Set pin modes
  pinMode(CLK, INPUT);
  pinMode(DT, INPUT);
  pinMode(SW, INPUT_PULLUP); // SW has a pull-up resistor
  // Initialize serial communication
  Serial.begin(115200);
  // Read initial state of CLK
  lastStateCLK = digitalRead(CLK);
  Serial.println("Rotary Encoder Initialized");
}
void loop() {
  // Read current state of CLK
  currentStateCLK = digitalRead(CLK);
  // Check if there is a change in state (with debouncing)
  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (currentStateCLK != lastStateCLK) {
      lastDebounceTime = millis(); // Reset debounce timer
      // Read DT to determine direction
      if (digitalRead(DT) != currentStateCLK) {
        counter++;
        Serial.println("Rotated Clockwise");
      } else {
        counter--;
        Serial.println("Rotated Counterclockwise");
      }
```

```
// Print the counter value
Serial.print("Counter Value: ");
Serial.println(counter);
}
lastStateCLK = currentStateCLK;
}
// Check if the switch is pressed (with simple debounce)
if (digitalRead(SW) == LOW) {
    delay(200); // Debounce delay for switch press
    Serial.println("Switch Pressed!");
    while (digitalRead(SW) == LOW); // Wait until the switch is released
}
```

## Key Details:

#### 1. Pin Assignments:

• The code uses **GPIO18** for CLK, **GPIO19** for DT, and **GPIO5** for SW as per your schematic.

#### 2. Debouncing:

• A debounce mechanism is implemented to avoid false triggers due to signal noise.

#### 3. Counter Updates:

• The counter increments or decrements based on the direction of rotation.

## 4. Switch Handling:

• The push button (SW) is monitored, and a message is displayed when it is pressed.

## **Testing:**

- Upload this code to your ESP32 using the Arduino IDE.
- Open the Serial Monitor at 115200 baud rate.
- Rotate the rotary encoder and observe the counter value changing only when you rotate it.
- Press the switch (SW) and verify its functionality.

This code should now work seamlessly with your circuit as per the updated pin configuration.

\*\*

# ets Jul 29 2019 12:21:46

rst:0x1 (POWERON\_RESET),boot:0x17 (SPI\_FAST\_FLASH\_BOOT) configsip: 0, SPIWP:0xee clk\_drv:0x00,q\_drv:0x00,d\_drv:0x00,cs0\_drv:0x00,hd\_drv:0x00,wp\_drv:0x00 mode:DIO, clock div:1 load:0x3fff0030,len:4916 load:0x40078000,len:16492 load:0x40080400,len:4 load:0x40080404,len:3524 entry 0x400805b8 ets Jul 29 2019 12:21:46

THIS WHAT I GOT DISPLAYED WHEN THE BOARD WAS RUNNING ON MY SERIAL MONITOR WITH 115200 BAUD RATE . CORRECT THE ISSUES . TELL ME WHAT ITS TELLING

The output you are seeing on the Serial Monitor indicates the ESP32's boot process and memory loading sequence. Here's a breakdown of the messages and steps to resolve the issue:

## What the Serial Monitor Output Means

#### 1. Boot Message:

rst:0x1 (POWERON\_RESET),boot:0x17 (SPI\_FAST\_FLASH\_BOOT)

- rst:0x1: Indicates the ESP32 was reset due to a power-on event.
- boot:0x17: Specifies the boot mode, which in this case is SPI\_FAST\_FLASH\_BOOT (standard boot mode for ESP32).

#### 2. Configuration Details:

```
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
```

• These lines show hardware configurations for flash memory and clock settings.

#### 3. Memory Loading:

```
load:0x3fff0030,len:4916
load:0x40078000,len:16492
load:0x40080400,len:4
load:0x40080404,len:3524
entry 0x400805b8
```

• These lines indicate that sections of the program are being loaded into memory. The entry line shows the starting address of the program.

#### 4. Repetition of Boot Message:

If this message repeats frequently without running your sketch, it suggests an issue with power stability or improper code execution.

## Possible Causes

### 1. Incorrect Baud Rate:

• Ensure that the Serial Monitor baud rate matches the Serial.begin() baud rate in your code (115200 in this case).

### 2. Power Issues:

• If the ESP32 is not receiving stable power, it may repeatedly reset. Ensure your power supply (USB or battery) provides sufficient current (at least 500mA).

### 3. Faulty Code or Sketch:

• If your uploaded code has errors or infinite loops, it may cause instability.

#### 4. Hardware Connections:

• Loose or incorrect wiring can lead to resets or improper operation.

#### 5. Boot Mode Issue:

• The ESP32 might not be entering normal operation mode due to incorrect pin states during boot.

## Steps to Resolve

#### 1. Verify Baud Rate:

• Open your Arduino IDE and ensure the Serial Monitor is set to 115200.

## 2. Check Power Supply:

- Use a reliable USB cable and ensure your computer's USB port can provide sufficient current.
- If using a battery, ensure it is fully charged and capable of delivering stable voltage (3.3V for ESP32).

## 3. Reset Board:

• Press the Enable button on your ESP32 board after uploading the sketch to restart it properly.

## 4. Test with Minimal Code:

Upload a simple sketch like this to verify basic functionality:

```
void setup() {
   Serial.begin(115200);
   Serial.println("ESP32 Test");
}
void loop() {
   Serial.println("Running...");
   delay(1000);
}
```

If this works correctly, your board is functioning properly.

#### 5. Inspect Connections:

• Double-check all connections in your circuit (rotary encoder, switches, etc.) to ensure they are correct and secure.

#### 6. Update ESP32 Library:

- Ensure you have the latest ESP32 library installed in Arduino IDE.
- 7. Reinstall Drivers (if COM port issues persist):
  - If you suspect USB driver issues, install the appropriate drivers for your board (e.g., CP2102 or CH340).

## **Additional Notes**

If none of these steps resolves the issue and you continue seeing repeated boot messages, it could indicate faulty hardware or an outdated ESP32 module version with known silicon bugs (e.g., older ESP-WROOM-32). In such cases, consider using an updated module like ESP-WROOM- $32E^{[4]}$ [5].

\*\*

## **Conclusion for Week 9 Documentation**

During Week 9, you focused on input devices, specifically integrating a rotary encoder with an ESP32 microcontroller. Below is a summary of the process and achievements:

## Key Tasks and Learnings

## 1. Rotary Encoder Integration:

- You explored the functionality of a rotary encoder by studying its datasheet and understanding how the CLK, DT, and SW pins work to detect rotational direction and button presses.
- Designed a schematic for the rotary encoder using KiCad, ensuring proper pin connections and pull-up resistors.

## 2. PCB Design:

- Created a custom PCB design incorporating the ESP32-WROOM-32 chip and rotary encoder.
- Used FreeRouter in KiCad for automated routing, adjusted trace widths, and resolved DRC errors by modifying pad properties and annular ring sizes.
- Successfully milled a double-sided PCB using Gerber files converted to PNG format.

## 3. ESP32 Library Installation:

• Installed the ESP32 library in Arduino IDE using the Additional Board Manager URLs method.

• Verified the installation by running sample sketches.

#### 4. Testing and Troubleshooting:

- Initial testing of the PCB showed issues with the rotary encoder functionality due to misplaced connections.
- With instructor guidance, corrected wiring errors and tested the encoder externally to resolve issues.

#### 5. NTP Server Integration:

- Incorporated date and time functionality into your module using an NTP server.
- Adjusted GMT offset values to correctly display IST (GMT+5:30) time.

## **Challenges Faced**

- Encountered multiple errors during PCB design, including padding issues and via hole dimensions, which were resolved through research and adjustments.
- Rotary encoder testing initially failed due to incorrect circuit connections, requiring external wiring for troubleshooting.
- Faced difficulties in achieving stable time synchronization with NTP servers but resolved them by fine-tuning offset values.

## Achievements

- Successfully designed, milled, and tested a custom PCB integrating an ESP32 microcontroller.
- Implemented basic functionality of the rotary encoder after resolving connection issues.
- Integrated NTP server functionality to display accurate date and time, which aligns with your final project goal of creating a productivity timer.

## Next Steps

- Refine the rotary encoder code further to ensure smooth operation without external wiring adjustments.
- Test additional input devices or sensors as needed for your final project.
- Continue optimizing PCB designs for reliability and ease of use in future iterations.

This week provided valuable insights into circuit design, troubleshooting, and microcontroller programming, laying a strong foundation for your final project development.

## make it brief to a paragraph . just tell what all things i have done for the documentation in the documentation tone as i have specified

For Week 9, I worked on designing and implementing input devices, focusing on a rotary encoder integrated with an ESP32 microcontroller. I created a detailed schematic in KiCad, ensuring proper connections for the rotary encoder, boot and enable switches, programmable LED, and RX/TX communication. The design included decoupling capacitors and a voltage regulator for stable operation. After resolving issues like pad sizing and routing errors, I generated Gerber files and milled a double-sided PCB. I programmed the ESP32 in Arduino IDE to test the rotary encoder functionality, troubleshooting initial wiring errors and refining the code for accurate rotation detection and switch presses. Additionally, I incorporated NTP server functionality to display real-time data, aligning with my final project goals.

\*\*

go through the documentation and tell whether its ok . this week was input devices week you could go through previous year student documentations that is available on the fabacdemy website . please note on searching remind the factor that its input devices week . tell the additional details which are required . the purpose of the documentation is that . a person refering the documentation should be able to replicate the thing as it self without much difficulties , they should understand the concepts .

Your documentation for Week 9 appears to cover the key aspects of the input devices assignment, but additional details can improve its clarity and replicability. Here's an assessment and suggestions:

## **Assessment of Current Documentation**

- 1. Schematic and Design:
  - The schematic is detailed and includes all necessary components like the rotary encoder, ESP32 microcontroller, voltage regulator, and supporting circuitry.
  - Ensure you provide clear explanations for each block in the schematic (e.g., why certain resistor values were chosen or the purpose of decoupling capacitors).
- 2. PCB Design and Fabrication:

• The PCB design process is mentioned, but include details about the software used (e.g., KiCad), milling machine settings, and troubleshooting steps during fabrication.

## 3. Programming:

- The code for interfacing with the rotary encoder is provided, but ensure you explain how it works (e.g., pin assignments, logic for detecting rotation direction, debounce mechanism).
- Include a link to the source code file for easy access.

#### 4. Testing:

- Document how you tested the rotary encoder functionality (e.g., tools used, expected vs. actual results).
- Mention any challenges faced during testing (e.g., wiring errors or power issues) and how you resolved them.

#### 5. Integration with NTP Server:

- Briefly describe the purpose of integrating an NTP server and how it aligns with your final project goals.
- Include code snippets or links for this part as well.

#### 6. Hero Shot:

• Include a clear image or video of your completed PCB working with the rotary encoder to demonstrate functionality.

## Additional Details Required

To make your documentation more replicable for others:

#### 1. Step-by-Step Process:

- Provide a chronological process from schematic design to PCB fabrication and programming.
- Include specific settings (e.g., trace width in KiCad, baud rate in Arduino IDE).

#### 2. Component List:

• Add a complete list of components with specifications (e.g., resistor values, capacitor ratings).

#### 3. Troubleshooting Tips:

• Document common issues encountered (e.g., incorrect pin connections) and solutions.

#### 4. Learning Outcomes:

• Reflect on what you learned about interfacing input devices with microcontrollers.

#### 5. Links to Resources:

• Provide links to datasheets, libraries, or tutorials that helped during the process.

## Conclusion

Your documentation is well-organized but can be enhanced by adding more explanations, stepby-step procedures, and troubleshooting tips. Including original design files, source code links, and a hero shot will ensure that anyone referring to it can replicate your work without difficulty while understanding the concepts involved.

\*\*

## **Evaluation of Documentation and Suggestions for Improvement**

Your documentation provides a good foundation for replicating the rotary encoder circuit with an ESP32, but some critical details need to be clarified or expanded to ensure replicability and understanding. Here's an evaluation based on your schematic and the principles of input devices:

## Strengths

## 1. Comprehensive Schematic:

- The schematic includes all necessary components: rotary encoder, ESP32 microcontroller, voltage regulator, decoupling capacitors, and supporting circuitry.
- Blocks are visually organized, making it easy to identify connections.

## 2. Key Components:

- Decoupling capacitors are correctly placed near the ESP32 power pins to stabilize voltage and filter out noise.
- Pull-up resistors are included for the rotary encoder and switches, ensuring proper signal levels.

## 3. Integration of Input Devices:

- The rotary encoder's CLK, DT, and SW pins are connected logically to GPIO pins on the ESP32.
- The programmable LED and RX/TX headers add flexibility for debugging or future expansion.

## **Areas for Improvement**

## 1. Explanations for Each Block:

- Add detailed explanations for each block in the schematic. For example:
  - Rotary Encoder: Describe how it works (e.g., phase-shifted signals on CLK and DT) and why pull-up resistors are necessary.
  - *Voltage Regulator*: Explain its role in providing stable 3.3V power to the ESP32.

 Decoupling Capacitors: Highlight their importance in filtering noise and preventing voltage drops during high-current events.

### 2. Component Specifications:

• Include a table listing all components with their values (e.g., resistor ratings, capacitor types) and part numbers for easy sourcing.

#### 3. Step-by-Step Process:

- Expand on how to go from schematic design to PCB fabrication:
  - Mention software settings (e.g., trace width in KiCad).
  - Provide milling machine parameters (e.g., spindle speed, feed rate).
  - Include troubleshooting tips (e.g., handling DRC errors).

#### 4. Programming Details:

- Explain the logic behind the code used for the rotary encoder (e.g., detecting rotation direction using CLK/DT signals).
- Provide links or references to libraries used (e.g., ezButton for debouncing).

#### 5. Testing Procedure:

- Document how you tested the rotary encoder functionality step-by-step.
- Include expected results and how errors were resolved (e.g., wiring corrections).

#### 6. Visuals:

• Add annotated images or videos of the assembled PCB and working setup to demonstrate functionality.

## Additional Suggestions

#### 1. Concept Explanation:

- Briefly explain what a rotary encoder is and its advantages over other input devices like potentiometers.
- Discuss why decoupling capacitors are critical for microcontroller circuits.

#### 2. Code Refinement:

- Ensure the code is optimized with comments explaining each section clearly.
- Provide instructions for uploading the code to the ESP32 using Arduino IDE.

#### 3. Replication Focus:

• Ensure every step is documented so that someone unfamiliar with your setup can replicate it without ambiguity.

## Conclusion

Your documentation is well-structured but needs more detailed explanations, component specifications, step-by-step instructions, and visuals to make it fully replicable. By addressing these areas, your work will serve as a comprehensive guide for others working on input devices in Fab Academy projects.

\*\*

## 1. Schematic and Design Documentation

The schematic design for the input device week integrates a rotary encoder with an ESP32 microcontroller, along with supporting components such as a voltage regulator, decoupling capacitors, pull-up resistors, and switches. Below is the detailed documentation:

## **Schematic Overview**

The schematic consists of several functional blocks:

## 1. Rotary Encoder:

- The rotary encoder (S1: WS-ENTV\_4820XX514001) is connected to GPIO pins of the ESP32 for detecting rotation direction and button presses.
- Pull-up resistors (R1, R2, R3: 10kΩ) ensure proper signal levels for CLK, DT, and SW pins.

## 2. ESP32 Microcontroller:

- The ESP32-WROOM-32 module serves as the central processing unit.
- Essential pins like EN (enable) and RX/TX are connected for programming and debugging.

## 3. Voltage Regulator:

- A linear voltage regulator (U2: NCP1117) steps down 5V to 3.3V to power the ESP32.
- Capacitors (C1: 10μF, C3: 1μF) are placed on the input and output sides of the regulator for stability.

## 4. Decoupling Capacitors:

• A 0.1µF capacitor (C2) is placed near the ESP32 power pins to filter high-frequency noise.

## 5. Boot and Enable Switches:

- Tactile switches (SW1, SW3: Omron B3SN) are used for boot mode selection and resetting the microcontroller.
- Pull-up resistors (R4, R5:  $10k\Omega$ ) ensure stable operation.

## 6. Programmable LED:

 An LED (D1: LED\_1206) is connected to a GPIO pin via a current-limiting resistor (R6: 100Ω).

#### 7. RX/TX Header:

• A pin header (J5: FTDI-compatible) provides external communication for programming/debugging.

#### 8. Pin Headers:

• Additional pin headers (J1, J2) are included for modularity and future expansion.

## **Design Considerations**

- Pull-Up Resistors: Ensure stable HIGH logic levels for input pins.
- **Decoupling Capacitors**: Filter out noise and stabilize voltage during high-current events.
- Voltage Regulation: Protects the ESP32 by providing a steady 3.3V supply.

## 2. Additional Details for Replicability

To ensure that anyone referencing your documentation can replicate your work seamlessly, include the following details:

## **Component Specifications**

Provide a comprehensive Bill of Materials (BOM) as shown, including part numbers and datasheets for sourcing components:

- Capacitors: C1 (10µF), C2 (0.1µF), C3 (1µF)
- Resistors: R1-R5 (10kΩ), R6 (100Ω)
- Rotary Encoder: WS-ENTV\_4820XX514001
- Voltage Regulator: NCP1117
- ESP32 Module: ESP32-WROOM-32
- Switches: Omron B3SN
- LED: LED\_1206

## Step-by-Step Process

Include detailed steps from design to testing:

#### 1. Schematic Design:

- Use KiCad or similar software to create the schematic.
- Explain each component's role in the circuit.

## 2. PCB Fabrication:

- Provide Gerber files and mention milling machine settings.
- Highlight troubleshooting steps during fabrication, such as resolving DRC errors or adjusting pad sizes.
- 3. Assembly Instructions:

- Describe soldering techniques and tools used.
- Include images of assembled PCB with labeled components.

#### 4. Programming:

- Provide source code and explain its functionality.
- Include instructions for uploading code via Arduino IDE.

#### 5. Testing Procedure:

- Detail how to test each part of the circuit, such as verifying rotary encoder signals using Serial Monitor.
- Include expected results and solutions to common issues.

#### Visuals

Add annotated images or videos of:

- Schematic design
- PCB layout
- Assembled board
- Working setup with rotary encoder functionality

## **Troubleshooting Tips**

Document challenges faced during design or testing, such as incorrect wiring or unstable signals, along with solutions.

## Conclusion

Your schematic is well-organized and functional but requires more detailed explanations of each block's purpose, step-by-step replication instructions, and visuals to enhance clarity. By adding these details, your documentation will serve as a comprehensive guide for anyone attempting to replicate your work during Fab Academy's input devices week.

\*\*

- 1. https://randomnerdtutorials.com/installing-the-esp32-board-in-arduino-ide-windows-instructions/
- 2. https://www.instructables.com/Installing-the-ESP32-Board-in-Arduino-IDE-Windows-/
- 3. https://www.tutorialspoint.com/esp32\_for\_iot/installing\_the\_esp32\_board\_in\_arduino\_ide.htm
- 4. https://randomnerdtutorials.com/esp32-troubleshooting-guide/
- 5. https://github.com/espressif/arduino-esp32/issues/4404