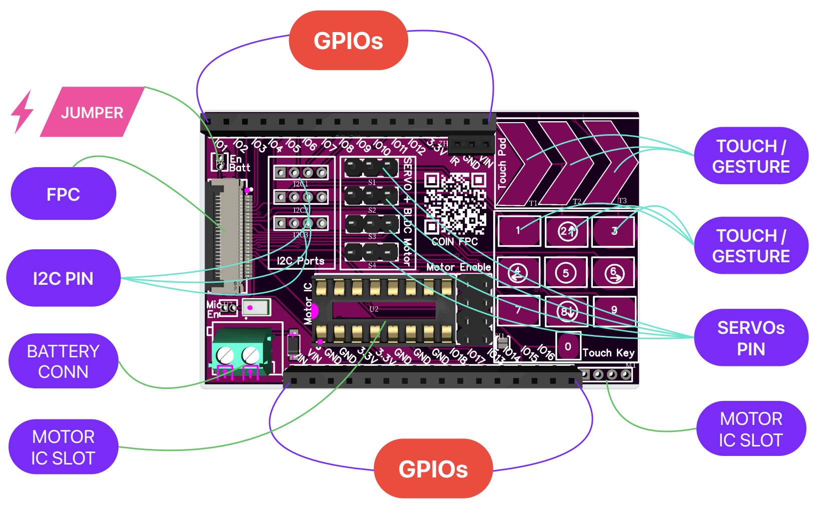
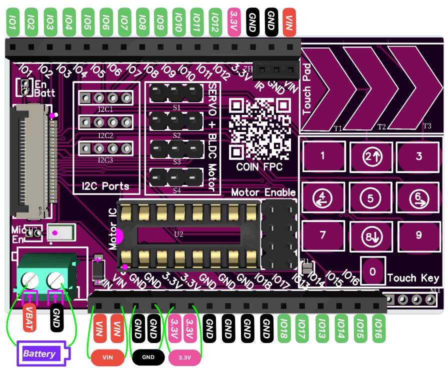
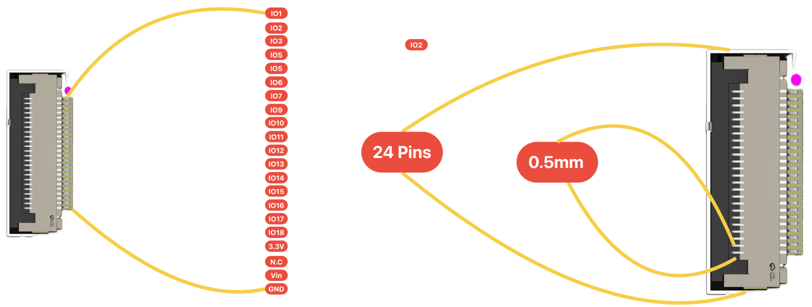
Coin FPC @IndusBoard V2


GPIOs
JUMPER
FPC
I2C PIN
BATTERY CONN
MOTOR IC SLOT
TOUCH / GESTURE
TOUCH / GESTURE
SERVOs PIN
MOTOR IC SLOT
GPIOs

Coin I/Os

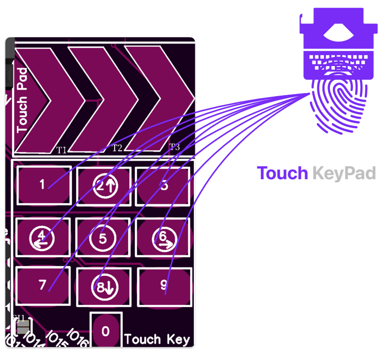

IO1
IO2
IO3
IO4
IO5
IO6
IO7
IO8
IO9
IO10
IO11
IO12
3.3V
GND
GND
VIN
I2C Ports
SERVO + BLDC Motor
COIN FPC
Motor Enable
Motor IC
Touch Pad
Touch Key
VBAT
GND
Battery
VIN
VIN
GND
GND
3.3V
3.3V
GND
GND
GND
GND
IO18
IO17
IO13
IO14
IO15
IO16
VIN
GND
3.3V

Coin FPC


IO1
IO2
IO3
IO5
IO5
IO6
IO7
IO9
IO10
IO11
IO12
IO13
IO14
IO15
IO16
IO17
IO18
3.3V
N.C
Vin
GND
IO2
24 Pins
0.5mm

Touch Pad

```
const int num_keys = 10;
int touch_pins[num_keys] = { 1, 3, 4, 7, 6, 5, 10, 11, 12,
13 };

void setup() {
  Serial.begin(115200);
  delay(1000);  // Give time to bring up serial monitor
  Serial.println("ESP32-S2 Touch Keypad Monitor");
  Serial.println("Calibrating baselines (ensure no touches
during this phase)...");
  calibrate_baselines();
  Serial.println("Monitoring touch values (Key order:
1,2,3,4,5,6,7,8,9,0):");
  Serial.println("Values:");
}

void loop() {
  // Read all touch pins and print in one line
  for (int i = 0; i < num_keys; i++) {
    uint32_t value = touchRead(touch_pins[i]);
    Serial.print(value);
    if (i < num_keys - 1) {
      Serial.print(" ");  // Space between values
    }
  }
  Serial.println();  // New line after the set
  delay(100);        // Adjust delay as needed (100ms for
~10Hz update rate)
}

void calibrate_baselines() {
  // Optional: Calibrate and print baselines (can be removed
if not needed)
  for (int i = 0; i < num_keys; i++) {
    uint32_t sum = 0;
    // Average 100 readings for stable baseline
    for (int j = 0; j < 100; j++) {
      sum += touchRead(touch_pins[i]);
      delay(2);  // Short delay between reads
    }
    uint32_t baseline = sum / 100;
    Serial.print("Baseline for pin ");
    Serial.print(touch_pins[i]);
    Serial.print(": ");
    Serial.println(baseline);
  }
  Serial.println("Calibration complete.");
}
```

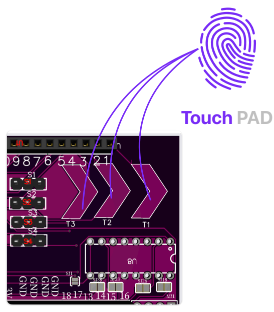

Touch Pad
1
2
3
4
5
6
7
8
9
0
Touch Key
Touch KeyPad

Pin Mapping: The touch pins are mapped to I/O pins on the board. Therefore, if you use these touch pins, the sensors connected to them will not be available during their use. Refer to the schematics design at the bottom of the datasheet for more detailed information on how the pins are multiplexed and mapped.

Sensor Drive

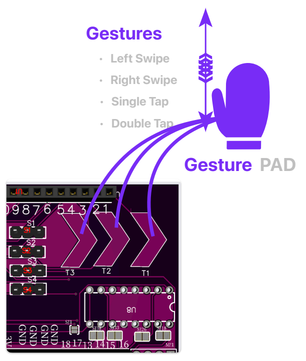
Touch Pad

```
#define TOUCH_PIN_1  1   // ESP32-S2 touch pin 1
#define TOUCH_PIN_3  3   // ESP32-S2 touch pin 3
#define TOUCH_PIN_4  4   // ESP32-S2 touch pin 4

int baseline1, baseline3, baseline4;
const int threshold = 1000; // Touch sensitivity threshold

void setup() {
    Serial.begin(115200);
    delay(1000); // Wait for stability

    // Read initial baseline values
    baseline1 = touchRead(TOUCH_PIN_1);
    baseline3 = touchRead(TOUCH_PIN_3);
    baseline4 = touchRead(TOUCH_PIN_4);

    Serial.println("Baseline Readings:");
    Serial.print("Pin 1: "); Serial.println(baseline1);
    Serial.print("Pin 3: "); Serial.println(baseline3);
    Serial.print("Pin 4: "); Serial.println(baseline4);

    Serial.println("Touch detection started...");



}



void loop() {
    int touchValue1 = touchRead(TOUCH_PIN_1);
    int touchValue3 = touchRead(TOUCH_PIN_3);
    int touchValue4 = touchRead(TOUCH_PIN_4);

    // Check if any pin is touched
    if (abs(touchValue1 - baseline1) > threshold) {
        Serial.println("Touch detected on Pin 1!");
    }
    if (abs(touchValue3 - baseline3) > threshold) {
        Serial.println("Touch detected on Pin 3!");
    }
    if (abs(touchValue4 - baseline4) > threshold) {
        Serial.println("Touch detected on Pin 4!");
    }

    delay(100); // Small delay to prevent excessive printing
}
```


Touch PAD
T3
T2
T1

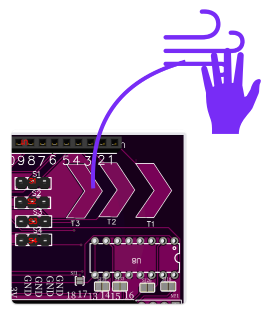
Touch Gesture

```
const int touchPins[] = {1, 3, 4};       // Touch pins
int baselineValues[3];                  // Baseline touch values
const int touchThreshold = 8000;        // Threshold for detection //
Time-related constants
const unsigned long tapTimeout = 250;    // Timeout to differentiate
taps
const unsigned long gestureTimeout = 500; // Timeout for gesture
detection
const unsigned long holdThreshold = 1000; // Threshold for tap hold
unsigned long lastTouchTime[3] = {0};   // Last touch time for each
pin
unsigned long gestureStartTime = 0;     // Start time of the current
gesture
int gestureSequence[10];                // Gesture sequence buffer
int gestureIndex = 0;                   // Current index in gesture
sequence

void setup() {
  Serial.begin(115200);
  delay(1000); // Give time to open the Serial Monitor
  Serial.println("ESP32 Refined Gesture Detection");
  // Calibrate baseline values
  for (int i = 0; i < 3; i++) {
    baselineValues[i] = touchRead(touchPins[i]);
    Serial.print("Baseline for Touch Pin ");
    Serial.print(touchPins[i]);
    Serial.print(": ");
    Serial.println(baselineValues[i]);
  }
}
void loop() {
  unsigned long currentTime = millis();
  for (int i = 0; i < 3; i++) {
    int touchValue = touchRead(touchPins[i]);
    int difference = baselineValues[i] - touchValue;
    if (abs(difference) > touchThreshold) {
      // If this is the first touch in the sequence, initialize
gesture
      if (gestureIndex == 0) {
        gestureStartTime = currentTime;
      }// Record the pin if it's a new touch
      if (currentTime - lastTouchTime[i] > tapTimeout) {
        gestureSequence[gestureIndex++] = i;
        lastTouchTime[i] = currentTime; // Prevent buffer overflow
        if (gestureIndex >= 10) {
          gestureIndex = 0;
        }}   Serial.print("Touch Detected on Pin ");
Serial.println(touchPins[i]);
    }}
  // Process gestures if enough time has elapsed
  if (gestureIndex > 0 && currentTime - gestureStartTime >
gestureTimeout) {
    processGesture();
    gestureIndex = 0; // Reset the gesture sequence
    }}
void processGesture() {
  if (gestureIndex == 1) {
    Serial.println("Single Tap Detected");
  } else if (gestureIndex == 2) {
    if (gestureSequence[0] == gestureSequence[1]) {
      Serial.println("Double Tap Detected");
    } else {
      detectSwipe();
    }
  } else if (gestureIndex >= 3) {
    detectSwipe();
  }}
void detectSwipe() {// Check for left and right swipes based on pin
order
  bool isLeftSwipe = (gestureSequence[0] == 0 && gestureSequence[1]
== 1 && gestureSequence[2] == 2);
  bool isRightSwipe = (gestureSequence[0] == 2 && gestureSequence[1]
== 1 && gestureSequence[2] == 0);
  if (isLeftSwipe) {
    Serial.println("Left Swipe Detected");
  } else if (isRightSwipe) {
    Serial.println("Right Swipe Detected");
  } else {
    Serial.println("Unknown Gesture Detected");
  }}
```


Gestures
Left Swipe
Right Swipe
Single Tap
Double Tap
Gesture PAD
T3
T2
T1

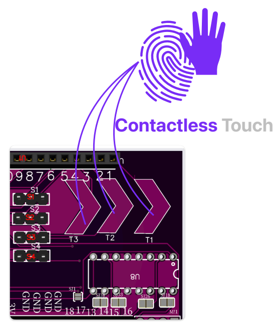
Proximity Sensing

```
#include <driver/touch_pad.h>
// Define the LED pin
const int ledPin = 33;
// Variable to keep track of the LED state
volatile bool isLedOn = false;
void setup() {
  Serial.begin(115200);
  delay(1000);
  // Set the LED pin as output
  pinMode(ledPin, OUTPUT);
  digitalWrite(ledPin, LOW); // Ensure LED is off initially
  // Initialize touch pins
  touchAttachInterrupt(4, onTouch, 20); // Adjust threshold for
proximity
  // touchAttachInterrupt(3, onTouch, 35);
  // touchAttachInterrupt(4, onTouch, 24);
  Serial.println("Touch sensor initialized for proximity
detection.");
}
void onTouch() {
  // Toggle the LED state
  isLedOn = !isLedOn;
  digitalWrite(ledPin, isLedOn ? HIGH : LOW);
  // Print the current state of the LED
  Serial.print("Proximity detected! LED is now ");
  Serial.println(isLedOn ? "ON" : "OFF");
}

void loop() {
  // Main loop is not needed for toggling since the logic is handled
in the interrupt
}
```



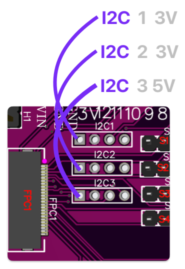
Contactless Touch

```
#include <driver/touch_pad.h>

void setup() {
  Serial.begin(115200);
  delay(1000);

  // Initialize touch pins
  touchAttachInterrupt(1, onTouch, 25); // Adjust threshold for
proximity
  touchAttachInterrupt(3, onTouch, 25);
  touchAttachInterrupt(4, onTouch, 25);

  Serial.println("Touch sensor initialized for proximity
detection.");
}

void onTouch() {
  // Logic for detecting touch or proximity
  Serial.println("Proximity detected!");
}

void loop() {
  // Your main loop logic here
}
```


Contactless Touch

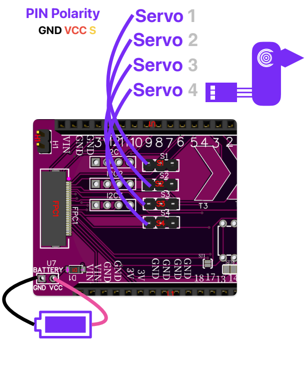
I2C Port

```
#include <Wire.h>

void setup() {
  Serial.begin(115200);
  while (!Serial);  // Wait for Serial Monitor
  Serial.println("\nI2C Scanner Initialized!");

  Wire.begin();  // Default I2C pins (SDA: GPIO8, SCL: GPIO9 on
ESP32-S2)
}

void loop() {
  Serial.println("Scanning for I2C devices...");

  byte count = 0;
  for (byte address = 1; address < 127; address++) {
    Wire.beginTransmission(address);
    if (Wire.endTransmission() == 0) {  // Device found
      Serial.print("I2C Device found at 0x");
      Serial.println(address, HEX);
      count++;
    }
    delay(10);
  }

  if (count == 0) {
    Serial.println("No I2C devices found!");
  } else {
    Serial.print("Total devices found: ");
    Serial.println(count);
  }

  Serial.println("\nScanning again in 5 seconds...\n");
  delay(5000);  // Wait before next scan
}
```


I2C 1 3V
I2C 2 3V
I2C 3 5V
I2C1
I2C2
I2C3
FPC1

Servo Driver

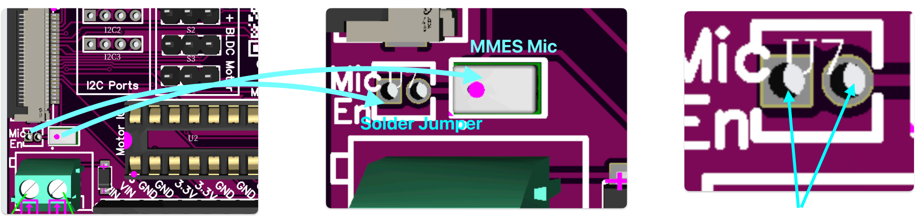
```
#include <Servo.h>

// Define servo objects
Servo servo1, servo2, servo3, servo4;

// Define servo pins
#define SERVO_PIN1 13
#define SERVO_PIN2 14
#define SERVO_PIN3 17
#define SERVO_PIN4 18

void setup() {
  Serial.begin(115200);

  // Attach each servo to its respective pin
  servo1.attach(SERVO_PIN1);
  servo2.attach(SERVO_PIN2);
  servo3.attach(SERVO_PIN3);
  servo4.attach(SERVO_PIN4);
}

void loop() {
  // Sweep all servos from 0° to 180° simultaneously
  for (int angle = 0; angle <= 180; angle++) {
    servo1.write(angle);
    servo2.write(angle);
    servo3.write(angle);
    servo4.write(angle);
    Serial.print("Angle: "); Serial.println(angle);
    delay(15);  // Delay for smooth movement
  }

  // Sweep all servos from 180° back to 0° simultaneously
  for (int angle = 180; angle >= 0; angle--) {
    servo1.write(angle);
    servo2.write(angle);
    servo3.write(angle);
    servo4.write(angle);
    Serial.print("Angle: "); Serial.println(angle);
    delay(15);
  }
}
```


PIN Polarity
GND VCC S
Servo 1
Servo 2
Servo 3
Servo 4
BATTERY
GND VCC

MEMS MIC

NOTE: The MEMS microphone used here has a built-in small amplifier circuit, and the IO pin is internally connected to pin 11 through a solder jumper.
In normal condition, it can be used to detect sound frequency and amplitude. It is suitable for detecting noise, classifying sounds using AI training, and small projects.
However, if you want to use it as a fully functional microphone for voice recognition or more accurate microphone-based projects, then you can add an external pre-amp circuit and noise filtration circuit. We have provided an option for that.
You can solder a header pin or wire to the first pad of the solder jumper and connect your external amplifier and noise filtration circuit for crystal-clear and high-quality audio output. The output of your amplifier or noise filtration circuit can be connected to any ADC pin or to the second solder pad.

Preparing MIC


I2C Ports
MMES Mic
Solder Jumper
Mic
En

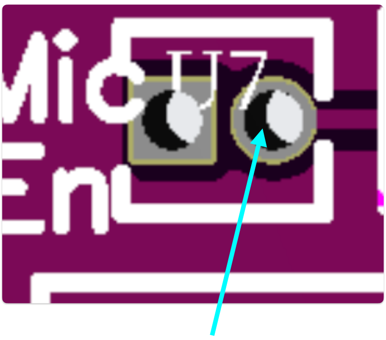
Short The Solder Jumper Using Wire

```
#define MIC_PIN 11          // ADC1_CH5 (GPIO 11) - connected to mic OUT on ESP32-S2
#define SAMPLE_RATE 10000   // Hz (samples per second; increase for faster plot, but Serial may lag)
#define BUFFER_SIZE 128     // Samples for RMS (higher = smoother envelope)
#define SAMPLING_INTERVAL_US (1000000 / SAMPLE_RATE)  // Microseconds between samples

int16_t buffer[BUFFER_SIZE];
int bufferIndex = 0;
float rmsValue = 0;

void setup() {
  Serial.begin(115200);
  analogReadResolution(12);        // 12-bit ADC (0-4095)
  analogSetAttenuation(ADC_11db);  // 0-3.3V range
  // Serial Plotter auto-starts; no text needed
}

void loop() {
  unsigned long startTime = micros();

  // Read sample
  int rawADC = analogRead(MIC_PIN);  // 0-4095
  int16_t amplitude = rawADC - 2048;  // De-bias (AC signal: -2048 to +2047)

  // Add to buffer for RMS
  buffer[bufferIndex] = amplitude;
  bufferIndex++;

  // Calculate RMS when buffer full
  if (bufferIndex >= BUFFER_SIZE) {
    bufferIndex = 0;
    long sumSquares = 0;
    for (int i = 0; i < BUFFER_SIZE; i++) {
      sumSquares += (long)buffer[i] * buffer[i];
    }
    rmsValue = sqrt((float)sumSquares / BUFFER_SIZE);
  }

  // Output for Serial Plotter: Waveform, then RMS (comma-separated for 2 traces)
  Serial.print(amplitude);  // Trace 1: Instantaneous wave/noise
  Serial.print(",");
  Serial.println(rmsValue);  // Trace 2: Smoothed envelope

  // For SINGLE TRACE (just waveform): Uncomment below, comment above
  // Serial.println(amplitude);

  // Timing loop
  while (micros() - startTime < SAMPLING_INTERVAL_US) {
    // Busy wait
  }
}
```

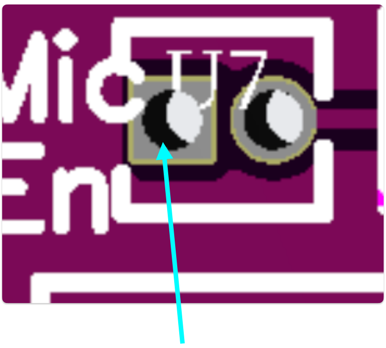
Amplifier Circuit Connection

NOTE: If you want to use it as a fully functional microphone for voice recognition or more accurate microphone-based projects, then you can add an external pre-amp circuit and noise filtration circuit. We have provided an option for that.

You can solder a header pin or wire to the first pad of the solder jumper and connect your external amplifier and noise filtration circuit for crystal-clear and high-quality audio output. The output of your amplifier or noise filtration circuit can be connected to any ADC pin or to the second solder pad.

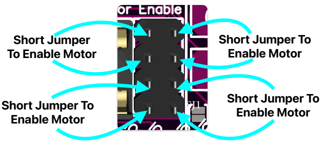


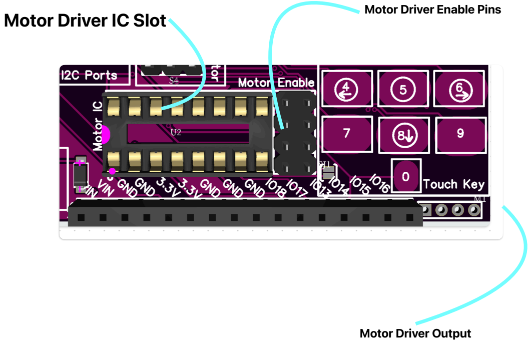
Connect the mic output to amplifier/noise filtration input circuit



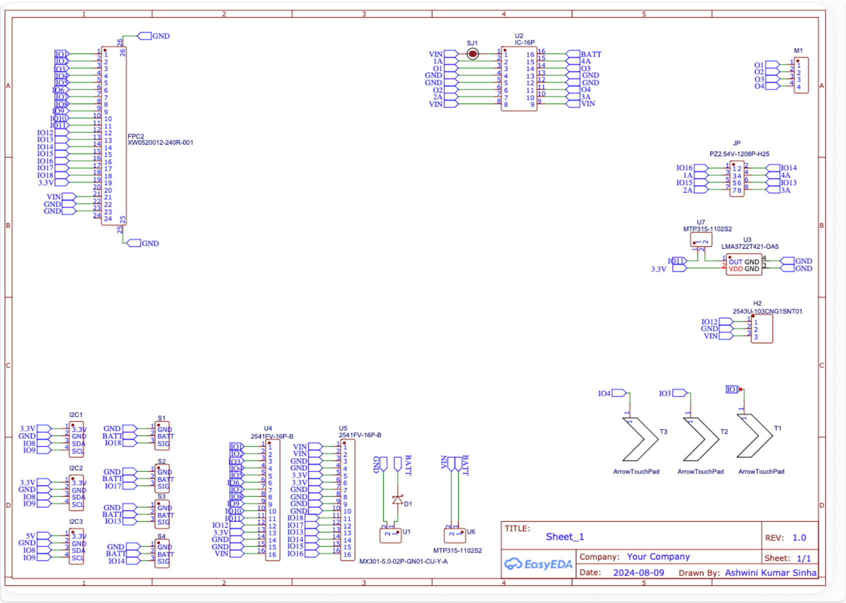
Connect the amplifier module output to other solder pin or any free GPIO on the board

Motor Driver


or Enable
Short Jumper To Enable Motor
Short Jumper To Enable Motor
Short Jumper To Enable Motor
Short Jumper To Enable Motor


Motor Driver IC Slot
Motor Driver Enable Pins
I2C Ports
S4
Motor Enable
Motor IC
U2
4
5
6
7
8
9
0
Touch Key
VIN
GND
GND
3.3V
3.3V
GND
GND
GND
IO18
IO17
IO13
IO14
IO15
IO16
Motor Driver Output

Coin FPC @IndusBoard V2 Pin Map


FPC2
XW0520012-240R-001
GND
IO1
IO2
IO3
IO4
IO5
IO6
IO7
IO8
IO9
IO10
IO11
IO12
IO13
IO14
IO15
IO16
IO17
IO18
3.3V
VIN
GND
GND
SJ1
U2
IC-16P
VIN
1A
O1
GND
GND
O2
2A
VIN
BATT
4A
O3
GND
GND
O4
3A
VIN
M1
O1
O2
O3
O4
JP
PZ2.54V-1208P-H25
IO16
1A
IO15
2A
IO14
4A
IO13
3A
U7
MTP315-1102S2
U3
LMA3722T421-OA5
IO11
3.3V
OUT GND
VDD GND
GND
GND
H2
2543U-103CNG1SNT01
IO12
GND
VIN
IO4
IO3
IO1
T3
T2
T1
ArrowTouchPad
ArrowTouchPad
ArrowTouchPad
I2C1
I2C2
I2C3
3.3V
GND
IO8
IO9
SDA
SCL
5V
S1
S2
S3
S4
GND
BATT
SIG
IO18
IO17
IO13
IO14
U4
2541FV-16P-B
U5
2541FV-16P-B
D1
U1
U6
MX301-5.0-02P-GN01-CU-Y-A
MTP315-1102S2
TITLE: Sheet_1
REV: 1.0
EasyEDA
Company: Your Company
Sheet: 1/1
Date: 2024-08-09
Drawn By: Ashwini Kumar Sinha