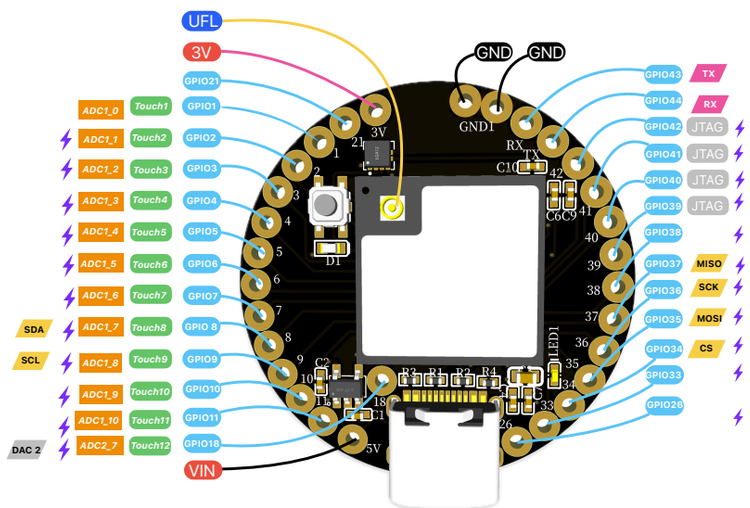
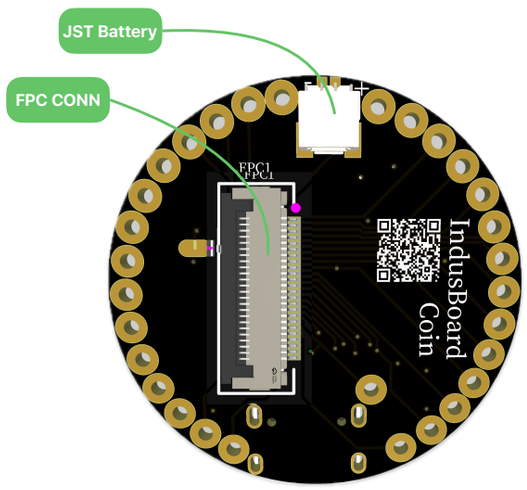
Coin V2

ADC Analoug To Digital Pins
Touch Capacitive Touch Sensor
PWM Pins
Peripheral Pins
JTAG JTAG Pins
Serial Port Pins
IO GPIO

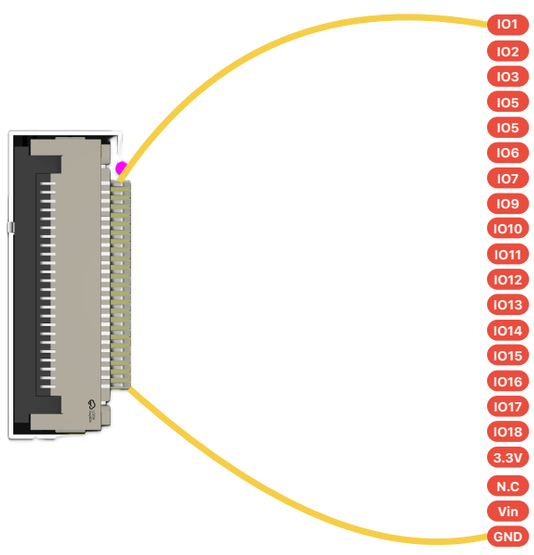
Onboard Sensors
- 3D Accelerometer
- 3D Magnetometer
- Temperature sensor
- 14 Capacitive Touch Sensor

Features
- 32-bit LX7 microcontroller
- Camera Interface
- ULP RISC-V Processor
- 4 MB flash and optional 2 MB PSRAM in chip package

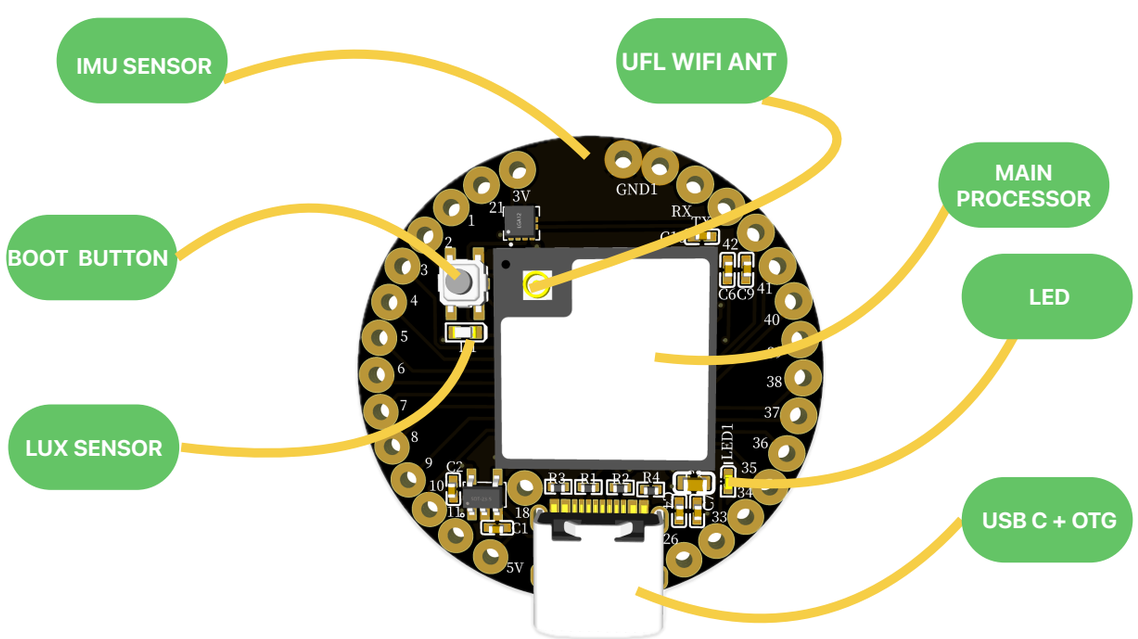


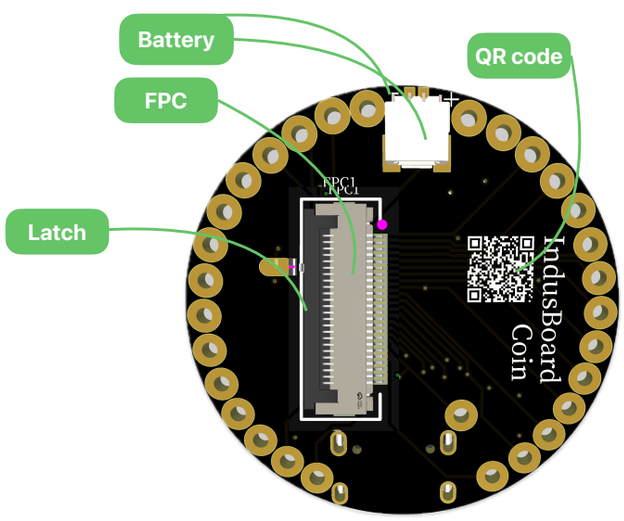


Coin FPC

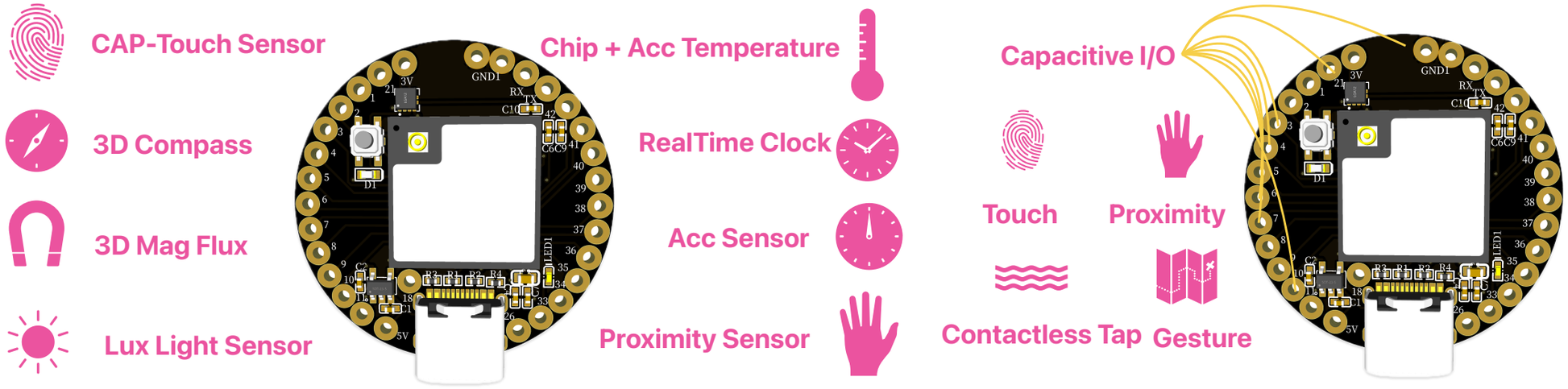


Coin Components

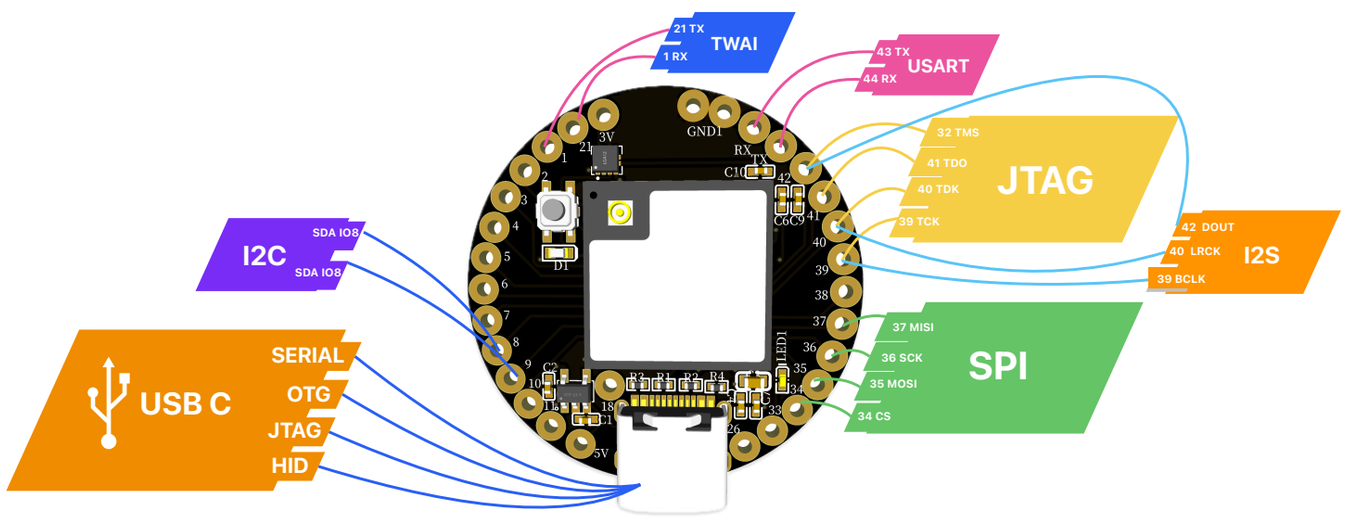




Coin Sensors



COIN Default Peripherals



SPI – Default pins: `SCK (36)`, `MISO (37)`, `MOSI (35)`, `CS (34)`

I2S – Audio interface (default for MIC: `BCLK (39)`, `LRCLK (40)`, `DOUT (42)`)

JTAG – Debugging interface (`TCK (39)`, `TDI (40)`, `TDO (41)`, `TMS (42)`)

CAN Bus (TWAI) – `TX (21)`, `RX (20)`, requires an external transceiver.

ADC (Analog Inputs) – `ADC1_CH0 to ADC1_CH9` (`GPIO1–10`)

I2C – `SDA (8)`, `SCL (9)` (default, but configurable)

Touch Sensors – `T0–T9` (e.g., `T4 = GPIO4`)

PWM (LEDC) – Available on most GPIOs for controlling LEDs, motors, etc.

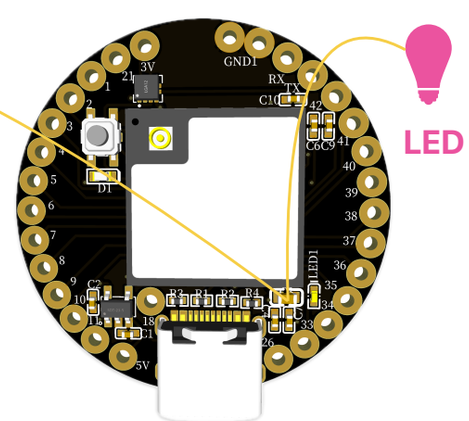
UART – `TX (43)`, `RX (44)` (default, software remappable)

Note: COIN I/O are highly flexible, and any of the pins can be used for any peripheral by defining the pin in the code above is only the default peripheral.

COIN Driving Codes

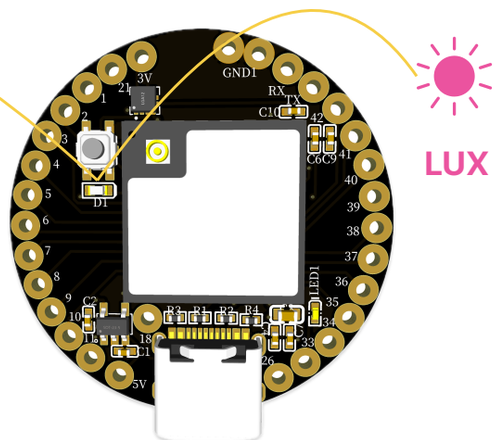
LED Drive

```
#define LED_PIN 33  // LED connected to pin 33

// PWM settings
#define PWM_CHANNEL  0
#define PWM_FREQ     5000  // 5 kHz frequency
#define PWM_RES      8     // 8-bit resolution (0-255)

void setup() {
    // Configure PWM
    ledcSetup(PWM_CHANNEL, PWM_FREQ, PWM_RES);
    ledcAttachPin(LED_PIN, PWM_CHANNEL);
}

void loop() {
    // Fade in
    for (int brightness = 0; brightness <= 255; brightness += 5) {
        ledcWrite(PWM_CHANNEL, brightness);
        delay(20);
    }

    // Fade out
    for (int brightness = 255; brightness >= 0; brightness -= 5) {
        ledcWrite(PWM_CHANNEL, brightness);
        delay(20);
    }
}
```



Lux/Light Sensor

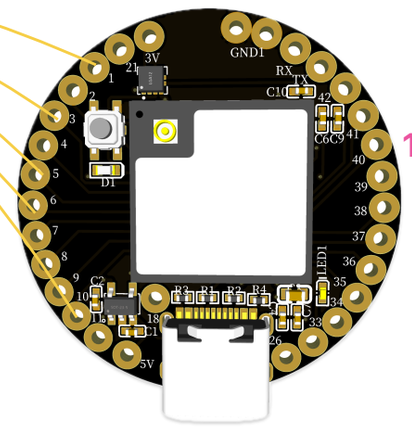
```
#define LIGHT_SENSOR_PIN 2  // XYC-PT19C-L1 connected to pin 3

void setup() {
    Serial.begin(115200);
    pinMode(LIGHT_SENSOR_PIN, INPUT);
}

void loop() {
    int adcValue = analogRead(LIGHT_SENSOR_PIN);  // Read raw ADC
value
    float voltage = adcValue * (3.3 / 4095.0);    // Convert ADC to
voltage (assuming 3.3V reference)

    // Approximate ADC to Lux mapping
    float lux = (adcValue / 4095.0) * 1000.0;  // Adjust based on
calibration

    Serial.print("ADC Value: ");
    Serial.print(adcValue);
    Serial.print(" | Voltage: ");
    Serial.print(voltage, 3);
    Serial.print("V | Lux: ");
    Serial.println(lux);

    delay(500);  // Delay for readability
}
```



Note:- Add 10K Resistor Between IO2 & GND To Activate Sensor

CAP-Touch Sensor

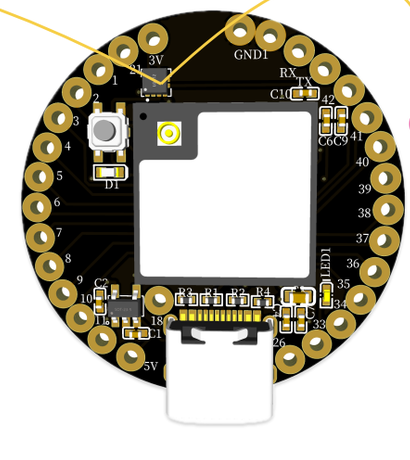
```python
void setup() {
  Serial.begin(115200);
  delay(1000); // Time to bring up Serial Monitor
  Serial.println("ESP32 Capacitive Touch Test");
}

// Store baseline values for each pin
int baseline[10];

void loop() {
  int touchPins[] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}; // ESP32 touch
pins

  for (int i = 0; i < 10; i++) {
    int currentValue = touchRead(touchPins[i]);

    // Initialize baseline values
    if (baseline[i] == 0) {
      baseline[i] = currentValue;
    }

    // Detect significant touch event (±1000 change)
    if (abs(currentValue - baseline[i]) > 1000) {
      Serial.print("Touch detected on pin: ");
      Serial.println(touchPins[i]);
    }
  }

  delay(100);
}
```



10 + Touch Sensor

Compass Sensor

```python
// Includes.
#include <LSM303AGR_MAG_Sensor.h>
#include <math.h>
#if defined(ARDUINO_SAM_DUE)
#define DEV_I2C Wire1   // Define which I2C bus is used. Wire1 for
the Arduino Due
#define SerialPort Serial
#else
#define DEV_I2C Wire    // Or Wire
#define SerialPort Serial
#endif
// Magnetometer sensor.
LSM303AGR_MAG_Sensor Mag(&DEV_I2C);
void setup() {
  SerialPort.begin(115200);
  // Initialize I2C bus.
  DEV_I2C.begin();
  // Initialize and enable magnetometer.
  Mag.begin();
  Mag.Enable();
}
void loop() {
  // Read magnetometer data.
  int32_t magnetometer[3];
  Mag.GetAxes(magnetometer);
  // Calculate heading in degrees.
  float heading = atan2(magnetometer[1], magnetometer[0]) * 180.0 /
M_PI;
  // Convert heading to range 0-360 degrees.
  if (heading < 0) {
    heading += 360;
  }
  // Output data.
  SerialPort.print("Compass Heading: ");
  SerialPort.print(heading, 2);
  SerialPort.println(" degrees");
  delay(500);
}
```

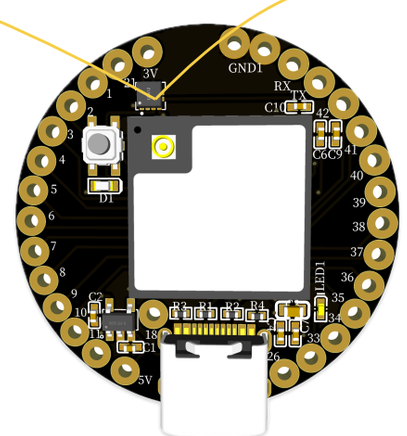


Compass

Note:- Add the "LSM303AGR(https://github.com/stm32duino/LSM303AGR)" Library to drive sensor

3D Mag Flux Sensor

```python
#include <LSM303AGR_MAG_Sensor.h>

#define DEV_I2C Wire  // Define I2C bus
LSM303AGR_MAG_Sensor Mag(&DEV_I2C);

// Define the area over which the flux is measured (in square
meters)
#define SENSOR_AREA 0.0001  // Example: 10mm x 10mm = 100 mm² =
0.0001 m²

void setup() {
  Serial.begin(9600);
  DEV_I2C.begin();

  // Initialize the magnetometer
  Mag.begin();
  Mag.Enable();
}

void loop() {
  int32_t magnetometer[3];
  Mag.GetAxes(magnetometer);  // Get raw magnetic field data (in
mGauss)

  // Convert mGauss to Tesla (1 mGauss = 1e-7 Tesla)
  float Bx = magnetometer[0] * 1e-7;
  float By = magnetometer[1] * 1e-7;
  float Bz = magnetometer[2] * 1e-7;

  // Calculate magnetic flux (Φ = B * A)
  float flux_x = Bx * SENSOR_AREA;
  float flux_y = By * SENSOR_AREA;
  float flux_z = Bz * SENSOR_AREA;

  // Total flux (assuming uniform field)
  float total_flux = sqrt(flux_x * flux_x + flux_y * flux_y + flux_z
* flux_z);

  // Display results
  Serial.print("Magnetic Flux (Weber) - X: ");
  Serial.print(flux_x, 8);
  Serial.print(" Wb, Y: ");
  Serial.print(flux_y, 8);
  Serial.print(" Wb, Z: ");
  Serial.print(flux_z, 8);
  Serial.print(" Wb, Total: ");
  Serial.print(total_flux, 8);
  Serial.println(" Wb");

  delay(500);
}
```

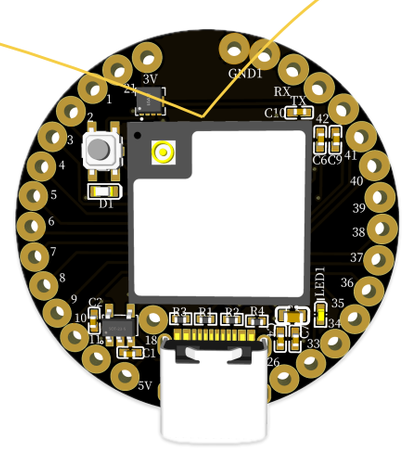


MAG Flux

Note:- Add the "LSM303AGR(https://github.com/stm32duino/LSM303AGR)" Library to drive sensor

3D ACC Sensor

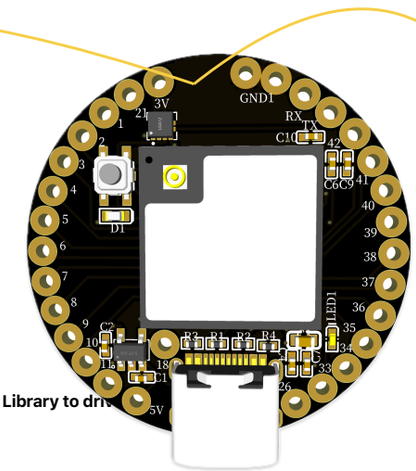
```python
#include <LSM303AGR_ACC_Sensor.h>

#define DEV_I2C Wire  // Define I2C bus
LSM303AGR_ACC_Sensor Acc(&DEV_I2C);
// Sensitivity for LSM303AGR (for ±2g mode: 1 mg/digit)
#define ACC_SENSITIVITY 0.001  // 1 mg = 0.001 g

void setup() {
  Serial.begin(115200);
  DEV_I2C.begin();

  // Initialize the accelerometer
  Acc.begin();
  Acc.Enable();
}

void loop() {
  int32_t accelerometer[3];
  Acc.GetAxes(accelerometer);  // Get raw acceleration data

  // Convert raw values to 'g' (gravity acceleration)
  float Ax = accelerometer[0] * ACC_SENSITIVITY;
  float Ay = accelerometer[1] * ACC_SENSITIVITY;
  float Az = accelerometer[2] * ACC_SENSITIVITY;

  // Display results
  Serial.print("Acceleration (g) - X: ");
  Serial.print(Ax, 4);
  Serial.print(", Y: ");
  Serial.print(Ay, 4);
  Serial.print(", Z: ");
  Serial.print(Az, 4);
  Serial.println();
  delay(500);
}
```



Acc

ACC Temp Sensor

```python
#include <LSM303AGR_ACC_Sensor.h>
#define DEV_I2C Wire  // Define I2C bus
LSM303AGR_ACC_Sensor Acc(&DEV_I2C);
// Sensitivity for LSM303AGR (for ±2g mode: 1 mg/digit)
void setup() {
  Serial.begin(115200);
  DEV_I2C.begin();
  // Initialize the accelerometer
  Acc.begin();
  Acc.EnableTemperatureSensor();  // Enable temperature sensor
}
void loop() {
  float temperature;
  Acc.GetTemperature(&temperature); // Get temperature data
  Serial.print(" | Temp (°C): ");
  Serial.print(temperature, 2);
  Serial.println();

  delay(500);
}
```

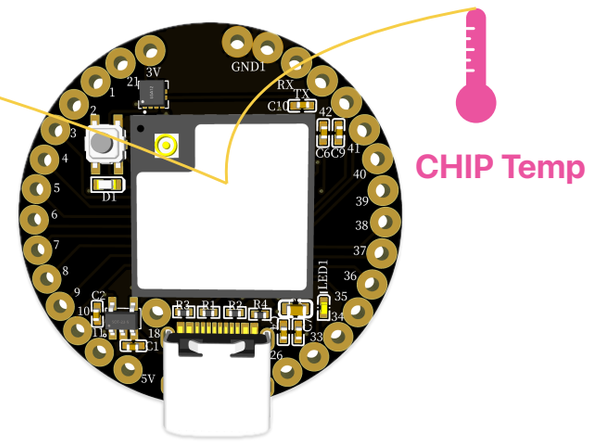


ACC Temp

Note:- Add the "LSM303AGR(https://github.com/stm32duino/LSM303AGR)" Library to dri

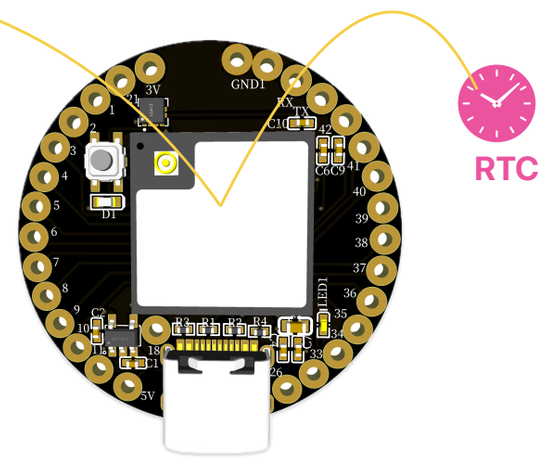
CHIP Temp Sensor

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

void loop() {
    float temperature = temperatureRead(); // Reads internal
temperature
    Serial.print("Internal Chip Temperature: ");
    Serial.print(temperature);
    Serial.println(" °C");
    delay(1000);
}
```



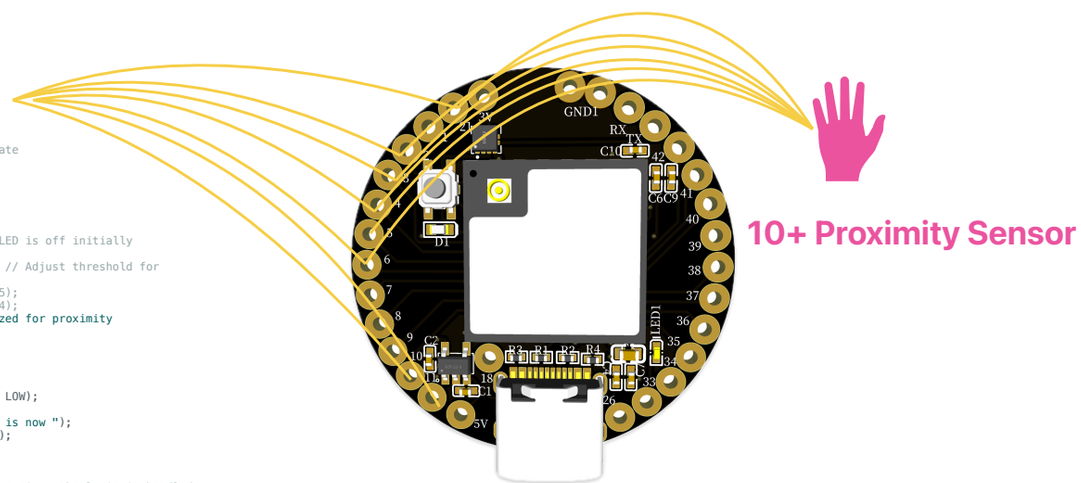
RTC Clock

```
#include <time.h>

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

    // Set the RTC time manually (YYYY, MM, DD, HH, MM, SS)
    struct tm t = {0};
    t.tm_year = 2024 - 1900;  // Year (since 1900)
    t.tm_mon = 1;             // Month (0-11, January = 0)
    t.tm_mday = 20;           // Day
    t.tm_hour = 10;           // Hours (0-23)
    t.tm_min = 30;            // Minutes
    t.tm_sec = 0;             // Seconds

    time_t t_of_day = mktime(&t);
    struct timeval now = { .tv_sec = t_of_day };
    settimeofday(&now, NULL); // Set system time

    Serial.println("RTC Time Set!");
}

void loop() {
    struct tm timeinfo;
    time_t now;
    time(&now);
    localtime_r(&now, &timeinfo);

    Serial.print("Time: ");
    Serial.print(timeinfo.tm_hour);
    Serial.print(":");
    Serial.print(timeinfo.tm_min);
    Serial.print(":");
    Serial.println(timeinfo.tm_sec);

    delay(1000);
}
```



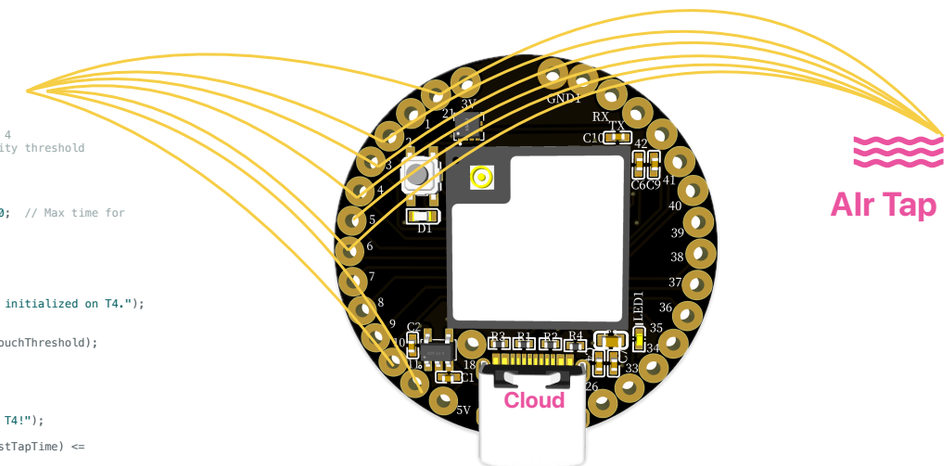
Proximity Sensor

```
#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
}
```

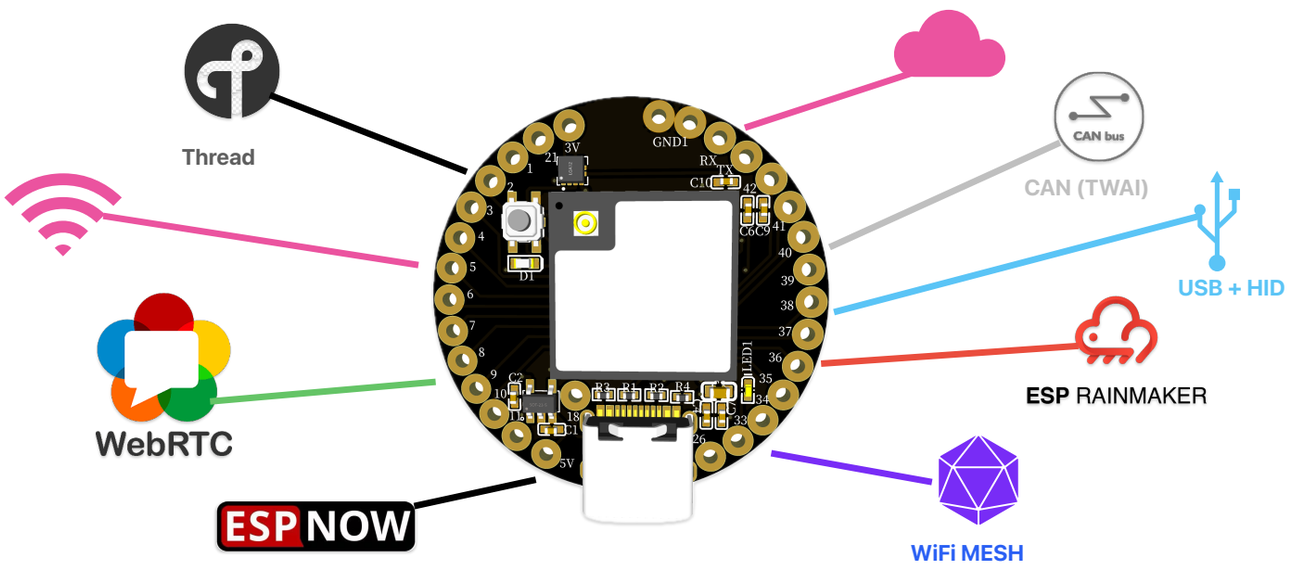


Contactless Tap

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

const int touchPin = 4;  // Using Touch Pin 4
const int touchThreshold = 30;  // Sensitivity threshold

// Tap detection variables
unsigned long lastTapTime = 0;
bool firstTapDetected = false;
const unsigned long doubleTapThreshold = 500;  // Max time for
double-tap (ms)

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

  Serial.println("Contactless tap detection initialized on T4.");

  // Attach interrupt to touch pin 4
  touchAttachInterrupt(touchPin, onTouch, touchThreshold);
}

void onTouch() {
  unsigned long currentTime = millis();

  Serial.println("Proximity tap detected on T4!");

  if (firstTapDetected && (currentTime - lastTapTime) <=
doubleTapThreshold) {
    Serial.println("Double-tap detected!");
    firstTapDetected = false;  // Reset for next detection
  } else {
    firstTapDetected = true;
    lastTapTime = currentTime;
  }
}

void loop() {
  // Nothing needed here, interrupts handle detection
}
```



COIN Connectivity



COIN Power Consumption

Power Consumption in Sleep Modes

Mode	Power Consumption (mA)	Description
Active Mode	**90-200mA**	Running at full speed (160MHz)
Idle Mode	**20-40mA**	CPU in idle state, no major processing
Light Sleep	**0.8-4mA**	CPU stopped, but peripherals active
Deep Sleep	**10-50μA**	CPU, most peripherals, and Wi-Fi off

📌 Deep sleep is the lowest power mode, consuming only ~10-50μA.

Power Consumption With WiFi

Wi-Fi Mode	Power Consumption (mA)	Description
Wi-Fi OFF	**20-40mA**	No Wi-Fi activity, only MCU running
Station Mode (STA) Connected	**50-120mA**	Connected to an access point
AP Mode (Soft Access Point)	**80-150mA**	Creating its own Wi-Fi network
STA + AP Mode (Both Active)	**120-250mA**	Running as both client and access point
Wi-Fi Transmission (TX Peak)	**250-350mA**	Sending data at max power
Wi-Fi Reception (RX)	**80-120mA**	Receiving data

📌 Wi-Fi transmission (TX) consumes the most power (~250-350mA).
📌 Using sleep modes with Wi-Fi reduces power consumption.

Power Consumption Idle

State	Power Consumption (mA)	Description
No Wi-Fi, CPU Running	**20-40mA**	Just running code, no connectivity
No Wi-Fi, Light Sleep	**0.8-4mA**	CPU sleeping, some peripherals on
No Wi-Fi, Deep Sleep	**10-50μA**	Everything off,

Power Consumption Sensors

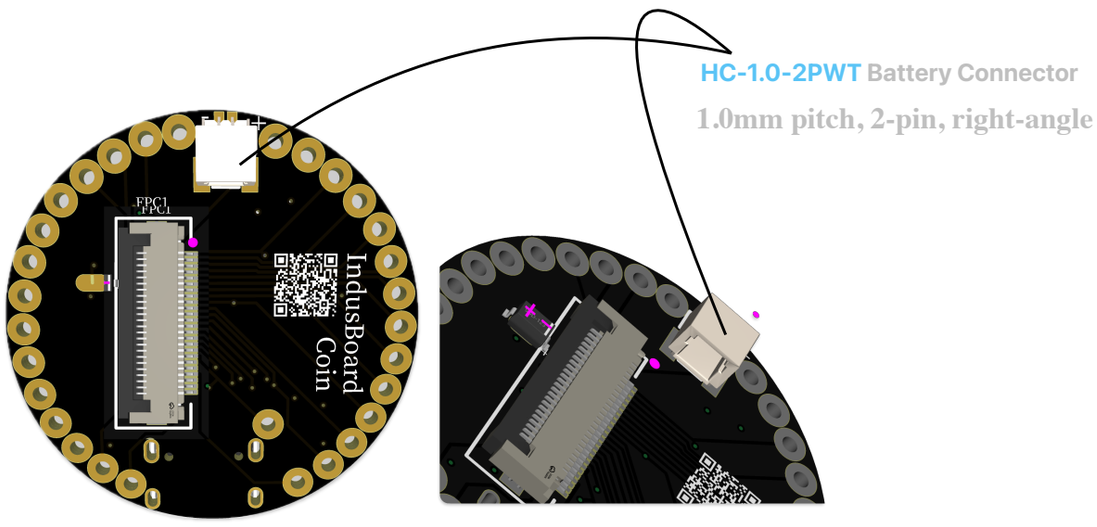
Sensor Mode	Power Consumption (μA / mA)	Description
Accelerometer Active	**130-180μA**	Normal operation
Accelerometer Low-Power	**20-50μA**	Lower resolution mode
Magnetometer Active	**100-110μA**	Normal magnetic field sensing
Magnetometer Low-Power	**10-20μA**	Reduced accuracy
Accelerometer + Magnetometer Active	**200-300μA**	Both sensors active
Standby Mode	**1-5μA**	Lowest power mode

COIN Pins

Pin	GPIO	Function	Additional Notes
1	**GPIO1**	ADC1_CH0, RTC_IO, Touch1	Analog input, touch sensor
2	**GPIO2**	ADC1_CH1, RTC_IO, Touch2	Analog input, touch sensor
3	**GPIO3**	ADC1_CH2, RTC_IO, Touch3	Analog input, touch sensor
4	**GPIO4**	ADC1_CH3, RTC_IO, Touch4	Analog input, touch sensor
5	**GPIO5**	ADC1_CH4, RTC_IO, Touch5	Analog input, touch sensor
6	**GPIO6**	ADC1_CH5, RTC_IO, Touch6	Analog input, touch sensor
7	**GPIO7**	ADC1_CH6, RTC_IO, Touch7	Analog input, touch sensor
8	**GPIO8**	ADC1_CH7, RTC_IO, Touch8, **I2C SDA**	Analog input, I2C SDA
9	**GPIO9**	ADC1_CH8, RTC_IO, Touch9, **I2C SCL**	Analog input, I2C SCL
10	**GPIO10**	ADC1_CH9, RTC_IO, Touch10	Analog input, touch sensor
11	**GPIO18**	PWM, RTC_IO, UART, I2C	General I/O, PWM
12	**GPIO21**	TWAI (CAN) TX	CAN TX pin
13	**GPIO26**	PWM, RTC_IO	General I/O, PWM
14	**GPIO33**	PWM, RTC_IO	General I/O, PWM
15	**GPIO34**	SPI CS, RTC_IO	SPI chip select (CS)
16	**GPIO35**	SPI MOSI, RTC_IO	SPI Master Out, Slave In (MOSI)
17	**GPIO36**	SPI SCK, RTC_IO	SPI Clock (SCK)

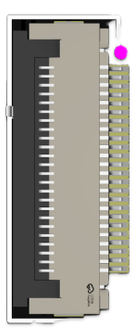
18	**GPIO37**	SPI MISO, RTC_IO	SPI Master In, Slave Out (MISO)
19	**GPIO38**	General I/O	No specific function
20	**GPIO39**	I2S BCLK, JTAG TCK	I2S Clock, JTAG debug
21	**GPIO40**	I2S LRCLK, JTAG TDI	I2S L/R Clock, JTAG debug
22	**GPIO41**	I2S DOUT, JTAG TDO	I2S Data Out, JTAG debug
23	**GPIO42**	I2S DIN	I2S Data In
24	**GPIO43**	UART TX, RTC_IO	Default TX for serial
25	**GPIO44**	UART RX, RTC_IO	Default RX for serial
26	**GPIO20**	TWAI (CAN) RX	CAN RX pin
27	**GPIO32**	JTAG TMS	JTAG debug pin
28	**GND**	Ground	Power ground
29	**3.3V**	Power Supply	3.3V power output
30	**Vin**	Power Input	5V power input

COIN JST Connector



Compatible Connector

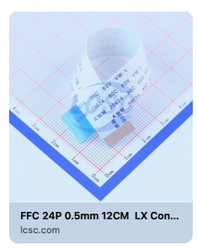
FPC Connector



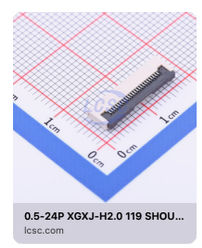
FPC Connector. 24Pin 0.5mm Pitch

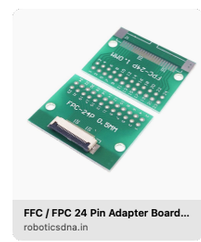
Ribbon Cable 24Pin 0.5mm Pitch

Compatible Ribbon

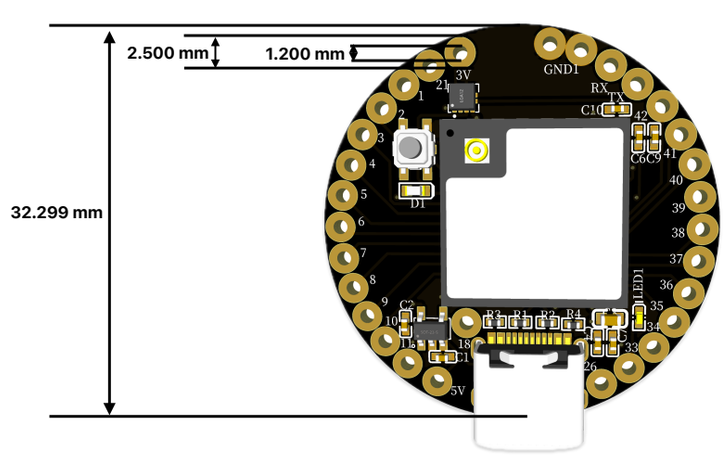


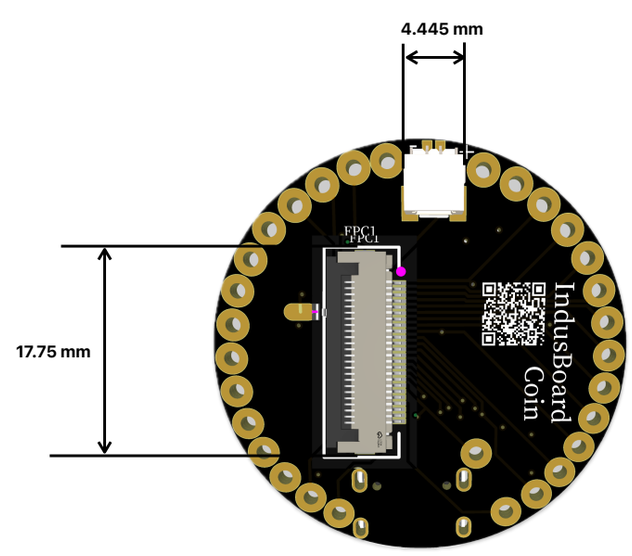
Compatible Connector





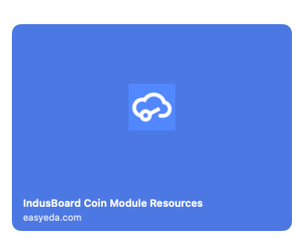
Coin Dimensions



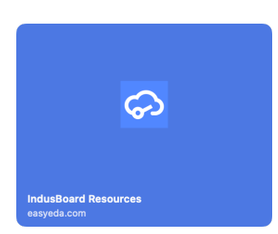


Coin EDA Design Resources

Circuit Designer



EasyEDA Footprint V1



EasyEDA Symbol V12