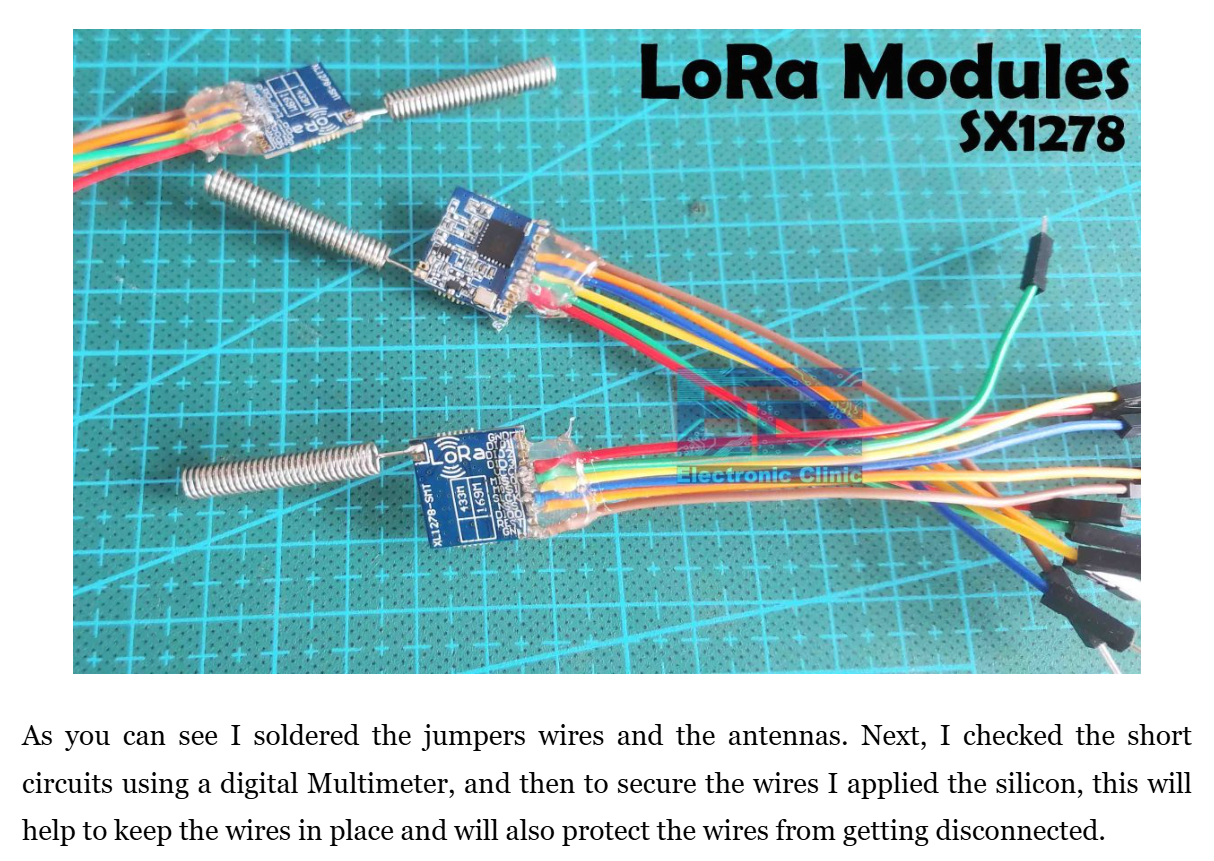
**SX1278- LoRa**

**Diagram:**

* **Supply Voltage Range**: 1.8V to 3.7V
* **Compatibility**: Can be used with both 3.3V and 5V controller boards
* **3.3V Compatible Boards**: Suitable for ESP8266, ESP32, Raspberry Pi Pico, etc.
* **Operational Temperature Range**: -40°C to +85°C

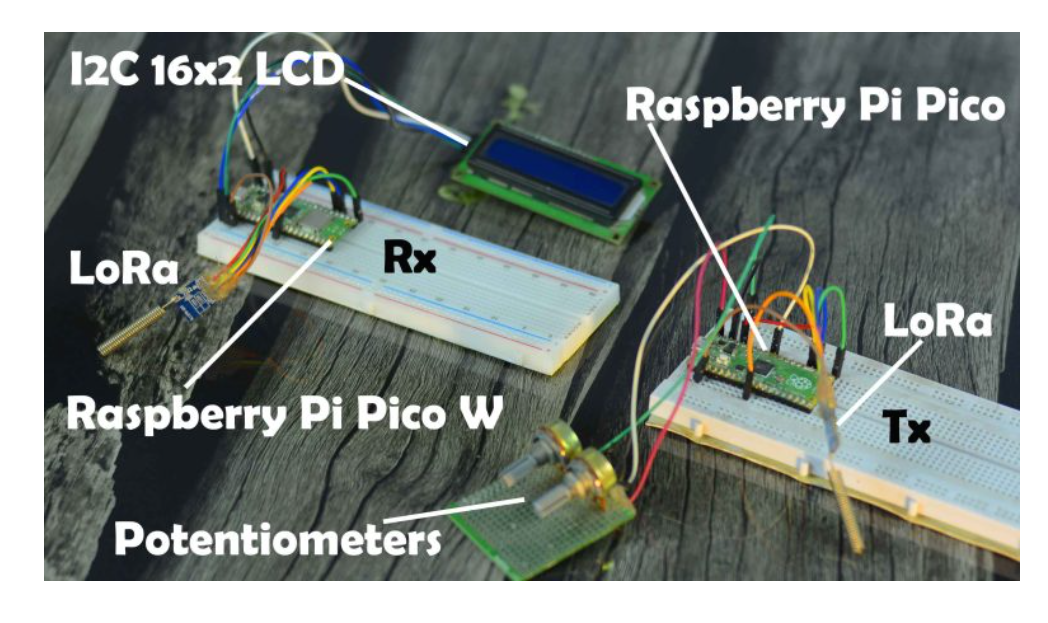
**Soldering of Jumper wires and Antenna**

****

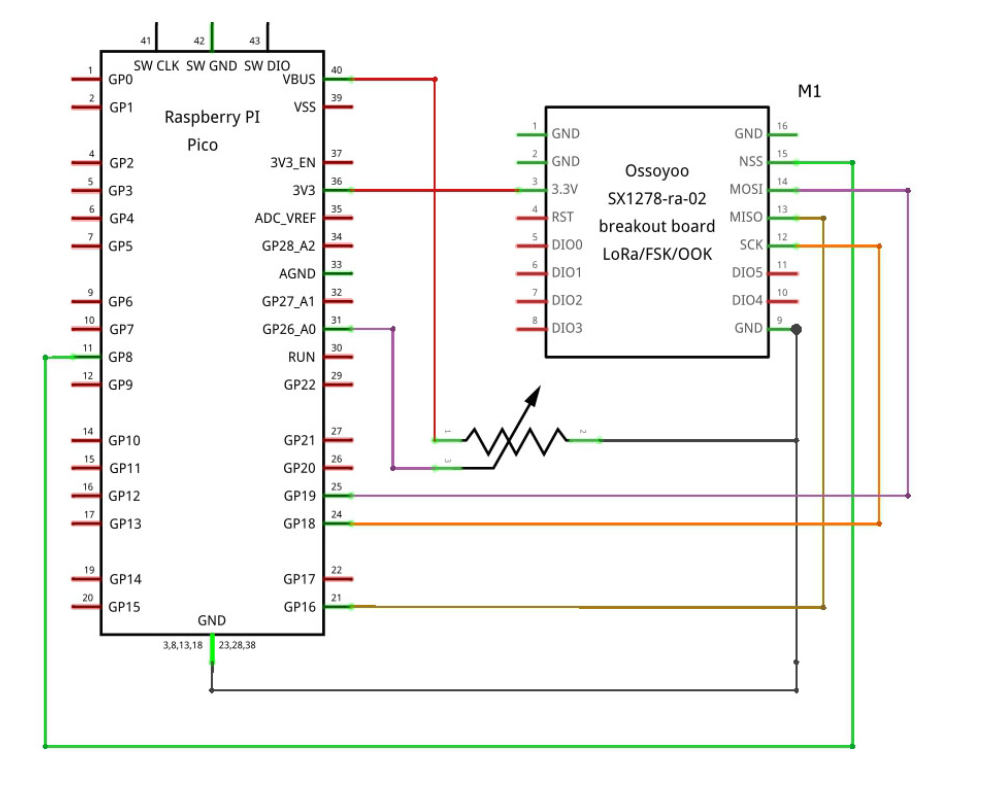
**Example:**

**The Raspberry Pi Pico reads the Potentiometer used as a sensor and sends the Potentiometer readings to the remote side receiver using the LoRa.**

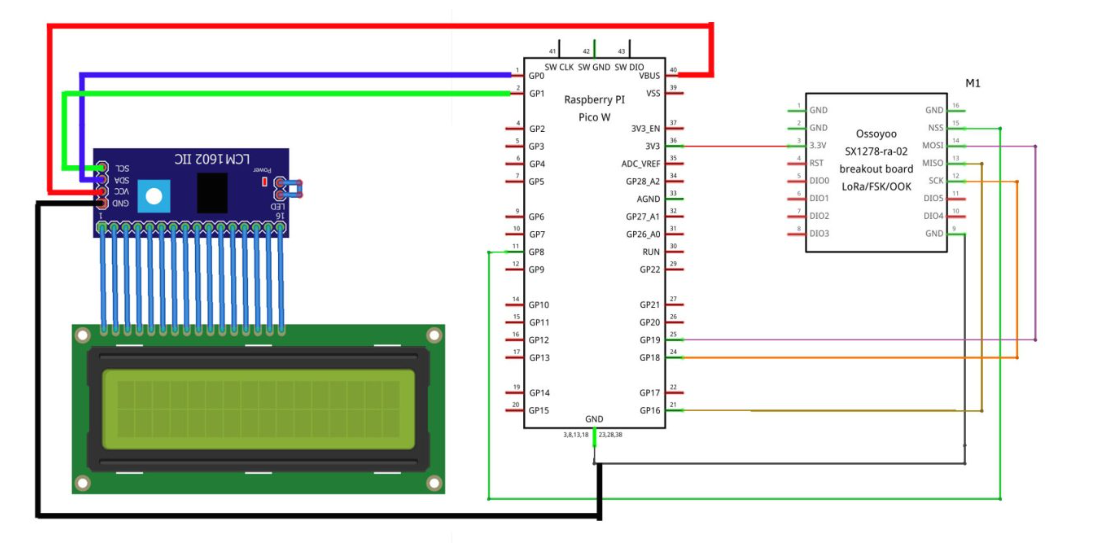
Programmed using: **Raspberry Pi Pico W and Adafruit Io**.



**TRANSMITTER**



**RECEIVER**

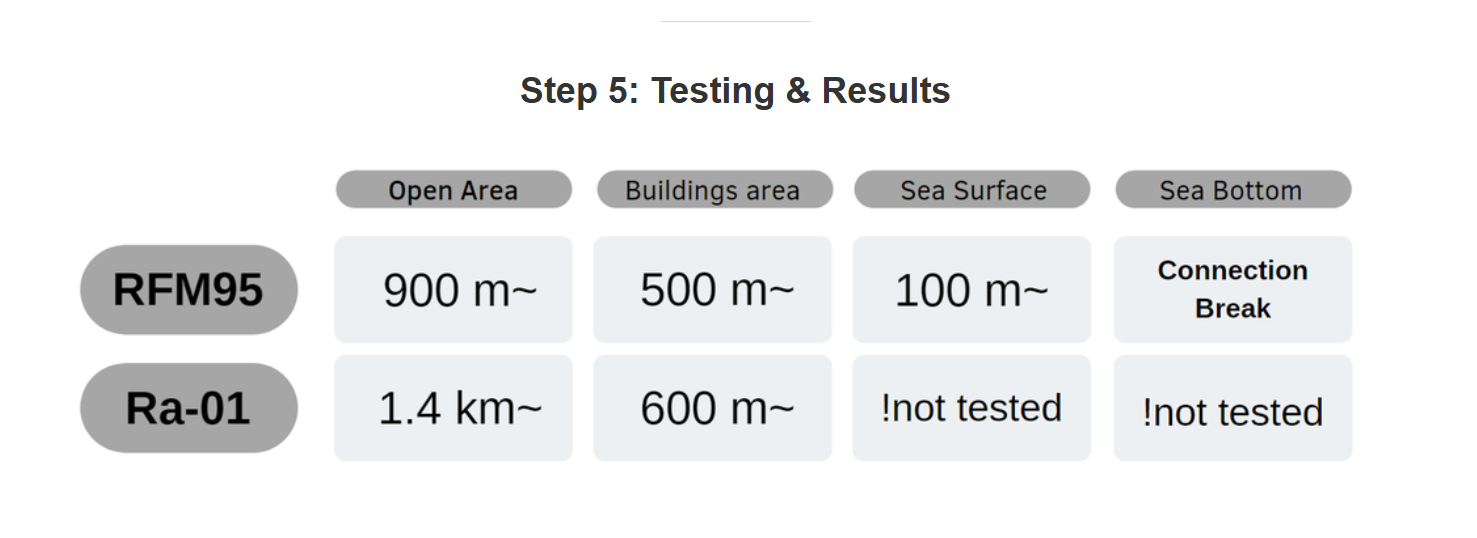
****

**YOUTUBE LINK:**

[**https://www.youtube.com/watch?v=kdQcQrzsgZM**](https://www.youtube.com/watch?v=kdQcQrzsgZM)

<https://www.electroniclinic.com/raspberry-pi-pico-w-with-lora-sx1278-for-sensor-monitoring/>

**EXPERIMENT 02:**



<https://www.youtube.com/watch?v=Ili8FZj0cEA>

**LoRa Technology Code Descriptions:**

**TRANSMITTER:**

**Code Explanation**

1. Accelerometer (GY-61AD / XL335):
   * This sensor outputs analog signals representing the X, Y, and Z acceleration axes. We’ll read these signals via analog pins.
2. Pulse Rate Sensor (SEN11574):
   * The pulse rate sensor outputs an analog signal that corresponds to the pulse rate. This value is read through an analog pin.
3. Peltier Pad (TEC1-12706):
   * Typically controlled via a transistor or relay. Here, we’ll just read a simulated temperature using an analog input to indicate its status.
4. GPS Module (NEO-6M):
   * This GPS module provides latitude and longitude coordinates via UART communication. It will use the Serial1 port on the Raspberry Pi Pico.
5. Temperature Sensor (DS18B20):
   * The DS18B20 temperature sensor communicates via the OneWire protocol, so we’ll use the OneWire and DallasTemperature libraries.

**Code Walkthrough**

1. Setup:
   * Initializes serial communications, sensor pins, LoRa settings, and the temperature sensor.
2. Loop:
   * Every 2 seconds, the code gathers data from all sensors and sends the information over LoRa.
3. readAndSendSensorData Function:
   * Reads data from:
     + Accelerometer: Reads X, Y, Z axis values.
     + Pulse Sensor: Reads pulse rate value.
     + Peltier Pad: Reads its simulated analog status.
     + Temperature Sensor: Requests and reads the temperature in Celsius.
     + GPS: Collects GPS latitude and longitude.
4. getGPSData Function:
   * Reads GPS data from the NEO-6M module and parses the GPGGA line for latitude and longitude.
5. sendMessage Function:
   * Sends the constructed message to the receiver, with the destination and source node addresses, and increments the message ID.

**RECEIVER:**

**Code Explanation**

1. **WiFi Setup (ESP8266)**:
   * Configures the Pico W to connect to a WiFi network via the ESP8266 module.
2. **MQTT Integration**:
   * Uses the MQTT protocol to publish sensor data (received via LoRa) to an MQTT broker, which can be viewed in an MQTT app.
3. **LoRa Receiver**:
   * Receives LoRa data and extracts sensor values (replacing the potentiometer data) to display on the LCD and publish via MQTT.

**Libraries Needed**

* Ensure the ESP8266 library and an MQTT library (like PubSubClient) are installed in the Arduino IDE.

**Key Changes**

1. **WiFi Connection**:
   * connectToWiFi() function establishes a connection to WiFi using the ESP8266.
2. **MQTT Connection**:
   * connectToMQTT() connects to the MQTT broker, with mqttServer, mqttPort, mqttUser, and mqttPassword settings provided.
3. **LoRa Message Parsing**:
   * The code parses incoming LoRa messages, extracting individual sensor values (acceleration, pulse rate, temperature, and GPS coordinates) and displaying the temperature and GPS coordinates on the LCD.
4. **MQTT Publishing**:
   * Constructs a JSON-like payload with sensor data and publishes it to the MQTT broker. The payload can be viewed on an MQTT app subscribed to the sensor/data topic.