Raspberry Pi wired connection to the PU

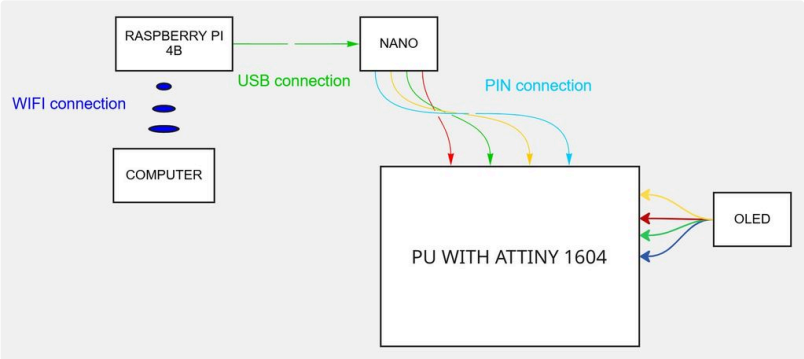
In this project, the objective is to wirelessly send a message from a computer to an `OLED` screen using a communication chain that involves a `Raspberry Pi`, an `Arduino Nano`, and an `ATtiny1604` microcontroller.

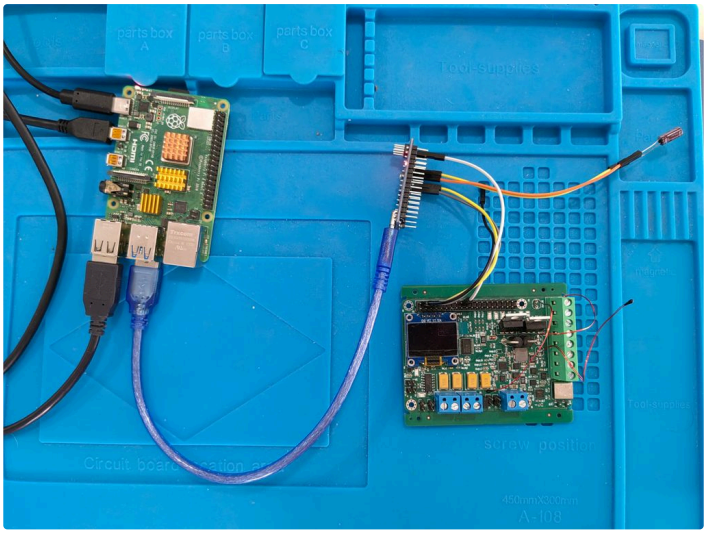
The process works as follows:

1. **Computer (`Windows`)**
   A `TCP` server is hosted on the computer, allowing the user to type and send text messages through a `Python` interface.
2. `Raspberry Pi`
   The `Raspberry Pi` acts as a `TCP` client, connecting to the computer's server over `Wi-Fi`. It receives the message, then forwards it via a `USB` serial connection to the `Arduino Nano`.
3. `Arduino Nano`
   The `Nano` receives the message over its `USB` serial port, processes it, and sends it via a `SoftwareSerial` connection to the `ATtiny1604`. It adds a termination character (`!`) to signal the end of the message.
4. `ATtiny1604` + `OLED`
   The `ATtiny1604` reads the incoming message via `SoftwareSerial`, detects the termination character, and displays the message on an `I2C`-connected `OLED` screen. The message scrolls across the screen and once completed, the `ATtiny` signals it's ready for the next one.

This project demonstrates wireless text transmission across multiple embedded platforms using both network (`Wi-Fi TCP`) and serial communication, combining modern `SBCs` (like the `Raspberry Pi`) with low-power microcontrollers for a compact display solution.

The message is sent to the `Raspberry Pi`, then it's resent to the `PU` through the `Arduino Nano`.





Prerequisites

- Have `Arduino IDE` installed on the computer.
- Have `Python` installed in the computer and the `Raspberry Pi`
- Have `ROS` installed in the computer and the `Raspberry Pi`. The information to do this process can be found here: How to use ROS on Arduino Nano

It is recommended to upload the scripts in the following order:

1. `PU` script:

```
#include <SoftwareSerial.h>
#include <U8g2lib.h>
#include <Wire.h>

// Pin configuration
const uint8_t SERIAL_RX_PIN = PIN_PA1;
const uint8_t SERIAL_TX_PIN = PIN_PA2;

// Configuration constants
const uint32_t SERIAL_BAUD_RATE = 57600;
const uint8_t TEXT_BUFFER_SIZE = 64;
const uint16_t SCROLL_DELAY_MS = 100;
const uint8_t DISPLAY_TEXT_Y_POSITION = 30;

SoftwareSerial softSerial(SERIAL_RX_PIN, SERIAL_TX_PIN);
U8G2_SSD1306_128X64_NONAME_1_HW_I2C display(U8G2_R0, U8X8_PIN_NONE);

// Message buffers
char inputBuffer[TEXT_BUFFER_SIZE];
char currentMessage[TEXT_BUFFER_SIZE];
uint8_t inputIndex = 0;

// Scrolling state
bool isScrolling = false;
int16_t scrollOffset = 0;
uint16_t currentMessageWidth = 0;
uint32_t lastScrollTimestamp = 0;

void setup() {
  display.begin();
  display.setFont(u8g2_font_ncenB08_tr);
  softSerial.begin(SERIAL_BAUD_RATE);
}

void drawScrollingText(const char* message, int16_t offset) {
  display.firstPage();
  do {
    display.setCursor(-offset, DISPLAY_TEXT_Y_POSITION); // Scroll text to the left
    display.print(message);
  } while (display.nextPage());
}

void loop() {
  // Handle scrolling
  if (isScrolling) {
    if (millis() - lastScrollTimestamp >= SCROLL_DELAY_MS) {
      drawScrollingText(currentMessage, scrollOffset);
```

```
      scrollOffset++;

      // Wait until the entire message, including the final character, has scrolled off-screen
      if (scrollOffset > currentMessageWidth + display.getStrWidth(" ")) {
        isScrolling = false;
        scrollOffset = 0;
        currentMessage[0] = '\0';
        inputBuffer[0] = '\0';
        inputIndex = 0;
        softSerial.println("READY"); // Signal readiness for the next message
      }

      lastScrollTimestamp = millis();
    }
    return; // Avoid processing new messages while scrolling is active
  }

  // Read incoming characters
  while (softSerial.available()) {
    char incomingChar = softSerial.read();

    if (incomingChar == '!') {
      inputBuffer[inputIndex] = '\0';
      strcpy(currentMessage, inputBuffer);
      currentMessageWidth = display.getStrWidth(currentMessage);
      scrollOffset = 0;
      isScrolling = true;
      inputIndex = 0;
    } else if (inputIndex < TEXT_BUFFER_SIZE - 1) {
      inputBuffer[inputIndex++] = incomingChar;
    }
  }
}
```

2. `Arduino Nano` script:

```
#include <SoftwareSerial.h>

// SoftwareSerial pin configuration for the OLED board
const uint8_t RX_PIN = 3;  // not used for sending, only for receiving if needed
const uint8_t TX_PIN = 4;  // output to OLED board
const uint32_t OLED_BAUD_RATE = 57600;

// Configuration for USB Serial (with Raspberry Pi)
const uint32_t USB_BAUD_RATE = 9600;

SoftwareSerial softSerial(RX_PIN, TX_PIN);

void setup() {
  Serial.begin(USB_BAUD_RATE);       // communication with Raspberry Pi
  softSerial.begin(OLED_BAUD_RATE);  // communication with OLED board

  Serial.println("Arduino Nano ready");
}

void loop() {
  if (Serial.available() > 0) {
    String data = Serial.readStringUntil('\n');
```

```
23
24      // Send message to the OLED board, adding '!' to indicate end of message
25      softSerial.print(data);
26      softSerial.print('!');
27
28      // For debugging, display what is being sent to the OLED
29      Serial.print("Sent to OLED: ");
30      Serial.println(data);
31    }
32  }
```

3. `Raspberry Pi` script:

```
1   #!/usr/bin/env python3
2   import socket
3   import serial
4   import time
5
6   # Server configuration (Windows PC)
7   WINDOWS_IP = '192.168.1.134'  # Change this if your IP changes
8   PORT = 65432
9
10  # Serial port configuration (Nano)
11  SERIAL_PORT = '/dev/ttyUSB0'  # Make sure this is correct
12  BAUDRATE = 9600
13
14  def main():
15      # Initialise serial connection
16      try:
17          ser = serial.Serial(SERIAL_PORT, BAUDRATE, timeout=1)
18          time.sleep(2)  # Wait for it to stabilise
19          print(f"[INFO] Serial connection established on {SERIAL_PORT} at {BAUDRATE} bps.")
20      except serial.SerialException as e:
21          print(f"[ERROR] Could not open serial port: {e}")
22          return
23
24      # Initialise TCP connection with the PC
25      with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
26          try:
27              s.connect((WINDOWS_IP, PORT))
28              print(f"[INFO] Connected to TCP server at {WINDOWS_IP}:{PORT}")
29          except Exception as e:
30              print(f"[ERROR] Could not connect to server: {e}")
31              return
32
33          # Main loop: receive messages over network and send to Nano
34          while True:
35              try:
36                  data = s.recv(1024)
37                  if not data:
38                      print("[INFO] Connection closed by the server.")
39                      break
40
41                  message = data.decode().strip()
42                  print(f"[RECV] {message}")
43
44                  # Send to Nano
45                  ser.write((message + '\n').encode('utf-8'))
```

```
                # (Optional) Read response from Nano
                while True:
                    response = ser.readline().decode('utf-8').strip()
                    if response:
                        print(f"[Nano] {response}")
                    else:
                        break

            except Exception as e:
                print(f"[ERROR] During receiving or sending: {e}")
                break

    ser.close()

if __name__ == '__main__':
    main()
```

4. `Windows` computer script:

```
import socket
import threading

# Server will listen on all interfaces on port 65432
HOST = '0.0.0.0'
PORT = 65432

clients = []
clients_lock = threading.Lock()  # Lock to ensure thread-safe operations on the clients list

# Function to handle individual client connections
def handle_client(conn, addr):
    print(f"[Connected by {addr}]")
    try:
        while True:
            data = conn.recv(1024)
            if not data:
                break  # Connection closed by client
            print(f"Message received from {addr}: {data.decode()}")
    except ConnectionResetError:
        # Handle case when client disconnects abruptly
        print(f"Connection closed by {addr}")
    finally:
        with clients_lock:
            clients.remove(conn)
        conn.close()
        print(f"Connection with {addr} closed")

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.bind((HOST, PORT))
    s.listen()
    print(f"[Server listening on {HOST}:{PORT}]")

    def accept_clients():
        while True:
            conn, addr = s.accept()
            with clients_lock:
                clients.append(conn)
```

```
            threading.Thread(target=handle_client, args=(conn, addr), daemon=True).start()

    threading.Thread(target=accept_clients, daemon=True).start()

    print("You can send messages to all connected clients.")

    while True:
        msg = input("Message to send: ")
        with clients_lock:
            for client in clients:
                try:
                    client.sendall(msg.encode())
                except Exception as e:
                    print(f"Error sending to client: {e}")
```

Connections between `Nano` and de `PU`

NANO	PU
D3	PA2 (PIN 8)
D4	PA1 (PIN 6)
GND	GND (PIN 20)
5V	VDD (PIN 4)

Steps to execute

Make the `Raspberry Pi` script executable with:

```
chmod +x nameOfScript.py
```

Execute `Windows` script with:

```
py nameOfScript2.py
```

Execute `Raspberry Pi` script with:

```
python3 nameOfScript.py
```

Now it is possible to write on the computer messages and it will print on the terminal of the `Raspberry` and on the `OLED` screen.

Testing:

1. `Windows`:

```
cd \completePath sendToRaspberry.py
[Server listening on 0.0.0.0:65432]
You can send messages to all connected clients.
Message to send:
[Connected by ('192.168.1.139', 59958)]
```

2. `Raspberry Pi`:

```
pi@raspberrypi:~ $ python3 serialTest4.py
[INFO] Serial connection established on /dev/ttyUSB0 at 9600 bps.
[INFO] Connected to TCP server at 192.168.1.134:65432
```

3. Sending a message:

```
cd \completePath sendToRaspberry.py
[Server listening on 0.0.0.0:65432]
You can send messages to all connected clients.
Message to send:
[Connected by ('192.168.1.139', 59958)]
hello
Message to send: bye
Message to send:
```

```
pi@raspberrypi:~ $ python3 serialTest4.py
[INFO] Serial connection established on /dev/ttyUSB0 at 9600 bps.
[INFO] Connected to TCP server at 192.168.1.134:65432
[RECV] hello
[Nano] Enviado a OLED: hello
Sent to OLED
[RECV] bye
[Nano] Enviado a OLED: bye
Sent to OLED
```

Output message on the `OLED` screen: