Set PU power profiles using Wi-Fi

In this project, the objective is to wirelessly send an option chosen from a computer in which there is a menu, to an `OLED` screen using a communication chain that involves a `Raspberry Pi`, an `Arduino Nano`, and an `ATtiny1604` microcontroller. It's similar to the previous project: Raspberry Pi wired connection to the PU .

The process works as follows:

1. **Computer (`Windows`)**
   A `TCP` server is hosted on the computer.

   A menu with 3 options (1 - Profile 1 (5V), 2 - Profile1 2 (20V), 3 - Exit).

   The option selected is sent to the Raspberry Pi.

2. `Raspberry Pi`
   The `Raspberry Pi` acts as a `TCP` client, connecting to the computer's server over `Wi-Fi`. It receives the option that has been chosen, then forwards it via a `USB` serial connection to the `Arduino Nano`.

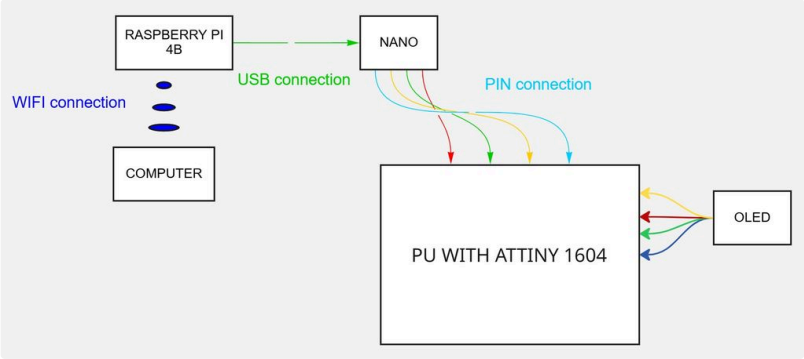
   Shows the option chosen on terminal and send it (number 1 or 2) to the nano.

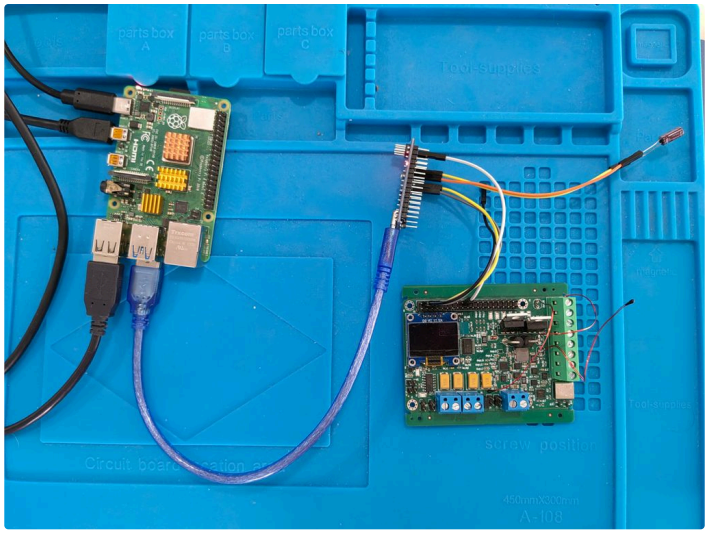
   Also shows if it was sent to the nano.

3. `Arduino Nano`
   The `Nano` receives the option from Raspberry Pi and send it to the `ATtiny1604`.
4. `ATtiny1604` + `OLED`
   The `ATtiny1604` changes the profile with the option chosen (1 or 2) via `SoftwareSerial`, and displays the previous profile and the chosen one for 3 seconds. After that time, it will show the message "Waiting for instructions" on an `I2C`-connected `OLED` screen.

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 chosen option 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 <Wire.h>
#include <SparkFun_STUSB4500.h>
#include <SoftwareSerial.h>
#include <Tiny4kOLED.h>

STUSB4500 usb;

// Constants for voltage settings and configuration
const uint8_t I2C_DEVICE_ADDRESS = 0x2B; // Default I2C address of STUSB4500
const uint8_t MAX_PDOS = 2; // Number of PDOs being configured (1 and 2)
const float TARGET_VOLTAGES[MAX_PDOS] = {5.0, 20.0}; // Desired voltages for each PDO (index 0 = PDO 1)
const float TARGET_CURRENT_A = 4.0; // Target current for PDO for profile 2
const int MAX_VOLTAGE_LIMIT_PERCENTAGE = 10; // Tolerance percentage for voltage limits

// Serial communication
const uint8_t SERIAL_RX_PIN = PIN_PA1; // RX pin for software serial
const uint8_t SERIAL_TX_PIN = PIN_PA2; // TX pin for software serial
const unsigned long SERIAL_BAUD_RATE = 57600; // Baud rate for both serials

// PDO configuration
const uint8_t DEFAULT_PDO = 2; // Initial default PDO to use
const uint8_t MIN_VALID_PDO = 1;
const uint8_t MAX_VALID_PDO = 2;

// Display settings
const uint16_t PROFILE_DISPLAY_DURATION_MS = 3000; // Duration to show profile change
const uint16_t STARTUP_DELAY_MS = 500; // Delay to allow USB controller initialization

SoftwareSerial softSerial(SERIAL_RX_PIN, SERIAL_TX_PIN); // Software serial for user input
```

```
30
31 int pdoNumber = DEFAULT_PDO; // Currently selected PDO
32
33 void setVoltageAndCurrent()
34 {
35   for (int pdo = MIN_VALID_PDO; pdo <= pdoNumber; pdo++)
36   {
37     if (pdo != 1)  // PDO 1 is fixed to 5V and may not support voltage changes on all hardware
38     {
39       usb.setLowerVoltageLimit(pdo, MAX_VOLTAGE_LIMIT_PERCENTAGE);
40       usb.setVoltage(pdo, TARGET_VOLTAGES[pdo - 1]); // Set target voltage; array is 0-indexed, PDOs start at 1
41     }
42
43     usb.setCurrent(pdo, TARGET_CURRENT_A);
44     usb.setUpperVoltageLimit(pdo, MAX_VOLTAGE_LIMIT_PERCENTAGE);
45   }
46 }
47
48 void showWaitingMessage()
49 {
50   oled.clear();
51   oled.setCursor(0, 1);
52   oled.print(F("Waiting for"));
53   oled.setCursor(0, 2);
54   oled.print(F("instructions..."));
55 }
56
57 void showProfileChange(float vOld, float cOld, float vNew, float cNew)
58 {
59   oled.clear();
60   oled.setCursor(0, 0);
61   oled.print(F("Previous:"));
62   oled.setCursor(0, 1);
63   oled.print(vOld, 1);
64   oled.print(F("V@"));
65   oled.print(cOld, 1);
66   oled.print(F("A"));
67
68   oled.setCursor(0, 2);
69   oled.print(F("Current:"));
70   oled.setCursor(0, 3);
71   oled.print(vNew, 1);
72   oled.print(F("V@"));
73   oled.print(cNew, 1);
74   oled.print(F("A"));
75 }
76
77 void setup()
78 {
79   Serial.begin(SERIAL_BAUD_RATE);
80   softSerial.begin(SERIAL_BAUD_RATE);
81   Wire.begin();
82
83   oled.begin();
84   oled.clear();
85   oled.setFont(FONT6X8);
86   oled.on();
87
```

```
  delay(STARTUP_DELAY_MS); // Allow USB controller to initialize

  if (!usb.begin(I2C_DEVICE_ADDRESS, Wire))
  {
    Serial.println("Cannot connect to STUSB4500.");
    while (true); // Halt if device is not found
  }

  Serial.println("Connected to STUSB4500!");
  Serial.println("Send '1' or '2' via SoftSerial to change the active PDO.");

  usb.setPdoNumber(pdoNumber);
  setVoltageAndCurrent();
  usb.write();  // Save changes to STUSB4500
  usb.read();   // Refresh data from device

  showWaitingMessage();
}

void loop()
{
  static int lastPdoNumber = pdoNumber;

  if (softSerial.available())
  {
    int input = softSerial.parseInt(); // Parse numeric input from serial

    // Check if the received input is a valid PDO number (within allowed range)
    if (input >= MIN_VALID_PDO && input <= MAX_VALID_PDO)
    {
      // Proceed only if the selected PDO is different from the current one
      if (input != lastPdoNumber)
      {
        // Read and display the current (old) profile
        float vOld = usb.getVoltage(lastPdoNumber);
        float cOld = usb.getCurrent(lastPdoNumber);

        pdoNumber = input;
        Serial.print("New PDO number received: ");
        Serial.println(pdoNumber);

        usb.setPdoNumber(pdoNumber);
        setVoltageAndCurrent();
        usb.write();  // Apply and save changes
        usb.read();   // Refresh device state

        // Read and display the new profile
        float vNew = usb.getVoltage(pdoNumber);
        float cNew = usb.getCurrent(pdoNumber);

        showProfileChange(vOld, cOld, vNew, cNew);
        delay(PROFILE_DISPLAY_DURATION_MS);

        showWaitingMessage();  // Return to idle screen

        lastPdoNumber = pdoNumber;  // Update state
      }
    }
```

```
    else
    {
      Serial.println("Invalid input. Please enter 1 or 2.");
      while (softSerial.available()) softSerial.read(); // Clear input buffer
    }
  }
}
```

2. `Arduino Nano` script:

```
#include <SoftwareSerial.h>

const uint8_t NANO_TX_TO_ATTINY_RX_PIN = 3;
const uint8_t OPTIONAL_RX_FROM_ATTINY_PIN = 2;

const unsigned long SERIAL_BAUD_RATE = 57600;      // Baud rate for both PC and ATtiny communication

const char PROFILE_1 = '1';
const char PROFILE_2 = '2';
const char NEWLINE_CHAR = '\n';

SoftwareSerial softSerial(OPTIONAL_RX_FROM_ATTINY_PIN, NANO_TX_TO_ATTINY_RX_PIN);

void setup() {
  Serial.begin(SERIAL_BAUD_RATE);
  softSerial.begin(SERIAL_BAUD_RATE);
  Serial.println("Send 1 or 2 to change profile:");
}

void loop() {
  if (Serial.available()) {
    char receivedChar = Serial.read();

    if (receivedChar == PROFILE_1 || receivedChar == PROFILE_2) {
      Serial.print("Sending profile: ");
      Serial.println(receivedChar);

      // Send to ATtiny with newline to trigger the change
      softSerial.write(receivedChar);
      softSerial.write(NEWLINE_CHAR);
    } else {
      Serial.println("Invalid input. Use 1 or 2.");
    }
  }

  // Receive messages from the ATtiny
  if (softSerial.available()) {
    char fromTiny = softSerial.read();
    Serial.write(fromTiny);
  }
}
```

3. `Raspberry Pi` script:

```
#!/usr/bin/env python3

import socket
import serial
```

```
import time

SERIAL_PORT_PATH = "/dev/ttyUSB0"                   # Serial device path connected to Arduino Nano
SERIAL_BAUD_RATE_BPS = 57600                         # Baud rate in bits per second (bps)
SERIAL_TIMEOUT_SECONDS = 1                           # Serial timeout duration in seconds
SERIAL_STARTUP_DELAY_SECONDS = 2                     # Delay in seconds for serial port to stabilize

REMOTE_SERVER_IP_ADDRESS = "192.168.1.133"           # IPv4 address of the remote TCP server
REMOTE_SERVER_TCP_PORT = 65432                        # TCP port number of the remote server
SOCKET_RECEIVE_BUFFER_SIZE_BYTES = 1024              # Maximum number of bytes to read from socket per call

VALID_PDO_COMMANDS_STR = ['1', '2']                   # Valid power delivery object commands as strings

def main():
    # Initialize serial connection to Arduino Nano
    try:
        serial_connection = serial.Serial(
            port=SERIAL_PORT_PATH,
            baudrate=SERIAL_BAUD_RATE_BPS,
            timeout=SERIAL_TIMEOUT_SECONDS
        )
        time.sleep(SERIAL_STARTUP_DELAY_SECONDS)  # Allow device to initialize serial connection
        print(f"[INFO] Connected to Nano via {SERIAL_PORT_PATH}")
    except Exception as error:
        print(f"[ERROR] Could not open serial port: {error}")
        return

    # Establish TCP connection to the Windows server
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
        try:
            server_socket.connect((REMOTE_SERVER_IP_ADDRESS, REMOTE_SERVER_TCP_PORT))
            print(f"[INFO] Connected to server at {REMOTE_SERVER_IP_ADDRESS}:{REMOTE_SERVER_TCP_PORT}")
        except Exception as error:
            print(f"[ERROR] Could not connect to server: {error}")
            return

        try:
            while True:
                # Receive bytes from socket, decode to UTF-8 string, strip whitespace
                received_data_str = server_socket.recv(SOCKET_RECEIVE_BUFFER_SIZE_BYTES).decode().strip()

                # Check if the connection was closed by the server
                if not received_data_str:
                    print("[INFO] Server closed the connection")
                    break

                # Verify that received command is valid (one of the defined PDO strings)
                if received_data_str in VALID_PDO_COMMANDS_STR:
                    print(f"[INFO] Received valid PDO command: {received_data_str}. Sending to Nano...")
                    # Send command over serial as bytes, with newline termination
                    serial_connection.write((received_data_str + '\n').encode())
                else:
                    print(f"[WARN] Ignored invalid command: {received_data_str}")

        except KeyboardInterrupt:
            print("\n[INFO] Manual interruption detected. Exiting...")

        except Exception as error:
```

```
            print(f"[ERROR] Communication error: {error}")

        finally:
            serial_connection.close()

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

4. `Windows` computer script:

```
#!/usr/bin/env python3
import socket
import threading
from typing import List, Tuple

SERVER_HOST: str = '0.0.0.0'
SERVER_PORT: int = 65432
BUFFER_SIZE_BYTES: int = 1024

PROFILE_1_MSG: str = "1\n"
PROFILE_2_MSG: str = "2\n"

clients: List[socket.socket] = []
clients_lock: threading.Lock = threading.Lock()

def handle_client(client_socket: socket.socket, client_address: Tuple[str, int]) -> None:
    print(f"[Connected by {client_address}]")
    try:
        while True:
            data: bytes = client_socket.recv(BUFFER_SIZE)
            if not data:
                break
            print(f"Message received from {client_address}: {data.decode().strip()}")
    except ConnectionResetError:
        print(f"Connection closed by {client_address}")
    finally:
        with clients_lock:
            if client_socket in clients:
                clients.remove(client_socket)
        client_socket.close()
        print(f"Connection with {client_address} closed")

def accept_clients(server_socket: socket.socket) -> None:
    while True:
        client_socket: socket.socket
        client_address: Tuple[str, int]
        client_socket, client_address = server_socket.accept()
        with clients_lock:
            clients.append(client_socket)
        threading.Thread(target=handle_client, args=(client_socket, client_address), daemon=True).start()

def send_to_all_clients(message: str) -> None:
    with clients_lock:
        for client in clients:
            try:
                client.sendall(message.encode())
            except Exception as ex:
                print(f"Error sending to client: {ex}")
```

```
49
50  def main() -> None:
51      with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
52          server_socket.bind((SERVER_HOST, SERVER_PORT))
53          server_socket.listen()
54          print(f"[Server listening on {SERVER_HOST}:{SERVER_PORT}]")
55
56          threading.Thread(target=accept_clients, args=(server_socket,), daemon=True).start()
57
58          while True:
59              print("\n--- MENU ---")
60              print("1 - Profile 1 (5V)")
61              print("2 - Profile 2 (20V)")
62              print("3 - Exit")
63              choice: str = input("Select an option: ").strip()
64
65              if choice == '1':
66                  print("Sending profile 1 (5V) to all clients...")
67                  send_to_all_clients(PROFILE_1_MSG)
68              elif choice == '2':
69                  print("Sending profile 2 (20V) to all clients...")
70                  send_to_all_clients(PROFILE_2_MSG)
71              elif choice == '3':
72                  print("Shutting down the server...")
73                  break
74              else:
75                  print("Invalid option, please try again.")
76
77  if __name__ == "__main__":
78      main()
```

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
```

It is now possible to enter one of the options via the computer. If either 1 or 2 is selected, the `PU` profile will be changed, and both the previous and current profiles will be displayed on the `OLED` screen.

Testing:

> To see the changes, it is necessary to disconnect and reconnect the USB-C cable. All other connections can remain in place; the USB-C can be plugged into the charger and the multimeter used to measure voltage and current.

1. `Windows`:

```
cd \completePath
\completePath\py scriptWindos.py
[Server listening on 0.0.0.0:65432]

--- MENU ---
1 - Profile 1 (5V)
2 - Profile 2 (20V)
3 - Exit
Select an option: [Connected by ('192.168.1.139', 50474)]
```

2. `Raspberry Pi`:

```
pi@raspberrypi:~ $ python3 serialTest5.py
[INFO] Connected to Nano via /dev/ttyUSB0
[INFO] Connected to server 192.168.1.133:65432
```

3. Sending the chosen option:

```
cd \completePath
```

```
\completePath\py scriptWindos.py
[Server listening on 0.0.0.0:65432]

--- MENU ---
1 - Profile 1 (5V)
2 - Profile 2 (20V)
3 - Exit
Select an option: [Connected by ('192.168.1.139', 50474)]
1
Sending profile 1 (5V) to all clients...

--- MENU ---
1 - Profile 1 (5V)
2 - Profile 2 (20V)
3 - Exit
Select an option: 2
Sending profile 2 (20V) to all clients...
--- MENU ---
1 - Profile 1 (5V)
2 - Profile 2 (20V)
3 - Exit
Select an option: 3
Shutting down the server...
```

```
pi@raspberrypi:~ $ python3 serialTest5.py
[INFO] Connected to Nano via /dev/ttyUSB0
[INFO] Connected to server 192.168.1.133:65432
[INFO] Received profile 1, sending to Nano...
[INFO] Received profile 2, sending to Nano...
```