Add ROS to the Power Unit board

This system connects a `ROS` (Robot Operating System) environment with a physical display using two `Arduino Nano` boards. Its primary purpose is to receive messages published within a `ROS` network and display them clearly on an `OLED` screen with automatic scrolling. The design is modular, dividing responsibilities between two devices to ensure simplicity and reliability.

The first Arduino functions as a bridge: it is used to upload the code to the `ATtiny1604` microcontroller via `jtag2updi` programming and also acts as the ROS subscriber by connecting to the `ROS` network using the `rosserial` library. It listens for string messages published on the `chatter` topic and forwards them via a `software serial` connection. This allows `ROS`-generated data to be transmitted to external components without relying on the main serial interface.

The second `Arduino` is responsible for displaying incoming messages on an `SSD1306 OLED` screen. It receives data over the software serial interface and displays each message with a smooth leftward scrolling animation. The display ensures that long messages are shown in full by automatically scrolling them across the screen. It does not accept new messages until the current one has been fully displayed. Once complete, it sends a `READY` signal to indicate it is prepared to receive the next message.

This setup provides a simple and effective way to visualise `ROS` messages in real time on a compact display module, making it ideal for debugging, status output, or user-facing robot feedback.

Important:

Make sure to have `WSL` and `ROS` installed. If not, follow this guide:
How to use ROS on Arduino Nano

It is **recommended** to use **two** `Arduino Nano` **boards** — one as a `jtag2updi` programmer and the other as the `ROS` subscriber. However, it is **also possible to use a single** `Nano`, although this requires reprogramming it between functions if something goes wrong during testing.

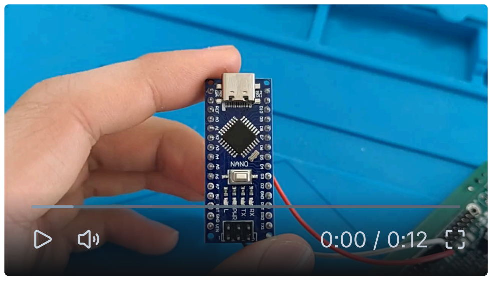
If only one `Nano` is used, its function will need to be alternated: first as a programmer to upload code to the `ATtiny1604`, and then as a receiver for incoming messages. With two `Nano` boards, it is only necessary to switch physical connections, avoiding the need to reprogram the board.

To upload code, use the connection: `D6` to `PA0`.
To receive messages, change the connection to: `D4` to `PA1`.

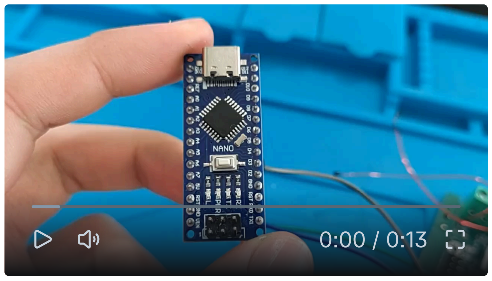
These steps are detailed in the corresponding section and are illustrated in the attached images and video. To prepare the `Nano` as a `jtag2updi` programmer, refer to this guide:
Connect the display

Once ready, upload the appropriate code to the `ATtiny1604`, making sure to correctly configure the board, programmer, chip, and port settings.

Conections:



Programmer



Subscriber

Code for ATiny1604:

```cpp
#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_PA0;

// 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 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;
    }
  }
}

```

Code explanation

Included libraries

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

- `SoftwareSerial`: allows the use of digital pins as additional serial (`RX`, `TX`) ports.
- `U8g2lib`: used to control `OLED` displays with various fonts and rendering functions.
- `Wire`: required for `I2C` communication, which the `OLED` display uses.

Pin configuration

```
const uint8_t SERIAL_RX_PIN = PIN_PA1;
const uint8_t SERIAL_TX_PIN = PIN_PA0;
```

- Assigns specific pins for serial communication using `SoftwareSerial` (`RX` on `PA1`, `TX` on `PA0`).

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;
```

- `SERIAL_BAUD_RATE`: defines the speed of serial communication (57,600 baud).
- `TEXT_BUFFER_SIZE`: maximum length of the text message.

- `SCROLL_DELAY_MS`: time in milliseconds between scroll steps.
- `DISPLAY_TEXT_Y_POSITION`: vertical position (in pixels) where the text appears on the `OLED` display.

Object initialisation

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

- `softSerial`: creates a software-based serial communication instance using the defined `RX` and `TX` pins.
- `display`: initialises the `OLED` display in 128x64 resolution using hardware `I2C`.

Message buffers and scrolling state

```
char inputBuffer[TEXT_BUFFER_SIZE];
char currentMessage[TEXT_BUFFER_SIZE];
uint8_t inputIndex = 0;

bool isScrolling = false;
int16_t scrollOffset = 0;
uint16_t currentMessageWidth = 0;
uint32_t lastScrollTimestamp = 0;
```

- `inputBuffer`: temporarily stores the incoming characters.
- `currentMessage`: holds the message to be scrolled on the display.
- `inputIndex`: tracks the current position in the input buffer.
- `isScrolling`: indicates whether scrolling is currently active.
- `scrollOffset`: the current horizontal offset of the scrolling text.
- `currentMessageWidth`: pixel width of the message.
- `lastScrollTimestamp`: stores the last time the text was scrolled.

Setup function

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

- Initialises the `OLED` display.
- Sets the font for displaying text.
- Begins serial communication via `SoftwareSerial`.

Draw scrolling text function

```
void drawScrollingText(const char* message, int16_t offset) {
    display.firstPage();
    do {
        display.setCursor(-offset, DISPLAY_TEXT_Y_POSITION);
        display.print(message);
    } while (display.nextPage());
}
```

- Draws the given text at a horizontal offset, creating a scrolling effect.
- Uses `firstPage()` and `nextPage()` to update the screen in multiple passes (required by `U8g2`).

Scrolling logic

```
if (isScrolling)
```

- Checks whether scrolling is currently active. If `true`, it proceeds to scroll the text.

Time-based scroll trigger

```
if (millis() - lastScrollTimestamp >= SCROLL_DELAY_MS) {
  drawScrollingText(currentMessage, scrollOffset);
  scrollOffset++;
```

- Uses `millis()` to measure how much time has passed since the last scroll.
- If enough time has passed (defined by `SCROLL_DELAY_MS`), the text should scroll one step.
- Calls a function to draw the current message on the screen with the current horizontal offset (`scrollOffset`).
- This makes the message appear to move leftwards on the display.
- Increases the offset to shift the message left by one step on the next update.

Stop scrolling when message has fully passed

```
if (scrollOffset > currentMessageWidth + display.getStrWidth(" ")) {
  isScrolling = false;
  scrollOffset = 0;
  currentMessage[0] = '\0';
  inputBuffer[0] = '\0';
  inputIndex = 0;
  softSerial.println("READY");
```

- Checks if the entire message (plus an extra space) has completely scrolled off the screen.
- Disables scrolling.
- Resets the scroll offset to the start.
- Clears out the message (`currentMessage`) and the input buffer (`inputBuffer`) by setting the first character to the null terminator.
- Resets the input index back to zero.
- Sends the message `"READY"` via serial to indicate that the system is ready to receive a new message.

```
lastScrollTime = millis();
```

- Updates the timestamp for the last scroll step, so the next delay can be measured accurately.

```
return;
```

- Exits the `loop()` function early if scrolling is in progress.
- This ensures no new message is processed until the current one has finished scrolling.

Read incoming serial data

```
while (softSerial.available()) {
```

- As long as there's data available on the serial port, enter the loop to read it.

```
char incomingChar = softSerial.read();
```

- Reads one character from the serial buffer and stores it in `incomingChar`.

Handle end of message

```
if (incomingChar == '!') {
  inputBuffer[inputIndex] = '\0';
  strcpy(currentMessage, inputBuffer);
  currentMessageWidth = display.getStrWidth(currentMessage);
  scrollOffset = 0;
  isScrolling = true;
  inputIndex = 0;
```

- Checks if the incoming character is `'!'`, which signals the end of the message.
- Terminates the `inputBuffer` string with a null character to make it a valid `C` string.
- Copies the fully received message from the buffer to `currentMessage`.
- Calculates the pixel width of the message for proper scrolling.
- Resets the scroll position.
- Activates scrolling.
- Clears the input index for the next message.

Store characters in input buffer

```
} else if (inputIndex < TEXT_BUFFER_SIZE - 1) {
  inputBuffer[inputIndex++] = incomingChar;
```

- If the incoming character is not `'!'` and there's still space in the buffer:
  - Adds the character to the `inputBuffer`.
  - Increments the input index for the next character.

6. Use the other `Nano` as a subscriber. To do this, upload the following code, and keep in mind it should not have anything connected.
7. Connect the other `Nano`, or the same one programmed in the same way but with a different physical connection. Use digital pin 4, and connect it to `PA1`, which is pin 3 counting from the top row

Code for Nano:

```
#include <ros.h>
#include <std_msgs/String.h>
#include <SoftwareSerial.h>

// define pin assignments for serial communication
const uint8_t RX_PIN = 3;  // receive pin for SoftwareSerial
const uint8_t TX_PIN = 4;  // transmit pin for SoftwareSerial
const uint32_t BAUD_RATE = 57600; // communication speed

// initialize SoftwareSerial with defined pins
SoftwareSerial softSerial(RX_PIN, TX_PIN);

// create a ROS node handle
ros::NodeHandle nh;

// callback function to handle incoming messages
void messageCb(const std_msgs::String& msg) {
  softSerial.println(msg.data);  // send received message to SoftwareSerial
}

```

```
// subscribe to the "chatter" topic and link it to the callback function
ros::Subscriber<std_msgs::String> sub("chatter", &messageCb);

void setup() {
  softSerial.begin(BAUD_RATE);  // start serial communication
  nh.initNode();  // initialize ROS node
  nh.subscribe(sub);  // subscribe to the topic
}

void loop() {
  nh.spinOnce();  // process incoming ROS messages
}
```

Code explanation:

Included libraries

```
#include <ros.h>
#include <std_msgs/String.h>
#include <SoftwareSerial.h>
```

- `ros.h`: this library provides functionality for communication between the microcontroller and a `ROS` network. It is part of the `rosserial` package, which allows microcontrollers to act as nodes in a `ROS` system.
- `std_msgs/String.h`: this is a `ROS` message type used for simple string data. It defines the structure of the messages sent over `ROS` topics.
- `SoftwareSerial.h`: this library enables serial communication on digital pins other than the hardware `UART`. It is useful when the main hardware serial port is already in use (e.g., by `USB`).

Pin and serial configuration

```
const uint8_t RX_PIN = 3;
const uint8_t TX_PIN = 4;
const uint32_t BAUD_RATE = 57600;
SoftwareSerial softSerial(RX_PIN, TX_PIN);
```

- `RX_PIN` and `TX_PIN` define which digital pins will be used for receiving and transmitting serial data via `SoftwareSerial`.
- `BAUD_RATE` sets the communication speed to 57600 bits per second.
- `softSerial` is an instance of the `SoftwareSerial` class, configured to use pins 3 (receive) and 4 (transmit).

This software serial link could be connected to another microcontroller, peripheral device, or even a display module.

ROS node handle and subscription

```
ros::NodeHandle nh;
```

- This object manages the connection to the `ROS` system. It handles:
  - Initialisation of the node
  - Topic subscription and publication
  - Message transmission and reception

```
void messageCb(const std_msgs::String& msg) {
```

```
softSerial.println(msg.data);
}
```

- This is the `callback` **function**. It is triggered **automatically** whenever a message is received on the subscribed topic.
- `msg` is an object of type `std_msgs::String`, and `msg.data` holds the actual string content.
- The message is printed (with a newline) via `softSerial.println()`, sending it to whatever device is connected to the `SoftwareSerial` port.

```
ros::Subscriber<std_msgs::String> sub("chatter", &messageCb);
```

- This line subscribes the node to the **"chatter"** topic.
- When a new message is published on the `"chatter"` topic, `messageCb` will be called.
- `std_msgs::String` ensures the expected message format is plain text.

Setup function

```
void setup() {
  softSerial.begin(BAUD_RATE);
  nh.initNode();
  nh.subscribe(sub);
}
```

- `softSerial.begin(BAUD_RATE)` starts the software serial connection at the specified baud rate.
- `nh.initNode()` initialises the `ROS` node. It sets up the communication link between the microcontroller and the `ROS` master (usually running on a `PC` or robot).
- `nh.subscribe(sub)` registers the subscriber to the `"chatter"` topic.

Loop function

```
void loop() {
  nh.spinOnce();
}
```

- `nh.spinOnce()` checks for and processes any incoming `ROS` messages.
- This function should be called **regularly and frequently**, hence its presence in the `loop()` function.
- Without this call, the node would not respond to published messages.

Execution

- Open `PowerShell` as administrator. (Check the attached documentation if it isn't shared.) Make sure the port is set as shared.

```
 C:\WINDOWS\system32> usbipd list
Connected:
BUSID  VID:PID    DEVICE                                                        STATE
1-3    413c:2003  Dispositivo de entrada USB                                    Not shared
1-4    1a86:7523  USB-SERIAL CH340 (COM5)                                       Shared
1-6    13d3:56dd  USB2.0 HD UVC WebCam                                          Not shared
1-10   8087:0aaa  Intel(R) Wireless Bluetooth(R)                                Not shared
```

- Connect the port. It should show as **attached**.

```
C:\WINDOWS\system32> usbipd attach --busid 1-4 --wsl
```

```
usbipd: info: Using WSL distribution 'Ubuntu-20.04' to attach; the device will be available in all WSL 2 distribu
usbipd: info: Detected networking mode 'nat'.
usbipd: info: Using IP address 172.22.224.1 to reach the host.
```

```
C:\WINDOWS\system32> usbipd list
Connected:
BUSID  VID:PID    DEVICE                                                        STATE
1-3    413c:2003  Dispositivo de entrada USB                                    Not shared
1-4    1a86:7523  USB-SERIAL CH340 (COM5)                                       Attached
1-6    13d3:56dd  USB2.0 HD UVC WebCam                                          Not shared
1-10   8087:0aaa  Intel(R) Wireless Bluetooth(R)                                Not shared
```

- Open the `Ubuntu` app or the terminal and activate `WSL`.

Terminal 1: activate ROS (server) with the command

```
$ roscore
... logging to /home/user/.ros/log/77c862d0-361e-11f0-8eea-9fd4f4449e6a/roslaunch-DESKTOP-R1R7VHK-964.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://DESKTOP-R1R7VHK:35599/
ros_comm version 1.17.0


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.17.0

NODES

auto-starting new master
process[master]: started with pid [974]
ROS_MASTER_URI=http://DESKTOP-R1R7VHK:11311/

setting /run_id to 77c862d0-361e-11f0-8eea-9fd4f4449e6a
process[rosout-1]: started with pid [984]
started core service [/rosout]
```

Terminal 2: connect the USB device

```
$ rosrun rosserial_python serial_node.py /dev/ttyUSB0
[INFO] [1747816570.351070]: ROS Serial Python Node
[INFO] [1747816570.356554]: Connecting to /dev/ttyUSB0 at 57600 baud
[INFO] [1747816572.467765]: Requesting topics...
[INFO] [1747816572.492291]: Note: subscribe buffer size is 280 bytes
[INFO] [1747816572.493680]: Setup subscriber on chatter [std_msgs/String]
```

Terminal 3: publish message

```
$ rostopic pub /chatter std_msgs/String 'data: "hello!"'
```

If there are any issues, use the following command, and try again.

```
$ set +H
```

There can be as many terminals as messages wanted to be sent, but they will be on queue until the current one is fully displayed.