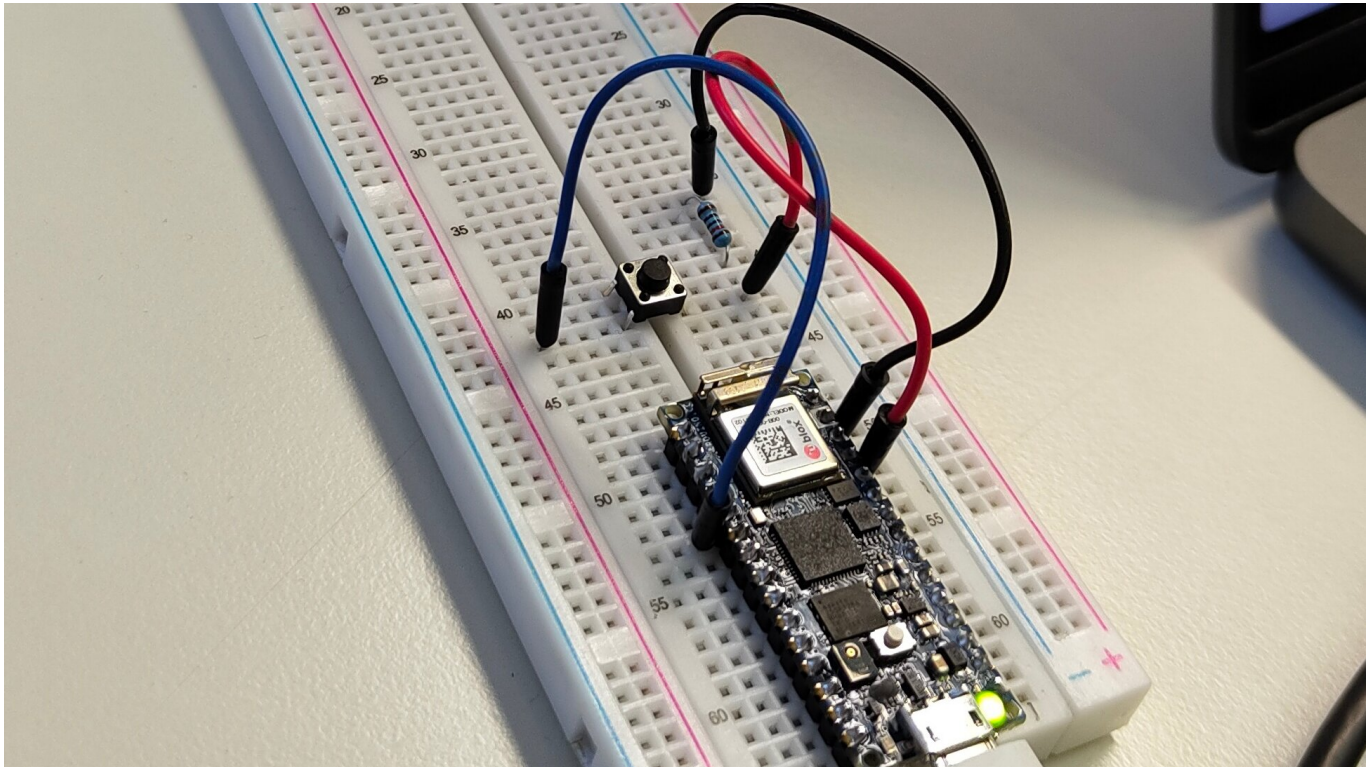


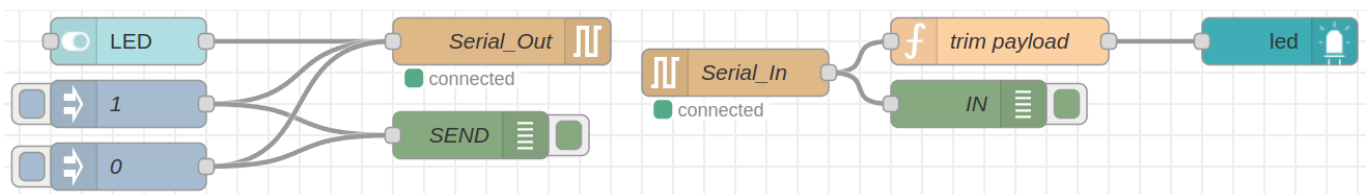
# Ubiquitous Computing: Lab 3

## Exercise 1: Building an Arduino-Node-Red Circuit for LED and Button Synchronization

### Task 1: Physical Button with Node Red Dashboard



In this lab, the objective was to create a system combining an Arduino and Node-RED to control an LED. For simplicity, the Arduino's built-in LED was used, and a physical button was added to toggle the LED on and off. Additionally, a web-based dashboard in Node-RED allowed the LED to be controlled remotely via the serial interface.



The setup involved configuring the Arduino to respond to both the physical button and a virtual button on the Node-RED dashboard. A visual indicator, represented as a light bulb icon in the dashboard, displayed the LED's current state. This status was updated in real-time by using Node-RED's **Serial In** node, which listened for updates sent by the Arduino over the serial connection.

To enable communication between the Arduino and Node-RED, the serial bus had to be properly configured. On Windows, the Arduino connects through **COM\_** devices, while on Linux, it uses **/dev/ttyUSB\_**, with the underscore replaced by the port number. The baud rate was set to **9600** to match the data transmission speed of the Arduino, ensuring smooth communication between the two systems.

Edit serial in node > Edit serial-port node

Delete

Cancel

Update

⚙ Properties

⚙

📄

📌 Name

Serial\_In

🔗 Serial Port

COM3

🔍

🔧 Settings

Baud Rate

▼ 9600

Data Bits

8 ▼

Parity

None ▼

Stop Bits

1 ▼

DTR

auto ▼

RTS

auto ▼

CTS

auto ▼

DSR

auto ▼

➡ Input

Optionally wait for a start character of , then

Split input 

on the character ▼

and deliver 

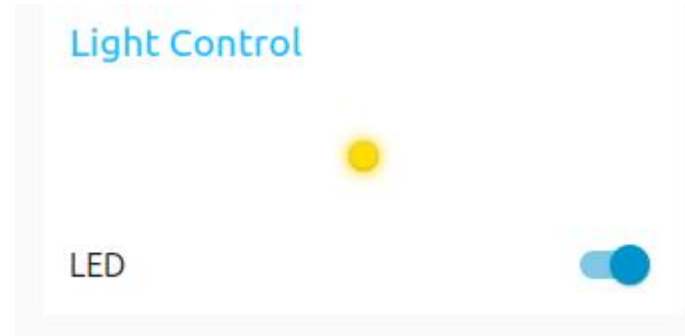
ASCII strings ▼

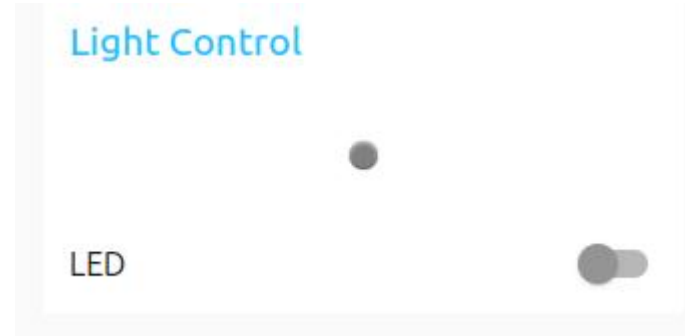
↔ Output

Add character to output messages

⇄ Request

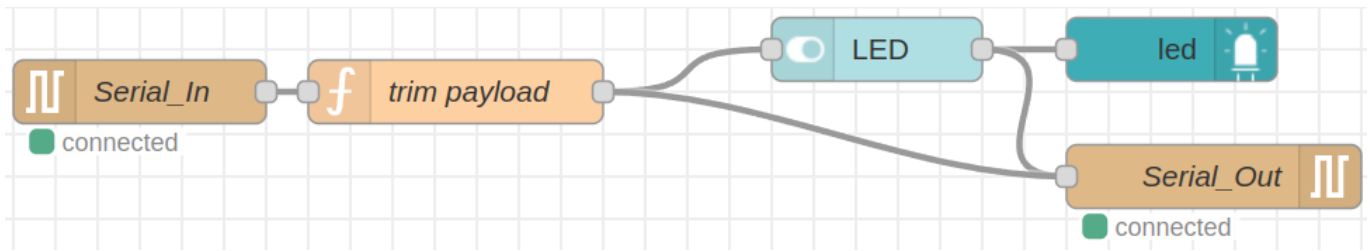
Default response timeout  ms

The image shows a 'Light Control' interface. At the top, the text 'Light Control' is displayed in blue. Below it, there is a glowing yellow circle representing a light source. At the bottom left, the text 'LED' is shown. To the right of 'LED' is a blue toggle switch that is currently turned on.

The image shows the same 'Light Control' interface. The glowing yellow circle is now a dark grey circle, indicating the light is off. The blue toggle switch at the bottom right is now turned off, appearing as a grey switch.

/

## Task 2: Synced Button



```
#define BUTTON_PIN 0x2
#define LED_PIN 0xD

int prevState = 0;
```

The Arduino code began with defining constants for the button and LED pins. The button was connected to pin D2, represented by the address 0x2, while the built-in LED was accessed at address 0xD. A variable, `prevState`, was introduced to track the previous state of the button. This was crucial to ensure that data was sent over the serial port only when the button's state changed, reducing unnecessary communication.

```
void setup() {
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT_PULLUP);
  Serial.begin(9600);
}
```

In the `setup` function, the LED pin was configured as an output, allowing it to be turned on or off by writing a digital signal to it. The button pin was set as an input with a pull-up resistor, which ensured the pin remained in a high state unless the button actively pulled it low. Serial communication was also initialized with a baud rate of 9600, setting up the Arduino to transmit and receive data at the appropriate speed.

```
void loop() {
  int buttonState = digitalRead(BUTTON_PIN);

  if (buttonState != prevState) {
    prevState = buttonState;
    Serial.println(buttonState);
    digitalWrite(LED_PIN, buttonState);
  }
  // ...
}
```

The `loop` function began by reading the current state of the button and comparing it to the previous state stored in `prevState`. If the state had changed, the new state was written to the serial port, and the LED was updated to reflect the button's state. This mechanism ensured that the LED's behavior remained consistent with the button's input, whether pressed or released.

```
// ...  
if (Serial.available() > 0) {  
  char command = Serial.read();  
  if (command == '1') {  
    digitalWrite(LED_PIN, HIGH);  
  } else if (command == '0') {  
    digitalWrite(LED_PIN, LOW);  
  }  
}  
}  
} // void loop();
```

The second part of the `loop` function handled incoming commands from Node-RED over the serial connection. If data was available, a single character was read and interpreted as a command. A `'1'` turned the LED on by setting the pin to high, while a `'0'` turned it off by setting the pin to low. This allowed the LED to be controlled not only by the physical button but also remotely through the Node-RED dashboard.

By combining the physical button, serial communication, and Node-RED's interface, this exercise demonstrated how to create a responsive system for LED control. The result was a functional and interactive circuit that integrated hardware and software seamlessly, showcasing the power of ubiquitous computing in action.