Polling Based Implementation:

Polling is a method where a device continuously checks the status of another device to determine if it needs attention. This involves sending requests or queries to the device, waiting for a response, and repeating this process in a loop until the desired condition or data is obtained.

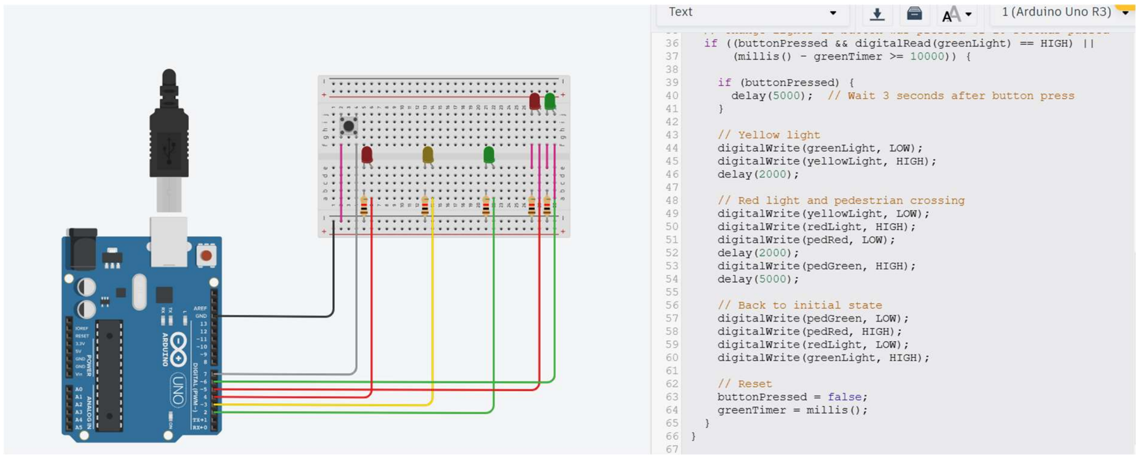
Advantages of polling include:

- **Simplicity**: It is a simpler mechanism to implement and provides straightforward control over the process.
- **Compatibility**: Polling can work on systems that do not support interrupt-driven mechanisms, making it widely applicable across various platforms.
- **Predictability**: Since the system checks for data at regular intervals, the behaviour is predictable and can be fine-tuned for specific performance requirements.

Disadvantages of polling include:

- **Resource Intensive**: Constantly checking for data or events consumes system resources, including CPU cycles, leading to inefficiency and wasted power.
- **Latency**: There can be a delay between the occurrence of an event and the system detecting it, especially if the polling interval is long, leading to higher latency in event processing.
- In real-time applications where immediate responses are critical, polling may not be suitable.

My Implementation:



```
  if ((buttonPressed && digitalRead(greenLight) == HIGH) ||
      (millis() - greenTimer >= 10000)) {

    if (buttonPressed) {
      delay(5000);  // Wait 3 seconds after button press
    }

    // Yellow light
    digitalWrite(greenLight, LOW);
    digitalWrite(yellowLight, HIGH);
    delay(2000);

    // Red light and pedestrian crossing
    digitalWrite(yellowLight, LOW);
    digitalWrite(redLight, HIGH);
    digitalWrite(pedRed, LOW);
    delay(2000);
    digitalWrite(pedGreen, HIGH);
    delay(5000);

    // Back to initial state
    digitalWrite(pedGreen, LOW);
    digitalWrite(pedRed, HIGH);
    digitalWrite(redLight, LOW);
    digitalWrite(greenLight, HIGH);

    // Reset
    buttonPressed = false;
    greenTimer = millis();
  }
}
```

- The button is working.
- When the system is not interrupted by a button, the green light remains on for about 10 seconds before going to yellow.
- Conversely, when it is pressed, the green light stays on for about 5 seconds and then goes to yellow.

Interrupt-based

- Interrupts are signals sent by devices to the CPU to indicate that they need attention. When a device has data ready or an event occurs, it sends an interrupt signal to the CPU, causing the CPU to stop its current activities and attend to the interrupting device.

- For example, a microcontroller can be interrupted by a sensor when it has a new value or by a serial port when it receives data.

- Interrupts can be either hardware (keyboard for example) or software based (components running on the processor).

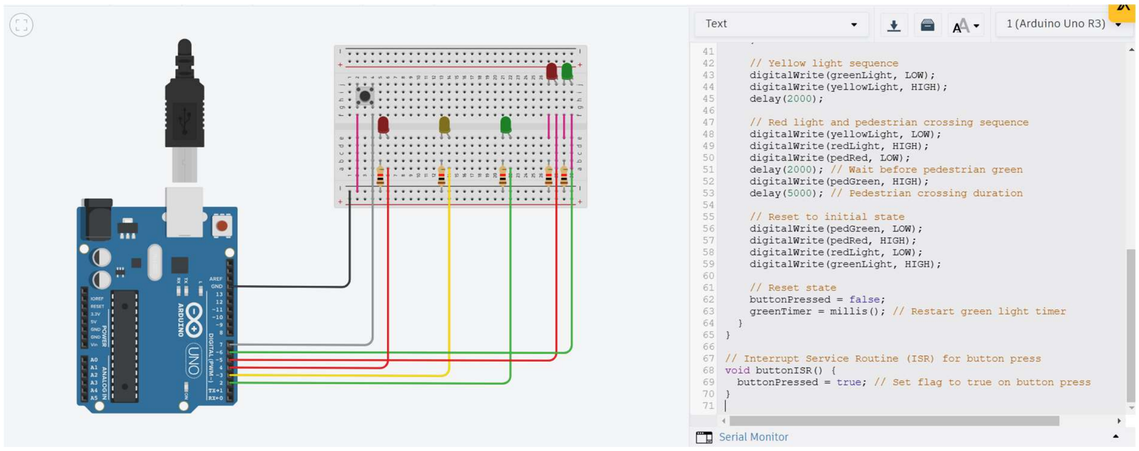
Advantages of interrupts include:

- **Efficiency**: Systems that employ interrupts are generally more efficient as the CPU can focus on other tasks while waiting for external events, rather than constantly polling.
- **Reduced Latency**: Devices can be serviced immediately when they have data to **transfer, leading to faster response times, reducing latency.**
- **Real-time Responsiveness**: Interrupts enable immediate responses to external events, making them essential for real-time applications where timely reactions are critical.
- **Prioritization**: Interrupts can be prioritized, allowing the system to respond to the most important events first, ensuring essential tasks are handled promptly.

Disadvantages of interrupts include:

- **Complexity**: Implementing and managing interrupts can be complex, involving intricate synchronization and handling mechanisms, especially in multi-device environments.
- **Task Interference**: In some cases, interrupts can restrict the simultaneous execution of tasks as the CPU's attention is frequently diverted to handle interrupt requests.

My implementation:



```

    // Yellow light sequence
    digitalWrite(greenLight, LOW);
    digitalWrite(yellowLight, HIGH);
    delay(2000);

    // Red light and pedestrian crossing sequence
    digitalWrite(yellowLight, LOW);
    digitalWrite(redLight, HIGH);
    digitalWrite(pedRed, LOW);
    delay(2000); // Wait before pedestrian green
    digitalWrite(pedGreen, HIGH);
    delay(5000); // Pedestrian crossing duration

    // Reset to initial state
    digitalWrite(pedGreen, LOW);
    digitalWrite(pedRed, HIGH);
    digitalWrite(redLight, LOW);
    digitalWrite(greenLight, HIGH);

    // Reset state
    buttonPressed = false;
    greenTimer = millis(); // Restart green light timer
  }
}

// Interrupt Service Routine (ISR) for button press
void buttonISR() {
  buttonPressed = true; // Set flag to true on button press
}
```

- The button doesn't work.

- Regardless of the button being pushed or not, the green button only goes to yellow after 10 seconds.

Code for interrupt based:

```
#include <Wire.h>

// Pin definitions
const int greenLight = 2;
const int yellowLight = 3;
const int redLight = 4;
const int pedRed = 5;
const int pedGreen = 6;
const int button = 7;

volatile bool buttonPressed = false; // Flag set by interrupt
unsigned long greenTimer = 0;

void setup() {
  // Set pin modes
  pinMode(greenLight, OUTPUT);
  pinMode(yellowLight, OUTPUT);
  pinMode(redLight, OUTPUT);
  pinMode(pedRed, OUTPUT);
  pinMode(pedGreen, OUTPUT);
  pinMode(button, INPUT_PULLUP); // Button with internal pull-up resistor

  // Attach interrupt to the button pin
  attachInterrupt(digitalPinToInterrupt(button), buttonISR, FALLING);

  // Start with green light on
  digitalWrite(greenLight, HIGH);
  digitalWrite(pedRed, HIGH);
  greenTimer = millis(); // Start the timer
}

void loop() {
  // Determine if the green light duration has passed or button was pressed
  if ((buttonPressed && digitalRead(greenLight) == HIGH) ||
      (millis() - greenTimer >= 10000)) {

    // If button was pressed, wait before proceeding
    if (buttonPressed) {
      delay(5000); // Preparation delay after button press
    }

    // Yellow light sequence
    digitalWrite(greenLight, LOW);
    digitalWrite(yellowLight, HIGH);
    delay(2000);

    // Red light and pedestrian crossing sequence
    digitalWrite(yellowLight, LOW);
    digitalWrite(redLight, HIGH);
    digitalWrite(pedRed, LOW);
    delay(2000); // Wait before pedestrian green
    digitalWrite(pedGreen, HIGH);
    delay(5000); // Pedestrian crossing duration

    // Reset to initial state
    digitalWrite(pedGreen, LOW);
    digitalWrite(pedRed, HIGH);
    digitalWrite(redLight, LOW);
    digitalWrite(greenLight, HIGH);

    // Reset state
    buttonPressed = false;
    greenTimer = millis(); // Restart green light timer
  }
}

// Interrupt Service Routine (ISR) for button press
void buttonISR() {
  buttonPressed = true; // Set flag to true on button press
}
```