# Project 1: Implementation of a Reactive State Machine

## 1. Introduction

The objective of this project was to design, develop, and implement a reactive state machine on an embedded system. Embedded systems are frequently event-driven, responding to environmental inputs and internal timers to transition between defined states of operation. This project models such a system using an Arduino Nano 33 BLE Sense microcontroller.

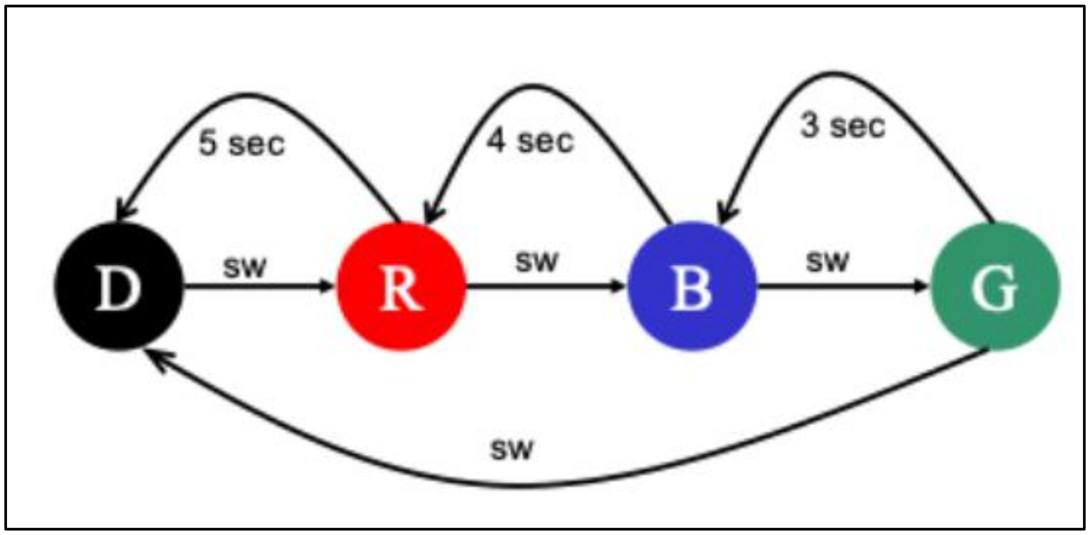

Figure 1. State transition diagram

The system's behavior was defined by a state transition diagram shown in *Figure 1*, where the system state is visually represented by a specific color on the board's built-in RGB LED. Transitions between these states are triggered by two types of events: a simulated user input (a "soft switch") and automated timeouts. The primary goal was to translate this abstract diagram into functional hardware and software, demonstrating a core competency in embedded systems design.

## 2. System Design and Components

To meet the project requirements, a system was designed using the onboard peripherals of the Arduino Nano 33 BLE Sense, requiring no external components. The design is broken down into its hardware and software components below.

### 2.1 Hardware Design

The hardware implementation is self-contained within the Arduino development board.

- **Microcontroller:** The **Arduino Nano 33 BLE Sense** was chosen as the central processing unit. Its powerful processor is more than capable of handling the state logic, and its built-in peripherals, specifically the RGB LED and USB connectivity, make it ideal for this project without needing external wiring.

- **Output (Visual Indicator):** The system state is displayed using the board's **built-in RGB LED**. This is a common-anode LED, meaning the common pin is connected to a high voltage source. To turn on a specific color (red, green, or blue), the corresponding pin (Pin 22, 23, or 24) must be set to LOW. Setting a pin to HIGH turns the color off. This understanding was critical for implementing the setLedColor() function correctly. The 'Dark' state is achieved by setting all three color pins to HIGH.
- **Input (The "Soft Switch"):** As the board lacks a physical button, a "soft switch" was designed to provide user input. The chosen implementation uses the **Serial Monitor** interface available through the Arduino IDE. The system is programmed to listen for an incoming character ('S' or 's') over the USB serial connection. When the character is received, it is interpreted as a switch press. This method is highly reliable, simple to implement, and provides an excellent channel for sending debugging messages, making it superior for development and demonstration purposes.

### 2.2 Software Design

The software was developed in the Arduino C/C++ environment with a focus on simplicity, readability, and functional correctness.

- **State Representation:** The system's states are represented with simple integers defined using the #define preprocessor directive (0 for Dark, 1 for Red, etc.). This approach is memory-efficient and makes the switch statements clean and easy to follow.
- **Core Logic:** The main program loop() is divided into two logical sections. The first section checks for user input from the serial port. If the 'S' or 's' key is detected, a switch statement evaluates the *currentState* variable and executes the appropriate state transition. The second section of the loop handles the automated timeouts.
- **Non-Blocking Timers:** A key challenge in any reactive system is handling time-based events without halting the entire program. A naive approach using the delay() function would render the system unresponsive to switch presses during the timeout periods. To overcome this, a **non-blocking timer** was implemented using the *millis()* function.
  - A global variable, *stateEnterTime*, records the timestamp (in milliseconds) when a new state is entered.
  - In each loop iteration, the code checks if the elapsed time (*millis() - stateEnterTime*) has exceeded the required duration for the current state (e.g., 5000ms for the RED state).
  - This allows the Arduino to continuously check for switch presses and handle other tasks while "waiting" for a timeout, which is a fundamental concept for building responsive embedded systems.

## 3. Experiment Design and Test Cases

To comprehensively validate the system's functionality, a series of test cases were designed to verify every transition shown in the state diagram. The experiment involves performing a specific action in each state and observing if the system responds with the expected outcome.

The following table details the 7 test cases that cover all required state transitions:

| **Test Case ID** | **Initial State** | **Trigger** | **Expected Outcome** | **Observed Outcome** |
|---|---|---|---|---|
| **T1** | Dark | Press Switch ('S') | State changes to Red; LED turns red. | Pass |
| **T2** | Red | Wait 5 seconds | State changes to Dark; LED turns off. | Pass |
| **T3** | Red | Press Switch ('S') | State changes to Blue; LED turns blue. | Pass |
| **T4** | Blue | Press Switch ('S') | State changes to Green; LED turns green. | Pass |
| **T5** | Green | Wait 3 seconds | State changes to Blue; LED turns blue. | Pass |
| **T6** | Blue | Wait 4 seconds | State changes to Red; LED turns red. | Pass |
| **T7** | Green | Press Switch ('S') | State changes to Dark; LED turns off. | Pass |

Each of these test cases was validated during the final demonstration. While not executed in a strict, linear T1-T7 order, the demonstration video shows a dynamic run-through where all transitions were successfully triggered, confirming the correctness of the hardware and software design.

## 4. Implementation and Demonstration

The final C++ code was uploaded to the Arduino Nano 33 BLE Sense using the Arduino IDE. The accompanying video demonstrates the project's successful implementation in a continuous, dynamic run that mirrors a real-world usage scenario.

In the video, the system is operated to showcase all 7 transitions outlined in the experiment design. The demonstration begins by showing the system timing out from Red back to Dark, then being reactivated and transitioned through the Red, Blue, and Green states via switch presses. It also clearly shows the various timeout behaviors, such as Green timing out to Blue, which in turn times out to Red. The demonstration confirms that all switch-based and time-based transitions function correctly and reliably in a fluid, interconnected sequence.

## 5. Challenges and Lessons Learned

The primary challenge in this project was not the state logic itself, but the implementation of the timers. My initial thought was to use delay(), but I quickly realized this would block the loop() and prevent the system from detecting the 'S' key press during a timeout period.

This led to the most important lesson learned: the critical difference between blocking and non-blocking code. By using the millis() function, I learned how to manage time-dependent events while keeping the system fully responsive to external inputs. This concept is fundamental to almost all advanced embedded systems, and this project provided a practical and clear example of its importance.

Furthermore, I observed that using the Serial Monitor as both an input mechanism and a debugging tool was incredibly efficient. It allowed me to print the current state and the reason for a transition (e.g., "Timeout: RED -> DARK"), which made troubleshooting and verification straightforward.

## 6. Conclusion

This project was a successful exercise in embedded systems design and development. The specified state machine was fully implemented on the Arduino Nano 33 BLE Sense by leveraging its built-in RGB LED and a serial-based soft switch. The final implementation correctly handles all 7 required state transitions, including both user-triggered and time-triggered events. The project reinforced my understanding of state-based logic, hardware-software integration, and the critical importance of non-blocking code for creating responsive, event-driven systems.

## 7. References

1. Arduino Language Reference. (2025). millis(). Arduino.cc. https://www.arduino.cc/reference/en/language/functions/time/millis/
2. Arduino Language Reference. (2025). Serial. Arduino.cc. https://www.arduino.cc/reference/en/language/functions/communication/serial/
3. Arduino Language Reference. (2025). digitalWrite(). Arduino.cc. https://www.arduino.cc/reference/en/language/functions/digital-io/digitalwrite/

## Final Source Code

```
// State Definitions
// Using simple numbers to represent the system states.
#define STATE_DARK  0
#define STATE_RED   1
#define STATE_BLUE  2
#define STATE_GREEN 3

// LED Pin Definitions
// Physical pins for the built-in RGB LED on the Nano 33 BLE Sense.
const int RED_PIN = 22;
const int GREEN_PIN = 23;
const int BLUE_PIN = 24;

// Variable to hold the current state, starting with DARK (0).
int currentState = STATE_DARK;

// Variable to store the last time a state with a timer was entered.
unsigned long stateEnterTime = 0;

void setup() {
  // Initialize serial communication at 115200 baud for the soft switch and debugging.
  Serial.begin(115200);

  // Set the LED pins as outputs.
  pinMode(RED_PIN, OUTPUT);
  pinMode(GREEN_PIN, OUTPUT);
  pinMode(BLUE_PIN, OUTPUT);

  // Start in the DARK state.
  setLedColor(STATE_DARK);
  Serial.println("System ready. Send 'S' to press the switch.");
}

void loop() {
  // Part 1: Check for the Switch Press ('S' key as a Serian Input)
  // This block checks if there is any data available to read from the serial port
  if (Serial.available() > 0) {
    char input = Serial.read();   // Read the incoming character.
    // Check if the input character is 'S' or 's' to trigger the switch.
    if (input == 'S' || input == 's') {
      // Use a switch statement to determine the next state based on the current state.
      switch (currentState) {
```

```
      // CASE: The system is currently in the DARK state.
      case STATE_DARK:
        Serial.println("Switch: DARK -> RED");
        // Transition to the RED state upon switch press.
        currentState = STATE_RED;
        break;

      // CASE: The system is currently in the RED state.
      case STATE_RED:
        Serial.println("Switch: RED -> BLUE");
        // Transition to the BLUE state upon switch press.
        currentState = STATE_BLUE;
        break;

      // CASE: The system is currently in the BLUE state.
      case STATE_BLUE:
        Serial.println("Switch: BLUE -> GREEN");
        // Transition to the GREEN state upon switch press.
        currentState = STATE_GREEN;
        break;

      // CASE: The system is currently in the GREEN state.
      case STATE_GREEN:
        Serial.println("Switch: GREEN -> DARK");
        // Transition back to the DARK state upon switch press, completing the cycle.
        currentState = STATE_DARK;
        break;
    }
    // Every time the state changes, update the LED and reset our timer.
    setLedColor(currentState);
    stateEnterTime = millis();
  }
}

// Part 2: Check for Timeouts
// This part runs continuously, checking if enough time has passed.
switch (currentState) {
  // CASE: The system is currently in the RED state.
  case STATE_RED:
    // Check if 5000ms (5 seconds) have passed since entering this RED state.
    if (millis() - stateEnterTime >= 5000) {
      Serial.println("Timeout: RED -> DARK");
      // If the timeout is reached, transition back to the DARK state.
      currentState = STATE_DARK;
```

```
        setLedColor(currentState);
        stateEnterTime = millis(); // Reset timer for the new state.
      }
      break;

    // CASE: The system is currently in the BLUE state.
    case STATE_BLUE:
      // Check if 4000ms (4 seconds) have passed since entering this BLUE state.
      if (millis() - stateEnterTime >= 4000) {
        Serial.println("Timeout: BLUE -> RED");
        // If the timeout is reached, transition back to the RED state.
        currentState = STATE_RED;
        setLedColor(currentState);
        stateEnterTime = millis(); // Reset timer for the new state.
      }
      break;

    // CASE: The system is currently in the GREEN state.
    case STATE_GREEN:
      // Check if 3000ms (3 seconds) have passed since entering this GREEN state.
      if (millis() - stateEnterTime >= 3000) {
        Serial.println("Timeout: GREEN -> BLUE");
        // If the timeout is reached, transition back to the BLUE state.
        currentState = STATE_BLUE;
        setLedColor(currentState);
        stateEnterTime = millis(); // Reset timer for the new state.
      }
      break;
  }
}

// This function sets the physical LED pins based on the current state number.
// The LED is common-anode, so LOW turns a color ON and HIGH turns if OFF.
void setLedColor(int state) {
  // Use a switch statement to apply the correct pin configuration for the desired color.
  switch (state) {

    // CASE: The desired state is DARK.
    case STATE_DARK:
      // Turn all colors OFF by setting their pins to HIGH.
      digitalWrite(RED_PIN, HIGH);
      digitalWrite(GREEN_PIN, HIGH);
      digitalWrite(BLUE_PIN, HIGH);
      break;
    // CASE: The desired state is RED.
    case STATE_RED:
      // Turn the RED color ON (LOW) and the others OFF (HIGH).
      digitalWrite(RED_PIN, LOW);
      digitalWrite(GREEN_PIN, HIGH);
      digitalWrite(BLUE_PIN, HIGH);
      break;

    // CASE: The desired state is BLUE.
    case STATE_BLUE:
      // Turn the BLUE color ON (LOW) and the others OFF (HIGH).
      digitalWrite(RED_PIN, HIGH);
      digitalWrite(GREEN_PIN, HIGH);
      digitalWrite(BLUE_PIN, LOW);
      break;

    // CASE: The desired state is GREEN.
    case STATE_GREEN:
      // Turn the GREEN color ON (LOW) and the others OFF (HIGH).
      digitalWrite(RED_PIN, HIGH);
      digitalWrite(GREEN_PIN, LOW);
      digitalWrite(BLUE_PIN, HIGH);
      break;
  }
}
```