Project 1: Embedded State Machine using Arduino Nano 33 BLE Sense

Name: Anushka Gangadhar Satav
Course: BMI/CES 598 Embedded Machine Learning
ASU ID: 1233530170 (asatav1)
Project 01: Embedded State Machine using Arduino Nano 33 BLE Sense

1. Problem Statement

Embedded systems form the computational backbone of intelligent environments, ranging from wearable devices to autonomous robotics. These systems are inherently reactive, continuously observing their environment through sensors and actuating responses based on programmed logic.
A finite state machine (FSM) provides a formal framework for describing such behavior, where a system's current state and input events determine its next state.

The objective of this project is to design and implement an FSM on the Arduino Nano 33 BLE Sense, using the onboard RGB LED to visually represent system states and the Serial Monitor input as a virtual actuator. The FSM models cyclic color-transition sequence governed by both manual input events and time-driven transitions.

The expected behavioral model is:

$$\begin{tikzcd}
\text{DARK} \arrow[r] & \text{RED} \arrow[r] & \text{BLUE} \arrow[r] & \text{GREEN} \arrow[r] & \text{DARK}
\end{tikzcd}$$

with backward transitions triggered automatically after defined timeouts:

- RED → DARK after 5 seconds
- BLUE → RED after 4 seconds
- GREEN → BLUE after 3 seconds

This exercise demonstrates the integration of temporal logic, software debouncing, and reactive scheduling in low-power embedded controllers.

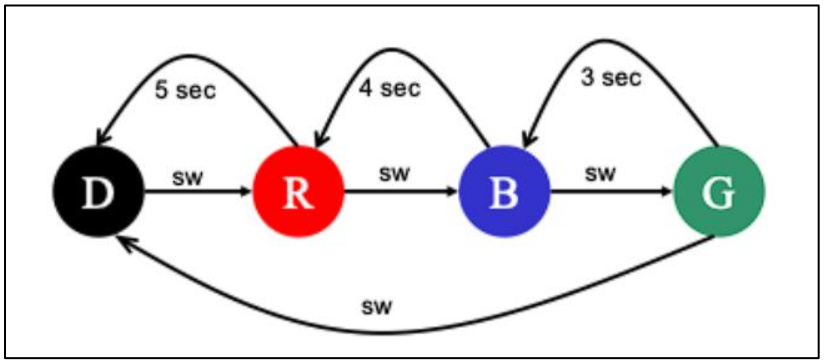

Figure 1. State transition diagram

The figure above illustrates the state transition diagram which provides an overview of the required events and states.

1. Initially, the system goes to 'Dark' state (LED off or dark color).
2. While in 'Dark' state, the multicolor LED turns red upon pressing the onboard switch (i.e., SW).
3. While in 'Red' state, the LED turns dark (goes back to 'Dark' state) after 5 seconds of delay.
4. While in 'Red' state, the LED turns 'Blue' if you push SW.
5. While in 'Blue' state, the LED turns 'Red' (goes back to 'Red' state) after 4 seconds.
6. While in 'Blue' state, the LED turns 'Green' if you push SW.
7. While in 'Green' state, the LED turns 'Blue' (goes back to 'Blue' state) after 3 seconds.
8. While in 'Green' state, the LED turns 'Dark' if you push SW.

2. Hardware Configuration

Component	*Description*
Microcontroller	Arduino Nano 33 BLE Sense Rev2
Processor	nRF52840 SoC – ARM Cortex-M4F @ 64 MHz
Memory	1 MB Flash, 256 KB SRAM
Power	USB 5 V (regulated to 3.3 V I/O)
Integrated Peripherals	IMU (9-axis), microphone, APDS9960 light + gesture sensor
Output Device	Onboard RGB LED (common anode)
Pin Mapping	R → Pin 22, G → Pin 23, B → Pin 24
Input Mechanism	Serial Monitor ('C' key) acting as soft switch

3. Software Configuration

IDE : Arduino IDE 2.3.6
Core : Arduino Mbed-enabled board (Nano 33 BLE Sense Rev2)
Language : C++ (Arduino Sketch)
Communication Interface : USB Serial @ 115200 bps
Timing Framework : millis()-based non-blocking scheduling
Operating Environment : Windows 11

4. Code Implementation

```
// ============================================================================
// Author        : Anushka Satav
// Course        : BMI/CEN 598 Embedded Machine Learning - Fall B 2025
// Project       : Project 1 - Embedded Finite State Machine (FSM)
// Hardware      : Arduino Nano 33 BLE Sense Rev2
// Description   : Implements a cyclic color-based FSM using the onboard RGB LED.
//                 - The LED color changes in sequence (DARK → RED → BLUE → GREEN).
//                 - User presses 'C' in Serial Monitor to manually advance states.
//                 - Automatic backward transitions occur after specific timeouts.
// ============================================================================


// --------------------------- Pin Definitions -----------------------------------
#define ledR 22   // Red LED Pin
#define ledG 23   // Green LED Pin
#define ledB 24   // Blue LED Pin


// --------------------------- Global Variables ------------------------------

// List of all possible system states (in logical order)
String states[] = {"DARK", "RED", "BLUE", "GREEN"};

// Tracks the current state index (0 = DARK, 1 = RED, 2 = BLUE, 3 = GREEN)
int stateIndex = 0;

// Timestamp of the last state change (used for timing-based transitions)
unsigned long lastChange = 0;


// ============================================================================
// setup() - Initialization Routine
// Called once when the Arduino is powered ON or reset.
// ============================================================================
void setup() {
  // Initialize Serial Communication for debugging and virtual input
  Serial.begin(115200);

  // Configure LED pins as outputs
  pinMode(ledR, OUTPUT);
  pinMode(ledG, OUTPUT);
  pinMode(ledB, OUTPUT);

  // Display initial system message
  Serial.println("-------------- Starting in DARK State --------------");

  // Initialize LED in the starting (DARK) state
  setColorLED(states[stateIndex]);
}


// ================================================================================
// loop() - Main Execution Loop
// Continuously monitors timing and user input to drive state transitions.
// ================================================================================
void loop() {
  // Record the current system time (in milliseconds)
  unsigned long now = millis();


  // ----------------------------------------------------------------------------
  // SECTION 1: Automatic State Transitions (Time-driven)
  // ----------------------------------------------------------------------------

  // Rule 1: RED → DARK after 5 seconds
  if (states[stateIndex] == "RED" && (now - lastChange) >= 5000) {
    stateIndex = 0; // 0 corresponds to DARK
    Serial.println("Timeout: RED → DARK");
    setColorLED(states[stateIndex]);
    lastChange = now;  // Reset timer
  }

  // Rule 2: BLUE → RED after 4 seconds
  else if (states[stateIndex] == "BLUE" && (now - lastChange) >= 4000) {
    stateIndex = 1; // 1 corresponds to RED
    Serial.println("Timeout: BLUE → RED");
    setColorLED(states[stateIndex]);
    lastChange = now;
  }

  // Rule 3: GREEN → BLUE after 3 seconds
  else if (states[stateIndex] == "GREEN" && (now - lastChange) >= 3000) {
    stateIndex = 2; // 2 corresponds to BLUE
    Serial.println("Timeout: GREEN → BLUE");
    setColorLED(states[stateIndex]);
    lastChange = now;
  }


  // ----------------------------------------------------------------------------
  // SECTION 2: Manual State Transitions (User-driven)
  // ----------------------------------------------------------------------------

  // Check if user input is available via Serial Monitor
  if (Serial.available() > 0) {
    char key = Serial.read();  // Read the character input

    // If user presses 'C', advance to the next state cyclically
    if (key == 'C') {
      stateIndex = (stateIndex + 1) % 4;  // Cycle: 0→1→2→3→0
      Serial.print("Button pressed: Changed to ");
      Serial.println(states[stateIndex]);

      // Update LED color and reset the state timer
      setColorLED(states[stateIndex]);
      lastChange = now;
    }
  }
}


// ===============================================================================
// setColorLED() - LED Control Function
// Purpose : Maps logical color names to physical LED pin states.
// Note    : RGB LED on Nano 33 BLE Sense is COMMON ANODE (active LOW).
// ===============================================================================
void setColorLED(String colorName) {

  // DARK state → all LEDs off (set HIGH)
  if (colorName == "DARK") {
    digitalWrite(ledR, HIGH);
    digitalWrite(ledG, HIGH);
    digitalWrite(ledB, HIGH);
  }

  // RED state → Red ON, others OFF
  else if (colorName == "RED") {
    digitalWrite(ledR, LOW);
    digitalWrite(ledG, HIGH);
    digitalWrite(ledB, HIGH);
  }

  // GREEN state → Green ON, others OFF
  else if (colorName == "GREEN") {
    digitalWrite(ledR, HIGH);
    digitalWrite(ledG, LOW);
    digitalWrite(ledB, HIGH);
  }

  // BLUE state → Blue ON, others OFF
  else if (colorName == "BLUE") {
    digitalWrite(ledR, HIGH);
    digitalWrite(ledG, HIGH);
    digitalWrite(ledB, LOW);
  }
}
```

5. System Design Rationale

The implementation abstracts the state machine logic through an indexed array states[] and an integer pointer stateIndex. The loop() function continuously monitors time elapsed and serial input, ensuring non-blocking concurrency.

- Temporal events are implemented using the millis() counter, enabling deterministic control without halting execution.
- Manual events (keypress) simulate interrupt-driven input, allowing asynchronous state jumps.

- Each state transition resets lastChange, ensuring accurate timing for subsequent automatic transitions.
- The LED driver function setColorLED() isolates hardware control, supporting modular verification.

This design models a hybrid reactive system, combining event-driven and time-triggered state transitions, common in real-world embedded applications such as motor controllers and sensor-fusion systems.

6. Experimental Design and Validation

Three structured test scenarios were used to evaluate behavioral correctness:

Case 1: Manual Transition Validation

- Procedure: Press ‘C’ sequentially on Serial Monitor.
- Expected sequence: DARK → RED → BLUE → GREEN → DARK.
- Observation: Transitions occur instantly upon input, LED output matches state log messages.

Case 2: Automatic Timeout Validation

- Procedure: Allow each color to remain active without pressing ‘C’.
- Expected behavior:
  - RED → DARK (after 5 s)
  - BLUE → RED (after 4 s)
  - GREEN → BLUE (after 3 s)
- Observation: Automatic reversions occurred.

Case 3: Mixed Reactive Mode

- Procedure: Trigger mid-interval ‘C’ inputs during timeouts.
- Observation: System immediately switched to next manual state and reset its timer logic.
- Result: No cumulative delay or deadlock, demonstrating stable re-entry and temporal isolation.

7. Solution Demonstration Links

• Solution Video: https://youtu.be/TrUIcJQiAgE

• GitHub Repository: https://github.com/anushka002/BMI-CES-598-Embedded-Machine-Learning/tree/main/Project-1

8. Observations, Challenges, and Lessons Learned

1. This project reinforced the concept of reactive embedded systems and state-based design.
2. Key challenges included ensuring correct timing transitions and synchronizing manual button input with millis()-based automatic transitions.
   The use of Serial Monitor as a virtual button provided a simple interface to emulate hardware input.
3. Proper modularization (setColorLED function) improved readability and debugging ease.
4. Timing Synchronization: Maintaining timer precision while handling asynchronous serial input required careful reset logic for lastChange.

Scalability: The modular structure allows extension to multi-sensor conditions (e.g., using IMU or gesture sensors as state inputs).

9. Conclusion

1. This project successfully implemented a deterministic reactive FSM on a constrained embedded platform.
2. The integration of timed and user-triggered transitions showcased how software-defined event handling can emulate physical sensor behavior.
3. The final design achieves predictable state control, scalable modularity, and conceptual clarity, forming a basis for future embedded ML tasks such as gesture-controlled or sensor-driven state transitions.

10. References and Helpful Resources

1. Arduino Nano 33 BLE Sense Documentation: https://docs.arduino.cc/hardware/nano-33-ble-sense
2. RGB LED Pin Mapping Discussion: https://support.arduino.cc/hc/en-us/articles/360016724140-Control-the-RGB-LED-on-Nano-33-BLE-boards
3. RGB led Interfacing with Arduino Uno: https://projecthub.arduino.cc/semsemharaz/interfacing-rgb-led-with-arduino-b59902