

Project 1 – State Machine Using Arduino Nano 33 BLE Sense

Student: Lana Popoola
Course: Embedded Systems / IoT Fundamentals
Board: Arduino Nano 33 BLE Sense
Switch Pin: D4 (External Push Button)
LED Pins: R = 22, G = 23, B = 24 (Active LOW)

1. Hardware Design

Overview

The **Arduino Nano 33 BLE Sense** has a built-in **RGB LED** connected internally to pins 22 (Red), 23 (Green), and 24 (Blue). The LED operates in **active-LOW** configuration, driving a pin LOW turns ON the corresponding color. Because the board lacks an onboard push-button for GPIO input, an **external tactile switch** was connected between **pin D4** and **GND**, with the internal pull-up resistor enabled (**INPUT_PULLUP**). Pressing the button pulls D4 LOW, triggering a state transition.

Component	Connection	Purpose
RGB LED	Built-in (pins 22–24)	Visualizes the current state
Push Button	D4 → GND	Input for state transition
Internal Pull-Up	Enabled	Provides a stable HIGH default

Hardware Behavior: State Machine Mapping

- Each LED color represents a unique state:
 - **Dark:** All pins HIGH (LED off)
 - **Red:** 22 LOW, others HIGH
 - **Blue:** 24 LOW, others HIGH
 - **Green:** 23 LOW, others HIGH
 - The button acts as the trigger for manual transitions.
 - Timers (`millis()`) handle automatic transitions.
-

2. State Machine Design and Logic

Current State	Event	Next State	Description
Dark	Button Press	Red	The LED turns red
Red	5s timeout	Dark	Auto returns to dark
Red	Button Press	Blue	Manual transition
Blue	4s timeout	Red	Auto return
Blue	Button Press	Green	Manual transition
Green	3s timeout	Blue	Auto return
Green	Button Press	Dark	Manual transition (back to start)

A total of **7 transitions** were implemented, matching the state diagram.

3. Experimental Design / Test Cases

Test Setup

- Board powered via USB.
- Serial monitor at 9600 baud for state logs.
- Button press simulated via jumper if needed.

Test Cases

	Description	Expected Result
1	System reset	LED = Dark (Off)
2	Press the button once	Dark → Red
3	Wait 5s (no press)	Red → Dark
4	Press while Red	Red → Blue
5	Wait 4s (no press)	Blue → Red
6	Press while Blue	Blue → Green

7	Wait 3s (no press)	Green → Blue
8	Press while Green	Green → Dark (restart)

All states and transitions were verified visually and through Serial output.

4. Challenges Faced and Solutions

Challenge	Cause / Observation	Solution / Action Taken
RGB LED not turning off properly	The built-in RGB LED on the Nano 33 BLE Sense operates as active LOW , which turns ON when the pin is set LOW instead of HIGH. Initially, the LED colors overlapped and did not go completely dark.	Corrected the logic by inverting the control signals (LOW = ON , HIGH = OFF) and explicitly setting all LED pins HIGH before lighting a new color.
Button input unstable (flickering)	The mechanical push button caused signal bounce , triggering multiple unintended state transitions.	Implemented a software debounce (200 ms) using millis() to ensure stable button detection.
Color overlap between transitions	Not resetting all LED pins before lighting a new color caused unwanted color blending.	An allOff() function was added that sets all LED pins HIGH before activating the next color.
The board is not responding to button presses.	The breadboard was partially damaged , causing poor electrical contact and an unreliable signal between D4 and GND. The switch sometimes didn't register presses.	Verified the issue by directly connecting jumper wires from D4 to GND to simulate button presses. Once confirmed, the wiring was adjusted and switched to a different breadboard row to ensure solid contact. This step helped confirm that the software logic was working and the fault was purely hardware-related
Initial unresponsive switch setup	The push button's four-legged configuration wasn't oriented	Reoriented the switch so that each side of the button used

	correctly power and signal were on the same side.	opposite legs. Added a test sketch that printed “Button Pressed” to the Serial Monitor to verify functionality before integrating it into the whole state machine.
Loss of timed state when using <code>delay()</code>	Using <code>delay()</code> prevented real-time button detection during waits.	Replaced all delays with non-blocking timing logic using <code>millis()</code> for smooth transitions and responsive input.

5. Lessons Learned

- Importance of understanding **active-LOW** hardware behavior.
 - Using **finite state machines (FSMs)** simplifies complex logic.
 - **Non-blocking timing** (`millis()`) is vital for real-time input.
 - Proper debounce handling prevents false triggers.
 - Testing every transition visually confirms FSM correctness.
-

6. Video Link:

<https://www.youtube.com/watch?v=jgZVr2BOntw>