Finite State Machine (FSM)

What is a Finite State Machine? (FSM)

Imagine your favorite toy robot.
The robot can do only **a few different things**:

- It can **sleep**.
- It can **walk**.
- It can **dance**.

These are like its **moods** or **modes**.
In computer words, we call them **states**.

States

States are like **moods**.

“The robot is SLEEPING.”
“The robot is WALKING.”
“The robot is DANCING.”

At any moment, the robot can be in **only one mood** (state).

Events

Something happens that can **change the mood**.
For example:

- Someone **presses a button**.
- A timer **beeps**.
- The robot **hears music**.

We call these things **events**.

Transitions

When an **event** happens, the robot **changes from one mood to another**.

For example:

- The robot is **sleeping**.
- You **press the button**.
- Now it **starts walking**.

Changing from **sleeping** → **walking** because of a button press is called a **transition**.

Actions

When the robot changes its mood or is in a mood, it can **do something**:

- If it is in **walking** state → move its legs.
- If it is in a dancing state → spin around.
- When going from **walking** → **dancing** → maybe play a sound.

These things are called **actions**.

All together:

- The robot has a few **states**.
- It listens for **events**.
- When an event happens, it can **transition** to a new state.
- In each state, or when changing, it can do **actions**.

Why do we use FSMs?

When we build small computers (like your ESP32), we want them to:

- Act in a few known modes.
- Change modes only when something happens.
- Do the right thing in each mode.

FSM helps us write the code in a way that is:

1. Clear
2. Organized
3. Easy to read

Example from your ESP32:

Think about an LED light:

- LED can be:
    1. OFF
    2. ON
    3. BLINKING

  These are the **states**.

If you **press a button**:

- If LED is OFF → turn ON.
- If LED is ON → start BLINKING.
- If LED is BLINKING → turn OFF.

Like playing with a toy:
Press once → light on
Press again → light blinks
Press again → light off
Press again → light on … and so on

Summary in baby words:

- A finite state machine (FSM) is like giving your robot or light a few moods.
- The robot watches for events like button press.
- It changes mood (state) when something happens.
- In each mood, it does different things.

That's it!
States → Events → Transitions → Actions

Drawing a simple FSM

Imagine we want to make a light that:

- Can be **OFF**
- Can be **ON**
- Can be **BLINKING**

We also have **one button** to control it.

Step 1: Draw the states

Think of each state as a **circle** with the name inside:

[OFF]   [ON]   [BLINKING]

These are the “moods” of the light.

Step 2: Draw the event

The event is **button press**.
Every time we press, something happens.

Step 3: Draw the transitions

Let’s add **arrows** to show what happens when we press the button.

```
 [ OFF ] ---button press---> [ ON ]
    ^                          |
    |                          |
    |                     button press
    |                          |
    |                          v
  [ BLINKING ] <---button press-- [ ON ]
```

To make it easier:

- From **OFF** → press → go to **ON**
- From **ON** → press → go to **BLINKING**
- From **BLINKING** → press → go to **OFF**

Then it keeps repeating.

Step 4: Think about actions

In each state, what does the light do?

State	**Action**
Off	Light is OFF
On	Light is always ON
Blinking	Light turns on/off every half second

Step 5: Say the story out loud

"The light is OFF.
I press the button → it turns ON.
I press again → it starts BLINKING.
I press again → it turns OFF.
Then repeat…"

That's your FSM!
A simple story of **states**, **events**, and **transitions**.

Why do we draw it?

Drawing the circles and arrows helps your brain:

1. See what states exist.
2. See how to move between them
3. Plan what code to write

Quick recap in very simple words:

- **States** → circles (OFF, ON, BLINKING)
- **Event** → button press
- **Transition** → arrow to next circle
- **Action** → what happens in each circle

It's just like drawing a cartoon or a comic:

- Each frame = a state
- Arrow = what makes the story move on

Part 1: Hardware configuration (in baby words)

We have:

- One **LED** (tiny light)
- One **button** (like a doorbell)
- ESP32 board (the small computer)

Wiring:

Part	ESP32 pin	Why
LED longer leg (+)	GPIO2	This pin will “push” electricity to turn LED on
LED shorter leg (–)	Resistor → GND	Resistor stops too much electricity, GND takes electricity away
One leg of button	GPIO4	ESP32 watches this pin to see if button is pressed
Other leg of button	GND	When you press, the pin sees 0V (LOW)

Why do we use resistor for LED?

- LED is like a little door → if too much electricity goes in, it can break.
- Resistor is like a gatekeeper → it only allows safe amount.

Why connect button between GPIO4 and GND?

- ESP32 keeps GPIO4 normally HIGH inside itself (INPUT_PULLUP).
- When you press, you “pull” it down to GND → ESP32 sees LOW.
- Simple and safe.

So now ESP32 can:

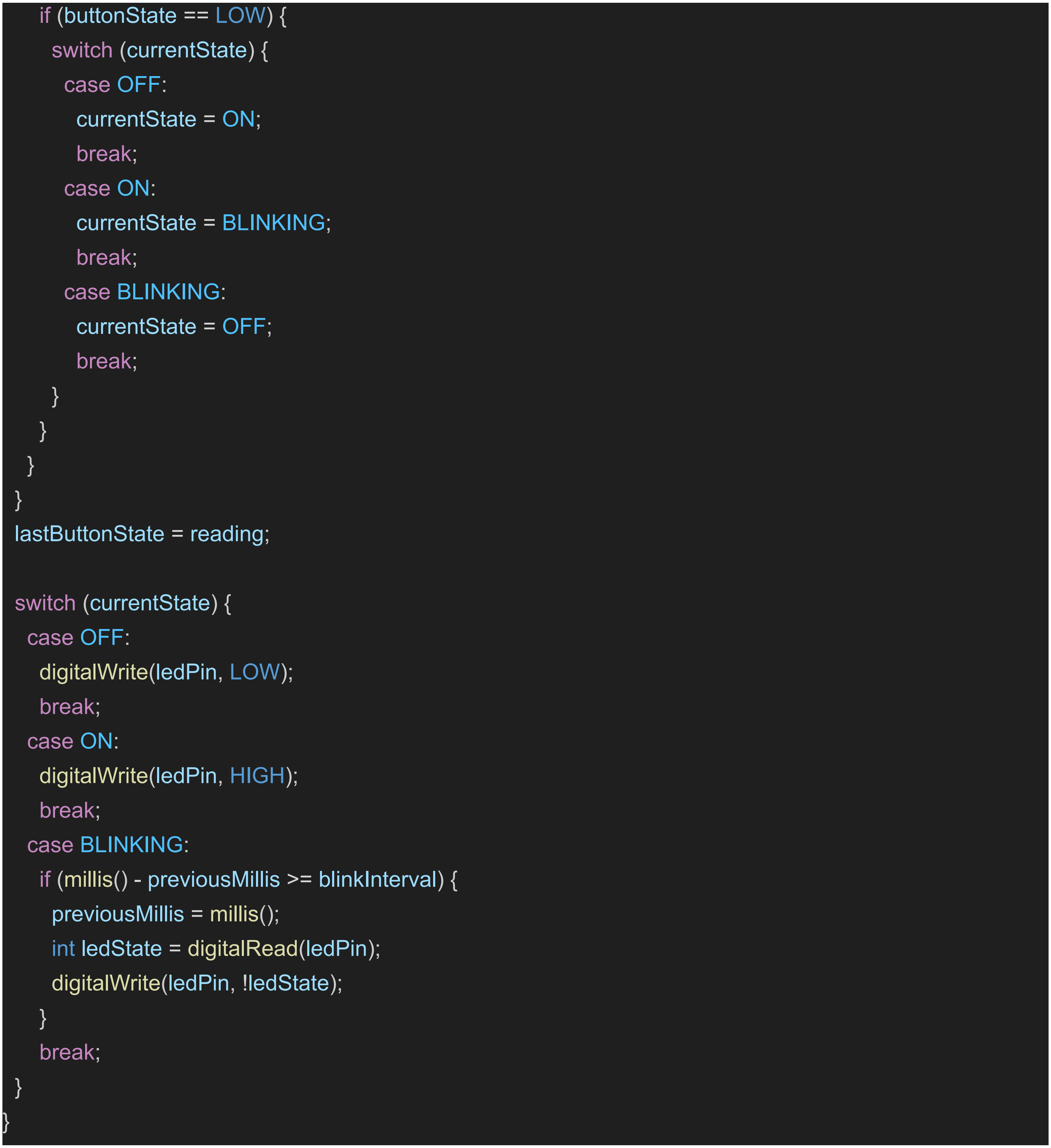
- See if button is pressed (GPIO4 becomes LOW)
- Turn LED on/off by controlling GPIO2

Pin summary:

Pin	Use
GPIO2	LED
GPIO4	Button (with internal pull-up)

Part 2: The code:

```
#include <Arduino.h>

// Define states
enum State { OFF, ON, BLINKING };
State currentState = OFF;

// Pins
const int ledPin = 2;
const int buttonPin = 4;

// Blinking
unsigned long previousMillis = 0;
const long blinkInterval = 500;

// Debounce
unsigned long lastDebounceTime = 0;
unsigned long debounceDelay = 50;
bool lastButtonState = HIGH;
bool buttonState = HIGH;

void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
}

void loop() {
  bool reading = digitalRead(buttonPin);

  if (reading != lastButtonState) {
    lastDebounceTime = millis();
  }

  if ((millis() - lastDebounceTime) > debounceDelay) {
    if (reading != buttonState) {
      buttonState = reading;
```



```
      if (buttonState == LOW) {
        switch (currentState) {
          case OFF:
            currentState = ON;
            break;
          case ON:
            currentState = BLINKING;
            break;
          case BLINKING:
            currentState = OFF;
            break;
        }
      }
    }
  }
  lastButtonState = reading;

  switch (currentState) {
    case OFF:
      digitalWrite(ledPin, LOW);
      break;
    case ON:
      digitalWrite(ledPin, HIGH);
      break;
    case BLINKING:
      if (millis() - previousMillis >= blinkInterval) {
        previousMillis = millis();
        int ledState = digitalRead(ledPin);
        digitalWrite(ledPin, !ledState);
      }
      break;
  }
}
```

```
#include <Arduino.h>
```

Tell ESP32:
"I want to use Arduino language and commands."

```
// Define states
enum State { OFF, ON, BLINKING };
State currentState = OFF;
```

We give names to moods: OFF, ON, BLINKING.
enum is like saying:

- OFF = 0
- ON = 1
- BLINKING = 2
  currentState keeps track of which mood we are in.
  We start in OFF.

```
// Pins
const int ledPin = 2;
const int buttonPin = 4;
```

Tell ESP32 which pins we use.

- The LED is on pin GPIO2.
- Button is on pin GPIO4.

```
// Blinking
unsigned long previousMillis = 0;
const long blinkInterval = 500;
```

For blinking:

- Remember the last time we changed LED.
- Blink every 500 milliseconds (half a second).

```
// Debounce
unsigned long lastDebounceTime = 0;
unsigned long debounceDelay = 50;
bool lastButtonState = HIGH;
bool buttonState = HIGH;
```

Button can be noisy: when you press, it can quickly go HIGH/LOW.

- debounceDelay = 50 milliseconds = wait a tiny moment to be sure.
- lastButtonState and buttonState remember what we saw last time.

setup() — first things ESP32 does at start

```
void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
}
```

Tell ESP32:

- The LED pin is OUTPUT → ESP32 can push voltage to turn LED on/off.
- Button pin is INPUT with PULLUP → normally stays HIGH until we press (then becomes LOW).

loop() — ESP32 keeps doing this forever

```
void loop() {
```

Like a merry-go-round: spins again and again, very fast.

Read button now

```
 bool reading = digitalRead(buttonPin);
```

Ask ESP32: “Is button pressed right now?”

- If HIGH → not pressed.
- If LOW → pressed.

Check if reading changed

```
 if (reading != lastButtonState) {
   lastDebounceTime = millis();
 }
```

If button state changed (from last time we looked):

- Remember **when** it changed → mark the time with millis() (number of milliseconds since ESP32 started).

Wait for debounce

```
 if ((millis() - lastDebounceTime) > debounceDelay) {
```

Wait 50ms → to be sure it’s not button “noise”.

If stable and changed, update button state

```
   if (reading != buttonState) {
     buttonState = reading;
```

Confirm new stable button state.

If the button is pressed now → change state!

```
  if (buttonState == LOW) {
    switch (currentState) {
      case OFF:
        currentState = ON;
        break;
      case ON:
        currentState = BLINKING;
        break;
      case BLINKING:
        currentState = OFF;
        break;
    }
  }
```

If the button is LOW → it means we pressed it.
Then:

- If the LED was OFF → turn ON.
- If LED was ON → start BLINKING.
- If BLINKING → turn OFF.

Like turning pages in our FSM story.

Remember last button state

```
   }
  }
  lastButtonState = reading;
```

Keep for next round to compare.

Now, do the action for current state

Use switch

```
 switch (currentState) {
```

Choose what to do based on mood.

OFF: turn LED off

```
   case OFF:
     digitalWrite(ledPin, LOW);
     break;
```

LED is off → no light.

ON: turn LED on

```
  case ON:
    digitalWrite(ledPin, HIGH);
    break;
```

LED stays on → always bright.

BLINKING: blink every 500ms

```
  case BLINKING:
    if (millis() - previousMillis >= blinkInterval) {
      previousMillis = millis();
      int ledState = digitalRead(ledPin);
      digitalWrite(ledPin, !ledState);
    }
    break;
  }
}
```

- Check if 500ms have passed since last blink.
- If yes → remember new times.
- Read LED state now (ON or OFF).
- Write the opposite → blink!

Like blinking eyes: open → close → open → close.

Summary:

- ESP32 remembers its mood (OFF, ON, BLINKING).
- Watch button → when pressed → change mood.
- In each mood:
  - OFF → LED off
  - ON → LED on
  - BLINKING → blink every half second

ESP32 keeps doing this very fast, forever.

Extra tips

- Use only safe pins for button (GPIO4, etc.)
- Use resistor for LED (220Ω–330Ω)
- `INPUT_PULLUP` keeps button pin HIGH, pressing button makes it LOW.
- `millis()` gives time in milliseconds → helps blinking and debounce.