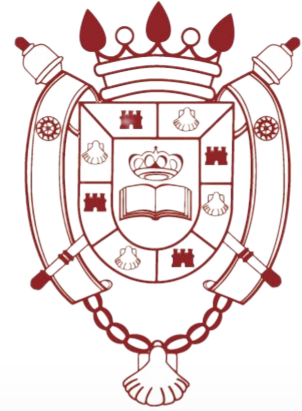
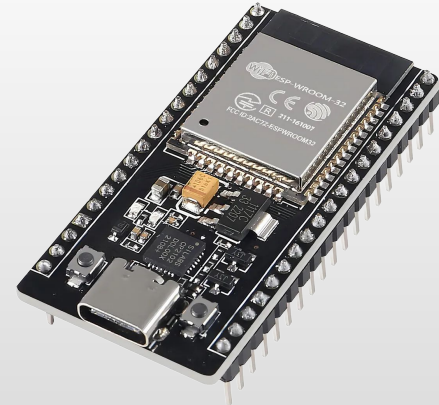


Internet de las Cosas con ESP32



Segunda Clase: Entrada digital,
PWM y programación no bloqueante

Ing. Fernando Raúl Vera Suasnívar



Universidad Nacional de Santiago del Estero - 2025

Objetivos

- 1) Controlar intensidad **LED con PWM**
- 2) Leer **pulsadores con antirrebote** (debounce)
- 3) Usar **millis()** para código **no bloqueante**
- 4) Diseñar **máquinas de estado multitarea**

"Cualquier cosa que pueda conectarse, se conectará."

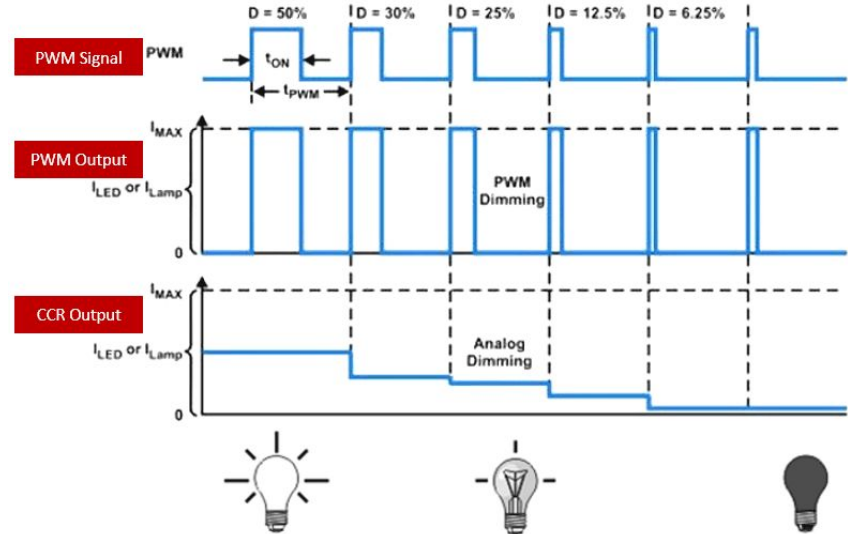
– Kevin Ashton, pionero en RFID e IoT (1999)

1.1 LED con PWM

El **PWM** permite
simular señales analógicas

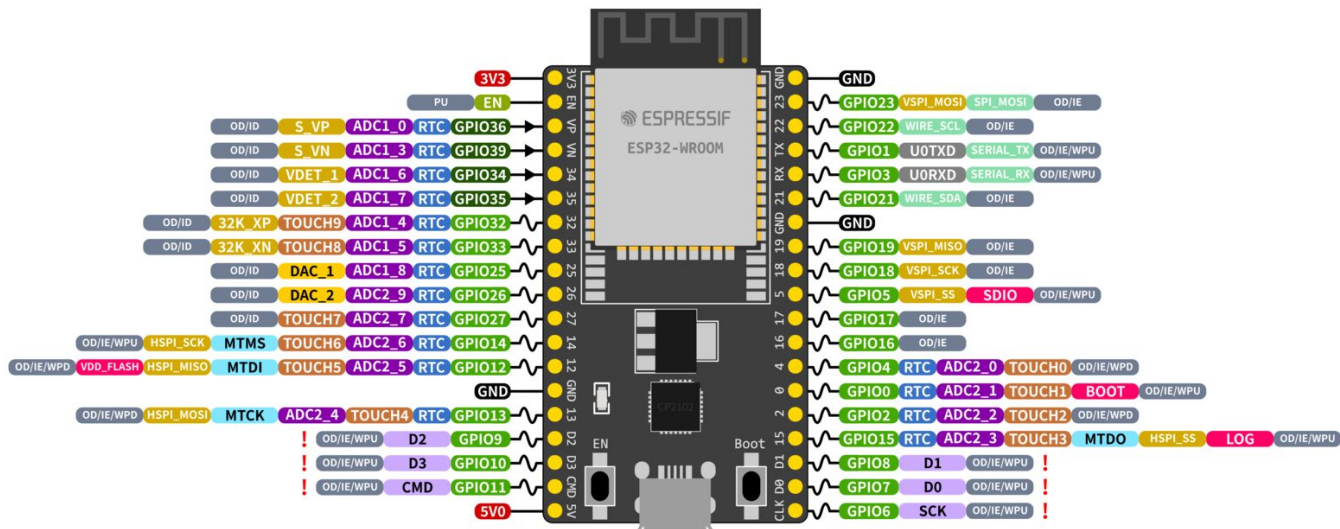
ESP32 estándar

16 canales, hasta 20 bits,
40 MHz o más



“La ingeniería es lograr con precisión lo que otros llaman imposible.” – Claude Shannon

ESP32-DevKitC



ESP32 Specs

32-bit Xtensa® dual-core @240MHz
 Wi-Fi IEEE 802.11 b/g/n 2.4GHz
 Bluetooth 4.2 BR/EDR and BLE
 520 KB SRAM (16 KB for cache)
 448 KB ROM
 34 GPIOs, 4x SPI, 3x UART, 2x I2C,
 2x I2S, RMT, LED PWM, 1 host SD/eMMC/SDIO,
 1 slave SDIO/SPI, TWAI®, 12-bit ADC, Ethernet



RTC: RTC Power Domain (VDD3P3_RTC)
GND: Ground
PWD: Power Rails (3V3 and 5V)
! Pin Shared with the Flash Memory
 Can't be used as regular GPIO

GPIO STATE

- WPU:** Weak Pull-up (Internal)
- WPD:** Weak Pull-down (Internal)
- PU:** Pull-up (External)
- IE:** Input Enable (After Reset)
- ID:** Input Disabled (After Reset)
- OE:** Output Enable (After Reset)
- OD:** Output Disabled (After Reset)

Consideraciones GPIO

Nivel lógico

3.3 V (⚠ no tolera 5 V)

Corriente

12 mA recomendado (40 mA máx.)

Restricciones

Boot: GPIO0, 2, 15

Flash: GPIO6–11


Solo entrada: GPIO34–39

Seguros para práctica

GPIO2, 4, 5,
12–19, 21–23,
25–27, 32–33

GPIO: General Purpose Input/Output

1.2 LED con PWM (Código)

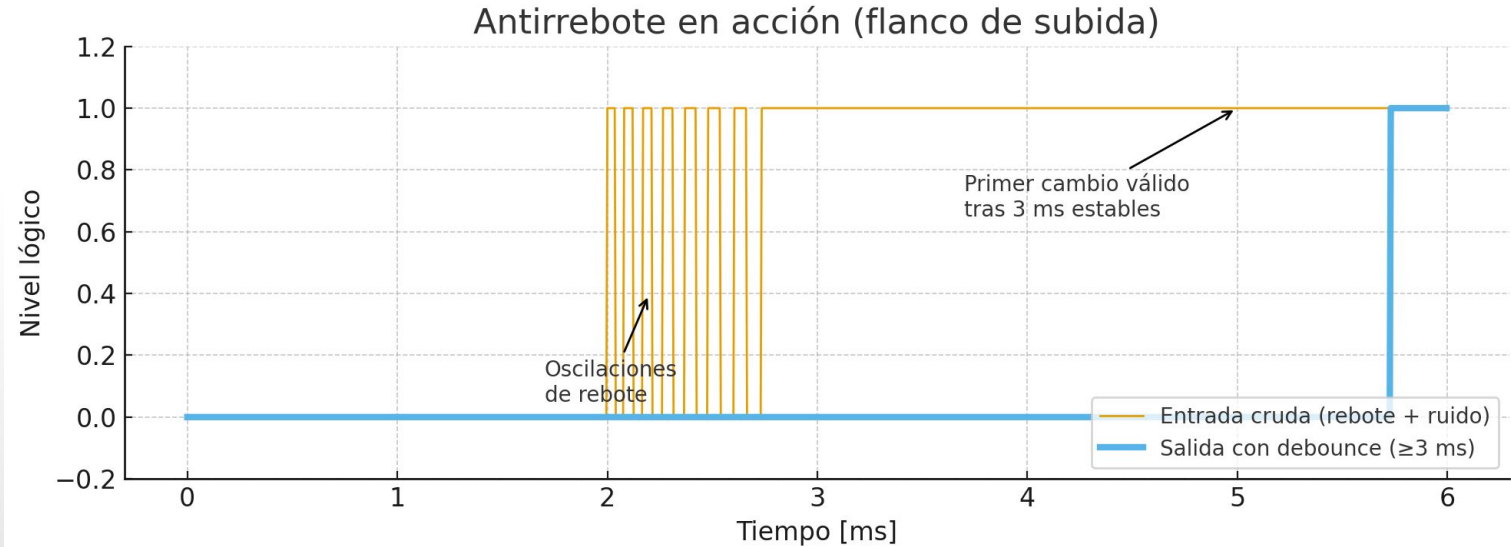


```
// Configura el canal 0 en 5 kHz
// con 8 bits de resolución (0-255).
ledcSetup(0, 5000, 8);

// Asocia el pin GPIO2 al canal 0 de PWM.
ledcAttachPin(2, 0);

// Envía un duty cycle del 50% (128/255)
// → LED a mitad de brillo.
ledcWrite(0, 128);
```

2.1 Pulsadores con antirrebote


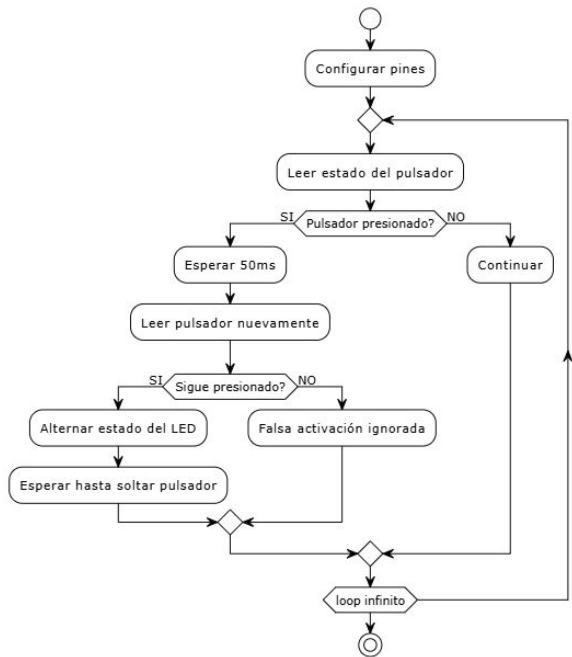


Un **botón rebota** y genera falsas lecturas.



Solución: Hardware RC o Software con tiempo mínimo.

2.2 Pulsadores con antirrebote (Código)



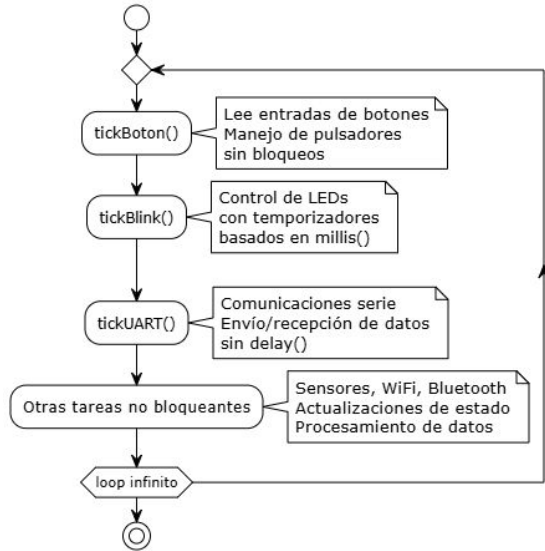
```
#include <Arduino.h>

const int buttonPin = 0;
const int ledPin = 2;

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

void loop() {
  if (digitalRead(buttonPin) == LOW) {
    delay(50);
    // espera establecimiento
    if (digitalRead(buttonPin) == LOW) {
      // Invierte el estado del LED
      // con cada pulsación
      digitalWrite(ledPin, !digitalRead(ledPin));
      // espera soltar
      while (digitalRead(buttonPin) == LOW);
    }
  }
}
```


3.1 Programación no bloqueante



delay() bloquea

el loop se detiene y el sistema "se pierde" eventos.

millis() permite temporizar sin frenar

multitarea cooperativa.

Patrón "elapsed time":

```
if (millis() - t0 >= intervalo) { ...; t0 = millis(); }
```

Varios timers virtuales

un `t0` por tarea (blink, sensores, serial, etc.).

Overflow seguro

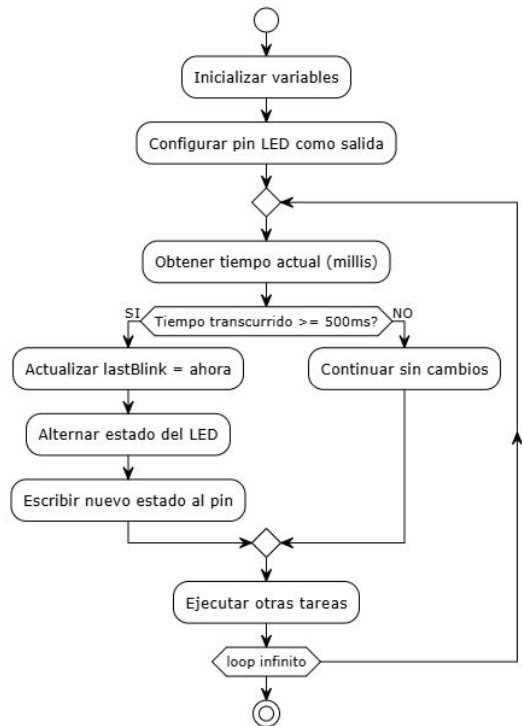
`millis()` reinicia cada ~49.7 días; la resta sigue siendo válida.

Estructura limpia

separar tareas en funciones

(`tickBlink()`, `tickBton()`, etc.) mejora legibilidad y testeo.

3.2 Blink no bloqueante (robusto)



```
#include <Arduino.h>

#define PIN_LED 2
#define BLINK_MS 500

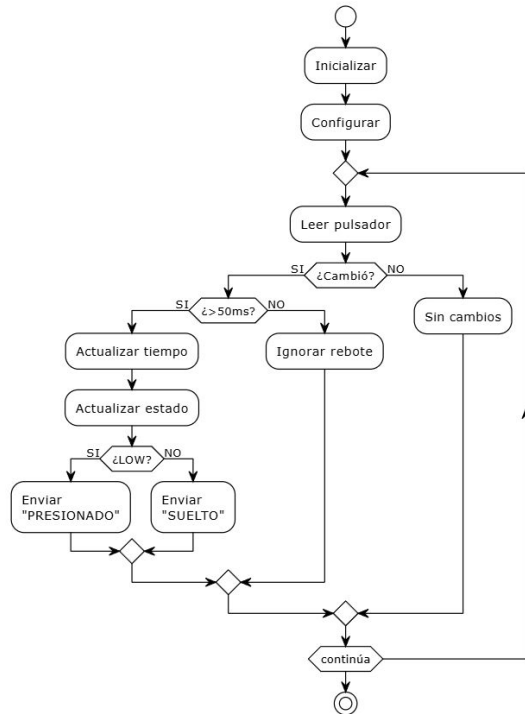
unsigned long lastBlink = 0;
bool ledState = false;

void setup() {
  pinMode(PIN_LED, OUTPUT);
}

void loop() {
  unsigned long ahora = millis();
  if (ahora - lastBlink >= BLINK_MS) {
    lastBlink = ahora;
    ledState = !ledState;
    digitalWrite(PIN_LED, ledState);
  }
  // aquí se pueden leer botones,
  // UART, sensores, etc.
}
```



3.3 Pulsador no bloqueante



```
#include <Arduino.h>

#define BUTTON_PIN 0
#define PIN_LED 2

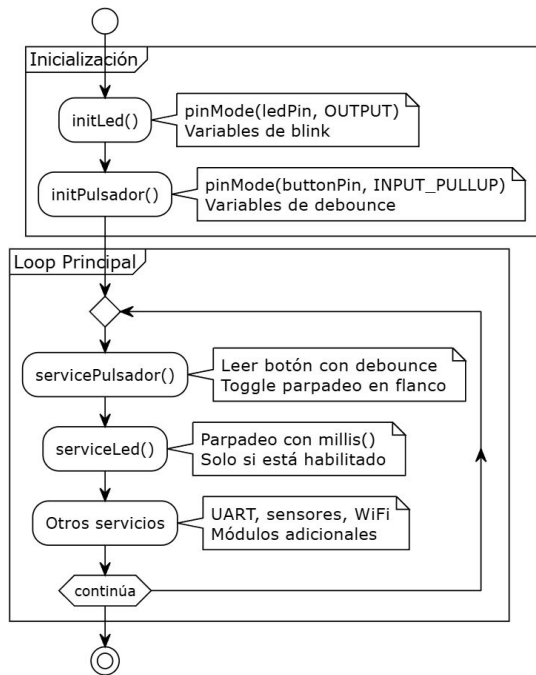
uint32_t lastDebounce = 0;
uint8_t stable = HIGH;

void setup() {
  Serial.begin(115200);
  pinMode(BUTTON_PIN, INPUT_PULLUP);
}

void loop() {
  int r = digitalRead(BUTTON_PIN);
  if (r != stable && millis() - lastDebounce > 50) {
    lastDebounce = millis();
    stable = r;
    Serial.println(
      stable == LOW ?
      "PRESIONADO" :
      "SUELTO"
    );
  }
}
```



3.4 Blink + Pulsador no bloqueantes (modular)



Separar inicialización y servicio en funciones:

initLed() / initPulsador()

→ configuración en setup()

serviceLed() / servicePulsador()

→ lógica en loop()

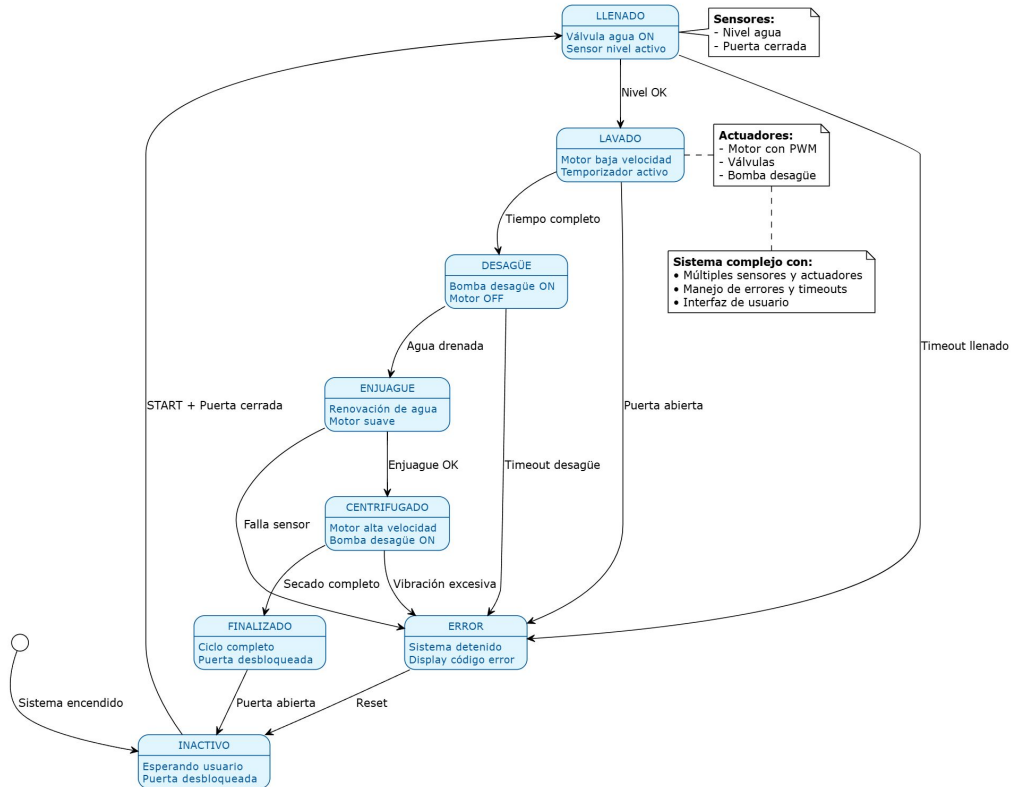
Cada función hace una sola tarea

→ más fácil de probar y mantener.

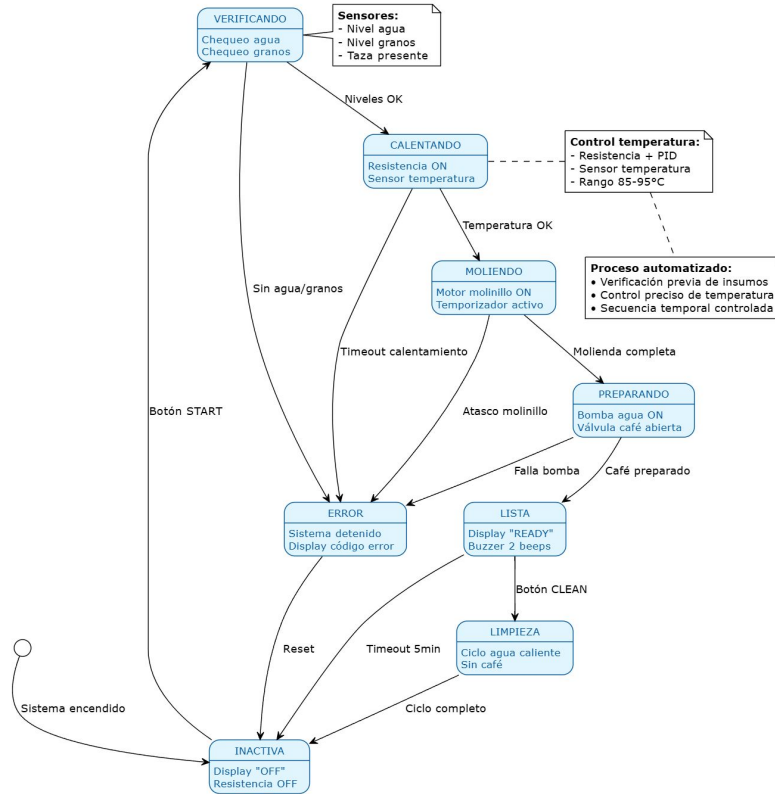
*Esquema ideal para crecer
en proyectos IoT más grandes*



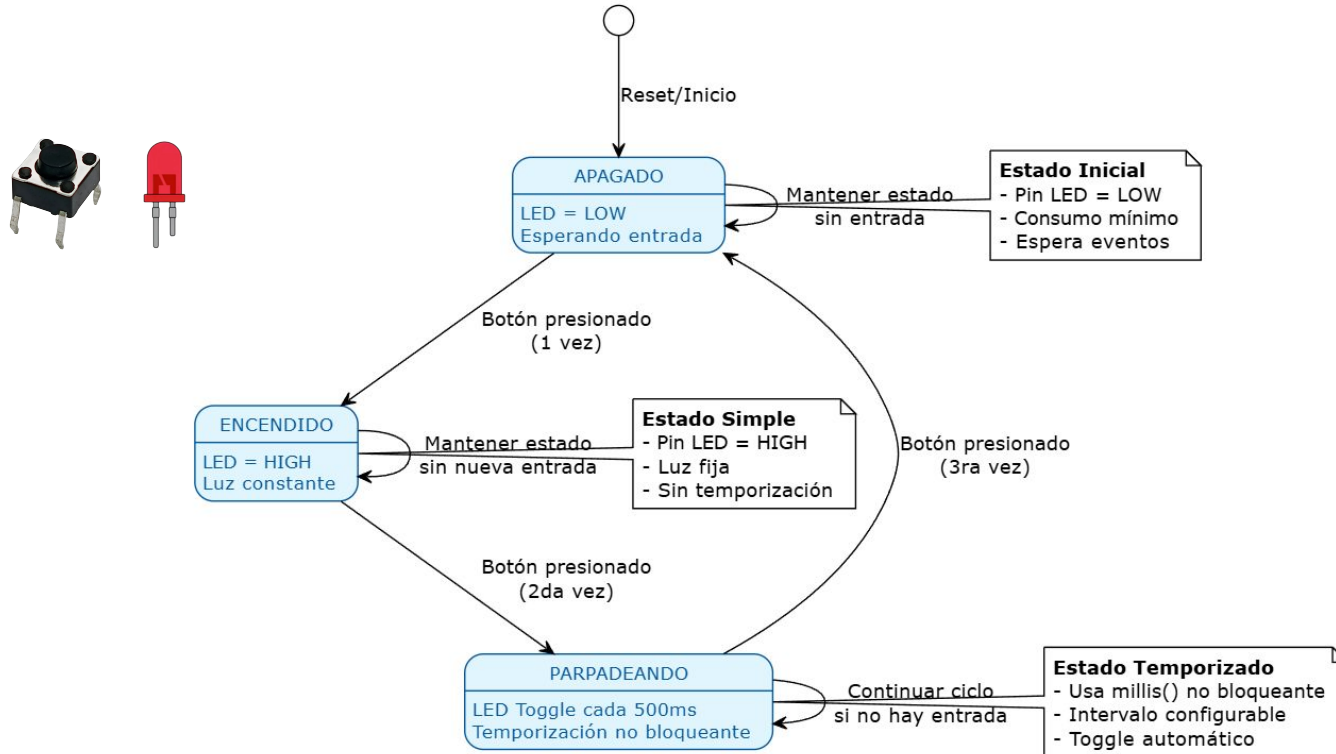
4.1 Máquinas de Estado Finito FSM



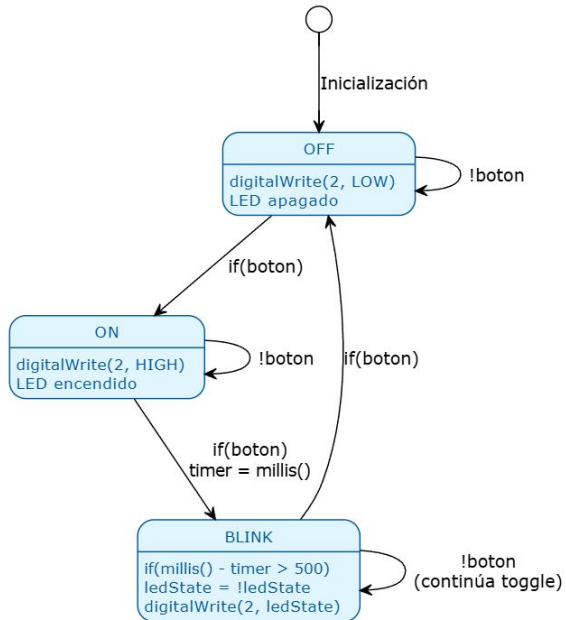
4.2 Máquinas de Estado Finito FSM



4.3 Máquinas de Estado Finito FSM



4.4 Máquinas de Estado Finito FSM



```
switch (state) {  
  
  case STATE_OFF:  
    digitalWrite(PIN_LED, LOW);  
    if (buttonPressed) {  
      buttonPressed = 0;  
      state = STATE_ON;  
    }  
    break;  
  
  case STATE_ON:  
    digitalWrite(PIN_LED, HIGH);  
    if (buttonPressed) {  
      buttonPressed = 0;  
      state = STATE_BLINK;  
    }  
    break;  
  
  case STATE_BLINK:  
    {  
      uint32_t ahora = millis();  
      if (ahora - lastBlink >= BLINK_MS) {  
        lastBlink = ahora;  
        ledState = !ledState;  
        digitalWrite(PIN_LED, ledState);  
      }  
      if (buttonPressed) {  
        buttonPressed = 0;  
        state = STATE_OFF;  
      }  
    }  
    break;  
  // ...  
}
```



¡Hora de practicar!

Poné a prueba lo aprendido

pulsador + LED + millis() + FSM

Construí tu propio código paso a paso

Experimentá, probá y no tengas miedo de equivocarte

"Lo que escucho lo olvido, lo que veo lo recuerdo, lo que hago lo aprendo."

— Confucio