Práctica 1: Entorno y simulación

Sistemas Embebidos

Jordi Blasco Lozano
DNI: 74527208D
Universidad de Alicante – Escuela Politécnica Superior
Sistemas Embebidos – Curso 2025/2026
Email: jbl42@alu.ua.es

Resumen

En esta práctica se aborda la familiarización con el entorno de desarrollo y simulación Wokwi para sistemas embebidos basados en ESP32. Se divide en dos partes: la Parte A, donde se implementa un circuito básico con un potenciómetro, una pantalla OLED y un LED utilizando una placa ESP32-C3 DevKitM-1; y la Parte B, donde se simula un pulsómetro completo con el sensor MAX30102, una pantalla OLED y un LED RGB sobre una ESP32-C3 SuperMini, incluyendo la creación de un chip personalizado para Wokwi que reproduce el comportamiento del sensor real.

1. Introducción

El objetivo de esta práctica es familiarizarse con el entorno de simulación Wokwi y con la plataforma ESP32-C3 para el desarrollo de sistemas embebidos. La práctica se divide en dos partes con complejidad creciente:

- **Parte A:** Montaje y simulación de un circuito básico con un potenciómetro (que simula un sensor de pulso), una pantalla OLED SSD1306 y un LED conectados a una ESP32-C3 DevKitM-1. Se realizan lecturas analógicas del potenciómetro, se mapean a valores de BPM y se muestran por pantalla.
- **Parte B:** Simulación completa de un sistema de pulsometría con el sensor MAX30102, una pantalla OLED y un LED RGB sobre una ESP32-C3 SuperMini. Se desarrolla un chip personalizado en Wokwi que emula fielmente el comportamiento del sensor real mediante comunicación I2C, permitiendo validar el código antes de la implementación en hardware físico.

2. Parte A: Circuito básico con potenciómetro, OLED y LED

2.1. Descripción del circuito

En esta primera parte se monta un circuito sencillo compuesto por los siguientes elementos:

- **ESP32-C3 DevKitM-1:** Microcontrolador principal que gestiona la lectura del sensor y el control de los componentes.
- **Potenciómetro:** Actúa como sensor analógico simulado. Su pin de señal (SIG) se conecta al pin GPIO 0 de la ESP32 para realizar lecturas analógicas. Los pines de alimentación (VCC) y tierra (GND) se conectan a 3.3 V y GND de la placa respectivamente.
- **Pantalla OLED SSD1306 (128×64):** Pantalla I2C que muestra los valores de BPM calculados. Se conecta mediante el bus I2C: SDA al pin GPIO 8 y SCL al pin GPIO 9. Su alimentación va a 3.3 V y el GND a tierra.

- **LED rojo con resistencia (22 Ω):** Indica actividad del sistema mediante parpadeos. El ánodo se conecta a través de la resistencia al pin GPIO 2 y el cátodo a GND.

Todas las tierras (GND) de los componentes se conectan al pin GND común de la placa ESP32, y las alimentaciones a la salida de 3.3 V. Esto es fundamental para que todos los componentes compartan la misma referencia de tensión y la comunicación I2C funcione correctamente.

2.2. Diagrama del circuito

A continuación se muestra el diagrama de conexiones en formato JSON utilizado por el simulador Wokwi:

```
{
  "version": 1,
  "author": "Anonymous maker",
  "editor": "wokwi",
  "parts": [
    { "type": "board-esp32-c3-devkitm-1", "id": "esp",
      "top": 38.1, "left": 5.82, "attrs": {} },
    { "type": "wokwi-led", "id": "led1",
      "top": 6, "left": -101.8,
      "attrs": { "color": "red" } },
    { "type": "wokwi-resistor", "id": "r1",
      "top": 80.75, "left": -67.2,
      "attrs": { "value": "22" } },
    { "type": "board-ssd1306", "id": "oled1",
      "top": 214.34, "left": 182.63,
      "attrs": { "i2cAddress": "0x3c" } },
    { "type": "wokwi-potentiometer", "id": "pot1",
      "top": 37.1, "left": -201.8, "attrs": {} }
  ],
  "connections": [
    [ "esp:TX", "$serialMonitor:RX", "", [] ],
    [ "esp:RX", "$serialMonitor:TX", "", [] ],
    [ "led1:A", "r1:1", "red", [ "v0" ] ],
    [ "r1:2", "esp:2", "red", [ "v0" ] ],
    [ "led1:C", "esp:GND.2", "black", [ "v0" ] ],
    [ "oled1:VCC", "esp:3V3", "red", [ "v0" ] ],
    [ "oled1:SDA", "esp:8", "blue", [ "v0" ] ],
    [ "oled1:SCL", "esp:9", "yellow",
      [ "h0.3", "v-124.8" ] ],
    [ "pot1:VCC", "esp:3V3", "red", [ "v0" ] ],
    [ "pot1:SIG", "esp:0", "orange",
      [ "v0.4", "h0.4", "v-86.8" ] ],
    [ "esp:GND.6", "oled1:GND", "black",
      [ "h94.72" ] ],
    [ "esp:GND.5", "pot1:GND", "black", [ "h0" ] ]
  ],
  "dependencies": {}
}
```

Listing 1: diagram.json – Parte A

2.3. Enlace a la simulación

La simulación completa de la Parte A está disponible en Wokwi en el siguiente enlace:

`https://wokwi.com/projects/456851474112359425`

2.4. Código fuente

El código del *sketch* implementa las siguientes funcionalidades:

1. **Inicialización:** Se configura la comunicación serie a 115200 baudios, se inicializan los pines GPIO, el bus I2C (SDA en pin 8, SCL en pin 9) y la pantalla OLED. Se muestra un mensaje de "Iniciando..." durante 2 segundos.

2. **Secuencia de parpadeo:** Se ejecutan 5 parpadeos rápidos del LED (100 ms encendido, 100 ms apagado) seguidos de una pausa de 1 segundo, indicando que el sistema está listo.

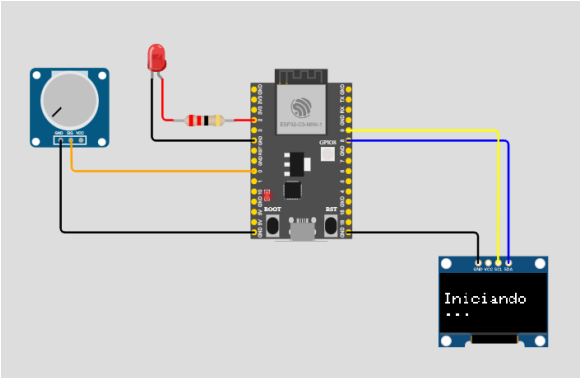
3. **Bucle principal:** Se lee el valor analógico del potenciómetro (rango 0–4095) y se mapea a un rango de BPM (40–180). El valor se muestra tanto por el monitor serie como en la pantalla OLED. El LED parpadea brevemente en cada ciclo para indicar actividad. El ciclo se repite cada 500 ms.

```
// Práctica 1 - Sistemas Embebidos - PARTE A
// Simulación con Wokwi: LED Blink, OLED y Potenciómetro

#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

// Definición de pines
#define LED_BLINK 2       // LED para secuencia de parpadeos
#define POT_PIN 0         // Potenciómetro (simula sensor MAX30105)
#define SDA_PIN 8         // I2C SDA
#define SCL_PIN 9         // I2C SCL

// Configuración OLED
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT,
                         &Wire, OLED_RESET);

int bpm = 0;
int lectura = 0;

void setup() {
  Serial.begin(115200);

  // Configurar pines
  pinMode(LED_BLINK, OUTPUT);
  pinMode(POT_PIN, INPUT);

  // Inicializar I2C
  Wire.begin(SDA_PIN, SCL_PIN);

  // Inicializar OLED
  if(!display.begin(SSD1306_SWITCHCAPVCC,
                    SCREEN_ADDRESS)) {
    Serial.println(
      F("Error: No se detectó la pantalla SSD1306"));
    for(;;);
  }

  // Mostrar mensaje inicial en OLED
  display.clearDisplay();
  display.setTextSize(2);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 20);
  display.println(F("Iniciando"));
  display.println(F("..."));
  display.display();

  Serial.println("Sistema iniciado - Parte A");
  Serial.println("Esperando 2 segundos...");

  delay(2000);

  // Secuencia de 5 parpadeos rápidos (100ms)
  // + pausa de 1 segundo
  Serial.println(
    "Ejecutando secuencia de blink: "
    "5 parpadeos rápidos");
  for(int i = 0; i < 5; i++) {
    digitalWrite(LED_BLINK, HIGH);
    delay(100);
    digitalWrite(LED_BLINK, LOW);
```

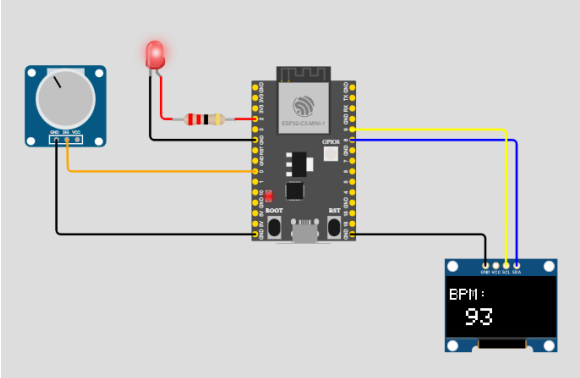
```
    delay(100);
    Serial.print("Parpadeo ");
    Serial.println(i + 1);
  }
  Serial.println("Pausa de 1 segundo...");
  delay(1000);
  Serial.println("Secuencia de blink completada");
  Serial.println(
    "Iniciando lectura de sensor simulado...");
}

void loop() {
  // Leer potenciómetro y simular sensor
  lectura = analogRead(POT_PIN);

  // Mapear lectura a BPM (40-180)
  bpm = map(lectura, 0, 4095, 40, 180);

  // Mostrar en Monitor Serial
  Serial.print("Lectura: ");
  Serial.print(lectura);
  Serial.print(" -> BPM: ");
  Serial.println(bpm);

  // Actualizar pantalla OLED con BPM
  display.clearDisplay();
  display.setTextSize(2);
  display.setCursor(0, 10);
  display.println(F("BPM:"));
  display.setTextSize(3);
  display.setCursor(20, 35);
  display.println(bpm);
  display.display();

  // Parpadeo simple del LED
  digitalWrite(LED_BLINK, HIGH);
  delay(50);
  digitalWrite(LED_BLINK, LOW);

  delay(450); // Total 500ms por ciclo
}
```

Listing 2: sketch.ino – Parte A

2.5. Capturas de la simulación



(a) Sistema inicializándose: pantalla OLED mostrando el mensaje “Iniciando...”.



(b) Sistema en funcionamiento: pantalla mostrando el valor de BPM leído del potenciómetro.

Figura 1: Capturas de la simulación de la Parte A en Wokwi.

3. Parte B: Pulsómetro con sensor MAX30102, OLED y LED RGB

3.1. Enfoque

Antes de realizar el montaje físico en el laboratorio, fue necesario recrear el sistema completo en Wokwi. Esta decisión se tomó por una razón práctica: no se disponia de una placa en casa y al día siguiente el montaje estaba ya desmontado.

Por lo que, se optó por simular al completo el circuito en Wokwi. De esta forma, se disponía de un diagrama limpio y visual que servía de referencia para reproducir el montaje físico de forma rápida y fiable durante las dos horas de clase disponibles. La estrategia consistía en tener la simulación completamente funcional, de modo que en la sesión práctica solo fuera necesario replicar el diagrama y cargar el código, optimizando así el tiempo limitado de laboratorio.

3.2. Descripción del circuito

El circuito de la Parte B se compone de los siguientes elementos, montados sobre una placa de conexiones (*breadboard*):

- **ESP32-C3 SuperMini:** Microcontrolador principal, elegido para que la simulación sea idéntica al escenario real de la práctica (a diferencia de la Parte A que usaba la DevKitM-1 estándar).
- **Sensor de pulso MAX30102:** Sensor óptico de pulsaciones y oximetría, comunicado por I2C (dirección 0x57). En la simulación se usa un chip personalizado de Wokwi.
- **Pantalla OLED SSD1306 (128×64):** Pantalla I2C (dirección 0x3C) que muestra el estado del sistema y los BPM medidos.
- **LED RGB (cátodo común):** Indica el estado del sistema mediante colores.
- **Breadboard:** Placa de conexiones para organizar el cableado de forma limpia.

3.2.1. Conexiones de alimentación

La alimentación se organiza aprovechando las líneas de la *breadboard*:

- El pin de 3.3 V de la ESP32 se conecta a la línea positiva superior de la *breadboard*.
- El pin GND se conecta a la línea negativa superior.
- Las líneas superiores e inferiores de la *breadboard* se interconectan (positiva con positiva, negativa con negativa) para disponer de puntos de alimentación en ambos lados. Esto permite conectar la tierra y el positivo de cada componente donde resulte más limpio el cableado.

3.2.2. Conexiones I2C (pantalla OLED y sensor MAX30102)

Tanto la pantalla OLED como el sensor MAX30102 comparten el mismo bus I2C:

- **SDA:** Conectado al pin GPIO 5 de la ESP32.
- **SCL:** Conectado al pin GPIO 6 de la ESP32.

Ambos dispositivos reciben alimentación desde las líneas positiva y negativa de la *breadboard*. El bus I2C permite conectar múltiples dispositivos en paralelo siempre que tengan direcciones distintas (0x3C para la OLED y 0x57 para el MAX30102).

3.2.3. Conexiones del LED RGB

El LED RGB es el único componente del circuito que presenta diferencias entre la simulación y el montaje real:

1. **Posición del cátodo:** En la simulación, la pata de tierra está en el segundo pin, mientras que en la práctica real se encuentra en el cuarto.
2. **Resistencias internas:** En el montaje real, el LED RGB incorpora resistencias internas dentro del propio encapsulado, por lo que no fue necesario añadir resistencias externas en el diagrama de simulación.
3. **Tipo de LED:** En la simulación se utilizó un LED de cátodo común, adaptándolo al componente disponible en Wokwi.

Las conexiones del LED RGB son:

- **Rojo (R):** Pin GPIO 7
- **Verde (G):** Pin GPIO 1
- **Azul (B):** Pin GPIO 0
- **Cátodo común (COM):** Conectado a GND

3.3. Chip personalizado MAX3010x para Wokwi

Dado que Wokwi no dispone de un componente nativo para el sensor MAX30102, se creó un chip personalizado que emula fielmente su comportamiento. Esto involucra dos archivos: un archivo JSON de definición y un archivo C con la lógica.

3.3.1. Definición del chip (JSON)

El archivo JSON define los pines y controles interactivos del chip:

```
{
  "name": "MAX3010x",
  "author": "you",
  "version": 1,
  "description":
    "MAX3010x pulse and proximity sensor",
  "pins": [
    "GND", "SCL", "SDA", "VIN",
    "", "", "", ""
  ],
  "controls": [
    { "id": "bpm", "label": "BPM",
      "type": "range",
      "min": 40, "max": 180, "step": 1 },
    { "id": "finger",
      "label": "Dedo (0=NO, 1=SI)",
      "type": "range",
      "min": 0, "max": 1, "step": 1 }
  ]
}
```

Listing 3: max3010x.chip.json

Se definen los 4 pines funcionales (GND, SCL, SDA, VIN) en un solo lado del chip. Para lograr esta disposición (4 pines en un lado y 0 en el otro), se añadieron 4 pines vacíos adicionales (), ya que Wokwi distribuye los pines equitativamente entre ambos lados del componente. Además, se incluyen dos controles interactivos: un deslizador de BPM (40–180) para ajustar la frecuencia cardíaca simulada, y un selector de dedo (0=no, 1=sí) para simular la presencia o ausencia del dedo sobre el sensor.

3.3.2. Lógica del chip (C)

El archivo `max3010x.chip.c` implementa la lógica completa del sensor simulado utilizando la API de chips personalizados de Wokwi. Sus principales características son:

- **Comunicación I2C:** Responde en la dirección 0x57 (la misma que el sensor real), implementando el protocolo completo de lectura/escritura de registros.
- **Mapa de registros:** Reproduce los registros esenciales del MAX30102 (FIFO, configuración de modo, configuración de partículas, amplitud de LEDs, ID de parte y revisión).
- **FIFO de datos:** Implementa un buffer circular de 32 posiciones donde se almacenan las muestras de los canales rojo e infrarrojo (18 bits cada una).
- **Generación de señal:** Genera una señal PPG (fotopletismografía) realista mediante la suma de dos funciones gaussianas que simulan el pulso sistólico y la onda dicrótica, con ruido añadido para mayor realismo.
- **Detección de dedo:** Cuando el control de "finger" está activo, genera valores IR en torno a 90000 (por encima del umbral de detección); cuando está inactivo, los valores caen a 10000 (por debajo del umbral).

De esta forma, por los pines I2C del chip sale exactamente la misma información que saldría si fuera un sensor MAX30102 real, permitiendo utilizar la librería SparkFun MAX3010x sin modificaciones.

```c
#include "wokwi-api.h"
#include <math.h>
#include <stdbool.h>
#include <stdint.h>
#include <stdlib.h>

#define I2C_ADDR 0x57

// Registros mínimos del MAX3010x
#define REG_INTSTAT1 0x00
#define REG_INTSTAT2 0x01
#define REG_INTEN1 0x02
#define REG_INTEN2 0x03
#define REG_FIFOWRITEPTR 0x04
#define REG_FIFOOVERFLOW 0x05
#define REG_FIFOREADPTR 0x06
#define REG_FIFODATA 0x07
#define REG_FIFOCONFIG 0x08
#define REG_MODECONFIG 0x09
#define REG_PARTICLECFG 0x0A
#define REG_LED1_PA 0x0C
#define REG_LED2_PA 0x0D
#define REG_REVID 0xFE
#define REG_PARTID 0xFF

#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

#define FIFO_DEPTH 32

typedef struct {
  i2c_dev_t i2c;
  timer_t timer;
  uint32_t attr_bpm;
  uint32_t attr_finger;
  uint8_t regs[256];
  uint8_t reg_ptr;
  bool expect_reg;
  uint8_t wr;
  uint8_t rd;
  uint8_t byte_idx;
  uint32_t fifo_red[FIFO_DEPTH];
  uint32_t fifo_ir[FIFO_DEPTH];
  float phase;
```

```
  uint32_t rng;
  uint32_t sample_rate_hz;
} state_t;

static inline uint32_t lcg(state_t *s) {
  s->rng = s->rng * 1664525u + 1013904223u;
  return s->rng;
}

static inline int32_t noise(state_t *s,
                            int32_t amp) {
  uint32_t r = (lcg(s) >> 8)
               % (uint32_t)(2 * amp + 1);
  return (int32_t)r - amp;
}

static inline uint32_t clamp18(int32_t v) {
  if (v < 0) v = 0;
  if (v > 0x3FFFF) v = 0x3FFFF;
  return (uint32_t)v;
}

static uint16_t decode_sample_rate(
    uint8_t spo2_cfg) {
  uint8_t sr = (spo2_cfg >> 2) & 0x07;
  switch (sr) {
    case 0: return 50;
    case 1: return 100;
    case 2: return 200;
    case 3: return 400;
    case 4: return 800;
    case 5: return 1000;
    case 6: return 1600;
    case 7: return 3200;
    default: return 100;
  }
}

static void fifo_clear(state_t *s) {
  s->wr = 0; s->rd = 0; s->byte_idx = 0;
  s->regs[REG_FIFOWRITEPTR] = 0;
  s->regs[REG_FIFOREADPTR] = 0;
  s->regs[REG_FIFOOVERFLOW] = 0;
}

static void update_timer(state_t *s) {
  uint16_t sr =
    decode_sample_rate(s->regs[REG_PARTICLECFG]);
  if (sr == 0) sr = 100;
  s->sample_rate_hz = sr;
  uint32_t period_us = 1000000u / (uint32_t)sr;
  if (period_us < 500) period_us = 500;
  timer_start(s->timer, period_us, true);
}

static void reset_device(state_t *s) {
  uint8_t part = s->regs[REG_PARTID];
  uint8_t rev = s->regs[REG_REVID];
  for (int i = 0; i < 256; i++) s->regs[i] = 0;
  s->regs[REG_PARTID] = part;
  s->regs[REG_REVID] = rev;
  s->regs[REG_FIFOCONFIG] = 0x10;
  s->regs[REG_MODECONFIG] = 0x03;
  s->regs[REG_PARTICLECFG] = 0x04;
  s->regs[REG_LED1_PA] = 0x1F;
  s->regs[REG_LED2_PA] = 0x1F;
  fifo_clear(s);
  update_timer(s);
}

static void fifo_push(state_t *s,
    uint32_t red18, uint32_t ir18) {
  uint8_t next = (uint8_t)((s->wr+1) & 0x1F);
  if (next == s->rd) {
    s->rd = (uint8_t)((s->rd+1) & 0x1F);
    s->regs[REG_FIFOREADPTR] = s->rd;
```

```
  }
  s->fifo_red[s->wr] = red18 & 0x3FFFF;
  s->fifo_ir[s->wr] = ir18 & 0x3FFFF;
  s->wr = next;
  s->regs[REG_FIFOWRITEPTR] = s->wr;
  s->regs[REG_INTSTAT1] |= 0x40;
}

static uint8_t fifo_read_byte(state_t *s) {
  if (s->wr == s->rd) return 0;
  uint32_t red = s->fifo_red[s->rd];
  uint32_t ir = s->fifo_ir[s->rd];
  uint8_t out = 0;
  switch (s->byte_idx) {
    case 0: out=(uint8_t)((red>>16)&0x03); break;
    case 1: out=(uint8_t)((red>>8)&0xFF); break;
    case 2: out=(uint8_t)(red&0xFF); break;
    case 3: out=(uint8_t)((ir>>16)&0x03); break;
    case 4: out=(uint8_t)((ir>>8)&0xFF); break;
    case 5: out=(uint8_t)(ir&0xFF); break;
    default: out=0; break;
  }
  s->byte_idx++;
  if (s->byte_idx >= 6) {
    s->byte_idx = 0;
    s->rd = (uint8_t)((s->rd+1) & 0x1F);
    s->regs[REG_FIFOREADPTR] = s->rd;
    if (s->wr == s->rd)
      s->regs[REG_INTSTAT1] &= (uint8_t)~0x40;
  }
  return out;
}

// --- Callbacks I2C ---

static bool on_connect(void *user_data,
    uint32_t address, bool read) {
  (void)address;
  state_t *s = (state_t *)user_data;
  if (!read) s->expect_reg = true;
  return true;
}

static bool on_write(void *user_data,
                     uint8_t data) {
  state_t *s = (state_t *)user_data;
  if (s->expect_reg) {
    s->reg_ptr = data;
    s->expect_reg = false;
    return true;
  }
  uint8_t reg = s->reg_ptr;
  s->regs[reg] = data;
  if (reg == REG_MODECONFIG) {
    if (data & 0x40) {
      reset_device(s);
      s->regs[REG_MODECONFIG] &=
        (uint8_t)~0x40;
    }
  } else if (reg == REG_PARTICLECFG) {
    update_timer(s);
  } else if (reg == REG_FIFOWRITEPTR) {
    s->wr = data & 0x1F;
  } else if (reg == REG_FIFOREADPTR) {
    s->rd = data & 0x1F;
    s->byte_idx = 0;
  }
  s->reg_ptr++;
  return true;
}

static uint8_t on_read(void *user_data) {
  state_t *s = (state_t *)user_data;
  if (s->reg_ptr == REG_FIFODATA)
    return fifo_read_byte(s);
  uint8_t reg = s->reg_ptr;
```

```
  uint8_t value = s->regs[reg];
  if (reg == REG_INTSTAT1 ||
      reg == REG_INTSTAT2)
    s->regs[reg] = 0;
  s->reg_ptr++;
  return value;
}

// --- Generación de señal PPG ---

static void on_timer(void *user_data) {
  state_t *s = (state_t *)user_data;
  if (s->regs[REG_MODECONFIG] & 0x80) return;

  uint32_t bpm = attr_read(s->attr_bpm);
  if (bpm < 30) bpm = 30;
  if (bpm > 220) bpm = 220;

  bool finger = attr_read(s->attr_finger) != 0;
  float sr = (float)(s->sample_rate_hz
                     ? s->sample_rate_hz : 100);
  float bps = (float)bpm / 60.0f;

  s->phase += (2.0f*(float)M_PI) * (bps / sr);
  if (s->phase >= (2.0f*(float)M_PI))
    s->phase -= (2.0f*(float)M_PI);

  float t = s->phase;
  float sigma1 = 0.14f;
  float sigma2 = 0.30f;
  float p1 = expf(
    -0.5f * (t/sigma1) * (t/sigma1));
  float p2 = 0.22f * expf(
    -0.5f * ((t-0.85f)/sigma2)
           * ((t-0.85f)/sigma2));

  int32_t base_ir  = finger ? 90000 : 10000;
  int32_t base_red = finger ? 75000 : 9000;
  int32_t amp_ir   = finger ? 32000 : 400;
  int32_t amp_red  = finger ? 22000 : 300;
  int32_t noise_amp = finger ? 180  : 35;

  int32_t ir = base_ir
    + (int32_t)(amp_ir * (p1 + p2))
    + noise(s, noise_amp);
  int32_t red = base_red
    + (int32_t)(amp_red * (p1 + p2))
    + noise(s, noise_amp);

  fifo_push(s, clamp18(red), clamp18(ir));
}

void chip_init(void) {
  state_t *s =
    (state_t *)calloc(1, sizeof(state_t));
  s->rng = 0xC001CAFE;
  s->attr_bpm = attr_init("bpm", 75);
  s->attr_finger = attr_init("finger", 1);
  s->regs[REG_PARTID] = 0x15;
  s->regs[REG_REVID] = 0x00;

  const i2c_config_t i2c_cfg = {
    .address = I2C_ADDR,
    .scl = pin_init("SCL", INPUT_PULLUP),
    .sda = pin_init("SDA", INPUT_PULLUP),
    .connect = on_connect,
    .write = on_write,
    .read = on_read,
    .disconnect = NULL,
    .user_data = s,
  };
  s->i2c = i2c_init(&i2c_cfg);

  const timer_config_t timer_cfg = {
    .callback = on_timer,
    .user_data = s,
```

```
  };
  s->timer = timer_init(&timer_cfg);
  reset_device(s);
}
```

Listing 4: max3010x.chip.c

3.4. Librerías utilizadas

```
# Wokwi Library List
# See https://docs.wokwi.com/guides/libraries

SparkFun MAX3010x Pulse and Proximity Sensor Library
Adafruit SSD1306
Adafruit GFX Library
```

Listing 5: libraries.txt – Parte B

Se utilizan tres librerías:

- **SparkFun MAX3010x Pulse and Proximity Sensor Library:** Proporciona funciones de alto nivel para comunicarse con el sensor MAX30102 por I2C y detectar latidos.
- **Adafruit SSD1306:** Driver para la pantalla OLED SSD1306.
- **Adafruit GFX Library:** Librería gráfica base para dibujar texto, formas y otros elementos en la pantalla.

3.5. Enlace a la simulación

La simulación completa de la Parte B está disponible en Wokwi:

`https://wokwi.com/projects/456308558605301761`

3.6. Código fuente

El código del *sketch* de la Parte B implementa un pulsómetro completo con las siguientes funcionalidades:

- **Detección de dedo:** Umbral de entrada a 50000 en la señal IR y umbral de salida a 42000 (histéresis para evitar falsos positivos).
- **Detección de latidos:** Doble mecanismo: se utiliza primero la función `checkForBeat()` de la librería SparkFun; si tras 3.5 segundos no se detectan latidos, se activa un algoritmo de *fallback* basado en umbrales sobre la línea base.
- **Promedio móvil:** Se mantiene un buffer circular de las últimas 4 mediciones de BPM y se calcula su media para suavizar las lecturas.
- **LED RGB indicador:**
  - **Apagado:** Sin dedo.
  - **Azul:** Dedo detectado, midiendo (sin datos aún).
  - **Verde:** BPM < 70 (ritmo bajo/normal).
  - **Amarillo (R+G):** BPM entre 70 y 110 (ritmo normal).
  - **Rojo:** BPM ≥ 110 (ritmo elevado).
  - **Blanco (flash):** Destello de 35 ms en cada latido detectado.
- **Pantalla OLED:** Muestra “Falta dedo”, “Midiendo...” o el valor de BPM actual según el estado.

```
#include <Wire.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>
#include "MAX30105.h"
#include "heartRate.h"

static const int PIN_I2C_SDA = 5;
static const int PIN_I2C_SCL = 6;
static const int PIN_LED_R = 7;
static const int PIN_LED_G = 1;
static const int PIN_LED_B = 0;

static const long FINGER_ENTER_THRESHOLD = 50000;
static const long FINGER_EXIT_THRESHOLD  = 42000;

static const unsigned long UI_REFRESH_MS = 200;
static const unsigned long FLASH_MS = 35;
static const unsigned long
    BEAT_MIN_INTERVAL_MS = 280;
static const unsigned long
    BEAT_MAX_INTERVAL_MS = 2000;
static const unsigned long
    FALLBACK_ACTIVATE_MS = 3500;
static const long FALLBACK_HIGH_OFFSET = 4500;
static const long FALLBACK_LOW_OFFSET  = 2200;

static const uint8_t RATE_SIZE = 4;

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64

Adafruit_SSD1306 display(SCREEN_WIDTH,
    SCREEN_HEIGHT, &Wire, -1);
MAX30105 particleSensor;

float beatsPerMinute = 0.0f;
int beatAvg = 0;
uint16_t rates[RATE_SIZE];
uint8_t rateSpot = 0;
uint8_t rateCount = 0;

bool fingerPresent = false;
bool flashActive = false;
unsigned long flashUntilMs = 0;
unsigned long lastUiMs = 0;
unsigned long fingerStartMs = 0;
unsigned long lastBeatEdgeMs = 0;
unsigned long lastBeatAnyMs = 0;
unsigned long lastFallbackPeakMs = 0;
bool fallbackAbove = false;
long irBaseline = 0;

void setRgb(bool r, bool g, bool b) {
  digitalWrite(PIN_LED_R, r ? HIGH : LOW);
  digitalWrite(PIN_LED_G, g ? HIGH : LOW);
  digitalWrite(PIN_LED_B, b ? HIGH : LOW);
}

void setBaseColorFromBpm(int bpm) {
  if (bpm < 70)
    setRgb(false, true, false);  // Verde
  else if (bpm < 110)
    setRgb(true, true, false);   // Amarillo
  else
    setRgb(true, false, false);  // Rojo
}

void flashWhite() {
  flashActive = true;
  flashUntilMs = millis() + FLASH_MS;
  setRgb(true, true, true);
}

void resetMeasurementState() {
  beatsPerMinute = 0.0f;
  beatAvg = 0;
```

```
  rateSpot = 0;
  rateCount = 0;
  for (uint8_t i = 0; i < RATE_SIZE; i++)
    rates[i] = 0;
  lastBeatEdgeMs = 0;
  lastBeatAnyMs = 0;
  lastFallbackPeakMs = 0;
  fallbackAbove = false;
  irBaseline = 0;
}

bool isValidBeatInterval(unsigned long dtMs) {
  return dtMs >= BEAT_MIN_INTERVAL_MS
      && dtMs <= BEAT_MAX_INTERVAL_MS;
}

void registerBeat(float bpm,
                  unsigned long nowMs) {
  if (bpm < 30.0f || bpm > 220.0f) return;
  beatsPerMinute = bpm;
  rates[rateSpot] = (uint16_t)(bpm + 0.5f);
  rateSpot = (rateSpot + 1) % RATE_SIZE;
  if (rateCount < RATE_SIZE) rateCount++;

  uint32_t sum = 0;
  for (uint8_t i = 0; i < rateCount; i++)
    sum += rates[i];
  beatAvg = (rateCount == 0) ? 0
          : (int)(sum / rateCount);

  lastBeatAnyMs = nowMs;
  setBaseColorFromBpm(
    (beatAvg > 0) ? beatAvg
    : (int)(beatsPerMinute + 0.5f));
  flashWhite();
}

void drawUi() {
  display.clearDisplay();
  display.setTextColor(SSD1306_WHITE);
  display.setTextSize(1);
  display.setCursor(0, 0);
  display.print("Pulsometro MAX3010x");
  display.drawLine(0, 10, 127, 10,
                   SSD1306_WHITE);

  if (!fingerPresent) {
    display.setTextSize(2);
    display.setCursor(0, 24);
    display.print("Falta dedo");
  } else if (beatAvg <= 0
         && beatsPerMinute <= 0.0f) {
    display.setTextSize(2);
    display.setCursor(0, 22);
    display.print("Midiendo...");
  } else {
    display.setTextSize(2);
    display.setCursor(0, 18);
    display.print("BPM:");
    display.setTextSize(3);
    display.setCursor(0, 38);
    display.print((beatAvg > 0) ? beatAvg
      : (int)(beatsPerMinute + 0.5f));
  }
  display.display();
}

void haltWithError(const char *message,
                   bool canDraw) {
  Serial.println(message);
  if (canDraw) {
    display.clearDisplay();
    display.setTextSize(1);
    display.setTextColor(SSD1306_WHITE);
    display.setCursor(0, 0);
    display.print(message);
```

```
    display.display();
  }
  while (true) {
    setRgb(true, false, false);
    delay(120);
    setRgb(false, false, false);
    delay(120);
  }
}

void setup() {
  Serial.begin(115200);
  pinMode(PIN_LED_R, OUTPUT);
  pinMode(PIN_LED_G, OUTPUT);
  pinMode(PIN_LED_B, OUTPUT);
  setRgb(false, false, false);

  Wire.begin(PIN_I2C_SDA, PIN_I2C_SCL);
  Wire.setClock(400000);

  bool displayReady =
    display.begin(SSD1306_SWITCHCAPVCC, 0x3C);
  if (!displayReady)
    haltWithError("ERROR OLED", false);

  display.clearDisplay();
  display.setTextSize(1);
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(0, 0);
  display.print("Iniciando...");
  display.display();

  if (!particleSensor.begin(Wire,
                            I2C_SPEED_FAST))
    haltWithError("ERROR MAX3010x", true);

  byte ledBrightness = 0x1F;
  byte sampleAverage = 4;
  byte ledMode = 2;
  int sampleRate = 100;
  int pulseWidth = 411;
  int adcRange = 4096;
  particleSensor.setup(ledBrightness,
    sampleAverage, ledMode, sampleRate,
    pulseWidth, adcRange);
  particleSensor.setPulseAmplitudeRed(0x1F);
  particleSensor.setPulseAmplitudeIR(0x1F);
  particleSensor.setPulseAmplitudeGreen(0x00);

  resetMeasurementState();
  fingerPresent = false;
  fingerStartMs = 0;
  lastUiMs = millis();
  drawUi();
}

void loop() {
  if (particleSensor.safeCheck(50)) {
    long irValue = particleSensor.getIR();
    unsigned long nowMs = millis();

    if (!fingerPresent
        && irValue >= FINGER_ENTER_THRESHOLD) {
      fingerPresent = true;
      fingerStartMs = nowMs;
      resetMeasurementState();
      irBaseline = irValue;
      setRgb(false, false, true);
    } else if (fingerPresent
        && irValue <= FINGER_EXIT_THRESHOLD) {
      fingerPresent = false;
      fingerStartMs = 0;
      resetMeasurementState();
      setRgb(false, false, false);
    }

```

```
    if (fingerPresent) {
      if (irBaseline == 0)
        irBaseline = irValue;
      else
        irBaseline =
          (irBaseline * 15 + irValue) / 16;

      bool beatCaptured = false;

      if (checkForBeat(irValue)) {
        if (lastBeatEdgeMs != 0) {
          unsigned long dtMs =
            nowMs - lastBeatEdgeMs;
          if (isValidBeatInterval(dtMs)) {
            float bpm =
              60000.0f / (float)dtMs;
            registerBeat(bpm, nowMs);
            beatCaptured = true;
          }
        }
        lastBeatEdgeMs = nowMs;
      }

      unsigned long referenceMs =
        (lastBeatAnyMs == 0) ? fingerStartMs
                             : lastBeatAnyMs;
      if (!beatCaptured && referenceMs != 0
          && (nowMs - referenceMs)
             >= FALLBACK_ACTIVATE_MS) {
        long highTh =
          irBaseline + FALLBACK_HIGH_OFFSET;
        long lowTh =
          irBaseline + FALLBACK_LOW_OFFSET;

        if (!fallbackAbove
            && irValue > highTh) {
          fallbackAbove = true;
          if (lastFallbackPeakMs != 0) {
            unsigned long dtMs =
              nowMs - lastFallbackPeakMs;
            if (isValidBeatInterval(dtMs)) {
              float bpm =
                60000.0f / (float)dtMs;
              registerBeat(bpm, nowMs);
            }
          }
          lastFallbackPeakMs = nowMs;
        }
        if (fallbackAbove
            && irValue < lowTh)
          fallbackAbove = false;
      }

      if (beatAvg <= 0
          && beatsPerMinute <= 0.0f
          && !flashActive)
        setRgb(false, false, true);
    }
  }

  if (flashActive
      && millis() >= flashUntilMs) {
    flashActive = false;
    if (!fingerPresent)
      setRgb(false, false, false);
    else if (beatAvg > 0
          || beatsPerMinute > 0.0f)
      setBaseColorFromBpm(
        (beatAvg > 0) ? beatAvg
        : (int)(beatsPerMinute + 0.5f));
    else
      setRgb(false, false, true);
  }

  if (millis() - lastUiMs >= UI_REFRESH_MS) {
    lastUiMs = millis();
```

```
    drawUi();
  }
}
```

Listing 6: sketch.ino – Parte B

3.7. Diagrama del circuito

El diagrama JSON del circuito de la Parte B es más complejo al incluir la *breadboard*, la ESP32-C3 SuperMini, el chip personalizado MAX3010x, la pantalla OLED y el LED RGB:

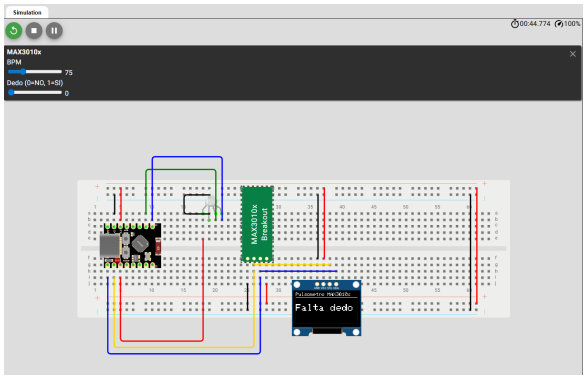
```
{
  "version": 1,
  "author": "Uri Shaked",
  "editor": "wokwi",
  "parts": [
    { "type": "wokwi-breadboard", "id": "bb1",
      "top": -444.6, "left": -371.6,
      "attrs": { "color": "#eeefed" } },
    { "type":
        "board-aitewinrobot-esp32c3-supermini",
      "id": "esp",
      "top": -391.77, "left": -326.14,
      "rotate": -90, "attrs": {} },
    { "type": "wokwi-rgb-led",
      "id": "rgb1",
      "top": -437.6, "left": -190.9,
      "attrs": { "common": "cathode" } },
    { "type": "board-ssd1306",
      "id": "oled",
      "top": -294.46, "left": -47.77,
      "attrs": { "i2cAddress": "0x3c" } },
    { "type": "chip-max3010x",
      "id": "chip1",
      "top": -401.1, "left": -157.28,
      "rotate": -90, "attrs": {} }
  ],
  "connections": [
    ["esp:TX","$serialMonitor:RX","",[]],
    ["esp:RX","$serialMonitor:TX","",[]],
    ["bb1:tp.3","bb1:5t.b","red",["v0"]],
    ["bb1:tn.2","bb1:4t.a","black",["v0"]],
    ["bb1:tn.15","bb1:19t.a","black",
      ["v-0.1","h-39.2","v28.8"]],
    ["bb1:20t.b","bb1:9t.a","green",
      ["v-76.8","h-105.6"]],
    ["bb1:10t.b","bb1:21t.b","blue",
      ["v-96","h105.6"]],
    ["bb1:tn.49","bb1:bn.49","black",["v0"]],
    ["bb1:tp.50","bb1:bp.50","red",["v0"]],
    ["bb1:25b.j","bb1:bn.20","black",["v0"]],
    ["bb1:28b.j","bb1:bp.22","red",["v0"]],
    ["bb1:26b.h","bb1:4b.i","gold",
      ["v115.2","h-211.2"]],
    ["bb1:3b.i","bb1:27b.i","blue",
      ["v115.2","h230.4"]],
    ["bb1:5b.i","bb1:18t.e","red",
      ["v96","h105.6"]],
    ["esp:0","bb1:10t.c","",["$bb"]],
    ["esp:1","bb1:9t.c","",["$bb"]],
    ["esp:2","bb1:8t.c","",["$bb"]],
    ["esp:3","bb1:7t.c","",["$bb"]],
    ["esp:4","bb1:6t.c","",["$bb"]],
    ["esp:5","bb1:3b.g","",["$bb"]],
    ["esp:6","bb1:4b.g","",["$bb"]],
    ["esp:7","bb1:5b.g","",["$bb"]],
    ["esp:8","bb1:6b.g","",["$bb"]],
    ["esp:9","bb1:7b.g","",["$bb"]],
    ["esp:10","bb1:8b.g","",["$bb"]],
    ["esp:5V","bb1:3t.c","",["$bb"]],
    ["esp:GND","bb1:4t.c","",["$bb"]],
    ["esp:3V3","bb1:5t.c","",["$bb"]],
    ["esp:RX","bb1:9b.g","",["$bb"]],
    ["esp:TX","bb1:10b.g","",["$bb"]],
```

```
    ["rgb1:R","bb1:18t.a","",["$bb"]],
    ["rgb1:COM","bb1:19t.b","",["$bb"]],
    ["rgb1:G","bb1:20t.a","",["$bb"]],
    ["rgb1:B","bb1:21t.a","",["$bb"]],
    ["chip1:GND","bb1:25b.f","",["$bb"]],
    ["chip1:SCL","bb1:26b.f","",["$bb"]],
    ["chip1:SDA","bb1:27b.f","",["$bb"]],
    ["chip1:VIN","bb1:28b.f","",["$bb"]],
    ["bb1:tn.29","bb1:36b.f","black",["v0"]],
    ["bb1:tp.30","bb1:37b.f","red",["v0"]],
    ["bb1:26b.g","bb1:38b.g","gold",["v0"]],
    ["bb1:27b.h","bb1:39b.h","blue",["v0"]],
    ["oled:GND","bb1:36b.j","",["$bb"]],
    ["oled:VCC","bb1:37b.j","",["$bb"]],
    ["oled:SCL","bb1:38b.j","",["$bb"]],
    ["oled:SDA","bb1:39b.j","",["$bb"]]
  ],
  "dependencies": {}
}
```

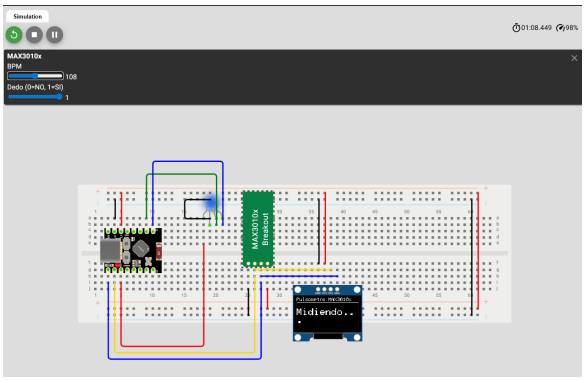
Listing 7: diagram.json – Parte B

3.8. Capturas de la simulación

A continuación se presentan las capturas de los diferentes estados del sistema simulado en Wokwi:



(a) Sin dedo: la pantalla muestra “Falta dedo” y el LED está apagado.



(b) Midiendo: dedo detectado, el sistema está procesando. LED azul.

Figura 2: Estados iniciales del pulsómetro simulado.

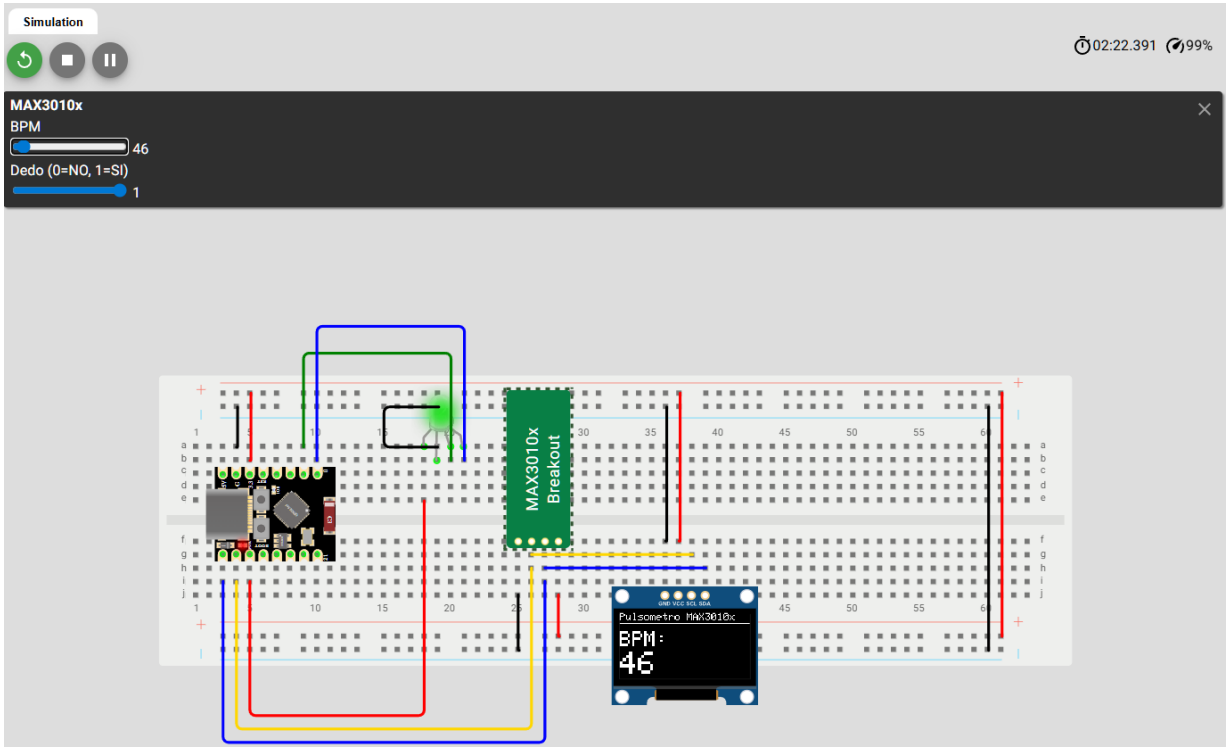


Figura 3: 46 BPM – LED verde (ritmo bajo).

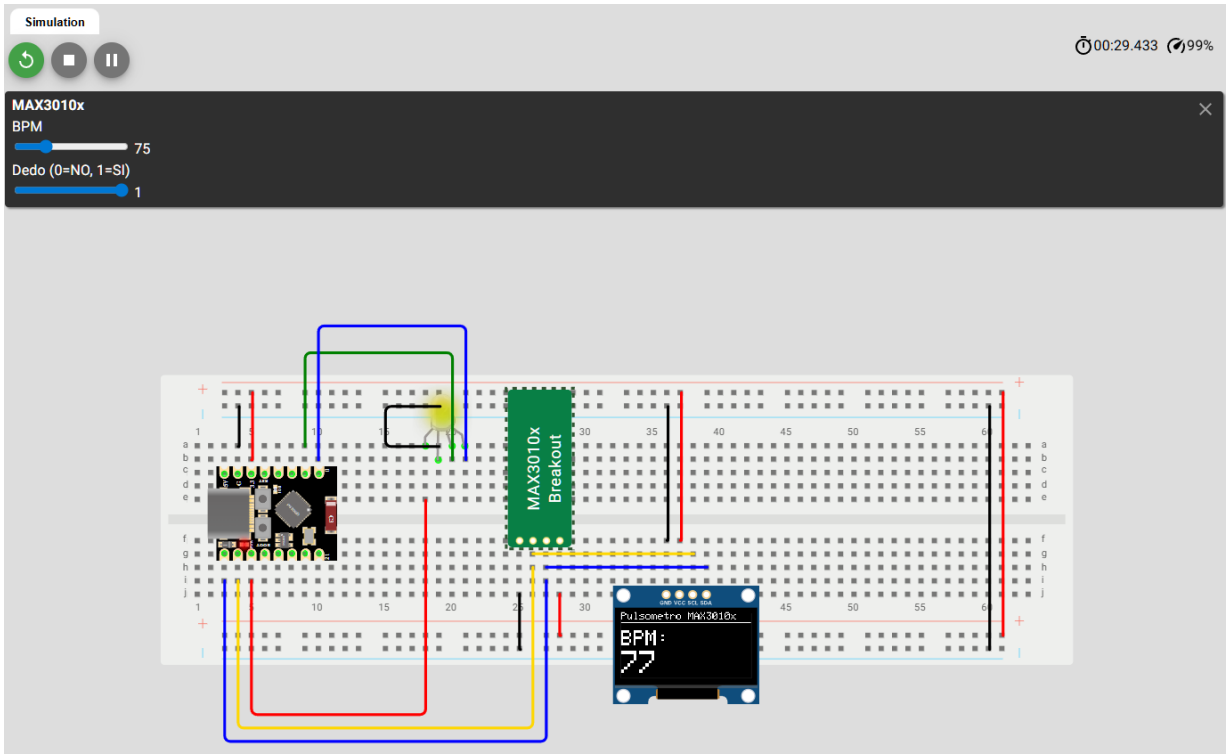


Figura 4: 75 BPM – LED amarillo (ritmo normal).

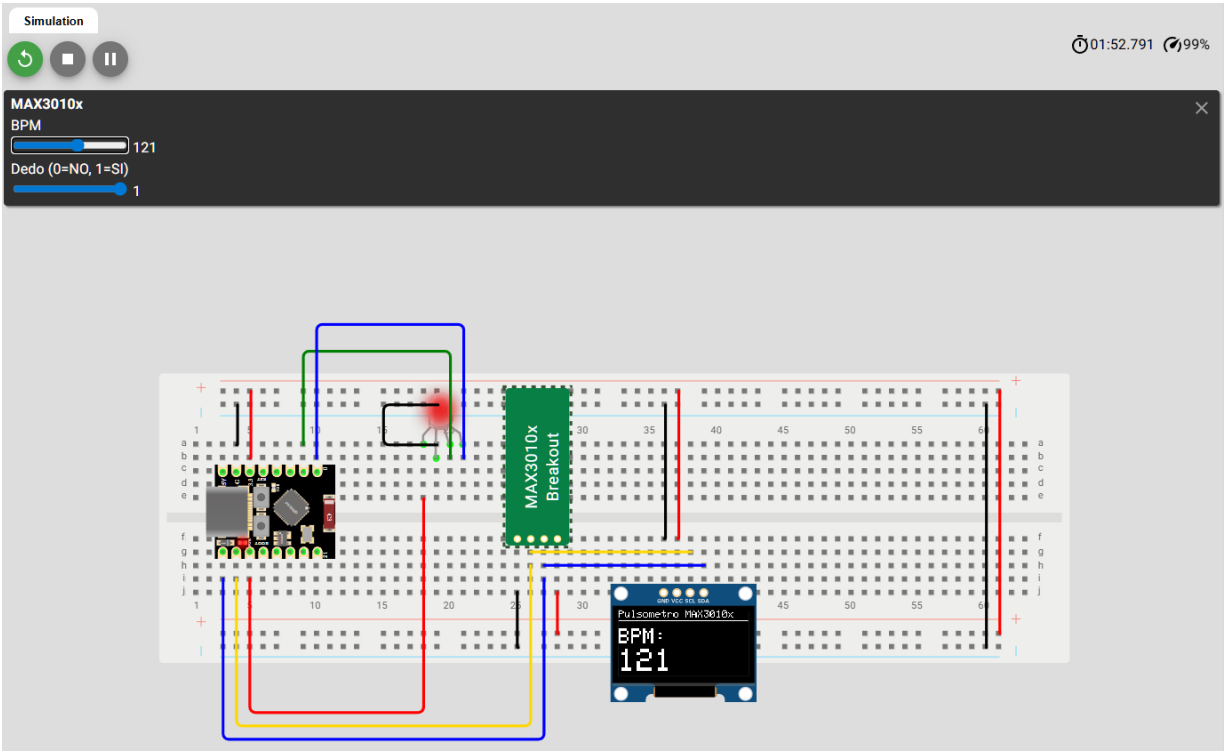


Figura 5: 121 BPM – LED rojo (ritmo elevado).

3.9. Comportamiento de convergencia del BPM

Un aspecto importante del sistema es que la pantalla no muestra inmediatamente el valor de BPM configurado en el sensor, sino que va convergiendo hacia él de forma progresiva. Esto se debe al diseño del algoritmo basado en un **promedio móvil** de las últimas 4 mediciones: cada nuevo latido detectado aporta una muestra al buffer circular, y el BPM mostrado es la media de las muestras disponibles.

Este comportamiento es deliberado y tiene dos razones fundamentales:

1. **Realismo:** Un sensor de pulso real no proporciona un valor instantáneo y exacto. Las mediciones individuales varían debido al ruido, movimiento del dedo, presión aplicada, etc. El promedio móvil filtra estas fluctuaciones y proporciona un valor estable.

2. **Compatibilidad con hardware real:** Si se programara el sistema para mostrar directamente un valor fijo sin filtrado, el código no funcionaría correctamente con el sensor físico, donde las lecturas son inherentemente ruidosas. El algoritmo de convergencia progresiva es necesario para obtener mediciones fiables tanto en simulación como en el montaje real.

Además, el mecanismo de *fallback* (activado tras 3.5 segundos sin detección de latidos por el algoritmo principal) garantiza que el sistema siempre acabe encontrando el ritmo incluso cuando la señal no es ideal, ajustándose dinámicamente a la línea base de la señal IR.

4. Parte práctica (montaje físico)

[Pendiente: incluir fotografías del montaje físico en el laboratorio una vez realizadas.]

5. Conclusiones

En esta práctica se ha logrado familiarizarse con el entorno de simulación Wokwi y con la plataforma ESP32-C3 para el desarrollo de sistemas embebidos. Se extraen las siguientes conclusiones:

- **Wokwi como herramienta de prototipado:** El simulador permite validar tanto el hardware (conexiones, componentes) como el software antes de pasar al montaje físico, reduciendo significativamente el tiempo de desarrollo en laboratorio.
- **Chips personalizados:** La capacidad de crear chips personalizados en Wokwi resulta especialmente útil cuando se trabaja con sensores no disponibles en el catálogo del simulador. El chip MAX3010x desarrollado reproduce fielmente el comportamiento del sensor real, incluyendo la comunicación I2C completa y la generación de señales PPG realistas.
- **Estrategia de simulación previa:** La decisión de simular el circuito completo antes de las sesiones de laboratorio demostró ser acertada, ya que permitió disponer de un diagrama de referencia limpio y un código ya validado, optimizando el uso del tiempo limitado de las prácticas presenciales.
- **Diseño de algoritmos robustos:** La implementación de un promedio móvil y mecanismos de *fallback* para la detección de latidos garantiza que el código sea funcional tanto en simulación como en hardware real, donde las condiciones de la señal son menos ideales.
- **Comunicación I2C:** Se ha comprendido el funcionamiento del bus I2C para conectar múltiples dispositivos (pantalla OLED y sensor de pulso) en el mismo bus, diferenciándolos por sus direcciones (0x3C y 0x57 respectivamente).