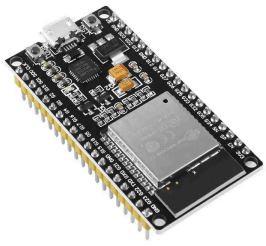
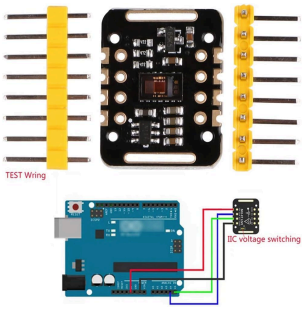
PRACTICA 5 : Buses de comunicación I (introducción y I2c)

MATERIAL

Para esta practica necesitaremos el microcontrolador ESP32, un Display oled I2C SSD1306 OLED, un Sensor Max3012 medidor de oxigeno en sangre y frecuencia cardiaca, un escáner I2C cables y una protoboard.





OBJETIVO Y FUNCIONALIDAD DE LA PRACTICA

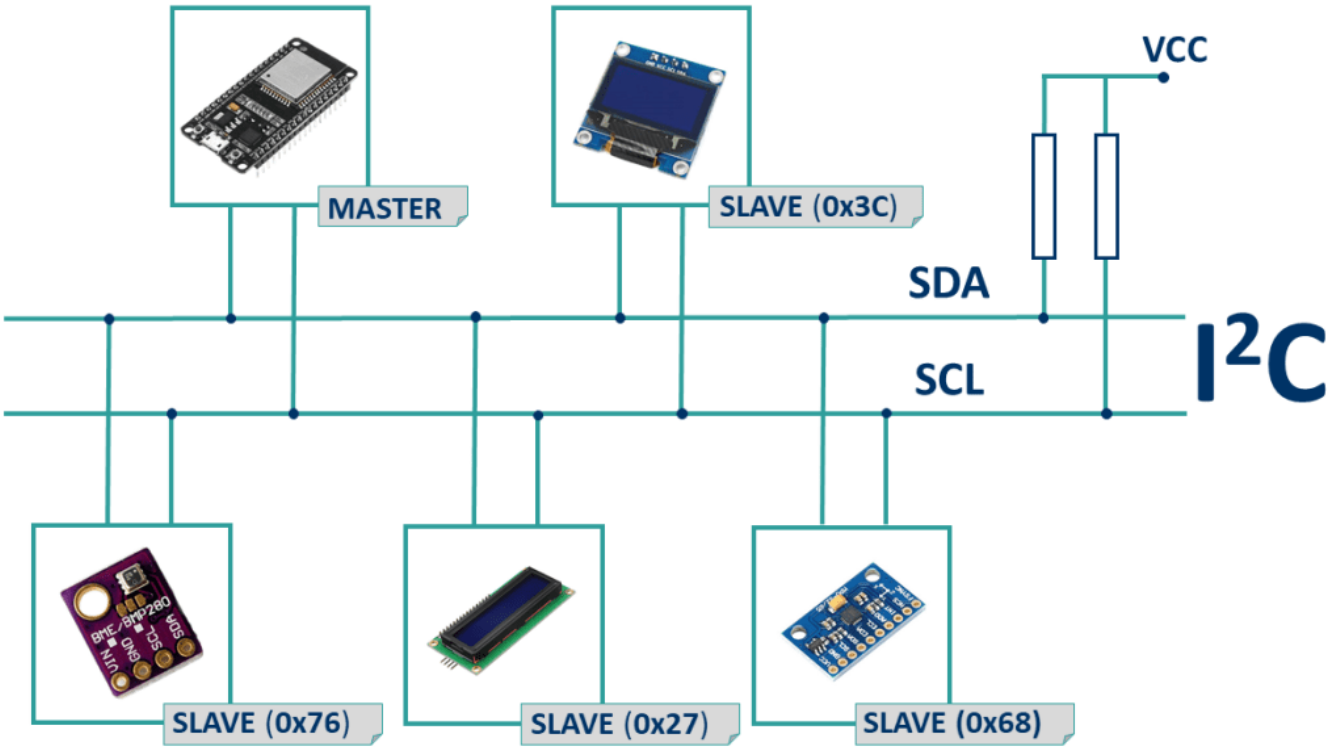
El objetivo de la practica es comprender el funcionamiento de los buses
sistemas de comunicación entreperiféricos; estos elementos pueden ser internos o externos al procesador.

Esta es la primera practica de 4 donde se verán los buses i2c, spi, i2s , usart.

Para lo cual realizaremos una practica en cada caso donde controlaremos un periférico de ejemplo

Ejercicio Practico 1 ESCÁNER I2C

Programar el siguiente codigo, colocar varios dispositivos I2C como en la figura siguiente



Código comentado

```ino
#include <Arduino.h>  // Librería base de Arduino
#include <Wire.h>     // Librería para comunicación I2C

void setup() {
  Wire.begin();       // Inicia la comunicación I2C
  Serial.begin(115200);  // Inicia la comunicación serie a 115200 baudios

// Si se usa en un Arduino Leonardo, espera a que el monitor serie esté listo
  while (!Serial);
  Serial.println("\nI2C Scanner");  // Mensaje inicial
}

void loop() {
  byte error, address;  // Variables para almacenar errores y direcciones I2C
  int nDevices = 0;     // Contador de dispositivos encontrados

  Serial.println("Scanning...");  // Mensaje indicando el inicio del escaneo

  // Bucle para recorrer todas las direcciones I2C posibles (1 a 127)
  for(address = 1; address < 127; address++) {
    // Inicia comunicación con la dirección actual
    Wire.beginTransmission(address);
    // Finaliza la transmisión y obtiene el estado
    error = Wire.endTransmission();

    // Si no hay error, se encontró un dispositivo en esta dirección
    if (error == 0) {
      Serial.print("I2C device found at address 0x");
      // Agregar un 0 si la dirección es menor a 0x10 para formato correcto
      if (address < 16) Serial.print("0");
      // Imprimir dirección en formato hexadecimal
      Serial.print(address, HEX);
      Serial.println("  !");

      // Incrementar el contador de dispositivos encontrados
      nDevices++;
    }
    else if (error == 4) {  // Error desconocido
      Serial.print("Unknown error at address 0x");
```

```
      if (address < 16) Serial.print("0");
      Serial.println(address, HEX);
    }
  }

  // Mensaje final según el número de dispositivos encontrados
  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");

  delay(5000);  // Esperar 5 segundos antes de volver a escanear
}
```

Foto montaje

https://drive.google.com/file/d/1vs1VN9QIzH5K3ueYBCw7uf7ni1jp9cpB/view?usp=share_link

https://drive.google.com/file/d/1GJvTIxlFmwjap0G-EwwYgzl7vHiS0zgy/view?usp=share_link

Descibir la salida por el puerto serie

Si no hay dispositivos o no lo encuentra:

> I2C Scanner
> Scanning...
> No I2C devices found
> done

Si ha encontrado el dispositivo:

> I2C Scanner
> Scanning...
> I2C device found at address 0x3C !
> done

Si ha encontrado más de un dispositivo:

> I2C Scanner
> Scanning...
> I2C device found at address 0x68 !

```
I2C device found at address 0x3C !
done
```

Si ha tenido un error desconocido en alguna dirección:

```
I2C Scanner
Scanning...
I2C device found at address 0x3C !
Unknown error at address 0x50
done
```

- El Monitor Serie muestra qué dispositivos están conectados al bus I2C y en qué direcciones.
- Si hay errores, indica problemas de conexión.
- El escaneo se repite cada 5 segundos para verificar cambios en la conexión.

Explicar el funcionamiento

- Se inicializa el bus I2C y la comunicación serie.
- Se recorre cada dirección I2C del 1 al 127.
- El ESP32 intenta comunicarse con cada dirección.
- Si un dispositivo responde, se muestra en el Monitor Serie.
- Si hay un error, también se notifica.
- Cada 5 segundos, se vuelve a escanear.

Ejercicio Practico 2

A realizar como ejercicio en casa

1. Realizar un programa que utilice un dispositivo i2c como por ejemplo alguno de los siguientes

---

Display oled I2C SSD1306 OLED (display de leds organicos)

referencia https://programarfacil.com/blog/arduino-blog/ssd1306-pantalla-oled-con-arduino/

o este otro formato

descargar la libreria https://github.com/lexus2k/ssd1306 medienate platformio

Código:

```
/*************************************************************************
 This is an example for our Monochrome OLEDs based on SSD1306 drivers

 Pick one up today in the adafruit shop!
 ------> http://www.adafruit.com/category/63_98

 This example is for a 128x32 pixel display using I2C to communicate
 3 pins are required to interface (two I2C and one reset).

 Adafruit invests time and resources providing this open
 source code, please support Adafruit and open-source
 hardware by purchasing products from Adafruit!

 Written by Limor Fried/Ladyada for Adafruit Industries,
 with contributions from the open source community.
 BSD license, check license.txt for more information
 All text above, and the splash screen below must be
 included in any redistribution.
 *************************************************************************/
#include <Arduino.h>
#include <SPI.h>
#include <Wire.h>
#include <Adafruit_I2CDevice.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>


 void testdrawline(void);      // Draw many lines

 void testdrawrect(void);      // Draw rectangles (outlines)

 void testfillrect(void);      // Draw rectangles (filled)

  void testdrawcircle(void );    // Draw circles (outlines)

  void testfillcircle(void );    // Draw circles (filled)

  void testdrawroundrect(void ); // Draw rounded rectangles (outlines)
```

```
  void testfillroundrect(void); // Draw rounded rectangles (filled)

  void testdrawtriangle(void);  // Draw triangles (outlines)

  void testfilltriangle(void);  // Draw triangles (filled)

  void testdrawchar(void);      // Draw characters of the default font

  void testdrawstyles(void);    // Draw 'stylized' characters

  void testscrolltext(void);    // Draw scrolling text

  void testdrawbitmap(void);    // Draw a small bitmap image

  void testanimate(const uint8_t *bitmap, uint8_t w, uint8_t h);


#define SCREEN_WIDTH 128 // OLED display width, in pixels
#define SCREEN_HEIGHT 32 // OLED display height, in pixels

// Declaration for an SSD1306 display connected to I2C (SDA, SCL pins)
// The pins for I2C are defined by the Wire-library.
// On an arduino UNO:       A4(SDA), A5(SCL)
// On an arduino MEGA 2560: 20(SDA), 21(SCL)
// On an arduino LEONARDO:   2(SDA),  3(SCL), ...
#define OLED_RESET     -1 // Reset pin # (or -1 if sharing Arduino reset pin)
#define SCREEN_ADDRESS 0x3C ///< See datasheet for Address; 0x3D for 128x64, 0x3C for 128x
Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

#define NUMFLAKES     10 // Number of snowflakes in the animation example

#define LOGO_HEIGHT   16
#define LOGO_WIDTH    16
static const unsigned char PROGMEM logo_bmp[] =
{ B00000000, B11000000,
  B00000001, B11000000,
  B00000001, B11000000,
  B00000011, B11100000,
  B11110011, B11100000,
  B11111110, B11111000,
```

```
  B01111110, B11111111,
  B00110011, B10011111,
  B00011111, B11111100,
  B00001101, B01110000,
  B00011011, B10100000,
  B00111111, B11100000,
  B00111111, B11110000,
  B01111100, B11110000,
  B01110000, B01110000,
  B00000000, B00110000 };

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

  // SSD1306_SWITCHCAPVCC = generate display voltage from 3.3V internally
  if(!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {
    Serial.println(F("SSD1306 allocation failed"));
    for(;;); // Don't proceed, loop forever
  }

  // Show initial display buffer contents on the screen --
  // the library initializes this with an Adafruit splash screen.
  display.display();
  delay(2000); // Pause for 2 seconds

  // Clear the buffer
  display.clearDisplay();

  // Draw a single pixel in white
  display.drawPixel(10, 10, SSD1306_WHITE);

  // Show the display buffer on the screen. You MUST call display() after
  // drawing commands to make them visible on screen!
  display.display();
  delay(2000);
  // display.display() is NOT necessary after every single drawing command,
  // unless that's what you want...rather, you can batch up a bunch of
  // drawing operations and then update the screen all at once by calling
  // display.display(). These examples demonstrate both approaches...
```

```
  testdrawline();      // Draw many lines

  testdrawrect();      // Draw rectangles (outlines)

  testfillrect();      // Draw rectangles (filled)

  testdrawcircle();    // Draw circles (outlines)

  testfillcircle();    // Draw circles (filled)

  testdrawroundrect(); // Draw rounded rectangles (outlines)

  testfillroundrect(); // Draw rounded rectangles (filled)

  testdrawtriangle();  // Draw triangles (outlines)

  testfilltriangle();  // Draw triangles (filled)

  testdrawchar();      // Draw characters of the default font

  testdrawstyles();    // Draw 'stylized' characters

  testscrolltext();    // Draw scrolling text

  testdrawbitmap();    // Draw a small bitmap image

  // Invert and restore display, pausing in-between
  display.invertDisplay(true);
  delay(1000);
  display.invertDisplay(false);
  delay(1000);

  testanimate(logo_bmp, LOGO_WIDTH, LOGO_HEIGHT); // Animate bitmaps
}

void loop() {
}

void testdrawline() {
  int16_t i;
```

```
display.clearDisplay(); // Clear display buffer

for(i=0; i<display.width(); i+=4) {
  display.drawLine(0, 0, i, display.height()-1, SSD1306_WHITE);
  display.display(); // Update screen with each newly-drawn line
  delay(1);
}
for(i=0; i<display.height(); i+=4) {
  display.drawLine(0, 0, display.width()-1, i, SSD1306_WHITE);
  display.display();
  delay(1);
}
delay(250);

display.clearDisplay();

for(i=0; i<display.width(); i+=4) {
  display.drawLine(0, display.height()-1, i, 0, SSD1306_WHITE);
  display.display();
  delay(1);
}
for(i=display.height()-1; i>=0; i-=4) {
  display.drawLine(0, display.height()-1, display.width()-1, i, SSD1306_WHITE);
  display.display();
  delay(1);
}
delay(250);

display.clearDisplay();

for(i=display.width()-1; i>=0; i-=4) {
  display.drawLine(display.width()-1, display.height()-1, i, 0, SSD1306_WHITE);
  display.display();
  delay(1);
}
for(i=display.height()-1; i>=0; i-=4) {
  display.drawLine(display.width()-1, display.height()-1, 0, i, SSD1306_WHITE);
  display.display();
  delay(1);
```

```
  }
  delay(250);

  display.clearDisplay();

  for(i=0; i<display.height(); i+=4) {
    display.drawLine(display.width()-1, 0, 0, i, SSD1306_WHITE);
    display.display();
    delay(1);
  }
  for(i=0; i<display.width(); i+=4) {
    display.drawLine(display.width()-1, 0, i, display.height()-1, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  delay(2000); // Pause for 2 seconds
}

void testdrawrect(void) {
  display.clearDisplay();

  for(int16_t i=0; i<display.height()/2; i+=2) {
    display.drawRect(i, i, display.width()-2*i, display.height()-2*i, SSD1306_WHITE);
    display.display(); // Update screen with each newly-drawn rectangle
    delay(1);
  }

  delay(2000);
}

void testfillrect(void) {
  display.clearDisplay();

  for(int16_t i=0; i<display.height()/2; i+=3) {
    // The INVERSE color is used so rectangles alternate white/black
    display.fillRect(i, i, display.width()-i*2, display.height()-i*2, SSD1306_INVERSE);
    display.display(); // Update screen with each newly-drawn rectangle
    delay(1);
  }
```

```
  delay(2000);
}

void testdrawcircle(void) {
  display.clearDisplay();

  for(int16_t i=0; i<max(display.width(),display.height())/2; i+=2) {
    display.drawCircle(display.width()/2, display.height()/2, i, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testfillcircle(void) {
  display.clearDisplay();

  for(int16_t i=max(display.width(),display.height())/2; i>0; i-=3) {
    // The INVERSE color is used so circles alternate white/black
    display.fillCircle(display.width() / 2, display.height() / 2, i, SSD1306_INVERSE);
    display.display(); // Update screen with each newly-drawn circle
    delay(1);
  }

  delay(2000);
}

void testdrawroundrect(void) {
  display.clearDisplay();

  for(int16_t i=0; i<display.height()/2-2; i+=2) {
    display.drawRoundRect(i, i, display.width()-2*i, display.height()-2*i,
      display.height()/4, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  delay(2000);
```

```
}

void testfillroundrect(void) {
  display.clearDisplay();

  for(int16_t i=0; i<display.height()/2-2; i+=2) {
    // The INVERSE color is used so round-rects alternate white/black
    display.fillRoundRect(i, i, display.width()-2*i, display.height()-2*i,
      display.height()/4, SSD1306_INVERSE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testdrawtriangle(void) {
  display.clearDisplay();

  for(int16_t i=0; i<max(display.width(),display.height())/2; i+=5) {
    display.drawTriangle(
      display.width()/2  , display.height()/2-i,
      display.width()/2-i, display.height()/2+i,
      display.width()/2+i, display.height()/2+i, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  delay(2000);
}

void testfilltriangle(void) {
  display.clearDisplay();

  for(int16_t i=max(display.width(),display.height())/2; i>0; i-=5) {
    // The INVERSE color is used so triangles alternate white/black
    display.fillTriangle(
      display.width()/2  , display.height()/2-i,
      display.width()/2-i, display.height()/2+i,
      display.width()/2+i, display.height()/2+i, SSD1306_INVERSE);
```

```
    display.display();
    delay(1);
  }

  delay(2000);
}

void testdrawchar(void) {
  display.clearDisplay();

  display.setTextSize(1);      // Normal 1:1 pixel scale
  display.setTextColor(SSD1306_WHITE); // Draw white text
  display.setCursor(0, 0);     // Start at top-left corner
  display.cp437(true);         // Use full 256 char 'Code Page 437' font

  // Not all the characters will fit on the display. This is normal.
  // Library will draw what it can and the rest will be clipped.
  for(int16_t i=0; i<256; i++) {
    if(i == '\n') display.write(' ');
    else          display.write(i);
  }

  display.display();
  delay(2000);
}

void testdrawstyles(void) {
  display.clearDisplay();

  display.setTextSize(1);             // Normal 1:1 pixel scale
  display.setTextColor(SSD1306_WHITE);        // Draw white text
  display.setCursor(0,0);             // Start at top-left corner
  display.println(F("Hello, world!"));

  display.setTextColor(SSD1306_BLACK, SSD1306_WHITE); // Draw 'inverse' text
  display.println(3.141592);

  display.setTextSize(2);             // Draw 2X-scale text
  display.setTextColor(SSD1306_WHITE);
  display.print(F("0x")); display.println(0xDEADBEEF, HEX);
```

```
  display.display();
  delay(2000);
}

void testscrolltext(void) {
  display.clearDisplay();

  display.setTextSize(2); // Draw 2X-scale text
  display.setTextColor(SSD1306_WHITE);
  display.setCursor(10, 0);
  display.println(F("scroll"));
  display.display();      // Show initial text
  delay(100);

  // Scroll in various directions, pausing in-between:
  display.startscrollright(0x00, 0x0F);
  delay(2000);
  display.stopscroll();
  delay(1000);
  display.startscrollleft(0x00, 0x0F);
  delay(2000);
  display.stopscroll();
  delay(1000);
  display.startscrolldiagright(0x00, 0x07);
  delay(2000);
  display.startscrolldiagleft(0x00, 0x07);
  delay(2000);
  display.stopscroll();
  delay(1000);
}

void testdrawbitmap(void) {
  display.clearDisplay();

  display.drawBitmap(
    (display.width()  - LOGO_WIDTH ) / 2,
    (display.height() - LOGO_HEIGHT) / 2,
    logo_bmp, LOGO_WIDTH, LOGO_HEIGHT, 1);
  display.display();
```

```
  delay(1000);
}

#define XPOS   0 // Indexes into the 'icons' array in function below
#define YPOS   1
#define DELTAY 2

void testanimate(const uint8_t *bitmap, uint8_t w, uint8_t h) {
  int8_t f, icons[NUMFLAKES][3];

  // Initialize 'snowflake' positions
  for(f=0; f< NUMFLAKES; f++) {
    icons[f][XPOS]   = random(1 - LOGO_WIDTH, display.width());
    icons[f][YPOS]   = -LOGO_HEIGHT;
    icons[f][DELTAY] = random(1, 6);
    Serial.print(F("x: "));
    Serial.print(icons[f][XPOS], DEC);
    Serial.print(F(" y: "));
    Serial.print(icons[f][YPOS], DEC);
    Serial.print(F(" dy: "));
    Serial.println(icons[f][DELTAY], DEC);
  }

  for(;;) { // Loop forever...
    display.clearDisplay(); // Clear the display buffer

    // Draw each snowflake:
    for(f=0; f< NUMFLAKES; f++) {
      display.drawBitmap(icons[f][XPOS], icons[f][YPOS], bitmap, w, h, SSD1306_WHITE);
    }

    display.display(); // Show the display buffer on the screen
    delay(200);        // Pause for 1/10 second

    // Then update coordinates of each flake...
    for(f=0; f< NUMFLAKES; f++) {
      icons[f][YPOS] += icons[f][DELTAY];
      // If snowflake is off the bottom of the screen...
      if (icons[f][YPOS] >= display.height()) {
        // Reinitialize to a random position, just off the top
```

```
        icons[f][XPOS]   = random(1 - LOGO_WIDTH, display.width());
        icons[f][YPOS]   = -LOGO_HEIGHT;
        icons[f][DELTAY] = random(1, 6);
      }
    }
  }
}
```

Funcionamento:

Este programa es una demo gráfica para pantallas OLED monocromáticas que utilizan el controlador SSD1306, conectadas por comunicación I2C.

1. Inicializa el display OLED conectado mediante I2C a la dirección 0x3C.
2. Muestra el logo de Adafruit durante 2 segundos.
3. Ejecuta una serie de efectos visuales uno tras otro:

   - Líneas animadas desde los bordes.
   - Rectángulos (simples y rellenos).
   - Círculos y círculos rellenos.
   - Rectángulos redondeados.
   - Triángulos.
   - Muestra texto en diferentes tamaños, estilos y colores.
   - Efecto de texto deslizante (scroll).
   - Dibuja un pequeño bitmap centrado (logo).

4. Realiza una animación tipo “copos de nieve” en bucle infinito usando el logo como partícula.

Salida monitor serie:

Si el display no se conecta correctamente aparece el siguiente mensaje por pantalla:

```
SSD1306 allocation failed
```

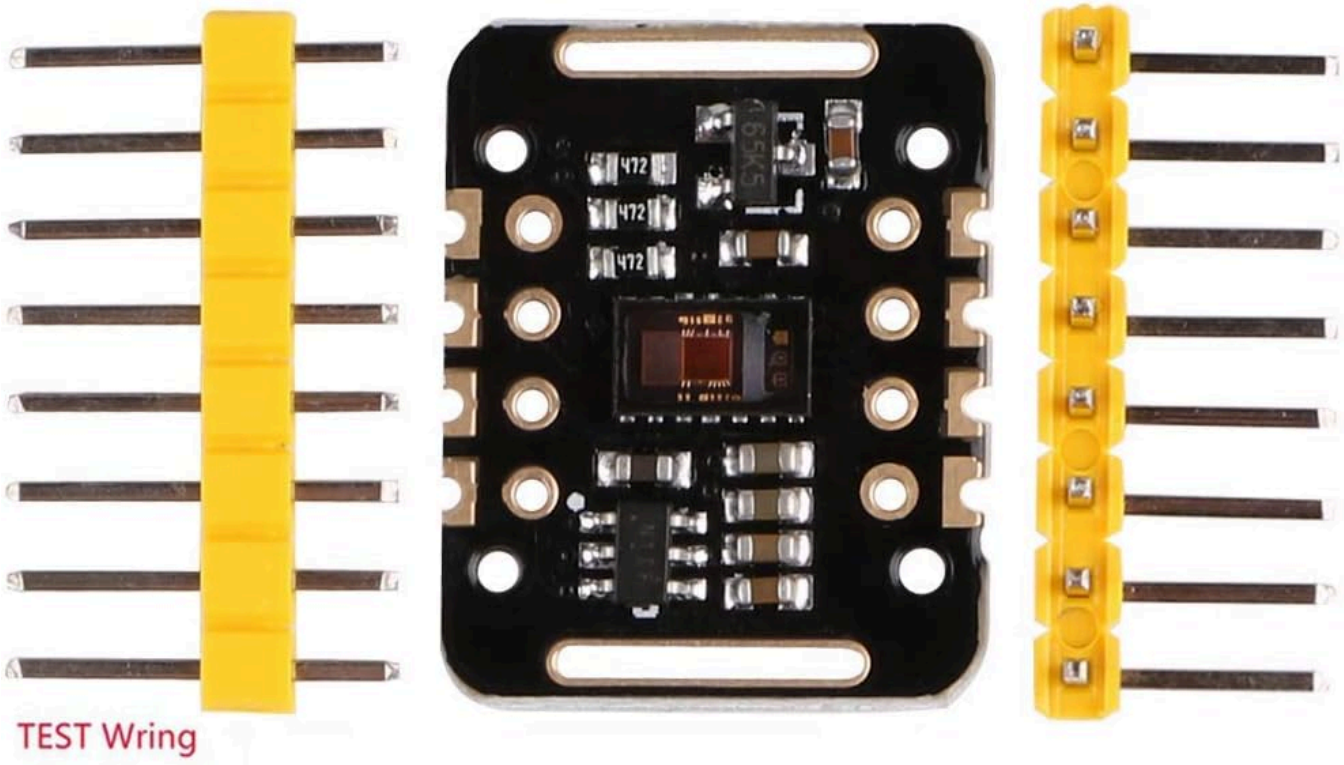
Durante la animación imprime información sobre cada “copo” que cae por la pantalla:

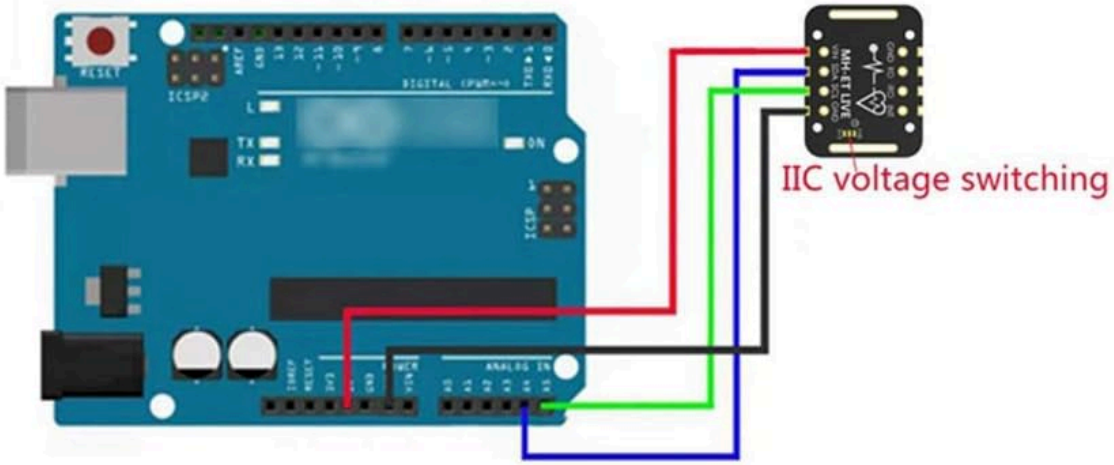
```
x: 23 y: -16 dy: 2
x: 56 y: -16 dy: 3
...
```

Video dimestración:

https://drive.google.com/file/d/1NKQJOf_JeiFLnGQJoev3SFA2sN5wBGqY/view?usp=share_link

Sensor Max3012 medidor de oxigeno en sangre y frecuencia cardiaca





referencia https://www.luisllamas.es/pulsimetro-y-oximetro-con-arduino-y-max30102/

Código:

```
/*
  MAX30105 Breakout: Output all the raw Red/IR/Green readings
  By: Nathan Seidle @ SparkFun Electronics
  Date: October 2nd, 2016
  https://github.com/sparkfun/MAX30105_Breakout

  Outputs all Red/IR/Green values.

  Hardware Connections (Breakoutboard to Arduino):
  -5V = 5V (3.3V is allowed)
  -GND = GND
  -SDA = A4 (or SDA)
  -SCL = A5 (or SCL)
  -INT = Not connected

  The MAX30105 Breakout can handle 5V or 3.3V I2C logic. We recommend powering the board w
  but it will also run at 3.3V.

  This code is released under the [MIT License](http://opensource.org/licenses/MIT).
*/
#include <Arduino.h>
#include <Wire.h>
#include "MAX30105.h"
#include <Adafruit_I2CDevice.h>
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#include "spo2_algorithm.h"

MAX30105 particleSensor;

#define MAX_BRIGHTNESS 255

#if defined(__AVR_ATmega328P__) || defined(__AVR_ATmega168__)
//Arduino Uno doesn't have enough SRAM to store 100 samples of IR led data and red led da
//To solve this problem, 16-bit MSB of the sampled data will be truncated. Samples become
uint16_t irBuffer[100]; //infrared LED sensor data
uint16_t redBuffer[100];  //red LED sensor data
#else
```

```
uint32_t irBuffer[100]; //infrared LED sensor data
uint32_t redBuffer[100];  //red LED sensor data
#endif

int32_t bufferLength; //data length
int32_t spo2; //SPO2 value
int8_t validSPO2; //indicator to show if the SPO2 calculation is valid
int32_t heartRate; //heart rate value
int8_t validHeartRate; //indicator to show if the heart rate calculation is valid

byte pulseLED = 11; //Must be on PWM pin
byte readLED = 13; //Blinks with each data read

void setup()
{
  Serial.begin(115200); // initialize serial communication at 115200 bits per second:

  pinMode(pulseLED, OUTPUT);
  pinMode(readLED, OUTPUT);

  // Initialize sensor
  if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) //Use default I2C port, 400kHz speed
  {
    Serial.println(F("MAX30105 was not found. Please check wiring/power."));
    while (1);
  }

  Serial.println(F("Attach sensor to finger with rubber band. Press any key to start conve
  while (Serial.available() == 0) ; //wait until user presses a key
  Serial.read();

  byte ledBrightness = 60; //Options: 0=Off to 255=50mA
  byte sampleAverage = 4; //Options: 1, 2, 4, 8, 16, 32
  byte ledMode = 2; //Options: 1 = Red only, 2 = Red + IR, 3 = Red + IR + Green
  byte sampleRate = 100; //Options: 50, 100, 200, 400, 800, 1000, 1600, 3200
  int pulseWidth = 411; //Options: 69, 118, 215, 411
  int adcRange = 4096; //Options: 2048, 4096, 8192, 16384

  particleSensor.setup(ledBrightness, sampleAverage, ledMode, sampleRate, pulseWidth, adcR
}
```

```
void loop()
{
  bufferLength = 100; //buffer length of 100 stores 4 seconds of samples running at 25sps

  //read the first 100 samples, and determine the signal range
  for (byte i = 0 ; i < bufferLength ; i++)
  {
    while (particleSensor.available() == false) //do we have new data?
      particleSensor.check(); //Check the sensor for new data

    redBuffer[i] = particleSensor.getRed();
    irBuffer[i] = particleSensor.getIR();
    particleSensor.nextSample(); //We're finished with this sample so move to next sample

    Serial.print(F("red="));
    Serial.print(redBuffer[i], DEC);
    Serial.print(F(", ir="));
    Serial.println(irBuffer[i], DEC);
  }

  //calculate heart rate and SpO2 after first 100 samples (first 4 seconds of samples)
  maxim_heart_rate_and_oxygen_saturation(irBuffer, bufferLength, redBuffer, &spo2, &validS

  //Continuously taking samples from MAX30102.  Heart rate and SpO2 are calculated every 1
  while (1)
  {
    //dumping the first 25 sets of samples in the memory and shift the last 75 sets of samp
    for (byte i = 25; i < 100; i++)
    {
      redBuffer[i - 25] = redBuffer[i];
      irBuffer[i - 25] = irBuffer[i];
    }

    //take 25 sets of samples before calculating the heart rate.
    for (byte i = 75; i < 100; i++)
    {
      while (particleSensor.available() == false) //do we have new data?
        particleSensor.check(); //Check the sensor for new data
```

```
      digitalWrite(readLED, !digitalRead(readLED)); //Blink onboard LED with every data re

      redBuffer[i] = particleSensor.getRed();
      irBuffer[i] = particleSensor.getIR();
      particleSensor.nextSample(); //We're finished with this sample so move to next sample

      //send samples and calculation result to terminal program through UART
      Serial.print(F("red="));
      Serial.print(redBuffer[i], DEC);
      Serial.print(F(", ir="));
      Serial.print(irBuffer[i], DEC);

      Serial.print(F(", HR="));
      Serial.print(heartRate, DEC);

      Serial.print(F(", HRvalid="));
      Serial.print(validHeartRate, DEC);

      Serial.print(F(", SPO2="));
      Serial.print(spo2, DEC);

      Serial.print(F(", SPO2Valid="));
      Serial.println(validSPO2, DEC);
    }

    //After gathering 25 new samples recalculate HR and SP02
    maxim_heart_rate_and_oxygen_saturation(irBuffer, bufferLength, redBuffer, &spo2, &vali
  }
}
```

Funcionamento:

Este programa utiliza el sensor MAX30105 para medir la frecuencia cardíaca (HR) y la saturación de oxígeno en sangre (SpO_2) a través de un dedo del usuario. La información se procesa y se muestra por el Monitor Serie.

1. Inicialización

    - Inicia la comunicación I2C con el sensor MAX30105.
    - Si no lo detecta, muestra un error y detiene el programa.
    - Solicita al usuario que presione una tecla para empezar la medición.
2. Configuración del sensor (brillo, modo, velocidad de muestreo, ancho de pulso, rango)
3. Adquisición de datos
    - El sensor toma 100 muestras de luz roja e infrarroja reflejada por el dedo.
    - Estos datos se almacenan en redBuffer[] e irBuffer[].
4. Cálculo inicial de HR y SpO_2

- Se analiza la señal usando el algoritmo maxim_heart_rate_and_oxygen_saturation(), que estima:
    - HR (latidos por minuto)
    - SpO_2 (porcentaje de saturación de oxígeno)
    - Y determina si los valores calculados son válidos.

5. Bucle de medición continua
    - En cada ciclo:
    - Se conservan las últimas 75 muestras.
    - Se toman 25 nuevas muestras (100 en total otra vez).
    - Se recalcula HR y SpO_2.

Salida monitor serie:

Cuando la lectura es correcta aparece el siguiente mensaje:

```
red=53240, ir=65432, HR=75, HRvalid=1, SPO2=97, SPO2Valid=1
```

Esta línea aparece una vez por muestra (25 veces por bucle) y representa una lectura del pulso óptico y los valores biométricos procesados.

Cuando la lectura es incorrecta o no has puesto el dedo en el sensor:

```
red=12000, ir=11000, HR=0, HRvalid=0, SPO2=0, SPO2Valid=0
```

Video demostració:

https://drive.google.com/file/d/1v5Q1t-eiQ0XvFpqBY-A2n-BcqG9aTyB-/view?usp=share_link

Ejercicio de subida de nota (muy valorado)

- Parte 1.- Realizar utilizando el display y el sensor anterior un dispositivo que muestre en display la frecuencia cardiaca y el contenido de oxigeno .

```
#include <Arduino.h>
#include <Wire.h>
#include "MAX30105.h"
#include <LiquidCrystal_I2C.h>
#include "spo2_algorithm.h"


#define SDA_PIN 8
#define SCL_PIN 9
#define LCD_ADDRESS 0x27  // Cambia a 0x3F si fuera necesario
#define LCD_COLUMNS 16
#define LCD_ROWS 2

MAX30105 particleSensor;
LiquidCrystal_I2C lcd(LCD_ADDRESS, LCD_COLUMNS, LCD_ROWS);

#if defined(__AVR_ATmega328P__) || defined(__AVR_ATmega168__)
uint16_t irBuffer[100];
uint16_t redBuffer[100];
#else
uint32_t irBuffer[100];
uint32_t redBuffer[100];
#endif

int32_t bufferLength;
int32_t spo2;
int8_t validSPO2;
int32_t heartRate;
int8_t validHeartRate;

byte pulseLED = 11;
byte readLED = 13;

void setup() {
  Serial.begin(115200);
  pinMode(pulseLED, OUTPUT);
  pinMode(readLED, OUTPUT);

  // Inicializar I2C con pines personalizados
```

```
  Wire.begin(SDA_PIN, SCL_PIN);

  // Inicializar LCD
  lcd.init();
  lcd.backlight();
  lcd.setCursor(0, 0);
  lcd.print("Iniciando LCD...");

  // Inicializar sensor
  if (!particleSensor.begin(Wire, I2C_SPEED_FAST)) {
    lcd.clear();
    lcd.setCursor(0, 0);
    lcd.print("Sensor MAX30105");
    lcd.setCursor(0, 1);
    lcd.print("no detectado!");
    while (1);
  }

  lcd.clear();
  lcd.setCursor(0, 0);
  lcd.print("Coloca el dedo");

  byte ledBrightness = 60;
  byte sampleAverage = 4;
  byte ledMode = 2;
  byte sampleRate = 100;
  int pulseWidth = 411;
  int adcRange = 4096;

  particleSensor.setup(ledBrightness, sampleAverage, ledMode, sampleRate, pulseWidth, adcR

  // Esperar que el usuario pulse una tecla en Serial
  Serial.println("Presiona una tecla para empezar");
  while (Serial.available() == 0);
  Serial.read();
}

void loop() {
  bufferLength = 100;
```

```
  for (byte i = 0; i < bufferLength; i++) {
    while (!particleSensor.available())
      particleSensor.check();

    redBuffer[i] = particleSensor.getRed();
    irBuffer[i] = particleSensor.getIR();
    particleSensor.nextSample();
  }

  maxim_heart_rate_and_oxygen_saturation(irBuffer, bufferLength, redBuffer,
                                         &spo2, &validSPO2, &heartRate, &validHeartRate);

  while (1) {
    for (byte i = 25; i < 100; i++) {
      redBuffer[i - 25] = redBuffer[i];
      irBuffer[i - 25] = irBuffer[i];
    }

    for (byte i = 75; i < 100; i++) {
      while (!particleSensor.available())
        particleSensor.check();

      digitalWrite(readLED, !digitalRead(readLED));
      redBuffer[i] = particleSensor.getRed();
      irBuffer[i] = particleSensor.getIR();
      particleSensor.nextSample();

      // Mostrar en la pantalla LCD
      lcd.setCursor(0, 0);
      lcd.print("HR:");
      if (validHeartRate)
        lcd.print(heartRate);
      else
        lcd.print("--");

      lcd.print(" bpm");

      lcd.setCursor(0, 1);
      lcd.print("SpO2:");
      if (validSPO2)
```

```
        lcd.print(spo2);
      else
        lcd.print("--");

      lcd.print(" %  ");

      // También puedes mantener Serial para debug si lo deseas
      Serial.print("HR=");
      Serial.print(heartRate);
      Serial.print(", SPO2=");
      Serial.print(spo2);
      Serial.println();
    }

    maxim_heart_rate_and_oxygen_saturation(irBuffer, bufferLength,
                                           redBuffer, &spo2, &validSPO2, &heartRate, &vali
  }
}
```

librerias necesarias:

```
lib_deps =
        marcoschwartz/LiquidCrystal_I2C@^1.1.4
        devxplained/MAX3010x Sensor Library@^1.0.5
        adafruit/Adafruit GFX Library@^1.12.0
        sparkfun/SparkFun MAX3010x Pulse and Proximity Sensor Library@^1.1.2
```

Funcionamiento:

1. Inicialización:

- Configura el sensor MAX30105 y la pantalla LCD.
- Espera que el usuario presione una tecla por monitor serie para empezar.

2. Lectura de datos:

- El sensor recoge 100 muestras de luz roja e infrarroja.
- Se calcula el HR (latidos por minuto) y SpO_2 (% oxígeno).
- Se actualizan continuamente cada segundo.

3. Salida en LCD:

- Línea 1: HR: 75 bpm (por ejemplo)
- Línea 2: SpO2: 98 %

4. LED (pin 13): Parpadea con cada nueva lectura del sensor.

Salida monitor serie:

El monitor serie muestra lo siguiente con cada nueva muestra:

```
HR=76, SPO2=98
```

Si el sensor no puede calcular correctamente, se muestra:

```
LCD: HR: -- bpm, SpO2: -- %
Serial: HR=0, SPO2=0
```

Video demostració:

https://drive.google.com/file/d/1POCz8aZEXP_RDvUqVMpF1vBY-mNZoNQL/view?usp=share_link

- Parte 2.- Generar una pagina web donde se pueda consultar dicha frecuencia y el contenido de oxigeno

Este apartado no he podido hacerlo, ya que mi router no permite conectar dispositivos como el ESP32-S3.