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

El objetivo de la practica es comprender el funcionamiento de los buses sistemas de comunicación entre periféricos; 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.

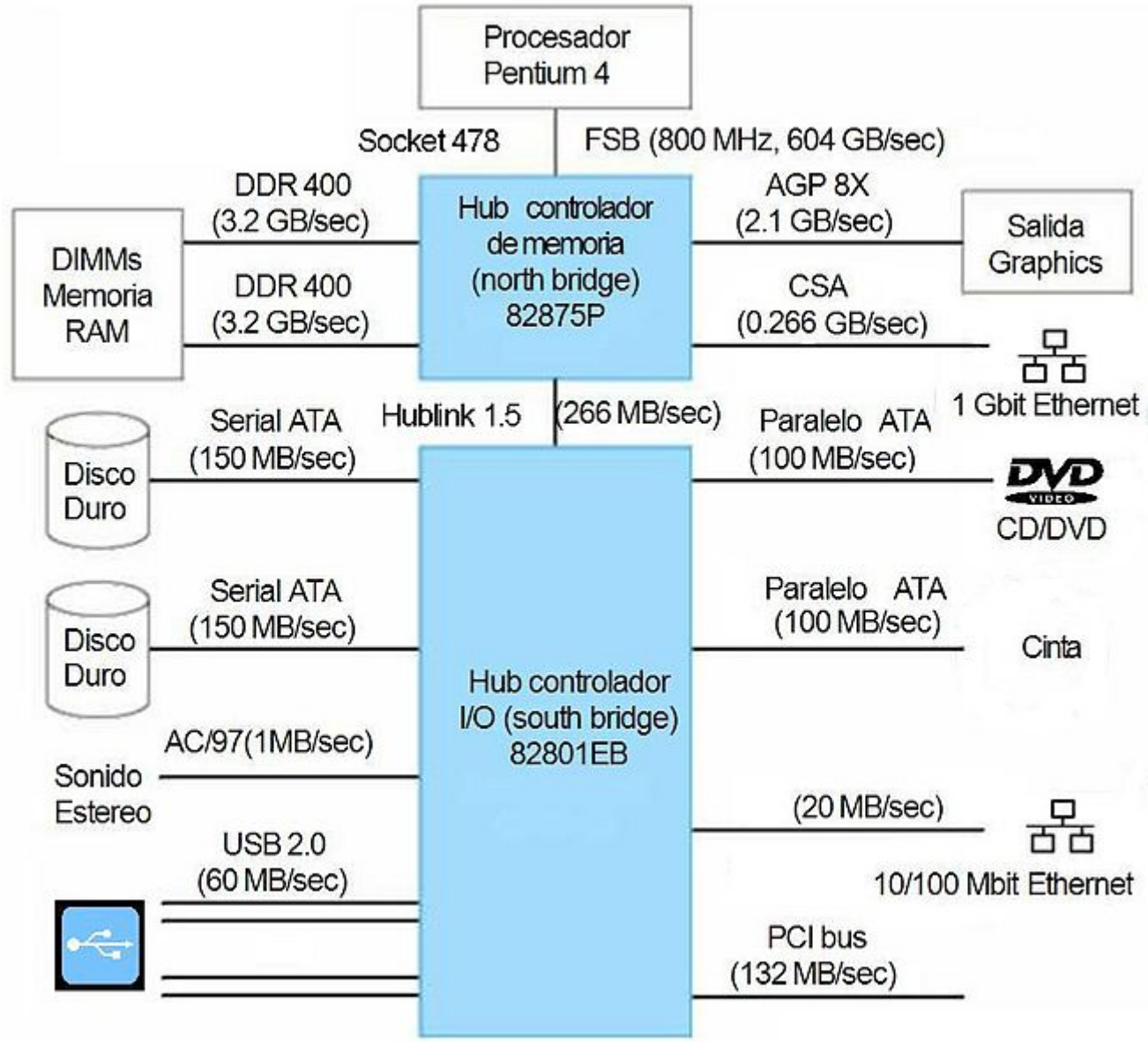
Para lo cual realizaremos una practica en cada caso donde controlaremos un periférico de ejemplo

Introducción teórica

La función del bus es permitir la conexión lógica entre los diferentes subsistemas que componen el computador. En su mayoría los buses están formados por conductores metálicos por los cuales se trasmiten señales eléctricas que son enviadas y recibidas con la ayuda de circuitos integrados que manejan un protocolo que les permite trasmitir datos útiles. Además de los datos el bus trasmite otras señales digitales como son las direcciones y señales de control.

Los buses definen su capacidad de acuerdo a la frecuencia máxima de envío y al ancho de los datos. Por lo general estos valores son inversamente proporcionales: si se tiene una alta frecuencia, el ancho de datos debe ser pequeño. Esto se debe a que la interferencia entre las señales (crosstalk) y la dificultad de sincronizarlas, crecen con la frecuencia, de manera que un bus con pocas señales es menos susceptible a esos problemas y puede funcionar a alta velocidad.

Todos los buses de computador tienen funciones especiales como las interrupciones y las DMA que permiten que un dispositivo periférico acceda a una CPU o a la memoria usando el mínimo de recursos.



Tipos de buses

Existen dos tipos que están clasificados por el método de envío de la información: **bus paralelo o bus serie**.

Hay diferencias en el rendimiento y hasta hace unos años se consideraba que el uso apropiado dependía de la longitud física de la conexión: para cortas distancias el bus paralelo, para largas el serial.

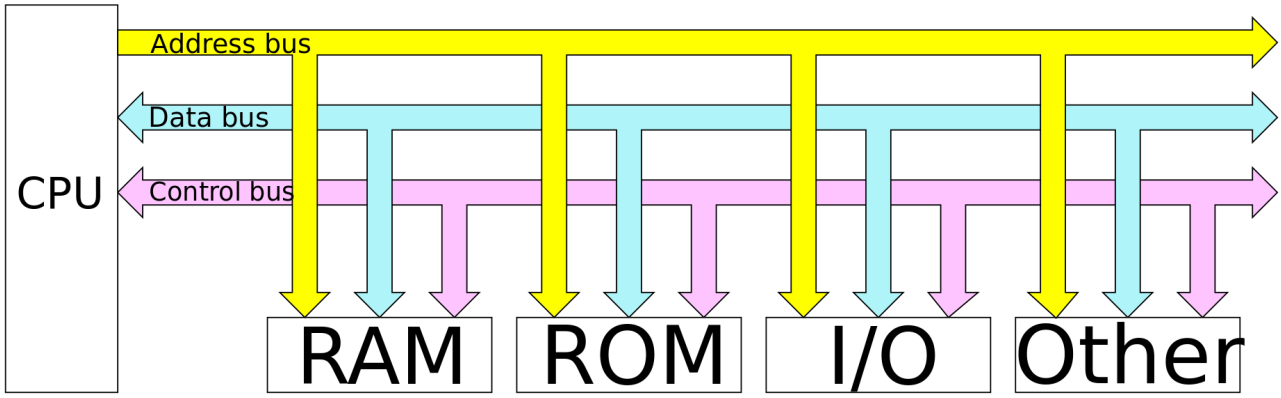
Bus paralelo

Es un bus en el cual los datos son enviados por bytes al mismo tiempo, con la ayuda de varias líneas que tienen funciones fijas. La cantidad de datos enviada es bastante grande con una frecuencia moderada y es igual al ancho de los datos por la frecuencia de funcionamiento. En los computadores ha sido usado de manera intensiva, desde el bus del

procesador, los buses de discos duros, tarjetas de expansión y de vídeo, hasta las impresoras.

El front-side bus de los procesadores Intel es un bus de este tipo y como cualquier bus presenta unas funciones en líneas dedicadas:

Las líneas de dirección son las encargadas de indicar la posición de memoria o el dispositivo con el que se desea establecer comunicación. Las líneas de control son las encargadas de enviar señales de arbitraje entre los dispositivos. Entre las más importantes están las líneas de interrupción, DMA y los indicadores de estado. Las líneas de datos transmiten los bits de forma aleatoria de manera que por lo general un bus tiene un ancho que es potencia de 2. Un bus paralelo tiene conexiones físicas complejas, pero la lógica es sencilla, que lo hace útil en sistemas con poco poder de cómputo. En los primeros microcomputadores, el bus era simplemente la extensión del bus del procesador y los demás integrados "escuchan" las línea de direcciones, en espera de recibir instrucciones. En el PC IBM original, el diseño del bus fue determinante a la hora de elegir un procesador con I/O de 8 bits (Intel 8088), sobre uno de 16 (el 8086), porque era posible usar hardware diseñado para otros procesadores, abaratando el producto.



Bus serie

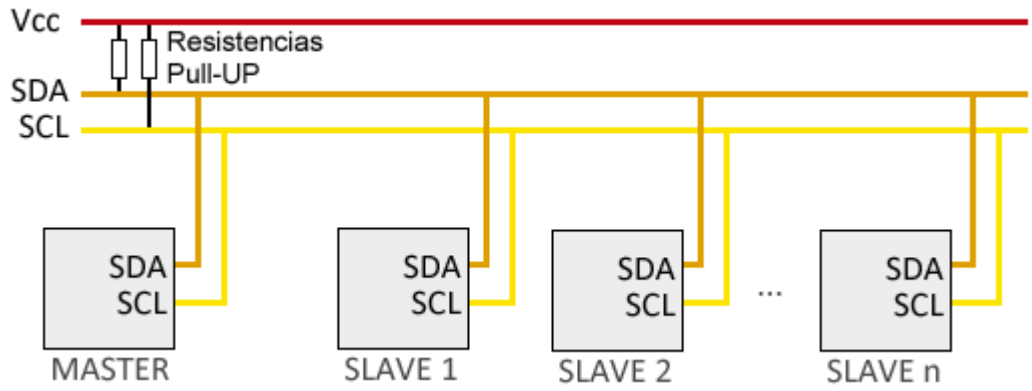
En este los datos son enviados, bit a bit y se reconstruyen por medio de registros o rutinas. Está formado por pocos conductores y su ancho de banda depende de la frecuencia. Aunque originalmente fueron usados para conectar dispositivos lentos (como el teclado o un ratón), actualmente se están usando para conectar dispositivos mucho más rápidos como discos duros, unidades de estado sólido, tarjetas de expansión e incluso para el bus del procesador.

Ejemplo de buses serie son USB,SATA(Serial ATA), RS232, SPI, I2C , I2S , nosotros en estas practicas no entraremos en el USB o el SATA ; pero si en todos los demás

BUS I2C

El estándar I2C (Inter-Integrated Circuit) fue desarrollado por Philips en 1982 para la comunicación interna de dispositivos electrónicos en sus artículos. Posteriormente fue adoptado progresivamente por otros fabricantes hasta convertirse en un estándar del mercado.

El bus I2C requiere únicamente dos cables para su funcionamiento, uno para la señal de reloj (CLK) y otro para el envío de datos (SDA), lo cual es una ventaja frente al bus SPI. Por contra, su funcionamiento es un poco más complejo, así como la electrónica necesaria para implementarla.



En el bus Cada dispositivo dispone de una dirección, que se emplea para acceder a los dispositivo de forma individual. Esta dirección puede ser fijada por hardware (en cuyo caso, frecuentemente, se pueden modificar los últimos 3 bits mediante jumpers o interruptores) o totalmente por software.

En general, cada dispositivo conectado al bus debe tener una dirección única. Si tenemos varios dispositivos similares tendremos que cambiar la dirección o, en caso de no ser posible, implementar un bus secundario.

El bus I2C tiene una arquitectura de tipo maestro-esclavo. El dispositivo maestro inicia la comunicación con los esclavos, y puede mandar o recibir datos de los esclavos. Los esclavos no pueden iniciar la comunicación (el maestro tiene que preguntarles), ni hablar entre si directamente.

Es posible disponer de más de un maestro, pero solo uno puede ser el maestro cada vez. El cambio de maestro supone una alta complejidad, por lo que no es algo frecuente.

El bus I2C es síncrono. El maestro proporciona una señal de reloj, que mantiene sincronizados a todos los dispositivos del bus. De esta forma, se elimina la necesidad de que cada dispositivo tenga su propio reloj, de tener que acordar una velocidad de transmisión y mecanismos para mantener la transmisión sincronizada (como en UART)

El protocolo I2C prevé resistencias de Pull-UP de las líneas a Vcc. En Arduino veréis que frecuentemente no se instalan estas resistencias, ya que la librería Wire activa las

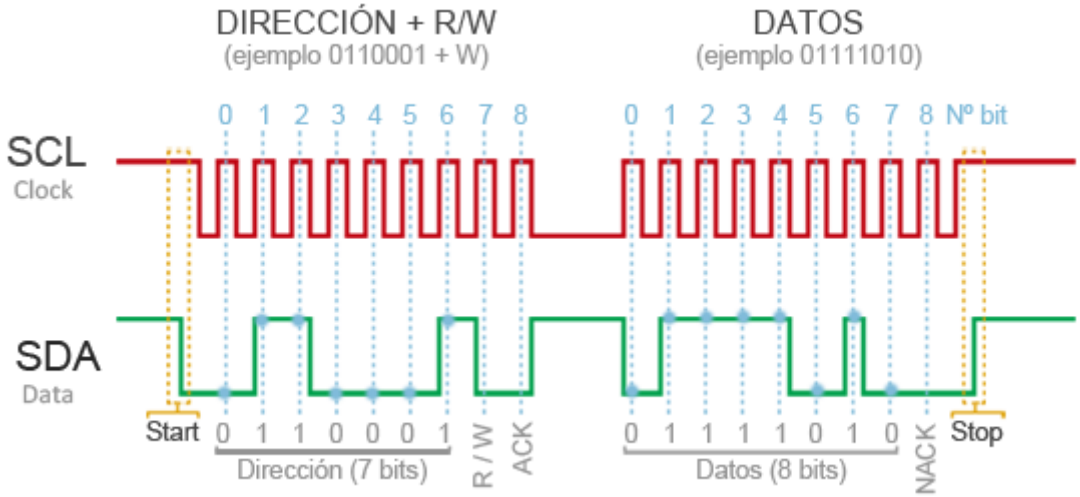
resistencias internas de Pull-UP. Sin embargo las resistencias internas tienen un valor de entre 20-30kOhmnios, por lo que son unas resistencias de Pull-UP muy blandas.

Usar unas resistencias blandas implica que los flancos de subida de la señal serán menos rápidas, lo que implica que podremos usar velocidades menores y distancias de comunicación inferiores. Si queremos emplear velocidades o distancias de transmisión superiores, deberemos poner físicamente resistencias de Pull-UP de entre 1k a 4K7.

Funcionamiento del bus I2C

Para poder realizar la comunicación con solo un cable de datos, el bus I2C emplea una trama (el formato de los datos enviados) amplia. La comunicación costa de:

- 7 bits a la dirección del dispositivo esclavo con el que queremos comunicar.
- Un bit restante indica si queremos enviar o recibir información.(R/W)
- Un bit de validación
- Uno o más bytes son los datos enviados o recibidos del esclavo.
- Un bit de validación



Con estos 7 bits de dirección es posible acceder a 112 dispositivos en un mismo bus (16 direcciones de las 128 direcciones posibles se reservan para usos especiales)

Este incremento de los datos enviados (18bits por cada 8bits de datos) supone que, en general, la velocidad del bus I2C es reducida. La velocidad estándar de transmisión es de 100khz, con un modo de alta velocidad de 400khz.

```
El estándar I2C define otros modos de funcionamiento, como un envío de
dirección de 8,10 y 12bits, o velocidades de transmisión de 1Mbit/s, 3.
4Mbit/s y 5Mbit/s. No suelen ser empleados en Arduino.
```

Ventajas y desventajas

Ventajas

- Requiere pocos cables
- Dispone de mecanismos para verificar que la señal hay llegado

Desventajas

- Su velocidad es media-baja
- No es full duplex (es half duplex) No se puede enviar y recibir simultáneamente
- No hay verificación de que el contenido del mensaje es correcto (solo de la llegada del mensaje)

referencia:

https://randomnerdtutorials.com/esp32-i2c-communication-arduino-ide/

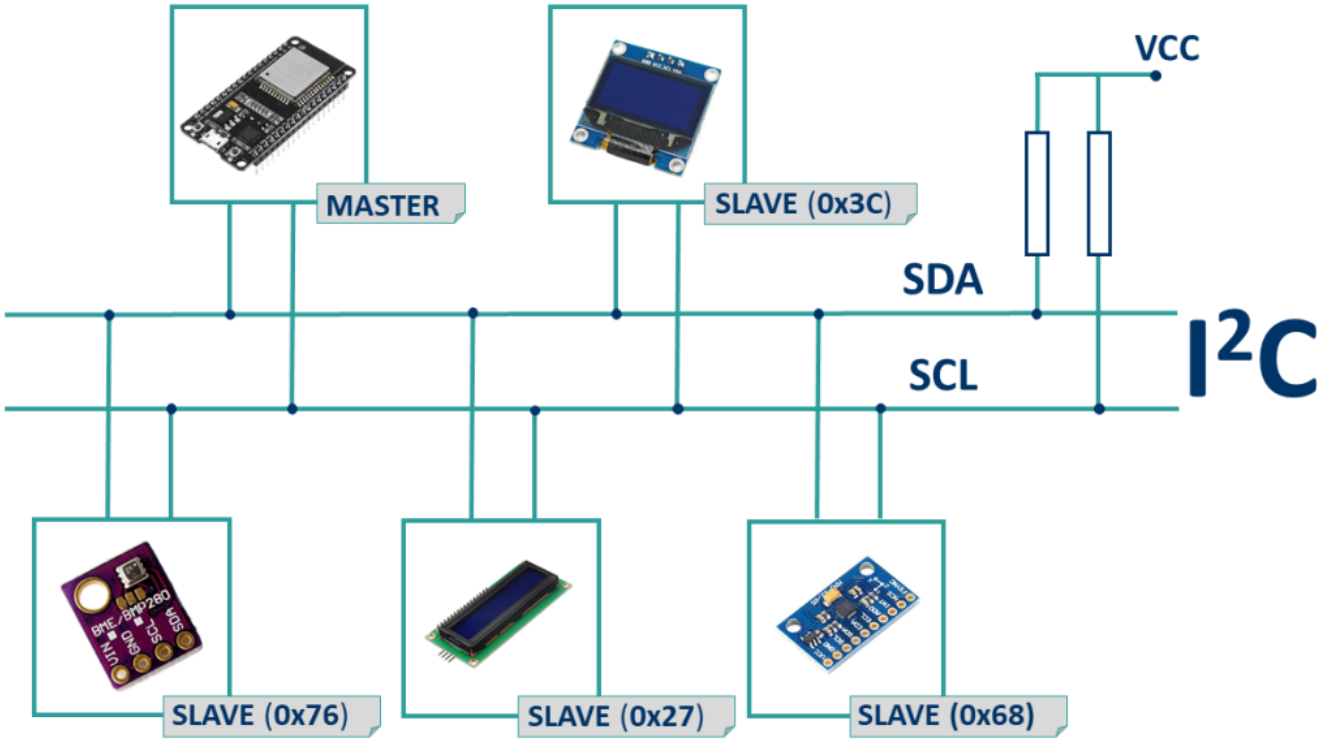
Ejercicio Practico 1 ESCÁNER I2C

Programar el siguiente codigo , colocar varios dispositivos I2C

en el caso de ESP32

I2C Device	ESP32
SDA	SDA (default is GPIO 21)
SCL	SCL (default is GPIO 22)
GND	GND
VCC	usually 3.3V or 5V

como en la figura siguiente



Las resistencias de pull-up dependen del numero de dispositivos ; aunque un valor usual es 3k3

```
#include <Arduino.h>

#include <Wire.h>


void setup()
{
  Wire.begin();

  Serial.begin(115200);
  while (!Serial);             // Leonardo: wait for serial monitor
  Serial.println("\nI2C Scanner");
}


void loop()
{
  byte error, address;
  int nDevices;

  Serial.println("Scanning...");

  nDevices = 0;
  for(address = 1; address < 127; address++ )
  {
    // The i2c_scanner uses the return value of
    // the Write.endTransmisstion to see if
    // a device did acknowledge to the address.
    Wire.beginTransmission(address);
    error = Wire.endTransmission();

    if (error == 0)
    {
      Serial.print("I2C device found at address 0x");
```

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

      nDevices++;
    }
    else if (error==4)
    {
      Serial.print("Unknown error at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.println(address,HEX);
    }
  }
  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");

  delay(5000);           // wait 5 seconds for next scan
}
```

1. Descibir la salida por el puerto serie
2. Explicar el funcionamiento

Ejercicio Practico 2

A realizar como ejercicio en casa

1. Realizar un programa que utilice un dispositivo i2c comopor 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/

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

```
/*********************************************************************
 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 128x32
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);
  }
```

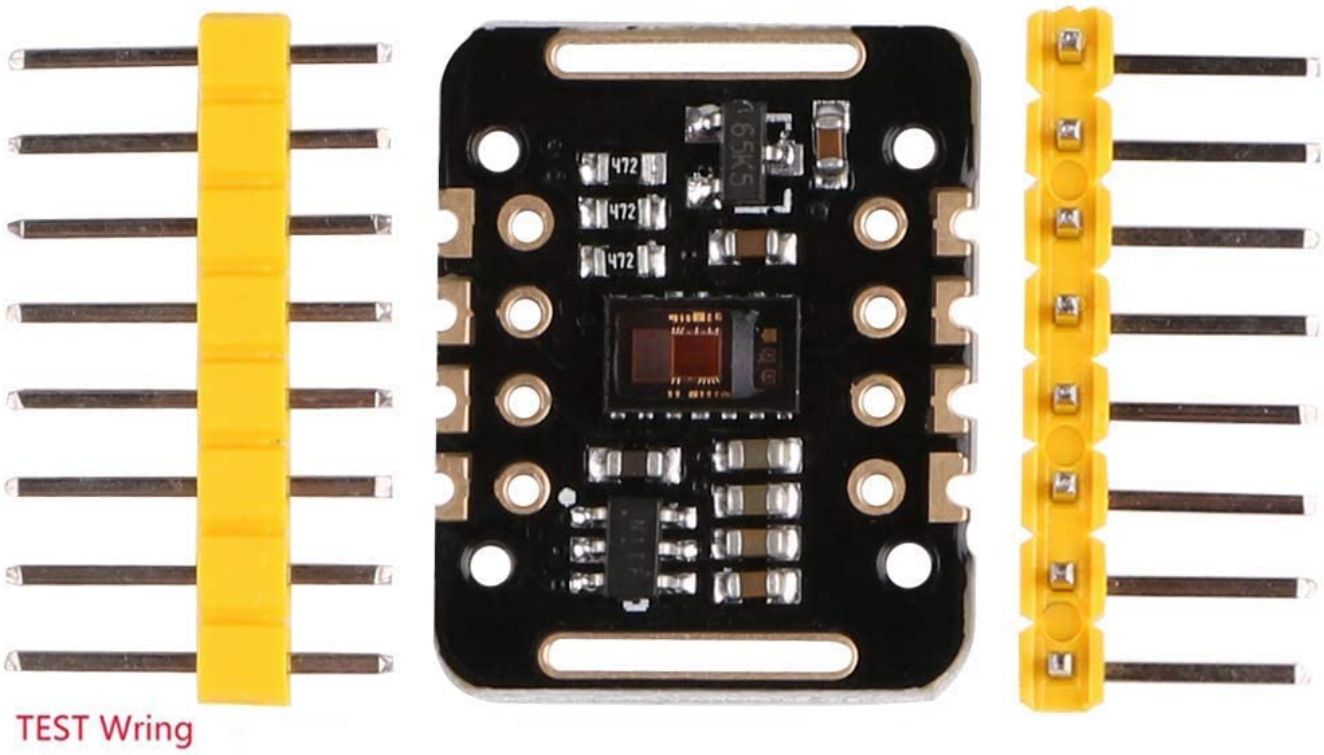
```
  for(i=0; i<display.width(); i+=4) {
    display.drawLine(display.width()-1, 0, i, display.height()-1, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  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,
```

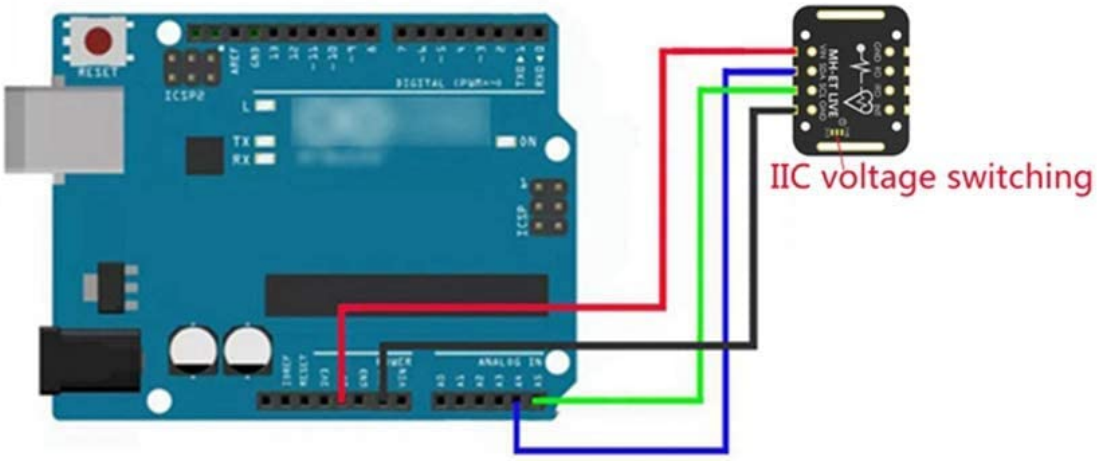
```
      display.height()/4, SSD1306_WHITE);
    display.display();
    delay(1);
  }

  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);
      }
    }
  }
}
```

Sensor Max3012 medidor de oxigeno en sangre y frecuencia cardiaca





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

```
/*
  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 with
  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 data in
//To solve this problem, 16-bit MSB of the sampled data will be truncated. Samples become 16-b
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 conversio
  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, adcRange
}

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, &validSPO2,

//Continuously taking samples from MAX30102.  Heart rate and SpO2 are calculated every 1 sec
while (1)
{
  //dumping the first 25 sets of samples in the memory and shift the last 75 sets of samples
  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 read

    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, &validSPO
```

```
  }
}
```

Para la realizacion de este ejercicio pueden utilizar cualquier elemento i2c que tengan disponible .

Necesario presentar en el informe y subirlo al github

1. fotos del montaje
2. salidas de depuracion (print...)
3. codigo generado
4. explicación del codigo

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