Laboratorio I2C

Belky Valentina Giron Lopez
est.belky.giron@unimilitar.edu.co
Docente: José De Jesús Rúgeles

Resumen — Durante la práctica se trabajó la comunicación digital utilizando el protocolo I2C, teniendo como dispositivo principal una Raspberry Pi Pico 2W programada con MicroPython. Se integraron dos esclavos en este caso una pantalla OLED con dirección 0x3C y el acelerómetro MPU6050 con dirección 0x68. Para validar la información transmitida en el bus se utilizó un analizador lógico, lo que permitió identificar las condiciones de inicio Start y parada Stop, las direcciones enviadas, la confirmación de los dispositivos mediante ACK/NACK y la transferencia de datos.

Abstract -- During the practical session, digital communication was worked on using the I2C protocol, with a Raspberry Pi Pico 2W programmed with MicroPython as the main device. Two slaves were integrated, in this case an OLED display with address 0x3C and the MPU6050 accelerometer with address 0x68. A logic analyzer was used to validate the information transmitted on the bus, which allowed us to identify the start and stop conditions, the addresses sent, the device acknowledgments via ACK/NACK, and the data transfer.

I. Introducción

En este laboratorio, la Raspberry Pi Pico 2W se configuró como maestro del bus, estableciendo comunicación con la pantalla OLED y el acelerómetro. El objetivo fue analizar la dinámica del protocolo en condiciones reales, identificando los elementos esenciales de la trasmisión como lo es el Start, dirección, ACK/NACK y Stop y relacionando los comandos enviados con las acciones visibles en la pantalla. El uso de un analizador lógico permitió observar las señales generadas, confirmar la información transmitida.

II. Desarrollo de experimentos

El montaje se realizó empleando la Raspberry Pi Pico 2W programada en MicroPython como maestro del bus I2C. A este se conectaron los dispositivos esclavos:

- OLED, con dirección 0x3C.
- MPU6050, con dirección 0x68.

Se utilizó la interfaz I2C de la Pico, asignando los pines GP5 (SCL) y GP4 (SDA). Ambos se conectaron en paralelo a estas líneas, siguiendo la arquitectura propia del protocolo. Además, cada dispositivo recibió su respectiva alimentación (VCC y GND).

Para registrar y analizar las señales se empleó un analizador lógico, conectado de la siguiente manera:

- CH0 → línea SCL
- CH1 → línea Tx
- CH2 → línea SDA
- GND

Con esta configuración se cargaron programas de prueba en Thonny.

Código empleado:

Listing 1: i2c_ping_0x3c_min.py

```
# i2c_ping_0x3c_min.py  (MicroPython RP2040)
import machine, time

i2c  = machine.I2C(1, scl=machine.Pin(15), sda=machine.Pin(14), freq
   =100000)
ADDR = 0x3C

time.sleep_ms(1000)  # tiempo para armar el analizador

print("Probing 0x3c ...")
try:
    # (addr, buffer, stop) -> sin keywords
    i2c.writeto(ADDR, b"", True)   # START, 0x3C(W), ACK, STOP
    print("ACK de 0x3c")
except OSError as e:
    print("NACK / no responde:", e)
    # Si tu firmware no permite buffer vac o, descomenta la
       siguiente l nea:
    # i2c.writeto(ADDR, b"\x00", True)  # ver s 0x3C(W), ACK, 0x00,
       ACK, STOP
```

Fig. 1. Código implementado en Thonny ACK.

El código implementado busca comprobar la existencia de la pantalla OLED en la dirección 0x3C. Al ejecutar el programa, si el dispositivo responde correctamente, se genera un ACK. Este reconocimiento ocurre en el noveno pulso de reloj, cuando el maestro libera la línea SDA y el esclavo la lleva a nivel bajo para confirmar la recepción.

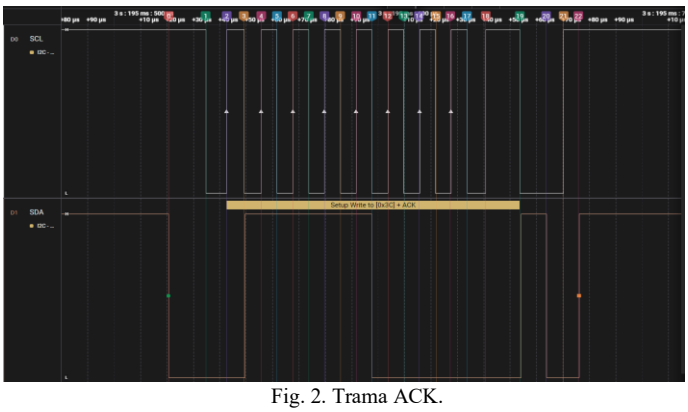


Fig. 2. Trama ACK.

En la trama se observan claramente:

- La condición de Start (SDA baja mientras SCL está en alto).
- La transmisión de la dirección 0x3C con bit de escritura.
- El ACK del esclavo al recibir la instrucción.
- La condición de Stop (SDA pasa de bajo a alto mientras SCL permanece en alto).

Cuando se intenta acceder a una dirección inexistente para este caso modificamos el ADDR a la dirección 0x3D, el dispositivo no responde. En este caso, el maestro envía la dirección, pero, en el noveno pulso, la línea SDA se mantiene en alto. El analizador marca este evento como NACK, confirmando que no hay ningún dispositivo en esa dirección como se puede evidenciar en la siguiente trama.

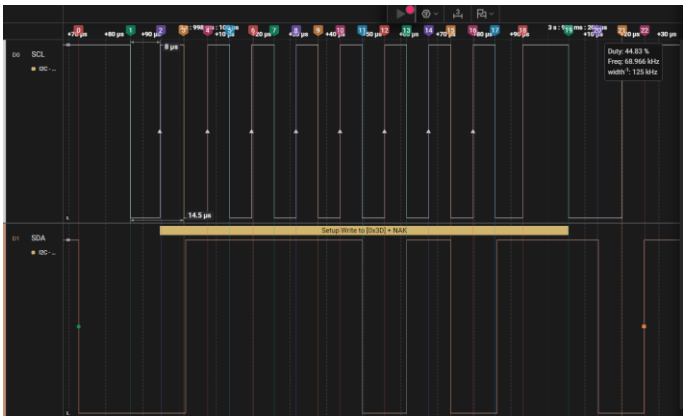


Fig. 3. Trama NACK.

Cuando el maestro termina de enviar los 8 bits (los 7 de la dirección y el bit de lectura/escritura), deja libre la línea SDA para que el dispositivo conectado pueda responder.

En la captura, la línea SDA nunca baja, se queda en “1”, lo que significa que ningún dispositivo respondió, eso es un NACK.

Además, el analizador muestra que todo ese byte tardó unos 143,5 microsegundos en transmitirse, lo cual corresponde a una velocidad muy cercana a los 100 kHz que se configuraron en el bus I2C.

Elemento	Captura ACK	Captura NACK
Start detectado (SDA↓ con SCL alto)	Sí	Sí
Octeto enviado (indicado por el decodificador)	0x78 (dirección 0x3C)	0x7A (dirección 0x3D)
Bit 9 (ACK=0 / NACK=1)	ACK = 0 (SDA en bajo)	NACK = 1 (SDA en alto)

Comparación ACK vs NACK

En ambos casos se mantienen las fases de Start, envío de dirección y Stop. Lo que varía es el noveno bit:

- En ACK, el esclavo baja la línea SDA confirmando recepción.
- En NACK, la línea permanece en alto, indicando que no existe un dispositivo en la dirección enviada.

Por qué con dirección incorrecta el esclavo no responde con ACK?

Cuando el maestro envía una dirección en el bus I²C, los dispositivos conectados la comparan con la suya propia.

Si alguno coincide, ese esclavo toma el control de la línea SDA en el noveno pulso de reloj y la lleva a nivel bajo, generando un ACK para confirmar que está disponible.

En cambio, cuando el maestro envía una dirección que no corresponde a ningún dispositivo presente, ninguno de ellos responde. Como consecuencia, la línea SDA se mantiene en nivel alto, y el analizador lógico lo interpreta como un NACK.

Se implementó el codigo i2c.scan():

Listing 2: scan_i2c_addr.py

```
# Digital Communication UMNG
# jose.rugeles@unimilitar.edu.co
# SCAN I2C ADDR- Raspberry Pi Pico

import machine

# Create I2C object
i2c = machine.I2C(1, scl=machine.Pin(15), sda=machine.Pin(14))

# Print out any addresses found
devices = i2c.scan()

if devices:
    for d in devices:
        print(hex(d))
```

Fig. 4. Código de escaneo implementado en Thonny.

Este código se impelente para detectar automáticamente los dispositivos en el bus. El analizador lógico permitió visualizar cómo el maestro recorre las direcciones, enviando tramas de escritura y recibiendo NACK en la mayoría de ellas, salvo en las direcciones reales (0x3C y 0x68), donde se observa el ACK.

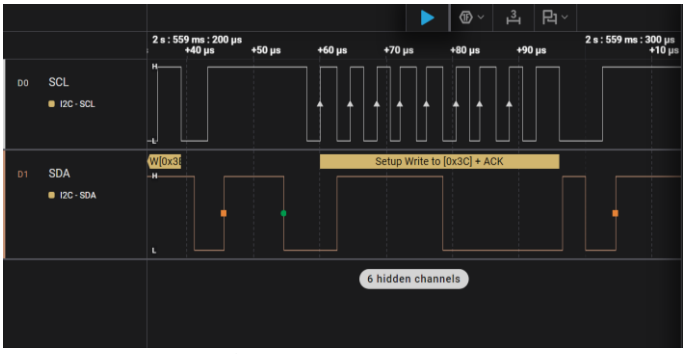


Fig. 5. Trama escaneo ACK.

Fig. 6. Trama escaneo NACK.

Esto nos permite detectar automáticamente los dispositivos conectados, sin necesidad de conocer previamente sus direcciones.

Control de la pantalla OLED

Con el programa OLED_demo_menu.py se ejecutaron distintas funciones sobre la pantalla, observando la relación entre los comandos transmitidos y las respuestas visibles:

- Apagar (0xAE): la pantalla se apaga completamente tras recibir el comando.
- Encender (0xAF): la pantalla se reactiva mostrando el contenido previo.
- Contraste (0x81 + valor): se ajusta la intensidad de los píxeles según el valor enviado.
- Invertir (0xA6 / 0xA7): alterna entre visualización normal e inversa.
- Texto/Animación (0x40 + datos): se envían secuencias de datos que generan texto o gráficos en pantalla.

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
import time, sys

# ---------- Config ----------
BUS_ID    = 0
PIN_SCL   = 13
PIN_SDA   = 12
FREQ      = 50_000       # 50 kHz: cómodo para analizador lógico (sube a 100k/400k si todo va bien)
CHUNK     = 16           # tamaño de trozo para write_data (8/16/32 suelen ir bien)
PAUSE_US  = 0            # micro-pausa entre trozos (0..100). Si hay errores, prueba 50.

CUR_FREQ = FREQ          # espejo de la frecuencia actual (algunos puertos no exponen i2c.freq())

# ---------- Subclase que trocea write_data ----------
class ChunkedSSD1306_I2C(SSD1306_I2C):
    def write_data(self, buf):
        # Enviar en una sola transacción: 0x40 (control byte de datos) + trozo.
        mv = memoryview(buf)
        for i in range(0, len(mv), CHUNK):
            part = bytes(mv[i:i+CHUNK])          # conversión explícita evita problemas con memoryview
            # Una sola transacción con el control byte al frente (SSD1306 lo espera así):
            self.i2c.writeto(self.addr, b'\x40' + part)
            if PAUSE_US:
```

Fig. 7. Código implementado en Thonny para control de pantalla OLLED.

Apagar

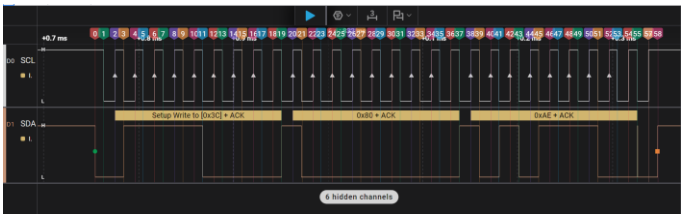
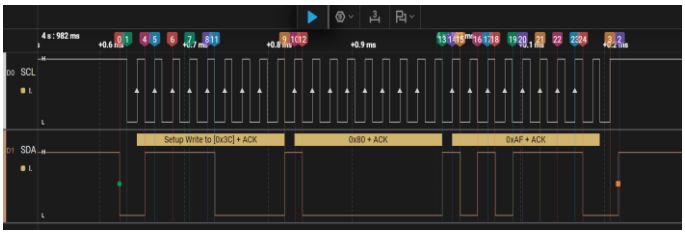


Fig. 8. Trama apagar.

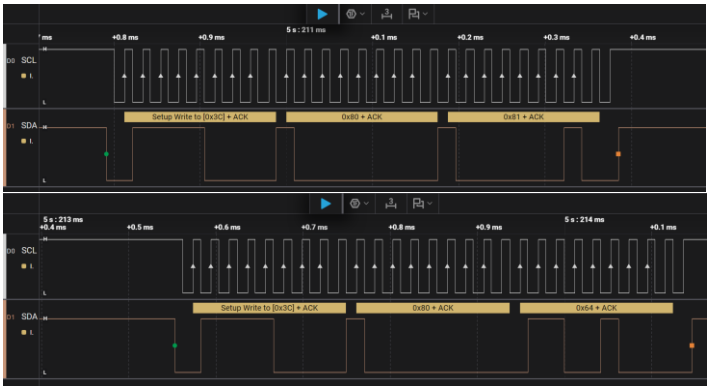
En la trama se visualiza primero la condición de inicio (Start), tras la cual el maestro envía la dirección 0x3C en modo de escritura. A continuación, se transmite el byte de control (0x80) seguido del comando 0xAE. Ambos son reconocidos por la pantalla OLED mediante la señal de ACK. La comunicación finaliza con la condición de Stop. Como resultado, la pantalla se apaga por completo y deja de mostrar la información que tenía cargada.

Encender



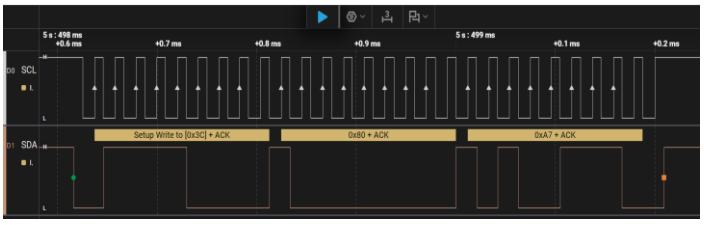
- En la trama: Start, dirección 0x3C, byte de control, comando 0xAF +ACK, Stop.
- En la pantalla: la OLED vuelve a encenderse mostrando la información que estaba guardada antes de apagarse.

Contraste 100



- En la trama: primero se envía el comando 0x81, y justo después otro byte con el valor del contraste elegido en este caso 0x64 que equivale a 100 en decimal. Ambos con ACK.
- En la pantalla: cambia la intensidad de los píxeles más brillantes si el valor es alto, más tenues si es bajo.

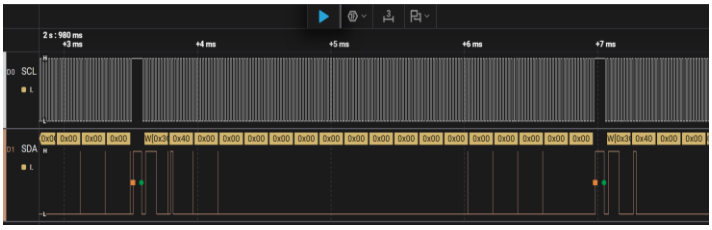
Inverso



- En la trama: aparece Start, dirección 0x3C, byte de control, para este caso el comando 0xA7 (invertir) o 0xA6 (volver a normal), ACK, Stop.

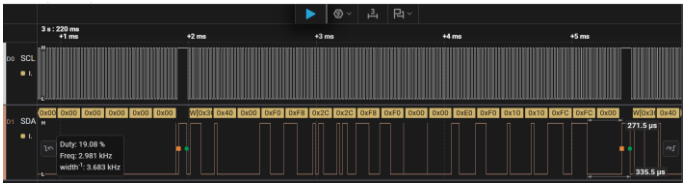
- En la pantalla: los colores se invierten los píxeles encendidos se ven apagados y los apagados se encienden.

Limpiar



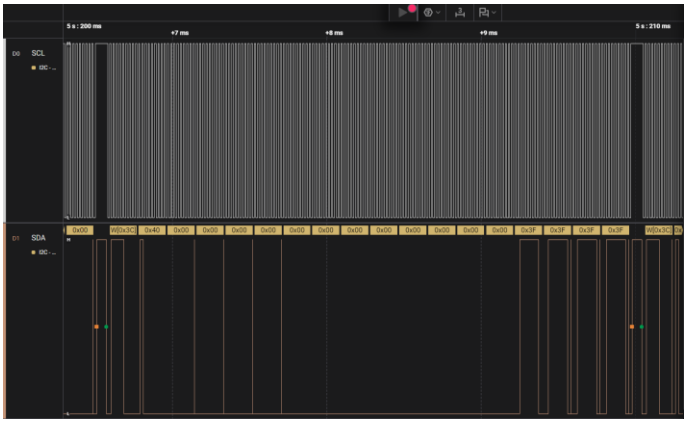
- En la trama: se envía una serie de datos usando el byte de control 0x40, con todos los valores en “0”.
- En la pantalla: la OLED se queda en negro porque se borran todos los píxeles.

Texto demo



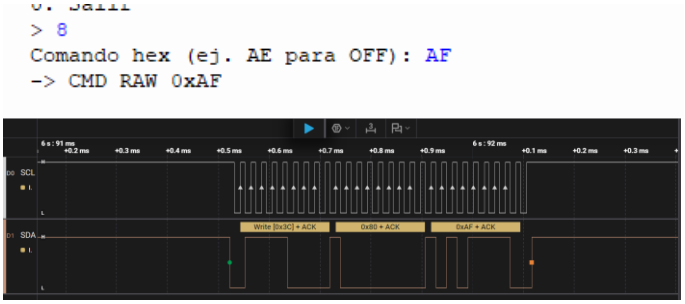
- En la trama: Start, dirección 0x3C, byte de control 0x40, secuencia de bytes que representan cada carácter, ACK en cada byte, Stop.
- En la pantalla: aparece texto escrito, tal como lo definió el código.

Animación



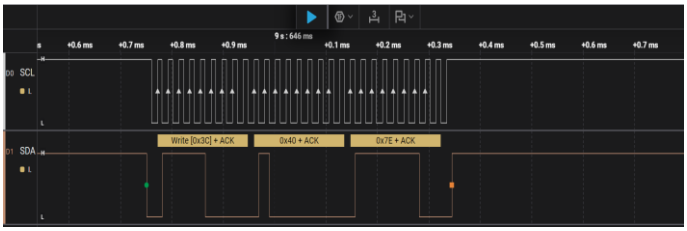
- En la trama: similar a texto, pero los datos (0x40 + secuencia de bytes) cambian rápido en el tiempo.
- En la pantalla: se ven gráficos que cambian como si fueran un dibujo animado**.**

Comando raw



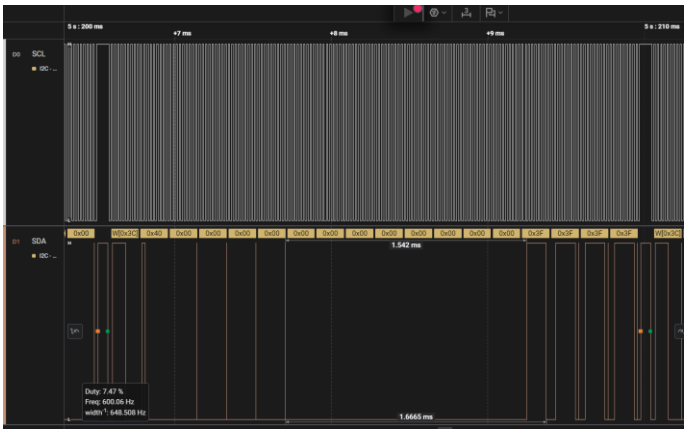
- En la trama: Start, dirección 0x3C, byte de control 0x80, comando elegido para este caso AF, ACK, Stop.
- En la pantalla: depende del comando enviado para est caso AF que es encender.

Dato raw



- En la trama: Start, dirección 0x3C, byte de control 0x40, datos enviados, ACK, Stop.
- En la pantalla: aparece lo que esos bytes representen (píxeles encendidos en posiciones concretas).

Cambio de frecuencia



- En la trama: no se ve un cambio en la estructura de los datos, sino en el espaciado de los pulsos del reloj (SCL). A mayor frecuencia, los pulsos se ven más juntos; a menor frecuencia, más separados.
- En la práctica: la comunicación será más rápida o más lenta, pero la OLED seguirá respondiendo a los mismos comandos.

REFERENCIAS

[1] Universidad Militar Nueva Granada. (2025). Guía de laboratorio: Comunicación I²C con Raspberry Pi Pico 2W y pantalla OLED SSD1306. Facultad de Ingeniería, Curso de Computadores II.

Link repositorio GitHub:
https://github.com/belkyvalentina11/Comunicaci-n-digital-.git