LABORATORIO DE CODIFICACIÓN HAMMING (7,4)

Vallentina Diaz Valbuena
est.vallentina.diaz@unimilitar.edu.co
Docente: José de Jesús Rugeles

I. RESUMEN

En este laboratorio se implementó un sistema de adquisición, codificación y transmisión de datos utilizando una Raspberry Pi Pico 2W y el sensor MPU6050. El objetivo principal fue aplicar la *codificación Hamming (7,4)* para proteger la integridad de las lecturas del acelerómetro antes de su transmisión. Cada muestra de 16 bits se dividió en cuatro *nibbles* de 4 bits, que fueron codificados individualmente en palabras de 7 bits, generando una trama final de 28 bits. Esta trama se transmitió mediante UART y se visualizó en un osciloscopio digital, donde se validó tanto la estructura de la señal como la correcta codificación. Se comprobó experimentalmente que los tiempos de bit y de trama medidos coincidían con los valores teóricos, confirmando la efectividad del esquema de codificación y la configuración del sistema.

II. ABSTRACT

In this lab, a data acquisition, encoding, and transmission system was implemented using a Raspberry Pi Pico 2W and the MPU6050 sensor. The main objective was to apply Hamming (7,4) encoding to protect the integrity of the accelerometer readings before transmission. Each 16-bit sample was divided into four 4-bit nibbles, which were individually encoded into 7-bit words, generating a final 28-bit frame. This frame was transmitted via UART and viewed on a digital oscilloscope, where both the signal structure and correct encoding were validated. It was experimentally verified that the measured bit and frame times coincided with the theoretical values, confirming the effectiveness of the coding scheme and the system configuration.

III. INTRODUCCIÓN

La transmisión confiable de datos en sistemas es un aspecto crítico en aplicaciones que requieren integridad y precisión en la comunicación de información. En este contexto, los códigos de detección y corrección de errores desempeñan un papel fundamental, especialmente en entornos donde el ruido o las interferencias pueden alterar la señal transmitida. Uno de los esquemas más utilizados para este propósito es el *código Hamming (7,4)*, el cual permite codificar palabras de 4 bits en palabras de 7 bits, añadiendo bits de paridad que facilitan la detección y corrección de errores individuales.

IV. DESARROLLO EXPERIMENTAL

En este laboratorio se implementó un sistema de adquisición y transmisión de datos utilizando una Raspberry Pi Pico 2W como unidad de procesamiento central. El sistema lee datos del sensor MPU6050, un módulo inercial que incluye acelerómetro y giroscopio, a través del protocolo I²C. Cada muestra de 16 bits correspondiente a una lectura del acelerómetro se divide en 4 *nibbles* (4 bits cada uno), los cuales son codificados individualmente mediante el algoritmo Hamming (7,4) con paridad par.

Posteriormente, la trama codificada de 28 bits es transmitida a través del módulo UART hacia un osciloscopio digital Tektronix TDS2012B del laboratorio, donde se visualiza la señal para su análisis. Este proceso permite no solo validar la correcta codificación y estructura de la trama, sino también observar el efecto de la *codificación Hamming* en la integridad de los datos transmitidos.

En la figura 1 se observa el diagrama del montaje y la conexión del osciloscopio digital en la salida Tx del UART del microcontrolador.

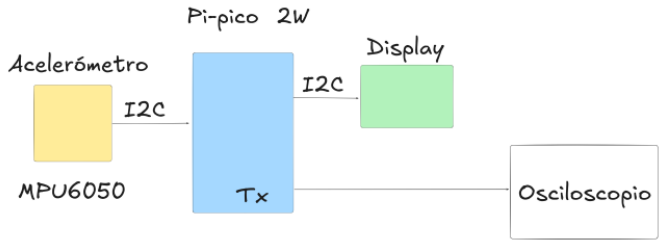

Figura 1. Diagrama del montaje

El proceso que se debe realizar se divide en tres fases:

1. *Toma de muestra del acelerómetro*

El proceso inicia con la lectura de la aceleración en el eje X, representada como un valor digital de 16 bits proveniente del sensor MPU6050. Esta palabra completa se divide en 4 segmentos de 4 bits cada uno, denominados *nibbles*, los cuales se identifican como A, B, C y D. Cada nibble corresponde a una porción específica de la palabra original: el nibble A abarca los bits menos significativos (posición 3-0), el B los bits 7-4, el C los bits 11-8 y el D los bits 15-12. Esta división facilita el manejo individual de cada segmento para la siguiente etapa de codificación, como se muestra a continuación:

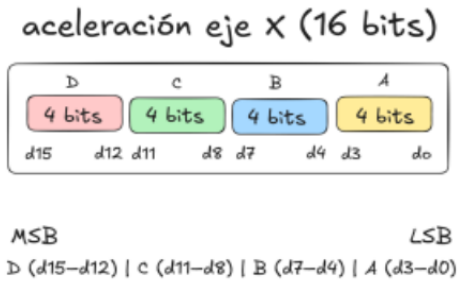

Figura 2. Primera fase del proceso

2. *Codificación Hamming (7,4)*

Cada uno de los cuatro *nibbles* obtenidos en la etapa anterior se procesa de forma independiente mediante el *codificador Hamming (7,4)*. Este algoritmo toma una palabra de 4 bits de datos y genera una palabra codificada de 7 bits, incorporando tres bits de paridad par (P1, P2, P4) según las relaciones establecidas. Como resultado, cada nibble (A, B, C, D) se transforma en una nueva palabra de 7 bits (A', B', C', D'), lo que permite posteriormente la detección y corrección de errores simples en la transmisión, como se observa en la siguiente figura:

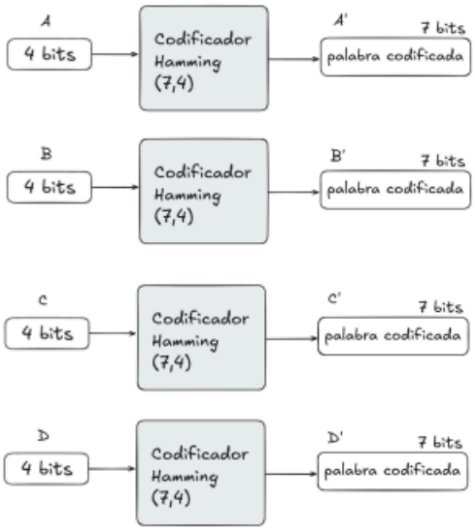

Figura 3. Segunda fase del proceso

3. *Resultado de la codificación*

Finalmente, las cuatro palabras codificadas de 7 bits cada una se encadenan para formar una trama única de 28 bits, que representa la muestra original del acelerómetro protegida contra errores. Esta trama ampliada está lista para ser transmitida a través del puerto UART hacia el osciloscopio, donde podrá visualizarse y validarse tanto la estructura de la señal como la efectividad de la *codificación Hamming* aplicada. De este modo, se asegura que la información del sensor pueda ser recuperada de manera fiable incluso en presencia de interferencias, como se muestra a continuación:

Figura 4. Tercera fase del proceso

Una vez se entiende el funcionamiento se implementa el siguiente programa en el entorno de desarrollo integrado Thonny con Python, el cual se aprecia a continuación:

```
from machine import I2C, Pin
import ssd1306
from imu import MPU6050
from time import sleep

# ===== Configuración I2C =====
i2c = I2C(0, scl=Pin(5), sda=Pin(4))
oled = ssd1306.SSD1306_I2C(128, 64,
i2c)
imu = MPU6050(i2c)
# ===== Pin de salida =====
tx_pin = Pin(15, Pin.OUT)
# ===== Codificador Hamming (7,4)
=====
def hamming74_encode(bits4):
    d3, d2, d1, d0 = bits4
    P1 = d3 ^ d1 ^ d0
    P2 = d3 ^ d2 ^ d0
    P4 = d3 ^ d2 ^ d1
    return [d3, d2, d1, P4, d0, P2,
P1]
# ===== Convierte un entero en lista
de bits (MSB primero) =====
def int_to_bits(value, nbits):
    return [(value >> i) & 1 for i
in range(nbits-1, -1, -1)]
# ===== Envía lista de bits por el
pin =====
def send_bits(bits):
    for b in bits:
        tx_pin.value(b)
        sleep(0.05)  # duración de
cada bit

while True:
    # 1) Leer eje X (16 bits,
entero)
    ax_raw = int(imu.accel.x * 1000)
# escalar a entero
    ax_raw &= 0xFFFF  # limitar a 16
bits
    # 2) Convertir a 16 bits
    bits16 = int_to_bits(ax_raw, 16)
```

```
    # 3) Dividir en bloques de 4
bits (A=LSB ... D=MSB)
    A = bits16[12:16]   # d3..d0
    B = bits16[8:12]    # d7..d4
    C = bits16[4:8]     # d11..d8
    D = bits16[0:4]     # d15..d12
    # 4) Codificar cada bloque en 7
bits
    A_ = hamming74_encode(A)
    B_ = hamming74_encode(B)
    C_ = hamming74_encode(C)
    D_ = hamming74_encode(D)
    # 5) Concatenar en 28 bits (D'
C' B' A')
    code28 = D_ + C_ + B_ + A_
    # 6) Mostrar en OLED con mejor
separación
    oled.fill(0)
    oled.text("AX RAW:", 0, 0)
    oled.text(str(ax_raw), 0, 10)
    oled.text("16 bits:", 0, 25)
    oled.text("".join(map(str,
bits16[:8])), 0, 35)   # primera
mitad
    oled.text("".join(map(str,
bits16[8:])), 0, 45)   # segunda
mitad
    oled.show()
    sleep(1)
    # 7) Mostrar 28 bits codificados
en consola
    print("ax:", ax_raw)
    print("bits16:", bits16)
    print("codificado (28b):",
code28)
    # 8) Enviar por pin
    send_bits(code28)
```

En pocas palabras este código lo que hace es permitir obtener las salida codificada de una muestra del acelerómetro, una vez conectamos los correspondientes pines como se muestran a continuación:

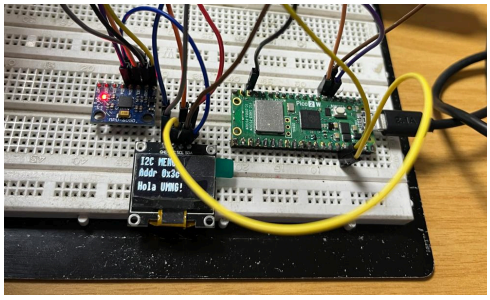

Imagen 1. Montaje físico

Una vez conectado se procede a correrlo y verlo desde el osciloscopio digital, la primera salida codificada se logra apreciar a continuación:

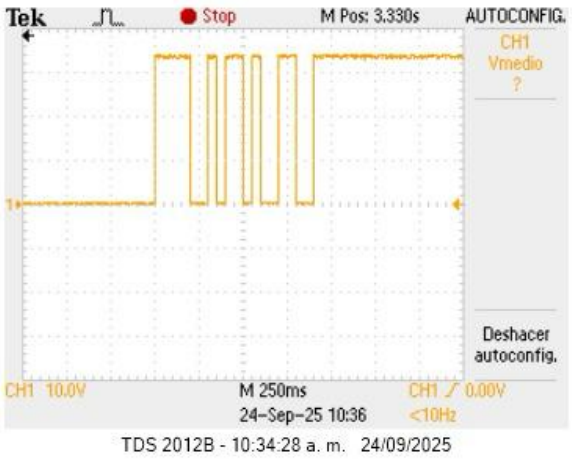

Imagen 2. Primera salida codificada

En la consola de Thonny se obtienen los siguientes datos de la primera salida codificada, donde se encuentra el valor del ax, los 16 bits y la codificación correspondiente:

```
ax: 850
bits16: [0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1, 0]
codificado (28b): [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1]
```

Imagen 3. Consola de la primera salida

La comprobación de los datos del osciloscopio digital con los datos de la consola se obtiene a continuación, donde cada segmento de color se refiere a un nibble distinto:

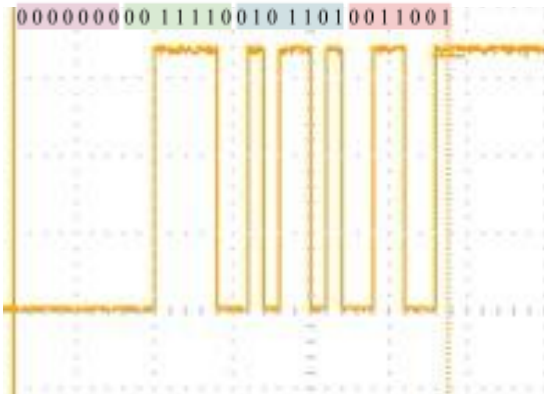

Imagen 4. Análisis de la primera salida

Además, por medio de los cursores del osciloscopio digital, se mide aproximadamente lo que vendría a ser el tiempo de bit:

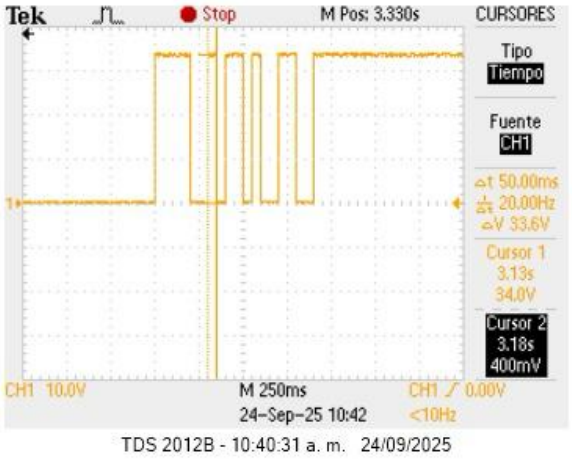

Imagen 5. Tiempo de bit de la primera salida

Por consiguiente, se puede decir que:

$$tiempo\ de\ trama = tiempo\ de\ bit \times \#bits$$

Reemplazando los datos utilizados:

$$tiempo\ de\ trama = 50\ ms \times 28 = 1,4\ s$$

Al medirlo con el osciloscopio se obtiene lo siguiente:

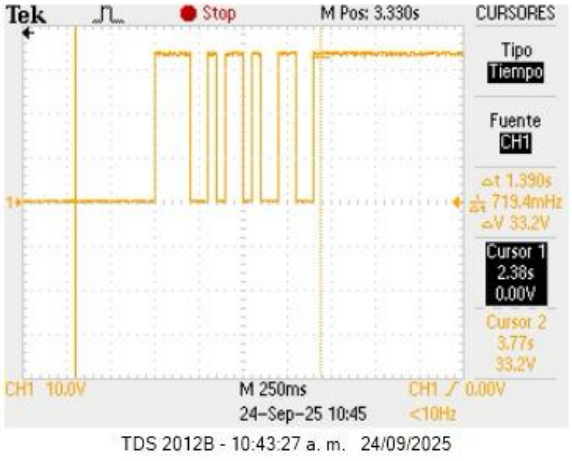

Imagen 6. Tiempo de trama de la primera salida

Entonces, se logra concluir que el tiempo de la trama medido en el osciloscopio coincide con el tiempo de la trama calculado.

Ahora bien, se hace otra prueba con una segunda salida codificada, que se logra observar a continuación:

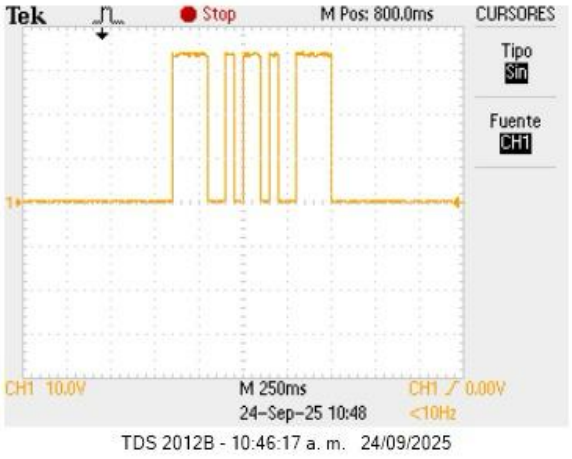

Imagen 7. Segunda salida codificada

En la consola de Thonny se obtienen lis siguientes datos de la segunda salida codificada, donde se encuentra el valor del ax, los 16 bits y la codificación correspondiente:

```
ax: 051
bits16: [0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 1, 0, 1, 1]
codificado (28b): [0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 0]
```

Imagen 8. Consola de la segunda salida

La comprobación de los datos del osciloscopio digital con los datos de la consola se obtiene a continuación, donde cada segmento de color se refiere a un nibble distinto:

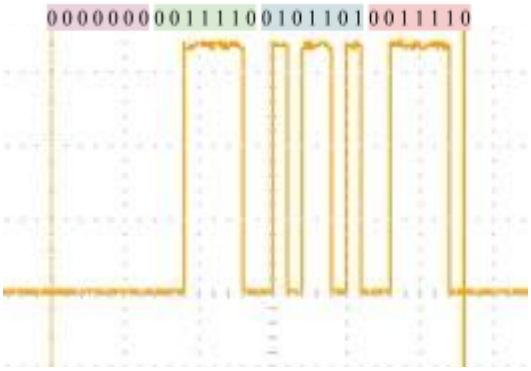

Imagen 9. Análisis de la segunda salida

Además, por medio de los cursores del osciloscopio digital, se mide aproximadamente lo que vendría a ser el tiempo de bit:

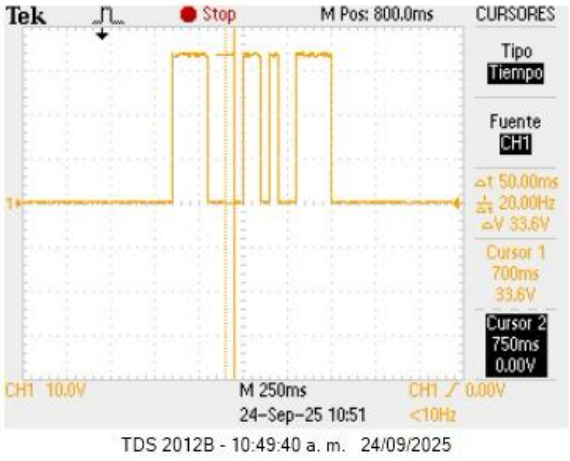

Imagen 10. Tiempo de bit de la segunda salida

Por consiguiente, se puede decir que como el tiempo de bit es el mismo que en la primera salida, al igual que el tiempo de trama, entonces al medirlo con el osciloscopio se obtiene lo siguiente:

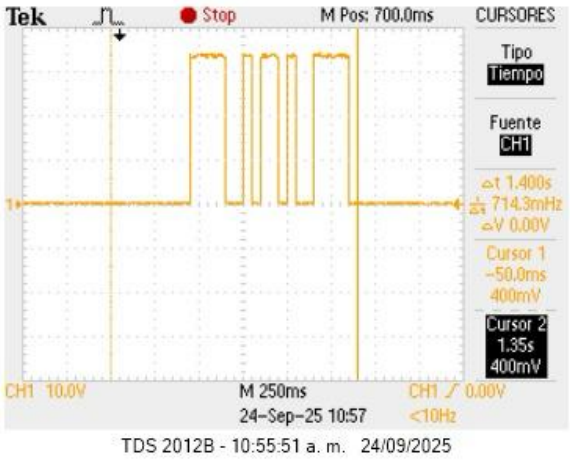

Imagen 11. Tiempo de trama de la segunda salida

Entonces, se logra concluir que el tiempo de la trama medido en el osciloscopio coincide con el tiempo de la trama calculado.

V. CONCLUSIONES

- Se logró implementar exitosamente un sistema que adquiere datos del sensor MPU6050, los codifica mediante *Hamming (7,4)* y los transmite de forma confiable a través de UART, validando el flujo completo de procesamiento de señales.

- La *codificación Hamming* permitió agregar redundancia a los datos, facilitando la detección y corrección de errores en la transmisión, lo que es esencial en entornos con posibles interferencias.

- La comparación entre los valores teóricos y los medidos en el osciloscopio mostró una coincidencia exacta en los tiempos de bit (50 ms) y de trama (1.4 s), lo que valida la configuración temporal del sistema y la correcta generación de la señal.

- El uso de la Raspberry Pi Pico 2W demostró ser una plataforma eficiente para el procesamiento en tiempo real y la implementación de protocolos de comunicación como I²C y UART en aplicaciones de adquisición de datos.

- La visualización de la trama codificada en el osciloscopio permitió un análisis detallado de la señal transmitida, verificando la correcta estructura de los *nibbles* codificados y la secuencia de bits generada.

- Este proyecto integra conceptos clave de sistemas integrados, procesamiento digital de señales y teoría de codificación, proporcionando una base sólida para el desarrollo de sistemas de comunicación robustos en aplicaciones industriales y académicas.

VI. REFERENCIAS

[1] Raspberry Pi Foundation, *Raspberry Pi Pico 2 W Datasheet*.

[2] Hamming, R. W. (1950). Error detecting and error correcting codes. The Bell System Technical Journal, 29(2), 147–160.

[3] Stallings, W. (2016). Data and Computer Communications (10th ed.). Prentice Hall. (Capítulo 6: Detección y corrección de errores).

[4] MicroPython Documentation. (2023). Machine Module: I2C, UART, and Pin Control.

[5] Forouzan, B. A. (2013). Data Communications and Networking (5th ed.). McGraw-Hill.
(Sección sobre códigos Hamming y detección de errores).

VII. REPOSITORIO DE GITHUB

https://github.com/Vallentincita/Comunicaci-n-digital-TEL-A-