

Universidad de Costa Rica

Escuela de Ingeniería Eléctrica

Laboratorio de Microcontroladores

Laboratorio 4: Sismógrafo

Pablo Durán Segura B62416

Sergio Garino Vargas B73157

31 de octubre de 2024

Índice

1. Resumen	3
2. Nota teórica	3
2.1. Registros	3
2.1.1. WHO_AM_I (0x0F)	4
2.1.2. CTRL_REG1 (0x20)	4
2.1.3. CTRL_REG4 (0x23)	4
2.1.4. STATUS_REG (0x27)	5
2.1.5. OUT_TEMP (0x26)	6
2.1.6. Registros del cambio angular en cada eje	6
2.2. Librerías	6
2.3. Alimentación	6
2.4. IoT	8
2.5. ThingsBoard	8
2.6. MQTT	8
2.7. Componentes	9
3. Desarrollo y análisis de resultados	10
3.1. Pantalla	10
3.2. Giroscopio	10
3.3. Batería	10
3.4. Comunicación serial	11
3.5. MQTT	12
3.6. ThingsBoard	13
4. Conclusiones	14
5. Recomendaciones	15

1. Resumen

En este informe se presenta una solución al laboratorio 4 del curso Laboratorio de Microcontroladores, el cual consiste en desarrollar una aplicación de sismógrafo en una placa de desarrollo STM32F429 Discovery Kit haciendo uso del giroscopio integrado, se debe mostrar el valor registrado en los tres ejes X, Y y Z así como la tensión que entrega una fuente de alimentación externa de 9 V en la pantalla integrada. Finalmente se hace que la aplicación sea IoT enviando los datos a ThingsBoard por medio del protocolo MQTT.

El código desarrollado se encuentra en el repositorio de GitHub: <https://github.com/PabloJoseD/Lab-4.git>. Si se desea probar el código es indispensable leer el README del repositorio.

Palabras claves: *MQTT, IoT, microcontrolador, ThingsBoard, comunicación serial*

2. Nota teórica

Para este laboratorio se utilizará la placa de desarrollo STM32F429 Discovery kit, entre sus características se destaca:

- Microcontrolador STM32F429ZIT6
 - Arquitectura ARM 32 bits Cortex-M4
 - 2 MB de memoria flash
 - 256 kB de memoria RAM
- Pantalla táctil LCD
 - Res 240x320 pixeles
 - Controlador gráfico ILI9341
 - Controlador táctil XPT2046
- Giroscopio MEMS L3GD20
- 6 LEDs
- 2 push button

2.1. Registros

Para poder programar el sismógrafo se debe acceder a algunos registros, por ejemplo en el giroscopio se trabaja con los siguientes registros:

2.1.1. WHO_AM_I (0x0F)

Este registro sirve como identificador del giroscopio, por esta razón siempre tiene el valor fijo 0xD4, de manera que cuando el microcontrolador lo lee sabe que está interactuando con el giroscopio.

Table 18. WHO_AM_I register							
1	1	0	1	0	1	0	0

Figura 1: Registro WHO_AM_I

2.1.2. CTRL_REG1 (0x20)

Este registro controla la configuración básica de encendido, frecuencia de salida, y el modo de operación del sensor. Este registro permite habilitar el giroscopio y definir sus parámetros de funcionamiento. A continuación, se desglosan los bits y su función:

- Bits [7:6] - DR1 y DR0: Configuran la frecuencia de salida (Data Rate) del giroscopio.
- Bits [5:4] - BW1 y BW0: Configuran el ancho de banda de los filtros de salida, afectando la suavidad de los datos y el tiempo de respuesta.
- Bit 3 - PD (Power Down): Controla el modo de encendido. Si está en 1, el giroscopio está activado; en 0, está en modo de bajo consumo.
- Bits [2:0] - Zen, Yen, Xen: Estos bits habilitan la medición en los ejes Z, Y y X respectivamente. Si están en 1, el eje correspondiente se habilita; si están en 0, el eje está deshabilitado.

Por ejemplo, establecer PD en 1 y Zen, Yen, Xen en 1 activaría el giroscopio en modo normal, habilitando los tres ejes.

Table 19. CTRL_REG1 register							
DR1	DR0	BW1	BW0	PD	Zen	Xen	Yen

Figura 2: Registro CTRL_REG1

2.1.3. CTRL_REG4 (0x23)

Con este registro se puede configurar la resolución en grados por segundo (dps) con la que mide el giroscopio, también permite configurar el comportamiento de los datos de salida como la lectura de los bytes, el endianness y el modo SPI. La siguiente figura describe lo que representa cada bit.

Table 29. CTRL_REG4 register

BDU	BLE	FS1	FS0	-	0 ⁽¹⁾	0 ⁽¹⁾	SIM
-----	-----	-----	-----	---	------------------	------------------	-----

1. This value must not be changed.

Table 30. CTRL_REG4 description

BDU	Block data update. Default value: 0 (0: continuous update; 1: output registers not updated until MSb and LSb reading)
BLE	Big/little endian data selection. Default value 0. (0: Data LSb @ lower address; 1: Data MSb @ lower address)
FS1-FS0	Full scale selection. Default value: 00 (00: 250 dps; 01: 500 dps; 10: 2000 dps; 11: 2000 dps)
SIM	SPI serial interface mode selection. Default value: 0 (0: 4-wire interface; 1: 3-wire interface).

Figura 3: Registro CTRL_REG4

2.1.4. STATUS_REG (0x27)

Este registro proporciona información sobre el estado de los datos en cada eje (X, Y, Z). Este registro indica si hay nuevos datos disponibles o si ha ocurrido un overrun (sobrescritura) en los datos antes de ser leídos. A continuación, se explica cada bit mediante una imagen.

Table 37. STATUS_REG register

ZYXOR	ZOR	YOR	XOR	ZYXDA	ZDA	YDA	XDA
-------	-----	-----	-----	-------	-----	-----	-----

Table 38. STATUS_REG description

ZYXOR	X, Y, Z -axis data overrun. Default value: 0 (0: no overrun has occurred; 1: new data has overwritten the previous data before it was read)
ZOR	Z axis data overrun. Default value: 0 (0: no overrun has occurred; 1: new data for the Z-axis has overwritten the previous data)
YOR	Y axis data overrun. Default value: 0 (0: no overrun has occurred; 1: new data for the Y-axis has overwritten the previous data)
XOR	X axis data overrun. Default value: 0 (0: no overrun has occurred; 1: new data for the X-axis has overwritten the previous data)
ZYXDA	X, Y, Z -axis new data available. Default value: 0 (0: a new set of data is not yet available; 1: a new set of data is available)
ZDA	Z axis new data available. Default value: 0 (0: new data for the Z-axis is not yet available; 1: new data for the Z-axis is available)
YDA	Y axis new data available. Default value: 0 (0: new data for the Y-axis is not yet available; 1: new data for the Y-axis is available)
XDA	X axis new data available. Default value: 0 (0: new data for the X-axis is not yet available; 1: new data for the X-axis is available)

Figura 4: Registro STATUS_REG

2.1.5. OUT_TEMP (0x26)

Este registro de 8 bits indica la temperatura del sensor en complemento a dos.

Table 35. OUT_TEMP register

Temp7	Temp6	Temp5	Temp4	Temp3	Temp2	Temp1	Temp0
-------	-------	-------	-------	-------	-------	-------	-------

Table 36. OUT_TEMP register description

Temp7-Temp0	Temperature data
-------------	------------------

Temperature data (1LSB/deg - 8-bit resolution). The value is expressed as two's complement.

Figura 5: Registro OUT_TEMP

2.1.6. Registros del cambio angular en cada eje

Los registros OUT_eje_L y OUT_eje_H permiten medir el valor del cambio angular para cada uno de los ejes del giroscopio, hay dos registros por eje, uno para la parte más significativa y otro para la menos significativa.

OUT_X_L (28h), OUT_X_H (29h)

X-axis angular rate data. The value is expressed as two's complement.

OUT_Y_L (2Ah), OUT_Y_H (2Bh)

Y-axis angular rate data. The value is expressed as two's complement.

OUT_Z_L (2Ch), OUT_Z_H (2Dh)

Z-axis angular rate data. The value is expressed as two's complement.

Figura 6: Registros de los ejes

Nota: La información presentada al inicio de la nota teórica y en la subsección “Registros” fue obtenida de las presentaciones y las hojas de datos proporcionadas.

2.2. Librerías

Para este laboratorio se utilizará la biblioteca de código abierto LibOpenCM3, la cual se puede encontrar en el repositorio de GitHub <https://github.com/libopencm3/libopencm3/wiki>, esta librería tiene como objetivo crear firmware para varios microcontroladores ARM Cortex-M0(+)/M3/M4, incluyendo STM32, Ti Tiva y Stellaris, NXP LPC 11xx, 13xx, 15xx, 17xx, Atmel SAM3, Energy Micro EFM32 y otros. [1]

2.3. Alimentación

Según las especificaciones eléctricas de la placa esta se puede alimentar a 3V o 5V mediante la entrada USB o una fuente externa. Ya que la fuente de alimentación será una batería cuadrada de

9V es necesario hacer un circuito tipo divisor de tensión que permita reducir esa tensión a 5V para no dañar la placa. El circuito a implementar tiene la siguiente topología.

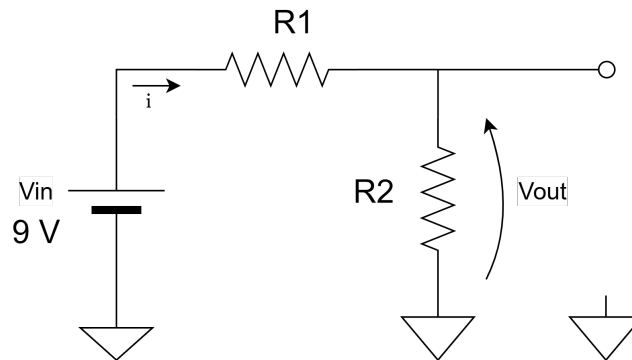


Figura 7: Divisor de tensiones

Utilizando la teoría de circuitos se sabe que la tensión a la salida esta dada por la ecuación de un divisor de tensiones.

$$V_{out} = V_{in} \cdot \frac{R_2}{R_1 + R_2} \quad (1)$$

Reacomodando los términos es posible encontrar una expresión que relaciona de forma directa las resistencias R_1 y R_2 de manera que se puede diseñar para los valores de tensión deseados.

$$\frac{V_{out}}{V_{in} - V_{out}} \cdot R_1 = R_2 \quad (2)$$

Sabiendo que el valor de la tensión a la entrada es 9V y a la salida deben ser 5V, la relación es:

$$\frac{5}{4} \cdot R_1 = R_2 \quad (3)$$

Con base en las resistencias disponibles se escogió una resistencia de $1\text{ k}\Omega$ para R_1 , de manera que la resistencia R_2 , debe ser de $1250\ \Omega$. Como no se cuenta con una resistencia comercial de ese valor se usarán dos resistores en serie de $1\text{ k}\Omega$ y $220\ \Omega$. Si se usan estos valores la tensión a la salida del divisor es de 4.94 V, por lo que el diseño es viable usando resistencias comerciales. A continuación se muestra una fotografía del circuito final.

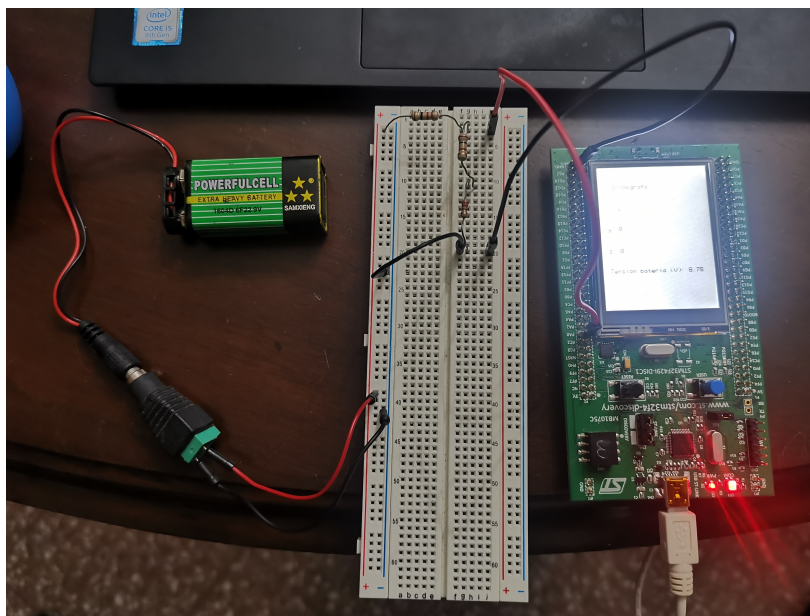


Figura 8: Modelo Pub/Sub MQTT

2.4. IoT

Es relevante conocer el concepto de Internet de las cosas (IoT) debido a que se desea comunicar el sismógrafo con Internet, de manera que se pueda ver desde una plataforma web los datos generados por el giroscopio.

El término IoT, o Internet de las cosas, se refiere a la red colectiva de dispositivos conectados y a la tecnología que facilita la comunicación entre los dispositivos y la nube, así como entre los propios dispositivos. Gracias a la llegada de los chips de ordenador de bajo coste y a las telecomunicaciones de gran ancho de banda, ahora tenemos miles de millones de dispositivos conectados a Internet. Esto significa que los dispositivos de uso diario, como los cepillos de dientes, las aspiradoras, los coches y las máquinas, pueden utilizar sensores para recopilar datos y responder de forma inteligente a los usuarios. [2]

2.5. ThingsBoard

Para visualizar los datos se usará la plataforma ThingsBoard de la escuela de ingeniería eléctrica. ThingsBoard es una plataforma de código abierto que permite desarrollar, manejar y escalar proyectos IoT. [3]

La plataforma es capaz de comunicarse por medio de varios protocolos, como por ejemplo HTTP, SNMP, CoAP, LwM2M, pero en este proyecto se va a utilizar MQTT.

2.6. MQTT

MQTT es un protocolo de transporte de mensajería de publicación/suscripción cliente-servidor. Es ligero, abierto, sencillo y está diseñado para ser fácil de implementar. Estas características lo hacen ideal para su uso en muchas situaciones, incluidos los entornos limitados como la comunicación

Máquina a Máquina (M2M) y la e Internet de las Cosas (IoT) en los que se requiere un pequeño y/o el ancho de banda de la red es escaso. de red. [4]

El protocolo MQTT fue inventado en 1999 por Andy StanfordClark (IBM) y Arlen Nipper (Arcom, ahora Cirrus Link). Ellos necesitaban un protocolo de mínima pérdida de batería y mínimo ancho de banda para conectar con oleoductos vía satélite. En dos inventores especificaron varios requisitos para el futuro protocolo: [4]

- Aplicación sencilla
- Entrega de datos con calidad de servicio
- Ligereza y eficiencia del ancho de banda
- Agnóstico a los datos
- Conciencia de sesión continua

El modelo Pub/Sub se puede describir con la siguiente imagen:

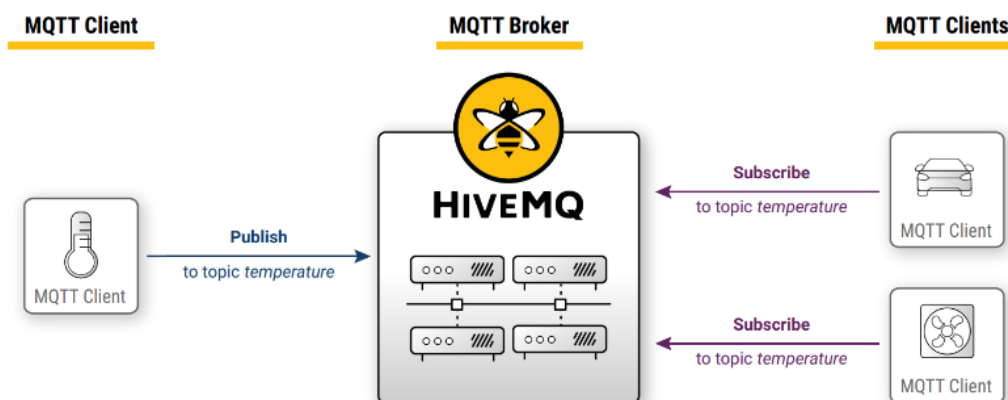


Figura 9: Modelo Pub/Sub MQTT

A diferencia del modelo Cliente/Servidor en MQTT se tiene un broker que sirve de mediador entre clientes, los cuales pueden publicar en algún tópico para enviar información, o bien el cliente puede suscribirse al tópico para recibir la información se esté publicando ahí.

Para comunicarse con el dashboard de ThingsBoard se utilizará el protocolo MQTT, esto usando programación en Python y el broker Mosquitto.

2.7. Componentes

En la siguiente tabla se listan los componentes usados en el proyecto.

Tabla 1: Componentes

Componente	Precio/unidad	Cantidad
STM32F429 Discovery Kit	48900	1
220 Ω res	25	1
1 k Ω res	70	2
Protoboard	5500	1
Bateria 9V	1400	1
Conector de bateria 9V	250	1
Total	56215	

3. Desarrollo y análisis de resultados

3.1. Pantalla

Para conectar la pantalla se utilizo el programa de ejemplo `lcd-serial`. A partir de este programa se realizaron ajustes para poder imprimir los tres ejes del giroscopio. Especialmente se utilizo la librería `gfx` para ajustar el tamaño del texto, elegir el color del mismo, poner un color de fondo, y acomodar cada linea de texto de modo que toda la información se pudiera leer correctamente. La librería cuenta con múltiples funciones para realizar todo esto. El texto se guardo en datos de tipo `char` ya que es necesario pasarle argumentos de este tipo a las funciones de la librería. Cada uno de los datos se imprimió por separado para facilitar la lectura de los mismos en el puerto serial.

3.2. Giroscopio

Para manejar el giroscopio se utilizo el protocolo SPI por medio de las funciones de la librería con el mismo nombre del protocolo. Dicha librería permite hacer acciones como por ejemplo elegir la tasa de baudios, escoger si el protocolo es full duplex o half duplex, hacer lecturas de datos y también enviarlos, entre otras funciones. Es importante mencionar que antes de iniciar el protocolo, se deben configurar las GPIOs que se utilizan para manejar las señales del protocolo. En este caso son la GPIO7, GPIO8 y GPIO9. Una vez inicializado el protocolo se puede conectar el giroscopio y leer las coordenadas. Es importante mencionar que se utilizo la sensibilidad que el fabricante recomienda (250dps), ya que de otra forma los datos que se leen son erróneos.

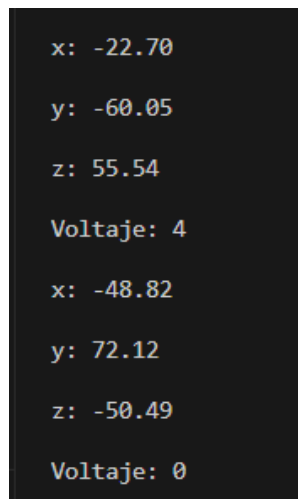
3.3. Batería

La alimentación del circuito se realizo mediante una batería de 9V, sin embargo el microcontrolador acepta como máximo 5V. Fue necesario hacer la conversión mediante código para que el microcontrolador calculara correctamente el nivel de la batería. Esto se hizo multiplicando el valor analógico 8.90, que es el máximo voltaje que la batería entregaba al momento de probar el circuito. Inicialmente se dividió este valor por 4095, sin embargo la conversión no estaba funcionando correctamente. También se intento calcular una función que defina la relación voltaje de la batería y voltaje del microcontrolador pero no funciono. Se opto por calcular el valor analógico que estaba dando esos resultados. Conociendo ese valor analogico se procedio a ajustar la formula del voltaje, el cual simplemente era dividir 8.90 entre 117.

3.4. Comunicación serial

El siguiente paso fue monitorear la comunicación serial, al contar con una única placa para el desarrollo del laboratorio se tomó la decisión de hacer esta parte simulada al inicio y una vez lograda la tarea se usan los datos reales generados por el microscopio, esta decisión también facilitó plantear una solución ya que fue una forma de dividir el trabajo en tareas más sencillas.

Cuando se habla de “hacer esta parte simulada” lo que se quiere decir es que los datos X, Y, Z y nivel de batería son generados por un script de Python y no por el microcontrolador, esto se hace muy sencillo ya que es solo generar valores aleatorios entre -100 y 100 para las coordenadas y entre 0 y 9 para la tensión de la batería, esos datos se envían igual que lo hace el microcontrolador, línea a línea y con un baud rate de 115200, tal como se ven en la siguiente imagen.



```
x: -22.70
y: -60.05
z: 55.54
Voltaje: 4
x: -48.82
y: 72.12
z: -50.49
Voltaje: 0
```

Figura 10: Datos aleatorios enviados por el puerto serial

Esos datos son recibidos en el otros extremos por otro script de Python, es cual sí es parte del laboratorio y se encarga de recibir, procesar y enviar estos datos a ThingsBoard, para recibir los datos se usa la librería `serial`, basta con configurar la comunicación serial y luego leer los datos línea por línea, esto se logra con pocas líneas de código. El siguiente código es solo una muestra y forma parte de un script que hace más tareas.

```
1 from serial import Serial
2
3 # Set up serial connection
4 ser = Serial('/dev/ttyACM0', 115200, timeout=1)
5 sleep(2)
6
7 # Read line
8 data = ser.readline().decode('utf-8').strip()
9
10 # Close connection
11 ser.close()
```

La comunicación se desarrolló de manera que se envía un dato por línea, tal como se ve en la figura 10, la razón de hacerlo así es facilitar la lectura de los datos y el envío a través de MQTT a la plataforma ThingsBoard, como se explicará más adelante.

Al igual que en el laboratorio 3 se usó el comando `socat` para comunicar los extremos de la comunicación serial, sin embargo, a la hora de obtener los datos desde el microcontrolador directamente, solo se tuvo que monitorear el archivo `/dev/ttyACM0` por el cual el microcontrolador envía los datos.

3.5. MQTT

Una vez leída cada línea se debe preparar para enviarla al ThingsBoard por MQTT, como se recibe línea a línea y el dato ya trae una palabra clave que identifica el dato solo de debe separar el string en etiqueta y valor, y convertirlo en un elemento estilo JSON, por ejemplo `{label: value}`. Con ese formato se envía a la plataforma IoT, para todo esto se utilizaron más librerías de Python, en específico JSON y `paho.mqtt.client`.

Para la comunicación por medio del protocolo MQTT se deben seguir varios pasos.

1. Especificar los detalles del broker, puerto, tópico y token de acceso del dispositivo creado en el ThingsBoard

```
1 # MQTT broker details
2 broker = "iot.eie.ucr.ac.cr"
3 port = 1883
4 topic = "v1/devices/me/telemetry"
5 access_token = "YI3VpgHK1lc5iCw02JvR"
```

2. Se utiliza el placeholder `on_connect()` de la librería para definir que hacer cuando la conexión es exitosa. No es estrictamente necesario pero es necesario para saber si la conexión fue exitosa. Fue de mucha ayuda para saber que en las primeras versiones del script habían reconexiones constantes con el broker.

```
1 # Callback for connection
2 def on_connect(client, userdata, flags, rc):
3     if rc == 0:
4         print("Connected to ThingsBoard")
5     else:
6         print("Connection failed")
```

3. Se crea una instancia de cliente MQTT, se configura con el token, se utiliza `on_connect` para indicarle lo que debe hacer al momento de que estable conexión con el broker y se da la instrucción de que se conecte.

```
1 # Set up the client
2 client = mqtt.Client()
3 client.username_pw_set(access_token)
4 client.on_connect = on_connect
5 client.connect(broker, port, keepalive=60)
```

4. Se inicia un loop infinito que ejecuta las instrucciones anteriores, además de que dentro de este bucle es donde se lee el puerto serial, se procesan los datos y se envían al dashboard.

```
1 # Start the loop to handle callbacks (blocking mode)
2 client.loop_start()
3
```

```

4 try:
5     while True:
6         # Read line
7         data = ser.readline().decode('utf-8').strip()
8         if data:
9             print(data)
10            data_to_send = data.split(': ')
11            label = data_to_send[0]
12            value = data_to_send[1]
13
14            # Publish data to MQTT broker
15            client.publish(topic, dumps({label: value}), qos=1)
16
17 except KeyboardInterrupt:
18     print("Exiting program")
19
20 finally:
21     client.loop_stop() # Stop the loop when exiting

```

3.6. ThingsBoard

Se utilizó como referencia el tutorial visto en clase para crear un dispositivo en ThingsBoard el cual pueda ser utilizada como cliente suscrito al tópico v1/devices/me/telemetry, en el se publican los datos del giroscopio.



Sismógrafo_Sergio_Pablo			
Device details			
Details Attributes Latest telemetry Alarms Events Relations Audit logs			
Telemetry			
<input type="checkbox"/>	Last update time	Key ↑	Value
<input type="checkbox"/>	2024-10-30 17:29:37	Voltaje	3
<input type="checkbox"/>	2024-10-30 17:29:37	x	36.34
<input type="checkbox"/>	2024-10-30 17:29:37	y	21.58
<input type="checkbox"/>	2024-10-30 17:29:37	z	62.31

Figura 11: Dispositivo cliente ThingsBoard

En la sección details→Check connectivity→MQTT→Linux se puede obtener que el comando para hacer una prueba de conexión es: `mosquitto_pub -d -q 1 -h iot.eie.ucr.ac.cr -p 1883 -t v1/devices/me/telemetry -u "YI3VpgHK1lc5iCw02JvRm" "{temperature:25}"`. De aquí se obtuvieron los datos usados en el script.

Una vez creado el dispositivo se hicieron unos widgets para darle una interpretación a los datos. Se añadió un indicador para la batería y un gráfico de barras verticales para los ejes X, Y y Z, este último es muy intuitivo ya que al mover el giroscopio el movimiento de las barras da a entender que un sismo está ocurriendo. También se añadió una tabla similar a lo que se ve en la figura 11



Figura 12: Widgets en ThingsBoard

4. Conclusiones

- Es posible desarrollar un sismógrafo con un microcontrolador y un giroscopio debido a su alta sensibilidad y se pueden mostrar los datos a usuarios en una interfaz gracias a ThingsBoard.
- La programación de un microcontrolador de 32-bits como el STM32F429ZIT6 es muy complicado, pero la tarea es factible gracias a librerías como libopencm3.
- El IoT se puede llevar a dispositivos que no tenga la posibilidad de conectarse a Internet, en este laboratorio el microcontrolador no tiene conexión a Internet, es a través de la comunicación serial que el microcontrolador se puede comunicar con el Internet.

5. Recomendaciones

- A la hora de resolver un problema grande se recomienda dividir ese problema en tareas pequeñas, esto permite una solución mas eficiente y eficaz que se puede traducir en tiempo ahorrado.
- Se recomienda dedicar suficiente tiempo a la comprensión del protocolo MQTT ya que es una tendencia en IoT, además para utilizarlo en Python es recomendable usar bibliotecas como `paho-mqtt` ya que tal como el microcontrolador, es muy difícil codificar el protocolo desde cero.
- Se recomienda usar widgets en ThingsBoard ya que los datos sin interpretación no son de mucha utilidad. En las imágenes PONER FOTOS DEL DASHBOARD se puede contrastar la diferencia entre los datos que le llegan al ThingsBoard y la visualización a través de widgets.
- Se recomienda evitar leer el puerto serial con mas de un programa o terminal, ya que esto puede generar errores a la hora de leer los datos. Es necesario que solo el script de python este leyendo el puerto serial para que funcione correctamente.

Referencias

- [1] ArminAustin200. libopencm3 lowlevel open-source library for arm cortex mcus. LibOpenCM3. [Online]. Available: <https://github.com/libopencm3>
- [2] AWS. ¿qué es iot (internet de las cosas)? Amazon Web Services. [Online]. Available: <https://aws.amazon.com/es/what-is/iot/#:~:text=El%20t%C3%A9rmino%20IoT%2C%20o%20Internet,como%20entre%20los%20propios%20dispositivos>.
- [3] ThingsBoard. What is thingsboard? ThingsBoard. [Online]. Available: <https://thingsboard.io/docs/getting-started-guides/what-is-thingsboard/>
- [4] HiveMQ. Mqtt mqtt 5 essentials. HiveMQ. [Online]. Available: <https://www.hivemq.com>