Universidad de Costa Rica
Escuela de Ingeniería Eléctrica
IE-0624 Laboratorio de Microcontroladores

Laboratorio #4

Grupo 01

Profesor: MSc. Marco Villalta Fallas

Estudiante: Luis Brenes Campos — **Carné:** C21324
Estudiante: Elizabeth Matamoros — **Carné:** C04652

I SEMESTRE 2025

Índice

1. Introducción

En este laboratorio se elaboró el diseño e implementación de un sismógrafo digital haciendo uso del microcontrolador STM32F429 y apoyándonos en la biblioteca libopencm3 para la gestión de periferia. El sistema integra un sensor giroscópico de tres ejes (X,Y,Z) cuyo muestreo se realiza a través de SPI y un circuito divisor de tensión que permite monitorizar el nivel de batería en un rango de 0 a 9 V sin comprometer la seguridad del ADC de 3.3 V. Adicionalmente, se incorporó un pulsador para activar o desactivar la transmisión de datos por USART/USB, y dos indicadores LED: uno que señala el estado de la comunicación y otro que alerta de batería baja al alcanzar el umbral de 7 V.

Para la supervisión local, se desarrolló una interfaz en pantalla LCD que despliega en tiempo real los valores de los tres ejes del giroscopio, el voltaje de la batería y el estado de enlace. De manera paralela, un script en Python se encarga de leer el puerto serial/USB, procesar la información recibida y publicarla en un dashboard de ThingsBoard, donde se configuraron gráficos y alarmas que facilitan el análisis remoto y la notificación automática en caso de condiciones críticas.

Los resultados obtenidos cumplen con los objetivos planteados: el registro de las oscilaciones fue estable y libre de pérdidas, la alarma de bajo voltaje se activó con precisión al nivel definido, y la transmisión remota mostró continuidad sin interrupciones. El uso de libopencm3 promovió una arquitectura de firmware modular, mientras que el control de versiones con Git garantizó la trazabilidad de cada contribución. En conjunto, este proyecto demuestra la viabilidad del STM32F429 como núcleo de un sistema IoT de adquisición y monitoreo en tiempo real.

2. Nota teórica

2.1. Información general del STM32F429

En esta sección se describirán las características generales del microcontrolador STM32F429ZIT6 con el fin de comprender su arquitectura, capacidades y recursos principales. Se usó como base el material brindado *Hojas de datos – STM32F429 Discovery kit* [1] [2] para extraer información precisa sobre memoria, conectividad, alimentación y otros aspectos relevantes que permitan apreciar cómo este dispositivo gestiona tareas de control y procesamiento en aplicaciones embebidas.

2.1.1. Memoria y Almacenamiento

- **Flash**: 2 MB de memoria interna en banco dual, con soporte de "read-while-write" que permite ejecutar código sin esperas desde Flash.
- **SRAM**:
  - 256 KB de SRAM de propósito general.
  - Además, 4 KB de Backup SRAM (BKPSRAM), accesible en modo VBAT para preservar datos durante apagado.
- **Memoria externa** (opcional): Controlador de memoria externa SDRAM: soporta hasta 64 Mbit (1 Mbit × 16 × 4 bancos), con interfaz síncrona a 80 MHz.
- **Aceleradores de memoria** (ART Accelerator^{TM} y Chrom-ART Accelerator^{TM}): mejoran el acceso a Flash y aceleran operaciones gráficas (llenado de rectángulos, copias con conversión de píxeles).

2.1.2. Comunicación y Conectividad

- **UART/USART**: Hasta 4 USART y 4 UART independientes, cada uno capaz de operar a 11.25 Mbit/s (máximo).
- **SPI/I^2**: 6 SPI que pueden alternar con funciones I^{2}S (audio). Velocidad SPI hasta 45 Mbit/s
- **I^{2}C/SMBus**: 3 I^{2}C con filtrado digital, empleables también como SMBus
- **CAN**: 2 controladores CAN (bxCAN), para redes de alta fiabilidad.
- **USB OTG**:
  - Controlador USB OTG Full-Speed (FS) y High-Speed (HS) con PHY interno (ULPI para HS).
  - Pines DP, DM, ID, VBUS, SOF, etc.
- **Ethernet MAC**: Ethernet 10/100 MAC con soporte IEEE 1588 v2. La PHY se conecta externamente mediante MII/RMII.
- **SDIO/MMC**: Interfaz SDIO para tarjetas SD/MMC con 8 líneas de datos (D0–D7), CMD, CLK.

- **Interfaz de cámara**(DCMI):Soporte para cámara en sensores con señales HSYNC, VSYNC, PCLK, datos de 14 bits.
- **Audio**:
  - 2 DAC de 12 bits y 3 ADC de 12 bits que pueden operar hasta 7.2 MSPS (en modo intercalado).
  - Interfaces I^2S y SAI con soporte TDM (el STM32F429 dispone de PLL de audio dedicado).
- **JTAG/SWD**: Puerto de depuración embebido (ARM SWJ-DP) que combina JTAG y SWD, reutiliza pines TMS/TCK como SWDIO/SWCLK.
- **SPI/I^2S para giroscopio L3GD20** (en placa Discovery):Giroscopio conectado a SPI5 (pines PF7: SCK, PF8: MISO, PF9: MOSI, PD7: CS).

2.1.3. Alimentación y Energía

- **Voltaje de alimentación (VDD/VSSA/VDDA)**: Rango de operación: 1.7 V a 3.6 V; típicamente 3.3 V para la mayoría de aplicaciones.
- **VBAT**: Pin VBAT en 1.65 V a 3.6 V, alimenta RTC y Backup SRAM cuando el resto del sistema está apagado; permite funcionamiento en modo VBAT (¡1 μA de consumo típico).
- **Regulador interno**: Regulador de 1.2 V con escalado de potencia (power-scaling) para reducir consumo en sistemas con batería.
- **Consumo en modos de bajo consumo**:
  - Modo Run: $\tilde{2}60$ μA/MHz a 180 MHz (periféricos apagados).
  - Modo Stop: $\tilde{1}00$ μA típico.
  - RTC en VBAT: ¡1 μA.
- **Cortocircuito de diodos protectores**(en Discovery): Circuito de conmutación automática cuando se alimenta por USB o fuente externa; diodos D1/D2 protegen las líneas de 5 V y 3 V.

2.1.4. Otras Características Relevantes

- **Núcleo**: ARM Cortex-M4 de 32 bits a hasta 180 MHz, con FPU de punto flotante simple y extensión DSP (MAC de ciclo único). 225 DMIPS.
- **Bus de sistema**:
  - Matriz AHB de 7 capas (32 bits) con hasta 10 masters y 8 slaves; ancho de banda elevado para memorias externas.
  - 2 controladores DMA generales, 1 para USB HS, 1 para Ethernet; cada DMA tiene 8 streams con FIFOs .

- **Controlador LCD-TFT embebido**: Soporta resoluciones hasta SVGA (800 × 600) en paralelo RGB de hasta 24 bits y 2 capas con blending; en placa Discovery usa interfaz paralela de 18 bits (LCD_R[7:0], LCD_G[7:0], LCD_B[7:0], HSYNC, VSYNC, PCLK).
- **Control de cache y aceleradores**:
  - L1 I-Cache de 4 KB y D-Cache de 4 KB; L2 Cache de 1 KB.
  - Memoria System SRAM interna de 112 KB aparte de los 256 KB de SRAM principal.
- **Seguridad y supervisión**:
  - Protector de memoria: MPU (Memory Protection Unit).
  - Supervisor de voltaje de alimentación (PVD) con interrupciones.
  - BOR (Brown-Out Reset) configurable.
- **Periféricos adicionales**:
  - WWDG (Window Watchdog), IWDG (Independent Watchdog), RTC con calendario.
  - Sensor de temperatura interna, generador de aleatoriedad (RNG).
  - Interfaz de tarjeta Smart Card (SMCARD), IrDA.

2.1.5. Diagrama de bloques

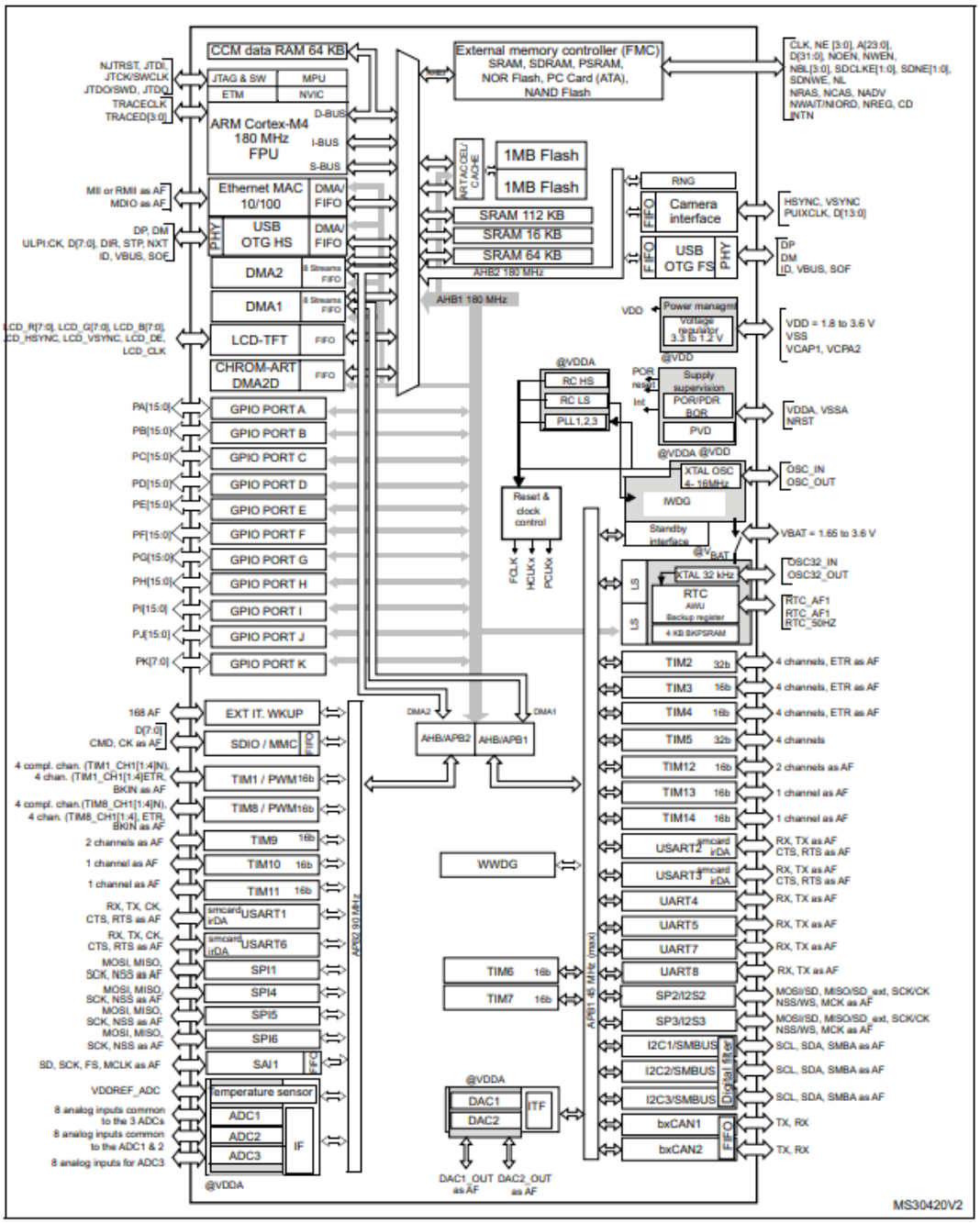

Figura 1: Diagrama de bloques del STM32F429 extraído de [1]

2.1.6. Diagrama de pines

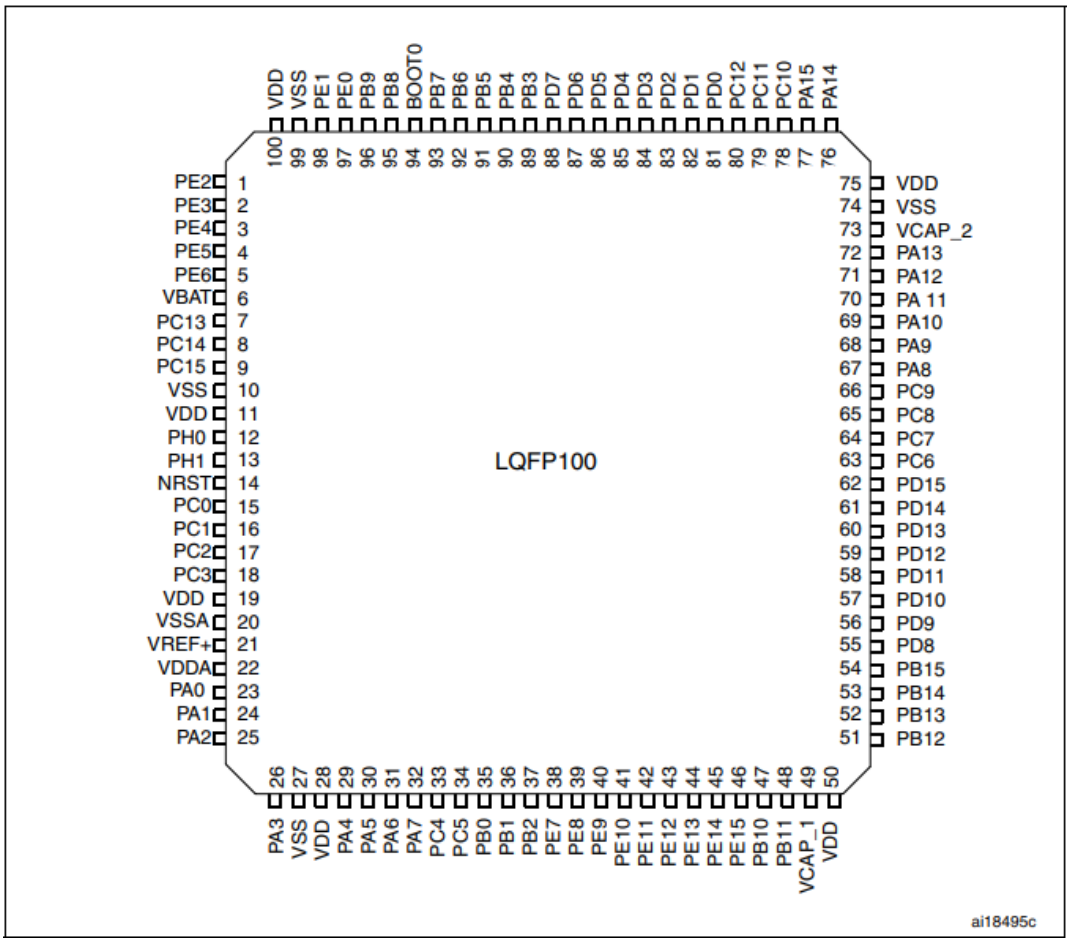

Figura 2: Diagrama de pines del STM32F429 extraído de [2]

2.1.7. Especificaciones eléctricas y operativas

Las características eléctricas se encuentran descritas en el datasheet general [2] para los modelos STM32F427xx STM32F429xx, se pueden ver las características de voltage y las de corriente en las siguientes figuras:

Características de Voltaje

Symbol	Ratings	Min	Max	Unit
V_{DD}–V_{SS}	External main supply voltage (including V_{DDA}, V_{DD} and VBAT)(1)	– 0.3	4.0	V
V_{IN}	Input voltage on FT pins(2)	V_{SS} – 0.3	V_{DD}+4.0	
	Input voltage on TTa pins	V_{SS} – 0.3	4.0	
	Input voltage on any other pin	V_{SS} – 0.3	4.0	
	Input voltage on BOOT0 pin	V_{SS}	9.0	
$\lvert\Delta V_{DDx}\rvert$	Variations between different V_{DD} power pins	-	50	mV
$\lvert V_{SSX} - V_{SS}\rvert$	Variations between all the different ground pins including V_{REF-}	-	50	
$V_{ESD(HBM)}$	Electrostatic discharge voltage (human body model)	see *Section 6.3.15: Absolute maximum ratings (electrical sensitivity)*		

Figura 3: Características de Voltage del STM32F429 extraído de [2]

Características de Corriente

Symbol	Ratings	Max.	Unit
ΣI_{VDD}	Total current into sum of all V_{DD_x} power lines (source)(1)	270	mA
ΣI_{VSS}	Total current out of sum of all V_{SS_x} ground lines (sink)(1)	– 270	
I_{VDD}	Maximum current into each V_{DD_x} power line (source)(1)	100	
I_{VSS}	Maximum current out of each V_{SS_x} ground line (sink)(1)	– 100	
I_{IO}	Output current sunk by any I/O and control pin	25	
	Output current sourced by any I/Os and control pin	– 25	
ΣI_{IO}	Total output current sunk by sum of all I/O and control pins (2)	120	
	Total output current sourced by sum of all I/Os and control pins(2)	– 120	
$I_{INJ(PIN)}$ (3)	Injected current on FT pins (4)	– 5/+0	
	Injected current on NRST and BOOT0 pins (4)		
	Injected current on TTa pins(5)	±5	
$\Sigma I_{INJ(PIN)}$ (5)	Total injected current (sum of all I/O and control pins)(6)	±25	

Figura 4: Características de Corriente del STM32F429 extraído de [2]

2.1.8. Periféricos

GPIO (General Purpose Input/Output)

Los pines GPIO se emplean para controlar los LEDs indicadores y para leer el estado del botón físico que habilita o deshabilita la transmisión de datos.

Los LEDs están configurados como salidas push-pull, lo que permite señalizar visualmente el estado de transmisión mediante parpadeos cuando está habilitada y mostrar la condición de "batería baja" con un encendido continuo o intermitente.

El botón está conectado a un pin GPIO en modo entrada con pull-up interno. Al presionarse, cambia su nivel lógico y provoca que el firmware cambie de estado, habilitando o deshabilitando la comunicación serial.

ADC (Convertidor Analógico-Digital)

El ADC interno de 12 bits se utiliza para medir la tensión resultante del divisor resistivo (9 V aproximadamente 3.3 V). Para ello, se selecciona un canal específico, por ejemplo PA0, que recibe directamente la señal del divisor. Cada conversión digital genera un valor entre 0 y 4095, el cual se traduce a voltaje utilizando como referencia $V < sub > DD < /sub > = 3.3$ V.

Este procedimiento de lectura continua permite determinar en tiempo real el nivel de batería; si el valor medido por el ADC cae por debajo de un umbral predeterminado, se activa automáticamente la lógica de "batería baja".

SPI (Serial Peripheral Interface)

SPI opera en modo maestro para comunicarse con dos dispositivos externos.

El giroscopio L3GD20 es un sensor MEMS de tres ejes conectado a través de las líneas SCK (reloj), MISO y MOSI (datos) y un pin CS (chip-select). Mediante SPI se envían comandos de configuración y se leen los registros que contienen la velocidad angular en los ejes X, Y y Z.

La pantalla TFT ILI9341 es un módulo gráfico a color que comparte las líneas SCK y MOSI, y utiliza pines adicionales DC (data/command), RST (reset) y CS (chip-select). A través de SPI se envían secuencias de comandos y datos para inicializar la pantalla y dibujar texto o gráficos que muestran los valores de los sensores y el nivel de batería.

USART (Universal Synchronous/Asynchronous Receiver Transmitter)

El periférico USART se configura en modo asíncrono para transmitir los datos recogidos (valores del giroscopio y voltaje de batería) hacia un script en Python en el PC.

La comunicación se establece a 115200 bps, con 8 bits de datos, sin paridad y 1 bit de stop. Cada paquete consiste en una cadena formateada que el script procesa y envía a ThingsBoard.

Timers (Temporizadores generales)

Un temporizador interno se utiliza para controlar los tiempos de parpadeo de los LEDs y regular la cadencia de lectura de sensores. Por ejemplo, el Timer 2 puede configurarse con un prescaler y un periodo que generen interrupciones o banderas cada cierto intervalo, como 100 ms o 500 ms. De esta manera se garantiza un muestreo periódico del giroscopio y la actualización de la pantalla sin bloquear el bucle principal.

DMA (Direct Memory Access)

En algunos casos, se emplea el controlador DMA para transferir datos del ADC a memoria sin intervención del CPU, optimizando eficiencia. Sin embargo, su uso es opcional y depende de si se desea reducir la carga de trabajo dentro del bucle principal.

RCC (Reset and Clock Control)

Antes de usar cualquier periférico, se habilitan los relojes correspondientes mediante el módulo RCC. Para ello, se activan las señales de reloj para GPIO, ADC, SPI, USART y Timer en los registros RCC_APB2ENR o RCC_APB1ENR según corresponda. De esta manera, cada bloque funcional recibe su fuente de reloj y puede operar a la frecuencia deseada.

2.2. STM32F429 Discovery Kit

En esta sección se presentarán tres componentes fundamentales que integran la placa STM32F429 Discovery: el sensor giroscópico MEMS L3GD20, la pantalla LCD TFT y la conectividad IoT para transmisión de datos. Primero se detallará el funcionamiento e integración del giroscopio L3GD20; más adelante se describirá la pantalla LCD y, finalmente, se explicará la parte de IoT para enviar la información recogida a una plataforma externa.

2.2.1. Sensor L3GD20/Giroscopio

En esta sección se describe el sensor giroscópico **MEMS L3GD20** integrado en la placa STM32F429 Discovery, con el objetivo de detallar su funcionamiento, prestaciones y la forma en que se conecta e interactúa con el microcontrolador STM32F429ZIT6. La información sobre el L3GD20 se extrajo de su hoja de datos correspondiente [3], complementada con los detalles de integración proporcionados en el manual del Discovery kit [1].

El L3GD20 es un giroscopio de tres ejes basado en tecnología MEMS (Micro-Electro-Mechanical Systems) diseñado para medir velocidades angulares alrededor de los ejes X, Y y Z. Sus características fundamentales incluyen:

- Rangos de medida angular seleccionables en ±250 dps, ±500 dps o ±2000 dps. Cada rango ofrece una sensibilidad distinta (por ejemplo, 8,75 mdps/LSB en ±250 dps).
- Interfaz de comunicación SPI de 4 hilos (modo 0/3) y conexión I^2C, con bus estándar hasta 10 MHz en SPI y hasta 400 kHz en I^2C. En la placa Discovery se emplea SPI (modo 0) para intercambiar datos rápidamente con el STM32F429.
- Consumo de energía típico de 6,1 mA en modo activo y menos de 0,18 μA en modo power-down.
- Rango de tensión de alimentación entre 2,4 V y 3,6 V.

- Frecuencia de salida (Output Data Rate) configurable en 95, 190, 380 o 760 Hz.
- Dos pines de interrupción (INT1, INT2) programables para señalizar datos listos o umbrales alcanzados en el buffer FIFO interno.
- Buffer FIFO de 32 muestras que permite almacenar lecturas sucesivas sin intervención constante del MCU.
- Filtros analógicos y digitales integrados para reducir ruido y mejorar la estabilidad en la medición de velocidades angulares.

Ahora bien en relación con la conexión en la STM32F429 Discovery Kit, según el manual de usuario de la placa [1], el L3GD20 está cableado al periférico SPI5 del STM32F429ZIT6 de la siguiente manera:

- **SPI5_SCK** a PF7
- **SPI5_MISO** a PF8
- **SPI5_MOSI** a PF9
- **Memory CS (Chip Select)** a PD7
- **INT1** (Data Ready) a PE0 (o PE1, configurable)
- **INT2** (FIFO Threshold/Overrun) a PE1 (o PE0, según configuración)
- **VDD / VDD_IO** a 3.3 V de la placa Discovery (proporcionado internamente)
- **GND** a Ground común de la Discovery

Estas conexiones garantizan que el MCU pueda inicializar el L3GD20 mediante SPI, configurar sus registros de control (CTRL1–CTRL5) y leer de forma periódica las salidas angulares de los ejes X, Y y Z.

Por otro lado, para iniciar el L3GD20, el firmware configura primero SPI5 en modo 0 (CPOL = 0, CPHA = 0) con una velocidad adecuada y asigna PF7, PF8 y PF9 como SCK, MISO y MOSI, mientras que PD7 se usa como GPIO para Chip Select. Luego escribe en CTRL_REG1 para activar el modo normal, habilitar los tres ejes y fijar la tasa de muestreo. A continuación, ajusta CTRL_REG4 para seleccionar el rango de medida y habilitar BDU si se requiere coherencia en las lecturas. Con el sensor listo, el firmware lee periódicamente los registros OUT_X_L a OUT_Z_H (ya sea en bucle o mediante interrupción “Data Ready”) y convierte los valores a (°/s). Opcionalmente, puede habilitar el modo FIFO para que el L3GD20 almacene varias muestras antes de que el MCU las lea, reduciendo tráfico en el bus SPI.

Este proceso asegura que se obtengan lecturas continuas y estables de los valores angulares, que luego el firmware puede convertir a unidades físicas (°/s) empleando la sensiblidad correspondiente al rango seleccionado.

Finalmente, es importante destacar que en la STM32F429 Discovery, el L3GD20 se utiliza generalmente para detectar rotaciones o inclinaciones de la placa, proporcionando valores en tiempo real que pueden mostrarse en la pantalla LCD o enviarse por UART a una PC/binario IoT. La interrupción INT1 puede conectarse a PE0 para avisar al STM32 cuando hay datos listos, permitiendo un muestreo eficiente sin sondeos constantes.

2.2.2. Pantalla LCD

En esta sección se describirá la pantalla **LCD TFT** integrada en la placa STM32F429 Discovery, detallando sus principales características, la forma en que se conecta al microcontrolador y las consideraciones de inicialización y configuración. La información proviene de la hoja de datos del controlador ILI9341 [4] y del manual de usuario de la placa Discovery [1].

La pantalla de la placa STM32F429 Discovery es un panel TFT de 2.4 pulgadas con una resolución de 240×320 píxeles y capacidad para mostrar 262 144 colores en formato de 18 bits. Su controlador es el **ILI9341**, el cual admite tanto un modo de interfaz paralelo RGB de 18 bits como un modo SPI de 4 hilos. En la Discovery, se emplea el bus FSMC/SDRAM del STM32F429 para gestionar las 18 líneas de datos paralelos y las señales de control necesarias. El ILI9341 opera a 3.3 V, mientras que la retroiluminación se alimenta a 5 V a través de un transistor integrado en la placa que permite encender y apagar el LED de forma controlada.

La conexión física entre el STM32 y la pantalla contempla varias señales mapeadas a distintos pines GPIO. Las líneas de datos D0–D17 se distribuyen en los puertos D, E y F, configurados por el FSMC como un bus de 18 bits de ancho. La señal HSYNC se conecta a PA4, la señal VSYNC a PG1 y el reloj de píxeles (PCLK) a PG7. La línea de comando/dato (D/C) está asignada al pin A16 del FSMC (internamente gestionado como AO1), mientras que las señales de lectura (NOE) y escritura (NWE) se ubican en PD4 y PD5, respectivamente. El chip select del banco FSMC1 (NE1) está en PD7, y el control de retroiluminación (LED_BL) se realiza mediante PC3, donde un transistor de potencia activa el LED cuando es necesario.

Para poner en marcha el ILI9341, el firmware debe seguir una secuencia de inicialización vía comandos por el bus FSMC antes de transferir datos de imagen. Primero, se aplica un reset al controlador activando la línea RESET (típicamente conectada a PE4) para reiniciar el dispositivo por completo. A continuación, se envía el comando “Sleep Out” (0x11) para sacarlo del modo de suspensión. Después se configura el formato de píxel mediante el comando 0x3A, seguido del valor 0x66 para establecer los 18 bits de color. Posteriormente, se ajustan los registros que controlan la tasa de refresco (comandos 0xB1, 0xB2 y 0xB3) según los valores recomendados por el fabricante. Finalmente, con el comando “Display ON” (0x29) se activa la salida de la pantalla y se puede comenzar a escribir datos de píxeles. Estas operaciones se realizan escribiendo en el registro de comandos del ILI9341 a través del FSMC configurado en modo multiplexado.

Una vez completada la inicialización, el firmware puede actualizar el contenido de la pantalla copiando bloques de píxeles al área de memoria correspondiente. Esto implica delimitar una ventana de dibujo con los comandos “Column Address Set” (0x2A) y “Page Address Set” (0x2B), y luego usar “Memory Write” (0x2C) para enviar los valores de color de cada píxel. Gracias a la rapidez del bus FSMC/SDRAM del STM32F429, la transferencia de datos hacia la TFT se realiza de manera muy eficiente, lo que permite mostrar texto y gráficos con actualizaciones fluidas y sin parpadeos perceptibles.

2.2.3. IoT

Para integrar los datos del STM32F429 con una plataforma IoT (ThingsBoard), el microcontrolador envía periódicamente, a través de su puerto UART, tramas de texto que incluyen los valores medidos (por ejemplo, nivel de batería y giroscopio) en un formato CSV. Un script en Python se encarga de abrir ese puerto serial, leer

cada línea completa y convertirla en un diccionario de pares “clave:valor”. A continuación, dicho script utiliza el cliente MQTT de ThingsBoard basado en Python (TBDeviceMqttClient) para autenticarse en el servidor MQTT de ThingsBoard mediante el token del dispositivo y publicar, en el tópico, un objeto JSON con la telemetría recibida. Este flujo aprovecha la librería oficial de ThingsBoard para Python, que simplifica la conexión y el envío de datos a la plataforma [5].

En el dashboard de ThingsBoard se define un “dispositivo” virtual al que se asocia un token de acceso; cuando el script publica telemetría en el tópico correspondiente, el servidor almacena esos datos y los hace disponibles para su visualización en tiempo real mediante widgets configurables [6].

Con estos componentes, el flujo completo de datos es:

- El STM32F429 envía por UART/USB una trama de texto con valores medidos.
- Un script en Python lee el puerto serial, parsea la cadena y construye un JSON con las lecturas.
- El script utiliza el cliente TBDeviceMqttClient para conectarse a mqtt.thingsboard.cloud:1883 (o la URL de su servidor), autenticarse con el token del dispositivo y publicar la telemetría en v1/devices/me/telemetry.
- ThingsBoard recibe la telemetría y actualiza el dashboard en tiempo real, incluyendo alertas si algún valor supera umbrales definidos.

De esta manera, se consigue un sistema IoT completo que permite supervisar los datos del giroscopio y del sensor de batería del STM32F429 desde cualquier ubicación con acceso a Internet.

2.3. Plataforma ThingsBoard

ThingsBoard es una plataforma IoT de código abierto diseñada para gestionar dispositivos, recolectar y procesar datos, así como visualizar información en dashboards personalizables. Soporta protocolos estándares como MQTT, HTTP y CoAP para la comunicación con los dispositivos, permitiendo tanto despliegues en la nube como en instalaciones propias (on-premises)[7]. A través de su interfaz web, facilita el aprovisionamiento de dispositivos y la creación de relaciones jerárquicas entre ellos (por ejemplo, ciudades, edificios, pisos, sensores) para organizar grandes implementaciones de manera eficiente [9].

En el núcleo de ThingsBoard, el “Transport Layer” recibe los mensajes entrantes desde los dispositivos y los enruta a un sistema de mensajería duradero, garantizando la entrega confiable de telemetría. Posteriormente, el “Core” maneja las API REST y WebSocket para exponer datos en tiempo real, mientras que el “Rule Engine” procesa dichas telemetrías para disparar alarmas o ejecutar flujos de trabajo automatizados según condiciones definidas [8]. Finalmente, los dashboards permiten crear widgets (gauges, gráficos de series de tiempo, indicadores de alerta) que se actualizan dinámicamente con los datos almacenados, brindando una visión clara y accesible del estado de los dispositivos.

2.4. Diseño de circuito

En esta sección se describe en detalle el diseño del divisor de tensión que convierte los 9 V de la batería externa a un nivel seguro de 3.3 V para el microcontrolador STM32F429. La explicación se fundamenta en los valores especificados en el datasheet del STM32F429ZIT6.

El STM32F429ZIT6 opera con una tensión de alimentación en el rango de 1.7 V a 3.6 V según el datasheet [2]. Adicionalmente, los pines de propósito general soportan como máximo ($V < sub > DD < /sub >$) + 0.3 V = 3.6 V para evitar inyección de corriente interna es indispensable que ningún pin de entrada reciba más de 3.3 V en operación normal[2].

Entonces, para asegurar que un pin de entrada analógica no supere 3.3 V, se emplea un divisor resistivo que reduzca los 9 V de la batería a ese nivel.

Ecuación del divisor y cálculo de resistencias

El divisor se modela con la siguiente fórmula, asumiendo que no hay carga externa que genere caída adicional:

$$V_{out} = V_{in} \cdot \frac{R_{bot}}{R_{top} + R2_{bot}} \tag{1}$$

donde:

- V_{out} = 3.3 V
- V_{in} = 9 V
- R_{top} = R1 + R_{pot}
- R_{bot} = R2

Se eligieron los siguientes valores para los componentes:

- **R1** = 9.80 kΩ
- **R2** = 9.94 kΩ
- **Potenciómetro** = 10 kΩ

Entonces, sustituyendo en la ecuación:

$$9 \cdot \frac{9.94k\Omega}{9.80k\Omega + Rpot + 9.94k\Omega} = 3.3 \tag{2}$$

Entonces despejamos R_{pot}:

$$Rpot \approx 7.37k\Omega \tag{3}$$

A continuación se muestra la simulación en SimulIDE que corresponde a ese cálculo:

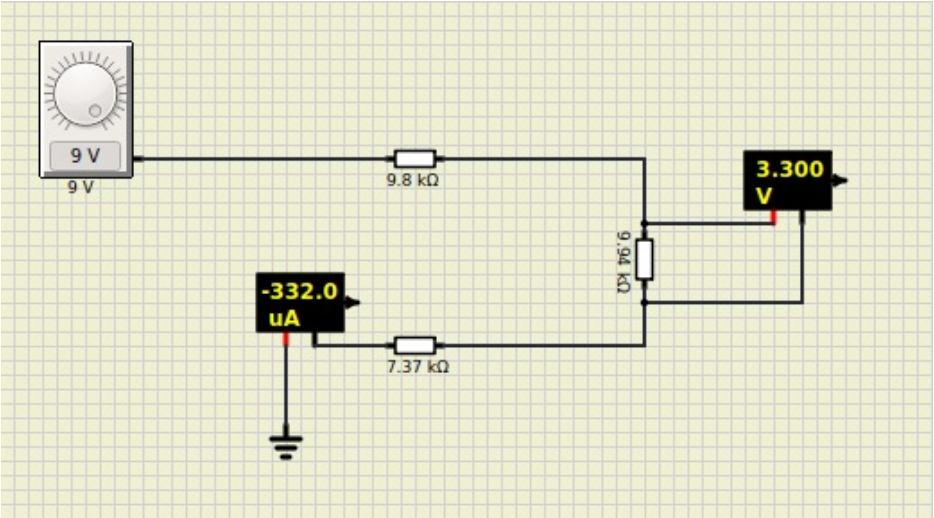

Figura 5: Simulación del divisor de tensión

Montaje físico en protoboard

La batería de 9 V se conecta directamente a los rieles de la protoboard, de modo que la línea superior suministra 9 V y la inferior actúa como tierra.

Luego, se coloca la resistencia fija superior de 9.80 kΩ entre el raíl de 9 V y el pin izquierdo del potenciómetro de 10 kΩ. El potenciómetro se monta de forma que su pin izquierdo reciba esa señal, su pin central sea el nodo de salida (aprox. 3.3 V) y su pin derecho vaya al raíl de tierra.

Finalmente, la resistencia fija inferior de 9.94 kΩ se conecta entre el pin derecho del potenciómetro y el raíl de tierra, cerrando así el divisor. El wiper del potenciómetro (nodo de 3.3 V) se lleva mediante un cable al pin ADC/GPIO del STM32F429, garantizando que el microcontrolador reciba siempre un voltaje seguro de 3.3 V.

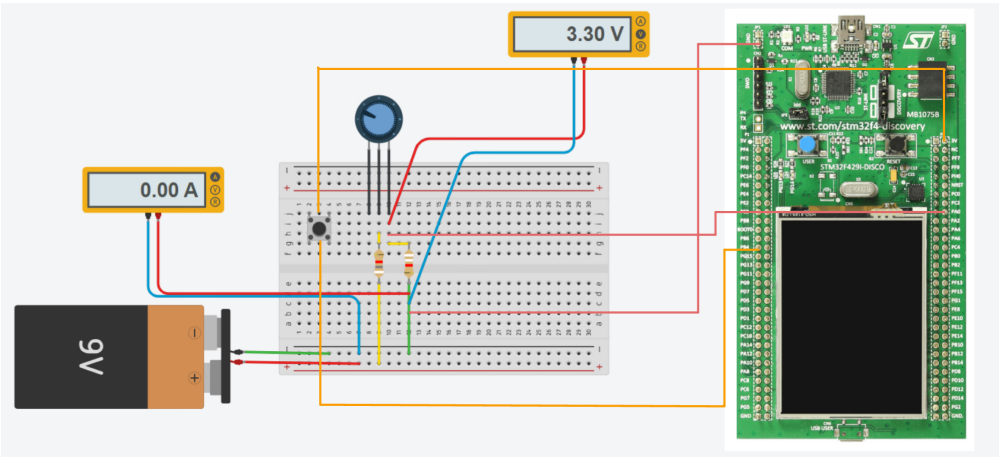

Figura 6: Simulación del diseño del circuito

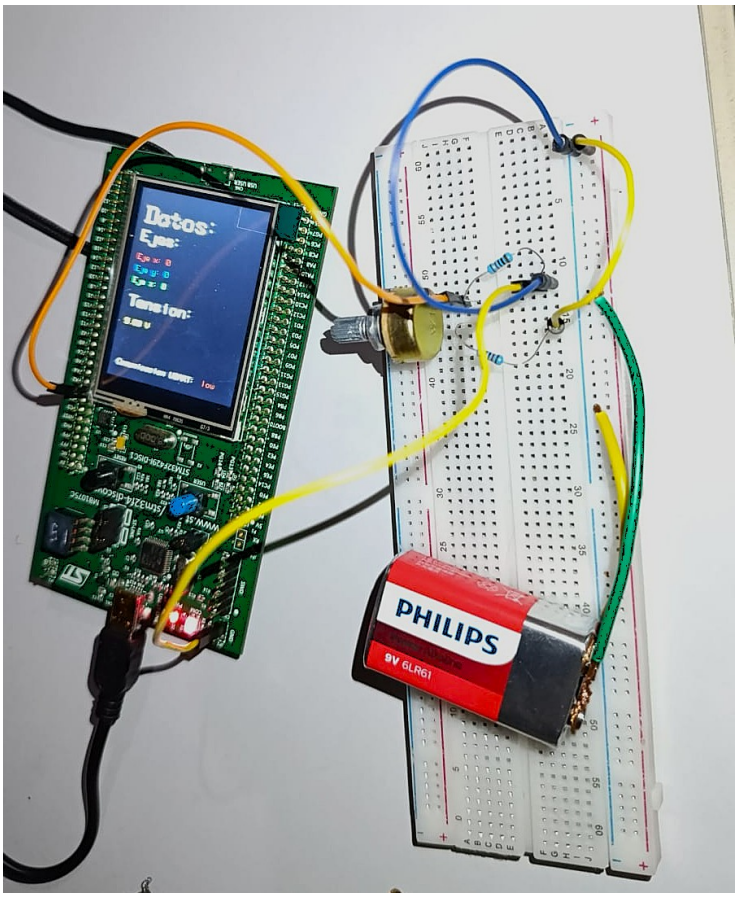

Figura 7: Implementación real del circuito

Validación práctica

A continuación se muestra una prueba para el valor de ajuste de la tensión de entrada para el MCU según el diseño creado:

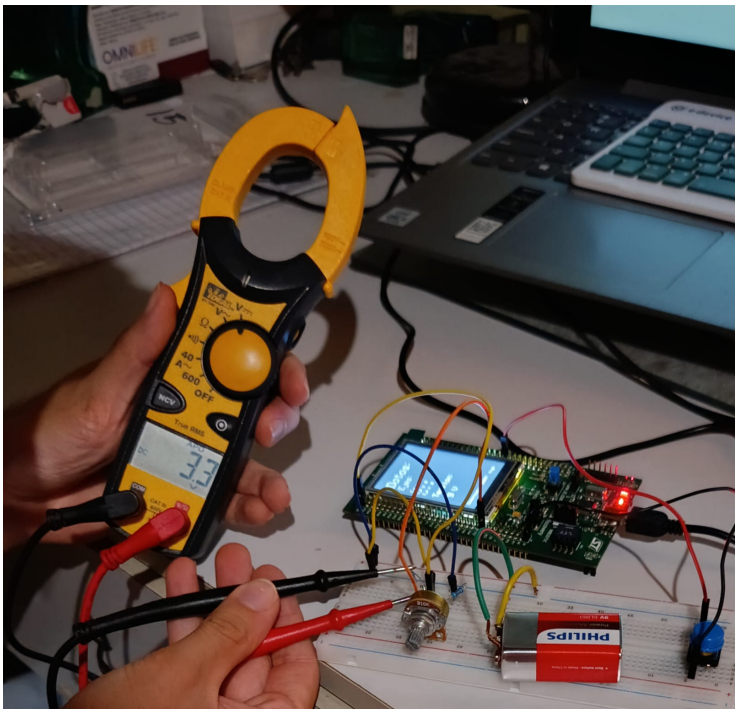

Figura 8: Validación para el valor de ajuste de la tensión

2.5. Lista de componentes y precios

A continuación se presenta una estimación de los costos, en colones costarricenses, de los componentes utilizados en el laboratorio. Los precios se obtuvieron de Mouser Costa Rica y Teltron Costa Rica.

Tabla 1: Lista de componentes y precios estimados en Costa Rica

Componente	**Descripción**	**Precio unitario (CRC)**
STM32F429 Discovery Kit	Kit de desarrollo STM32F429	₡17 644
Resistencia 10 kΩ	Carbono 1 W 5 % (aprox. 9.8 kΩ)	₡100
Resistencia 10 kΩ	Carbono 1 W 5 % (aprox. 9.94 kΩ)	₡100
Potenciómetro 10 kΩ	Eje media caña, B10K	₡275
Batería alcalina 9 V	Alcalina 9 V (supermercado)	₡1400
Botón	—	₡750

2.6. Conceptos/temas del laboratorio

STM32F429 Discovery Kit

El kit STM32F429 Discovery está basado en el microcontrolador STM32F429ZIT6 (Cortex-M4 a 180 MHz con FPU), incluye pantalla TFT LCD de 2.4 (240×320 px), giroscopio L3GD20 de tres ejes, seis LEDs y dos botones.

Dispone de memoria SDRAM de 64 Mbit, conector USB OTG y cabezas de expansión en LQFP144 para GPIO, ADC, DAC y múltiples buses de comunicación. Integra la interfaz ST-LINK/V2-B para depuración, comunicaciones CDC y alimentación por USB o fuente externa (3–5 V)[10].

LibOpenCM3

LibOpenCM3 es una biblioteca en C para controladores ARM Cortex-M, incluyendo la serie STM32F4, que permite desarrollar firmware sin IDEs propietarios. Requiere instalar un toolchain (por ejemplo, gcc-arm-none-eabi) y herramientas ST-Link. Proporciona funciones como rcc_periph_clock_enable, spi_set_master_mode y gpio_set para configurar relojes, GPIO, SPI, USART y otros periféricos, facilitando ejemplos de "blink", ADC, LCD o comunicación serial[10].

Sensor L3GD20 (giroscopio MEMS)

El L3GD20 es un giroscopio MEMS de bajo consumo que mide la velocidad angular en los ejes X, Y y Z, comunicándose por SPI (SPI5 en la placa) o I²C. Para usarlo, se habilita el reloj de SPI5 y los pines SPI, se inicializa el módulo SPI con prescaler, polaridad y fase adecuados, y se configuran registros como CTRL_REG1 (habilita ejes y modo de potencia) y CTRL_REG4 (rango en dps). La lectura se realiza bajando CS, enviando la dirección OR 0x80 y recibiendo el dato de salida[11].

Interfaz SPI (Serial Peripheral Interface)

SPI es un protocolo full-duplex maestro-esclavo que utiliza cuatro líneas: SCLK, CS (SS), MOSI y MISO. En STM32 se configura mediante funciones de LibOpenCM3 para ajustar velocidad (prescaler), polaridad (CPOL) y fase (CPHA). Al iniciar una transferencia, se baja CS, se transmiten datos (o comandos) y, al finalizar, se eleva CS para delimitar cada transacción[11].

Pantalla ILI9341 (TFT LCD)

La ILI9341 es una pantalla TFT a color de 240×320 px con controlador ILI9341 y táctil XPT2046, conectada por SPI (SPI5). Su inicialización implica habilitar el reloj de SPI y los pines de control (CS, DC, RST) y enviar una secuencia de comandos conforme al datasheet para establecer modos de visualización. Con LibOpenCM3 se emplean funciones como spi_send y spi_read para transmitir comandos, dibujar píxeles y mostrar texto en la pantalla integrada[11].

3. Desarrollo/análisis

3.1. Análisis programa

En esta sección se explica cómo funciona el firmware del sismógrafo. Primero se describen las llamadas de inicialización de periféricos, y luego se detalla el bucle principal, donde ocurren las lecturas de sensores, la actualización de indicadores visuales y la transmisión de datos a ThingsBoard.

3.1.1. Diagrama de bloques

En el diagrama se muestra cómo el firmware arranca inicializando periféricos y luego, en cada ciclo, lee el voltaje y el giroscopio, controla el LED de batería baja, envía datos por UART si está habilitado, actualiza la pantalla TFT y gestiona el botón para alternar la transmisión antes de esperar 100 ms y repetir.

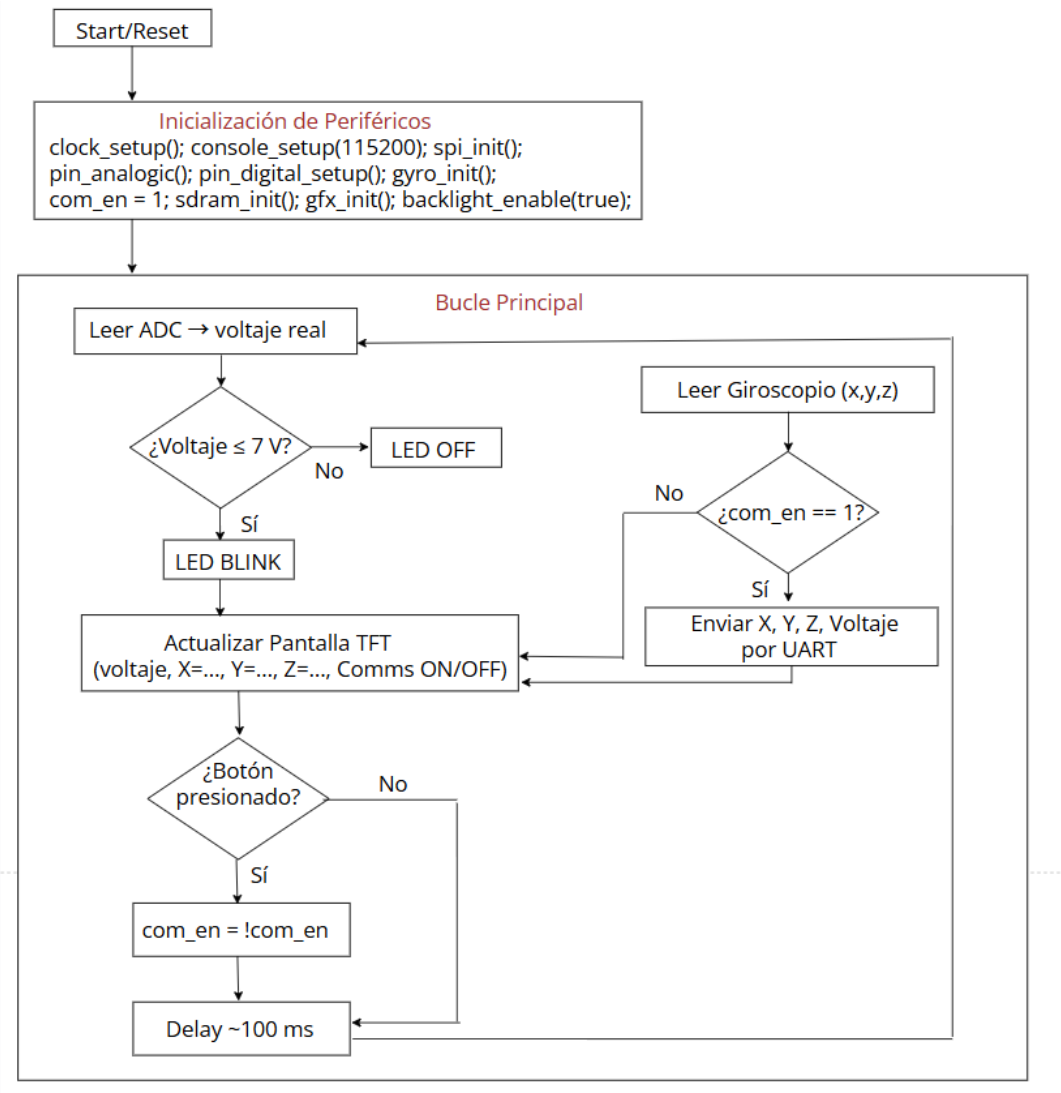

Figura 9: Diagrama de bloques para la lógica del programa

3.1.2. Inicialización de Periféricos

Al inicio de `main.c` se invocan todas las funciones necesarias para configurar cada subsistema, el orden de las mismas se muestra a continuación:

```
clock_setup();
console_setup(115200);
spi_init();
pin_analogic();
pin_digital_setup();
gyro_init();
com_en = 1;
sdram_init();
gfx_init();
backlight_enable(true);
```

La función `clock_setup()` activa el cristal externo (HSE) y configura el PLL para que el microcontrolador opere a 168 MHz. Gracias a esto, UART, SPI y ADC reciben el reloj adecuado. Por otro lado, `console_setup(115200)` deja USART1 listo a 115 200 bps (8N1) utilizando PA9 como TX y PA10 como RX, lo que permite enviar mensajes de texto a un computador. Luego, `spi_init()` configura el periférico SPI5 en modo maestro (CPOL=0, CPHA=0) y asigna PC1 como *chip select* (CS) para el giroscopio L3GD20.

Seguidamente, `pin_analogic()` habilita ADC1 y configura el pin PA0 como entrada analógica, de modo que pueda leer el voltaje proveniente de un divisor que reduce 0–9 V a 0–3.3 V. Inmediatamente, `pin_digital_setup()` habilita GPIOB y configura PB4 como entrada sin pull-up ni pull-down para que actúe como botón que alterna la transmisión de datos. Cuando se llama a `gyro_init()`, el L3GD20 se inicializa en modo de medición continua a ±500 dps; internamente, esta rutina baja CS, programa los registros de control (CTRL_REG1 y CTRL_REG4) y vuelve a subir CS.

Posteriormente, se asigna `com_en = 1`. Esta variable global controla si el firmware envía datos por UART. Luego, con `sdram_init()` se inicializa el controlador FMC y la SDRAM externa (banco 2 mapeado en 0xD0000000). Aunque en esta aplicación no se emplea la SDRAM como búfer de pantalla, su inicialización es obligatoria para cumplir con los requisitos de la placa Discovery. Finalmente, `gfx_init()` invoca `lcd_init()`, que envía la secuencia de comandos necesarios para poner la pantalla TFT (ILI9341) en modo de trabajo; acto seguido, `backlight_enable(true)` enciende la retroiluminación para que el contenido sea visible.

3.1.3. Bucle Principal

Una vez completada la configuración de periféricos, el firmware entra en un bucle infinito (`while (1)`) que se ejecuta aproximadamente cada 100 ms. En cada iteración se repiten las siguientes tareas:

Lectura del Voltaje de Batería

```
uint16_t raw = read_adc_naiive(0);
float volts_f = (raw * 3.3f) / 4095.0f;
```

```
float volts_real = volts_f * ((9940.0f + 9800.0f + 7370.0f) / 9940.0f);
if (volts_real <= 7.0f)
    set_led(true);
else
    set_led(false);
```

La llamada a `read_adc_naiive(0)` devuelve un valor entre 0 y 4095 del ADC1 (canal 0). Ese valor se convierte primero a un voltaje aproximado entre 0 y 3.3 V, y luego se ajusta al rango real 0–9 V mediante el factor del divisor de resistencias (9940 Ω, 9800 Ω y 7370 Ω). Si el voltaje real es menor o igual a 7 V, el LED de "batería baja" parpadea alternando llamadas a `set_led(true)` y `set_led(false)` en cada iteración. En caso contrario, el LED permanece apagado.

Lectura del Giroscopio

```
int32_t x, y, z;
gyro_read_xyz(&x, &y, &z);
```

Al invocar `gyro_read_xyz(&x,&y,&z)`, el firmware baja la línea CS del L3GD20 y lee en modo *burst* los registros de salida X_L/H, Y_L/H y Z_L/H. Cada par de bytes se combina en un entero con signo (16 bits), luego se multiplica por la constante `L3GD20_SENS_500` (0.0175) para convertirlo a grados por segundo. De esta forma, las variables `x`, `y` y `z` contienen la velocidad angular en cada eje.

Transmisión por UART para ThingsBoard

```
if (com_en) {
    print_decimal(x); console_puts("\t");
    print_decimal(y); console_puts("\t");
    print_decimal(z); console_puts("\t");
    print_decimal((int)volts_real); console_puts("\n");
}
```

Si `com_en == 1`, el firmware envía por USART1 cuatro valores separados por tabuladores: los tres ejes del giroscopio (X, Y, Z) y el voltaje de la batería (convertido a entero). Un script Python, ejecutándose en la PC a 115 200 bps, lee cada línea (por ejemplo, `''12=538''`), divide la cadena por "=" y publica esos valores mediante MQTT en ThingsBoard. De este modo, el dashboard refleja en tiempo real tanto el voltaje de la batería como los valores angulares del giroscopio.

Actualización de la Pantalla TFT

```
char buf[32];
gfx_setTextColor(LCD_WHITE, LCD_BLACK);
sprintf(buf, "Bat: %.2f V", volts_real);
gfx_drawString(0, 0, buf);
sprintf(buf, "X=%ld Y=%ld Z=%ld", x, y, z);
```

```
gfx_drawString(0, 16, buf);
sprintf(buf, "Comms: %s", com_en ? "ON" : "OFF");
gfx_drawString(0, 32, buf);
```

Cada llamada a `sprintf` genera una línea de texto que se dibuja en la pantalla usando `gfx_drawString()`. La primera fila muestra "`Bat: X.XX V`", la segunda "`X=... Y=... Z=...`" y la tercera "`Comms: ON`" o "`Comms: OFF`" según el valor de `com_en`. La función `gfx_drawString` recorre cada carácter, consulta su patrón en `font-7x12.c` y dibuja píxel a píxel con `lcd_draw_pixel()` (en `lcd-spi.c`). Si no se limpia la zona antes de escribir, pueden quedar restos de texto anteriores, por lo que conviene usar `gfx_fillRect()` o refrescar toda la pantalla antes de reescribir.

Detección de Botón para Alternar `com_en`

```
if ((GPIOB_IDR & GPIO4) == 0) {
    com_en = !com_en;
}
```

El pin PB4, configurado con `pin_digital_setup()`, detecta el estado del botón. Si `GPIOB_IDR & GPIO4` es cero (botón presionado), se invierte `com_en`. Al cambiar esta bandera, se suspende o reanuda la transmisión por UART y, en la siguiente iteración, el texto de "Comms" en pantalla cambia entre "ON" y "OFF". El rebote mecánico (debounce) se gestiona de forma implícita al esperar el siguiente ciclo del bucle.

Retardo de Muestreo

```
for (volatile int i = 0; i < 80000; i++) __asm__("nop");
msleep(100);
```

Este bucle de instrucciones NOP (aproximadamente 80 000 iteraciones) junto con la llamada a `msleep(100)` generan un retardo total de cerca de 100 ms. De esta manera, la frecuencia de muestreo queda alrededor de 10 Hz, suficiente para capturar variaciones de un sismógrafo de laboratorio.

3.1.4. Organización de Módulos

Para facilitar la comprensión, a continuación se agrupan los archivos según su responsabilidad principal:

- **Reloj** (`clock.c`/`clock.h`): `clock_setup()` configura el sistema a 168 MHz.
- **Consola UART** (`console.c`/`console.h`): `console_setup(115200)` inicializa USART1; `print_decimal()`, `console_putc()` y `console_puts()` envían texto.
- **Giroscopio** (`devices.c`/`devices.h`):
  - `spi_init()` prepara SPI5 para L3GD20.
  - `gyro_init()` configura el sensor.
  - `gyro_read_xyz()` lee los valores crudos y los convierte a dps.
  - Constantes relevantes: `GYR_OUT_X_L`, `GYRO_RNW_BIT`, `L3GD20_SENS_500`.

- **ADC y Divisor de Voltaje** (`devices.c`/`devices.h`):
  - `pin_analogic()` configura PA0 como entrada analógica.
  - `read_adc_naiive()` devuelve un valor 0–4095.
  - En `main.c` se convierte raw → 0–3.3 V → 0–9 V.
- **GPIO Digital** (`devices.c`/`devices.h`):
  - `pin_digital_setup()` configura PB4 para el botón.
  - `set_led(bool)` controla el LED en PG13.
  - `extern uint8_t com_en` habilita o inhabilita el envío por UART.
- **SDRAM** (`sdram.c`/`sdram.h`): `sdram_init()` inicializa FMC y la SDRAM externa (no se usa como búfer activo).
- **Gráficos y Pantalla TFT** (`gfx.c`/`gfx.h`, `lcd-spi.c`/`lcd-spi.h`, `font-7x12.c`):
  - `gfx_init()` invoca `lcd_init()` para la pantalla ILI9341.
  - `gfx_drawString()` dibuja texto punto a punto usando la fuente de 7×12 píxeles.
  - `backlight_enable(true)` enciende la retroiluminación.
- **main.c**: Orquesta la inicialización de todos los módulos y, en el bucle infinito, lee sensores, actualiza indicadores, controla LEDs y envía datos a ThingsBoard.

3.1.5. Flujo Completo de `main.c`

1. **Inicialización:**
   - `clock_setup()`
   - `console_setup(115200)`
   - `spi_init()`
   - `pin_analogic()`
   - `pin_digital_setup()`
   - `gyro_init()`
   - `com_en = 1`
   - `sdram_init()`
   - `gfx_init()`
   - `backlight_enable(true)`
2. **Bucle Infinito:**
   a) Leer ADC, convertir a voltaje real y controlar LED de “batería baja”.
   b) Leer giroscopio (`gyro_read_xyz()`), obtener X, Y, Z en dps.

c) Si `com_en == 1`, enviar "X[TAB]Y[TAB]Z[TAB]Voltaje" a ThingsBoard.

d) Actualizar pantalla TFT con "Bat: X.XX V", "X=... Y=... Z=..." y "Comms: ON/OFF".

e) Leer PB4; si está presionado, invertir `com_en` (pausa breve para debounce).

f) Retardo de $\tilde{1}$00 ms: bucle de NOP + `msleep(100)`.

3.1.6. Implementación en Python para ThingsBoard

Además del firmware en el STM32, se desarrolló otro código en Python cuya finalidad es leer datos del puerto serie (USB) y enviarlos a un dashboard en la plataforma IoT ThingsBoard. Este script está organizado en una clase principal llamada `IOTClient`, que agrupa todas las funciones necesarias para: establecer la conexión por USB al sismógrafo, convertir lecturas a formato JSON y publicar dichas lecturas mediante MQTT en ThingsBoard.

La estructura general del código aprovecha cuatro módulos clave:

- `time`: permite introducir pausas controladas entre cada envío de datos.
- `json`: convierte la información de sensores (giroscopio y batería) en cadenas JSON válidas.
- `serial`: gestiona la comunicación vía puerto serie (por ejemplo, `/dev/ttyACM0` a 115200 bps).
- `paho.mqtt.client`: proporciona la interfaz para conectarse a un broker MQTT y publicar mensajes.

La clase `IOTClient` se encarga de tres tareas fundamentales. En primer lugar, establece la configuración de la conexión serial: al crear una instancia de `IOTClient`, se abre el puerto serie para que las lecturas que envía el sismógrafo sean recibidas correctamente por el programa Python.

En segundo lugar, la misma clase configura el cliente MQTT. Para ello, utiliza la biblioteca `paho.mqtt.client`, se autentica con el token que proporciona ThingsBoard y se conecta al servidor. De este modo, queda preparado para publicar, en el tópico `v1/devices/me/telemetry`, cada mensaje JSON que represente las mediciones del giroscopio y el voltaje de batería.

Finalmente, el método `run()` de `IOTClient` implementa un bucle continuo que realiza tres pasos en cada iteración: primero, lee una línea del puerto serie con `ser.readline()`, la decodifica y la separa por tabuladores para extraer los valores `x`, `y`, `z` y `voltaje`. A continuación, construye un diccionario Python con esos datos y lo convierte a formato JSON usando `json.dumps()`. Por último, publica el JSON en ThingsBoard mediante `client.publish(...)` y llama a `time.sleep(t)` para esperar un intervalo de tiempo fijo antes de repetir el proceso.

De manera más concisa, el flujo en `IOTClient` podría resumirse así:

```
class IOTClient:
    def __init__(self, port, baud, token):
        # Abre el puerto serie y configura MQTT
    def run(self):
        while True:
            line = ser.readline().decode().strip()
            x, y, z, v = line.split('\t')
            data = {"gyro_x": int(x), "gyro_y": int(y),
                    "gyro_z": int(z), "battery": int(v)}
```

```
        client.publish('v1/devices/me/telemetry', json.dumps(data))
        time.sleep(0.1)
```

Cuando se ejecuta el script directamente (`if __name__ == "__main__":`), se crea una instancia de `IOTClient` con los parámetros deseados (puerto, baudios y token) y se llama a `run()`. Así, la transmisión de datos desde el sismógrafo al servidor MQTT de ThingsBoard es continua y automática.

3.1.7. Configuración en ThingsBoard

Una vez el script Python está en funcionamiento, en ThingsBoard se utiliza una cuenta para:

1. **Registrar el dispositivo:** Ingresar el token generado por ThingsBoard en el script Python para autenticar la publicación de telemetría.
2. **Crear widgets en el dashboard:**
   - Un gráfico de línea para cada eje del giroscopio (X, Y y Z), mostrando valores en tiempo real.
   - Un indicador numérico o gauge para el voltaje de batería.
   - Opcionalmente, un widget de alarma que se active cuando el voltaje caiga por debajo de cierto umbral.
3. **Configurar el refresco:** Establecer la frecuencia de actualización de los widgets de acuerdo al intervalo de publicación (aprox. 10 lecturas por segundo).

Con esta integración, el dashboard de ThingsBoard presenta en tiempo real las lecturas del giroscopio y el nivel de batería, permitiendo monitorizar el comportamiento del sismógrafo desde cualquier dispositivo con acceso a Internet.

3.1.8. Resultados en el ThingsBoard

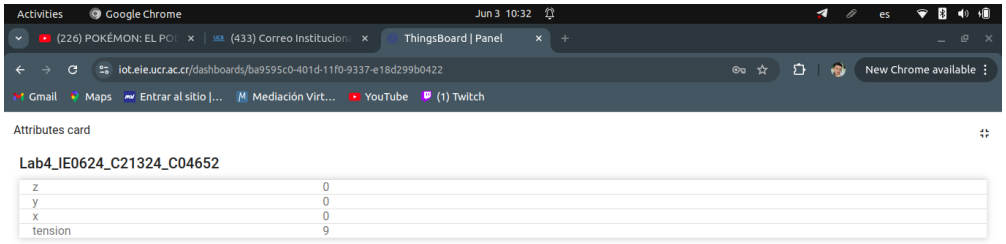

Figura 10: ThingsBoard sin movimiento del STM

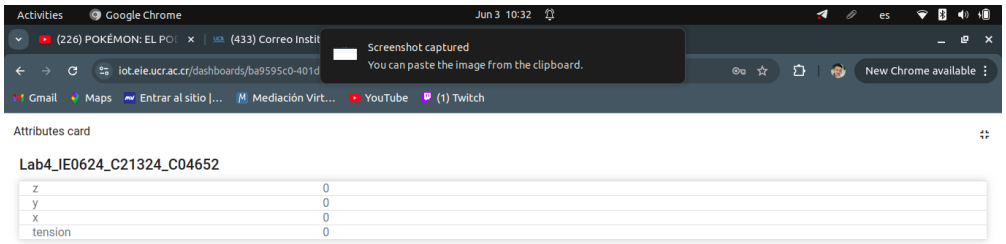

Figura 11: ThingsBoard con movimiento del STM

3.2. Análisis electrónico

3.2.1. Funcionamiento del giroscopio L3GD20

El L3GD20 es un sensor MEMS de tres ejes que mide rotación angular alrededor de los ejes X, Y y Z. Internamente, utiliza un conjunto de estructuras resonantes piezoeléctricas que se deforman cuando el dispositivo gira, generando cargas proporcionales a la velocidad angular. Para nuestro diseño, el giroscopio se alimenta con 3.3 V (conectado a VDD) y sus salidas digitales se comunican por SPI. El pin CS (chip select) está conectado a PC1 en modo GPIO, lo que permite al microcontrolador habilitar o deshabilitar el sensor según sea necesario.

Al iniciar, el firmware baja CS para tomar el control del L3GD20 y programa el registro CTRL_REG1 (direccion 0x20) con el valor que enciende el dispositivo y activa los ejes X, Y y Z. A continuación, vuelve a subir CS y baja de nuevo para escribir en CTRL_REG4 (dirección 0x23), definiendo la escala de medición a ±500 dps (con la constante de sensibilidad 0.0175 dps/LSB). Una vez configurado, el sensor produce datos en format 16 bits en los registros OUT_X_L/H, OUT_Y_L/H y OUT_Z_L/H.

Durante cada lectura, el firmware baja CS, envía por SPI el comando “dirección de OUT_X_L — bit de lectura — bit de auto-incremento” para activar un burst de lectura, y luego recibe seis bytes consecutivos. Estos seis bytes se agrupan en tres valores “low+high” de 16 bits con signo, que se multiplican por 0.0175 para obtener la velocidad angular en grados por segundo. Finalmente, se sube CS y se procesan dichas lecturas en el micro.

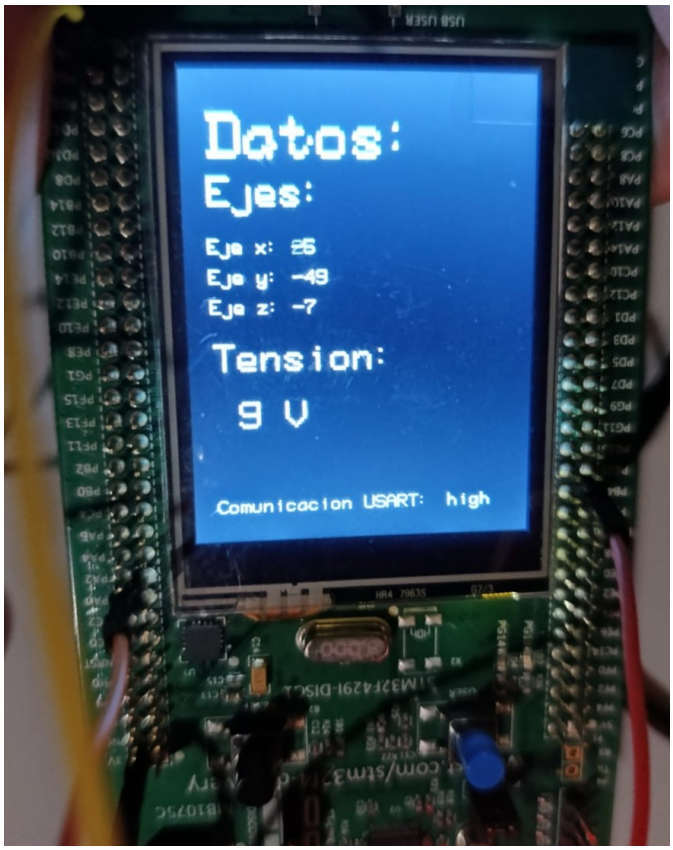

Figura 12: Evidencia del funcionamiento del giroscopio

3.2.2. Visualización en la pantalla TFT (ILI9341)

La pantalla TFT ILI9341 funciona con un controlador que recibe comandos y datos vía interfaz SPI a frecuencias típicas de hasta 20 MHz. En nuestro diseño, se utiliza SPI5 del STM32F429 para comunicar con la TFT, con pines SCK (PF7), MOSI (PF9) y MISO desconectado (no se usa). El pin CS de la pantalla está asignado a PC2, el pin DC (data/command) a PD13 y el pin RESET a PD14. Además, la retroiluminación se controla mediante PD12.

Durante la inicialización, el firmware baja RESET para reiniciar el controlador, espera unos milisegundos y lo sube. A continuación, envía la secuencia de configuración de registros (Frame Rate Control, Power Control, Gamma, Memory Access) solicitada por la hoja de datos del ILI9341. Cada comando se transmite con DC a 0 y cada dato con DC a 1, precedido de la señal de chip select (CS baja). Una vez concluida la secuencia, la pantalla queda en modo de escritura en memoria de video, permitiendo dibujar píxeles.

Para mostrar texto, se usa una fuente de 7×12 px almacenada en `font-7x12.c`. El firmware, al llamar a `gfx_drawString(x, y, texto)`, recorre cada carácter, obtiene su bitmap correspondiente y lo envía como secuencia de píxeles RGB565 a través de SPI (usando `lcd_drawPixel()`). Antes de cada actualización, se fija el área de dibujo (rectángulo) y se envían los datos de color. La retroiluminación, controlada por PD12, se deja siempre encendida para garantizar visibilidad. Gracias a este esquema, la pantalla muestra simultáneamente el voltaje de batería, los valores X/Y/Z del giroscopio y el estado “Comms: ON/OFF”.

Figura 13: Evidencia del funcionamiento del pantalla

3.2.3. Indicadores LED y botón de habilitación

En la placa STM32F429 Discovery hay varios LEDs integrados (por ejemplo, PG13). En nuestro proyecto, PG13 se utiliza como indicador de "batería baja" o de "comms activas", según se programe. El pin PG13 se configura como salida push-pull en 3.3 V. Para encender el LED, el micro escribe un 1 en el registro BSRR correspondiente; para apagarlo, escribe un 0 en el registro BRR. Cuando el voltaje de batería cae por debajo de 7 V, el firmware alterna PG13 ON/OFF en cada iteración ($\tilde{1}$00 ms) para indicar alerta.

El botón de habilitación de comunicaciones está conectado a PB4, configurado como entrada digital sin pull-up ni pull-down. Al leer `GPIOB_IDR & GPIO4`, si se detecta nivel bajo (botón presionado), el firmware invierte la bandera `com_en`. Cuando `com_en` está en 1, el GPIO PB4 no cambia el estado de transmisión; al cambiar a 0, se interrumpe el envío por UART y el LED de "comms" (PG13) se apaga. Para evitar rebotes, se deja el tiempo de retardo natural entre iteraciones ($\tilde{1}$00 ms) sin necesidad de un circuito de debouncing adicional.

Cada uno de estos bloques electrónicos (giroscopio, pantalla TFT, LEDs y botón) se ilustra mediante la imagen correspondiente, mostrando conexiones de pines, resistencias de polarización y rutas de señal.

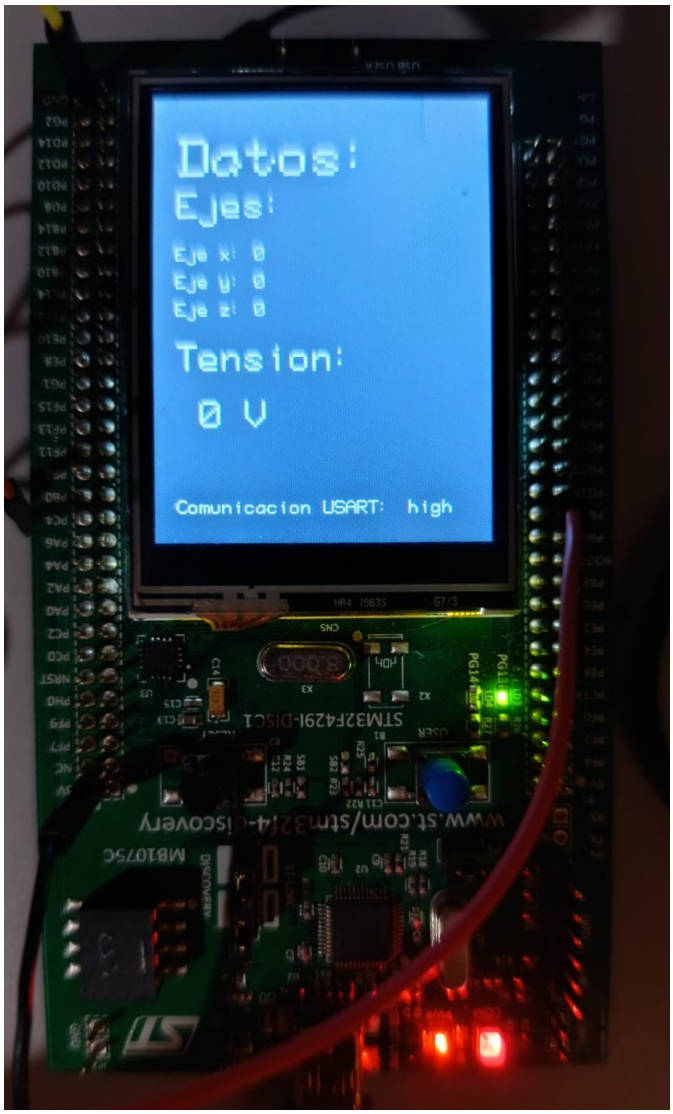

Figura 14: Evidencia del funcionamiento del los LED's

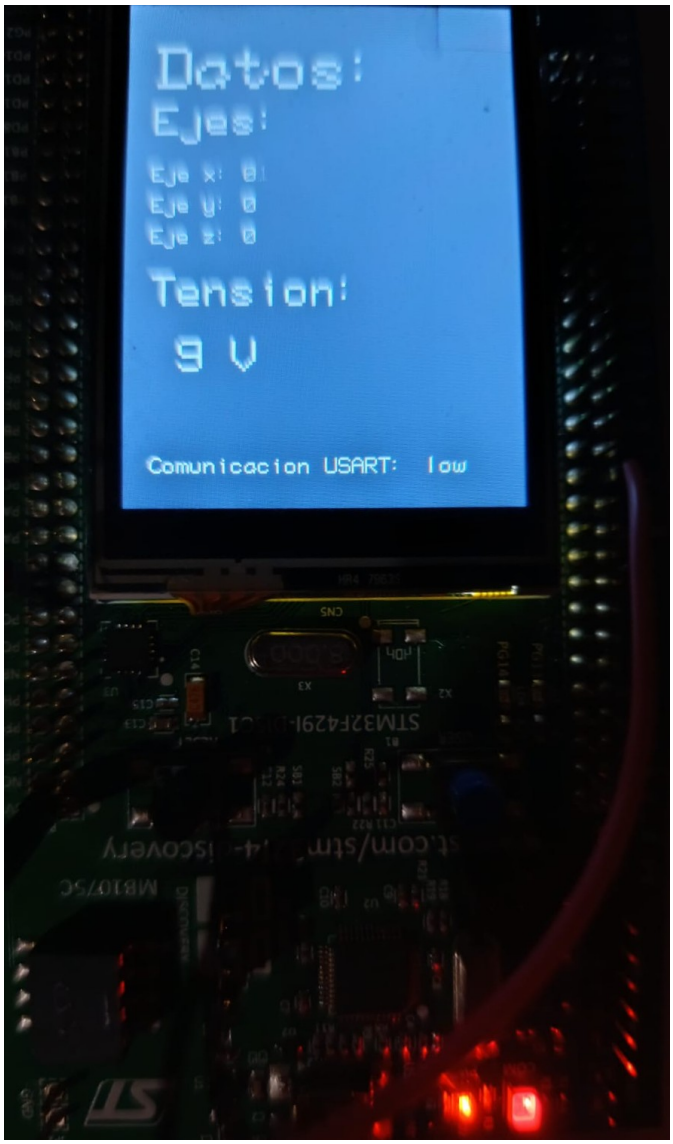

Figura 15: Evidencia del funcionamiento del los LED's

4. Git y enlace a video para comprobación de funcionamiento

El desarrollo completo del laboratorio, que incluye el código fuente, las simulaciones y la documentación del laboratorio, se encuentra disponible en el repositorio oficial del curso en Git UCR, en la carpeta correspondiente al Laboratorio #4 `https://git.ucr.ac.cr/laboratorio_microcontroladores_i2025/laboratorio_4`.

Además, en el siguiente enlace se encuentra un video que comprueba el funcionamiento del sismógrafo `https://youtube.com/shorts/h4fl7fI-_S4?feature=share`.

5. Conclusiones y recomendaciones

5.1. Conclusiones

- El STM32F429 Discovery Kit integrado con libopencm3 permitió implementar con éxito el sismógrafo digital.
- La conexión a ThingsBoard mediante un script en Python funcionó correctamente, garantizando el envío en tiempo real de datos.
- Los periféricos del STM32F429 (SPI, ADC, GPIO, USART, timers) demostraron ser versátiles para lectura de sensores y control de indicadores.
- El giroscopio L3GD20 y la pantalla ILI9341 validaron que se puede monitorear y mostrar datos de forma efectiva usando hardware accesible.
- La combinación de hardware de bajo costo y software libre (libopencm3, Python) permitió crear un sistema de monitoreo eficiente y asequible.

5.2. Recomendaciones

- Probar distintos valores de resistencias o emplear un regulador de 3.3 V para asegurar lecturas estables en el ADC.
- Implementar un filtrado sencillo (media móvil o promedio) en las lecturas del giroscopio para reducir el ruido.
- Considerar el uso de un buffer o regulador de 3.3 V para alimentar cargas adicionales sin afectar la tensión de salida.
- Basarse en ejemplos de libopencm3-examples para configurar periféricos (SPI, ADC, USART) y organizar el código en ese entorno.

Referencias

[1] STMicroelectronics. *Discovery Kit for STM32F429/439 Lines User Manual* (UM1670). STMicroelectronics, 2013.

[2] STMicroelectronics. *STM32F429ZIT6 Datasheet* (DB0765). STMicroelectronics, 2015.

[3] STMicroelectronics. *L3GD20 MEMS Gyroscope Datasheet*. STMicroelectronics, agosto 2013.

[4] STMicroelectronics. *ILI9341 TFT LCD Driver Datasheet*. STMicroelectronics, 2011.

[5] ThingsBoard Community. *Python Client SDK – TBDeviceMqttClient Documentation*. ThingsBoard, 2023. (URL: `https://thingsboard.io/docs/reference/python-client-sdk/`)

[6] Sergii Gnodtsev. *Using ThingsBoard and Mbed to Read Sensor Data from STM32 B-L475E-IOT01A*. Foro en ForGge, 2018. (URL: `https://forgge.github.io/B-L475E-IOT01A-thingsboard-read-temperature-and-humidity-using-mbed.html`)

[7] ThingsBoard Community Edition. *What is ThingsBoard?* ThingsBoard.io, 2024. (URL: `https://thingsboard.io/docs/getting-started-guides/what-is-thingsboard/`)

[8] ThingsBoard Community Edition. *ThingsBoard Transports and Core Architecture*. ThingsBoard.io, 2024. (URL: `https://thingsboard.io/docs/reference/`)

[9] Dusun IoT. *What is ThingsBoard? Architecture, Working Principle, and Uses*. DusunIoT.com, 2022. (URL: `https://www.dusuniot.com/blog/a-brief-guide-and-description-of-thingsboard/`)

[10] M. Villalta Fallas. *STM32F429 Discovery Kit y LibOpenCM3*. Presentación oficial de laboratorio, Universidad de Costa Rica, 2025.

[11] M. Villalta Fallas. *Integración L3GD20 e ILI9341 con STM32F429*. Presentación oficial de laboratorio, Universidad de Costa Rica, 2025.