Universidad de Costa Rica

Escuela de Ingeniería Eléctrica

IE0624 - Laboratorio de Microcontroladores

Informe del Laboratorio 4

STM32: GPIO, ADC, timer, comunicaciones, Iot

Profesor

Marco Villalta Fallas

Jose Daniel Blanco Solis, B71137

III 2024

18 de Febrero de 2025

Índice

Resumen

Este informe muestra el desarrollo e implementación de un sistema basado en el microcontrolador STM32F429 Discovery kit, con el objetivo de monitorear y procesar datos provenientes de sensores, se describen los fundamentos teóricos, el diseño del sistema, la programación del firmware y la integración con una plataforma IoT para la visualización remota de los datos, se documentan los procedimientos de configuración del hardware, la comunicación entre módulos y la validación del sistema, finalmente, se presentan los resultados obtenidos y las conclusiones derivadas del proceso de desarrollo.

1. Introducción

El objetivo principal de este laboratorio es desarrollar un sistema que integra sensores, procesamiento de señales y comunicación con una plataforma IoT, con lo que se obtiene una implementación que permite la captura y análisis de datos de manera eficiente, al comunicar los resultados en tiempo real para su posterior análisis.

Para este caso se implementa un sistema de detección de pendientes utilizando un STM32F429 Discovery kit, este sistema emplea sensores para tomar la información del giroscopio, temperatura y nivel de batería, una pantalla LCD para visualizar los valores y un canal de comunicación serial que permite la transferencia de datos a un dashboard de una plataforma Iot (thingsboard).

2. Nota teórica

2.1. STM32F429

El microcontrolador STM32F429ZIT6 es un dispositivo basado en el núcleo ARM Cortex-M4, opera con un rango de tensión de alimentación de 1,8 V a 3,6 V dependiendo del modo los pines de entrada y salida pueden suministrar o consumir hasta 25 mA cada uno, también incorpora una unidad de punto flotante (FPU), cuenta con una memoria de 2 MB de Flash y 256 KB de SRAM y además, incorpora un controlador LCD-TFT, lo que facilita la integración con pantallas gráficas sin necesidad de hardware adicional.

En cuanto a sus capacidades de conversión analógica, dispone de tres convertidores analógico-digitales (ADCs) de 12 bits y dos convertidores digital-analógicos (DACs) de 12 bits, su sistema de temporización cuenta con 17 temporizadores integrados de los cuales 12 son de 16 bits y 2 de 32 bits, todos operan hasta 180 MHz, posee soporte para modos como Input Capture, Output Compare y PWM, soporta interfaces serial wire debug (SWD) y JTAG.

En términos de conectividad, ofrece 168 pines de entrada y salida (I/O) todos con capacidad de generar interrupciones, incorpora 21 interfaces de comunicación entre las que se tienen I2C, USART, SPI, SAI y CAN, también cuenta con una interfaz avanzada USB 2.0 para aplicaciones que implican transferencia de datos.

2.2. Discovery Kit

El STM32F429 Discovery Kit es una plataforma de desarrollo diseñada para evaluar y crear aplicaciones utilizando el microcontrolador STM32F429ZIT6, ofrece una amplia variedad de periféricos y recursos que facilitan la creación de prototipos y la realización de pruebas en sistemas, su diseño integra una pantalla QVGA TFT LCD de 2.4", proporcionando una interfaz gráfica para aplicaciones con visualización en tiempo real, cuenta con una interfaz USB OTG con conector Micro-AB lo que permite la comunicación con dispositivos externos, también incluye una memoria SDRAM de 64 Mbit para un rendimiento adecuado en aplicaciones exigentes.

El sistema incorpora elementos de control y depuración, cuenta con seis LEDs indicadores cada uno con una función específica: LD1 para comunicaciones USB, LD2 para indicar el estado de encendido a 3,3 V, LD3 y LD4 para uso del usuario y LD5 y LD6 destinados a la función USB OTG, además, integra dos botones push-button uno de ellos programable para ser utilizado por el usuario y otro dedicado al reset del sistema, la placa incluye un ST-LINK/V2-B on-board, lo que facilita la programación y el debugging sin necesidad de hardware adicional.

Uno de los componentes clave es el sensor de movimiento giroscópico ST MEMS I3G4250D de tres ejes, el cual mide la velocidad angular en los ejes X, Y y Z es altamente sensible y preciso permitiendo detectar variaciones en la orientación del sistema en tiempo real, su función principal es monitorear cambios angulares significativos, lo que lo hace ideal para aplicaciones que requieren seguimiento de movimiento o estabilización.

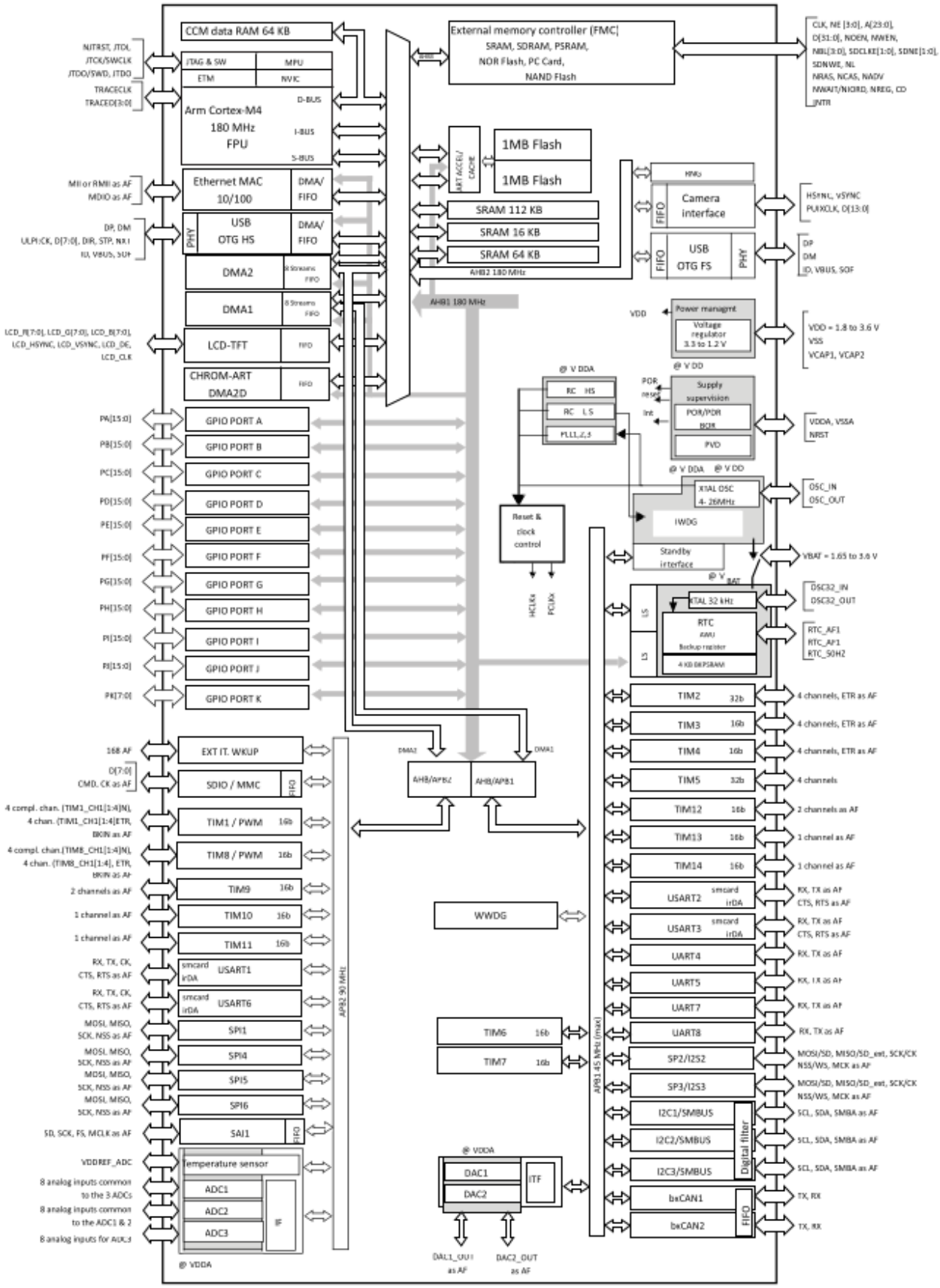

Figura 1: Diagrama de bloques del STM32F429.

2.3. Libopencm3

Es un proyecto de código abierto cuyo objetivo es proporcionar una biblioteca de firmware libre y gratuita para una amplia variedad de microcontroladores basados en la arquitectura ARM Cortex-M, este proyecto ha evolucionado para ofrecer soporte a microcontroladores de diversos fabricantes, su propósito es facilitar el desarrollo de aplicaciones mediante una API estandarizada permitiendo un acceso uniforme a los periféricos y funcionalidades de estos microcontroladores sin depender de bibliotecas propietarias.

Para compilar los ejemplos incluidos en la biblioteca y generar los archivos ejecutables en formato .elf es necesario ejecutar el comando make dentro del directorio `/libopencm3-examples`, para convertir el archivo .elf en un archivo .bin compatible con el microcontrolador se debe utilizar el comando `arm-none-eabi-objcopy -O binary lcd-serial.elf lcd-serial.bin`, lo que genera un archivo binario que puede ser cargado directamente en la memoria del microcontrolador, para programar el código en el STM32F429i se carga el archivo binario en la dirección de memoria adecuada y se reinicia el microcontrolador con el comando `st-flash --reset write lcd-serial.bin 0x8000000`. En el desarrollo de esta práctica, se parte del diseño basado en el ejemplo lcd-serial, el cual permite explorar y probar la comunicación con pantallas LCD mediante la biblioteca ofreciendo una base sólida para futuras implementaciones.

2.4. Thingsboard

Para la implementación del laboratorio se requiere establecer una conexión con un dashboard que permita la visualización en tiempo real de los datos obtenidos del sistema, este debe mostrar los valores proporcionados por el giroscopio, la temperatura del entorno y el nivel de la batería, con lo cual se facilita el monitoreo y análisis de estos parámetros, el diseño del dashboard se encuentra disponible aquí donde se tiene una vista general de la interfaz y funcionalidad permitendo interpretar de manera eficiente los datos adquiridos por el microcontrolador.

Para establecer la comunicación entre el dashboard y el STM32F429i es necesario implementar un archivo de conexión message queuing telemetry transport (MQTT) el cual es un protocolo de mensajería ampliamente utilizado en aplicaciones de internet de las cosas (IoT) debido a su capacidad de operar en redes de ancho de banda limitado, en este laboratorio se desarrolló un archivo de configuración MQTT basado en los ejemplos disponibles en el repositorio de ejemplos de ThingsBoard, con lo cual se permite que el STM32F429i envíe datos de manera estructurada al dashboard.

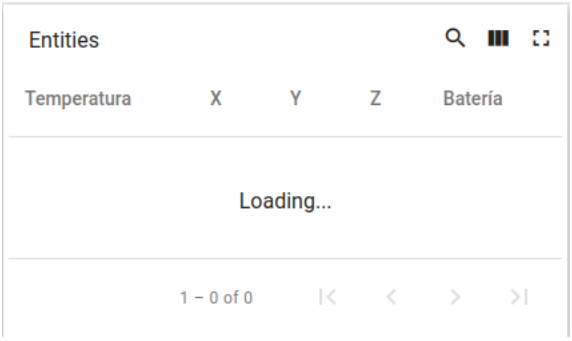

Figura 2: Dashboard de Thinksboard.

3. Desarrollo/Análisis

3.1. Implementación

Para el desarrollo y prueba del código en el microcontrolador STM32F429i se realizan modificaciones en el archivo Makefile añadiendo dos nuevas reglas, la primera automatiza la compilación del código en C y la carga del archivo .bin en el microcontrolador. La segunda regla facilita la comunicación con el dashboard mediante el protocolo MQTT permitiendo la transmisión eficiente de datos en tiempo real como valores del giroscopio, temperatura y estado de la batería, a continuación se presenta el código diseñado.

```
PROJECT     = firmware
BUILD_DIR   = bin
DEVICE      = stm32f429zit6u
OOCD_FILE   = board/stm32f4discovery.cfg

# Directorios compartidos
SHARED_DIR  = ../include/mems \
              ../include/lcd-serial \
              ../include/button \
              ../include/labo \
              ../include/battery

# Archivos fuente
CFILES      = mems.c \
              clock.c console.c font-7x12.c gfx.c lcd-spi.c sdram.c \
              button.c \
              labo.c \
              detector.c \
              battery.c

PREFIX  ?= arm-none-eabi-
CC      = $(PREFIX)gcc
CXX     = $(PREFIX)g++
LD      = $(PREFIX)gcc
OBJCOPY = $(PREFIX)objcopy
OBJDUMP = $(PREFIX)objdump
OOCD    ?= openocd

OPT  ?= -Os
CSTD ?= -std=c99

VPATH     += $(SHARED_DIR)
INCLUDES  += $(patsubst %,-I%, . $(SHARED_DIR))

OPENCM3_DIR = ../libopencm3
OPENCM3_INC = $(OPENCM3_DIR)/include
```

```
37 INCLUDES   += $(patsubst %,-I%, . $(OPENCM3_INC))
38
39 TGT_CPPFLAGS  = -MD -Wall -Wundef $(INCLUDES)
40 TGT_CFLAGS    = $(OPT) $(CSTD) -ggdb3 -fno-common -ffunction-sections
       -fdata-sections
41 TGT_CFLAGS   += -Wextra -Wshadow -Wno-unused-variable -Wimplicit-
     function-declaration
42 TGT_CFLAGS   += -Wredundant-decls -Wstrict-prototypes -Wmissing-
     prototypes
43
44 TGT_CXXFLAGS  = $(OPT) $(CXXSTD) -ggdb3 -fno-common -ffunction-
     sections -fdata-sections
45 TGT_CXXFLAGS += -Wextra -Wshadow -Wredundant-decls -Weffc++
46
47 TGT_ASFLAGS   = $(OPT) $(ARCH_FLAGS) -ggdb3
48
49 TGT_LDFLAGS   = -T$(LDSCRIPT) -L$(OPENCM3_DIR)/lib -nostartfiles
50 TGT_LDFLAGS  += $(ARCH_FLAGS) -specs=nano.specs -Wl,--gc-sections
51 TGT_LDFLAGS  += -u _printf_float -mfloat-abi=hard
52
53 ifeq ($(V),99)
54 TGT_LDFLAGS  += -Wl,--print-gc-sections
55 endif
56
57 LDLIBS  = -Wl,--start-group -lc -lgcc -lnosys -Wl,--end-group
58
59 OBJS = $(CFILES:%.c=$(BUILD_DIR)/%.o)
60 GENERATED_BINS = $(PROJECT).elf $(PROJECT).bin $(PROJECT).map $(
     PROJECT).list $(PROJECT).lss
61
62 all: $(PROJECT).elf $(PROJECT).bin
63
64 $(BUILD_DIR)/%.o: %.c
65         @printf "  CC\t$<\n"
66         @mkdir -p $(dir $@)
67         $(Q)$(CC) $(TGT_CFLAGS) $(CFLAGS) $(TGT_CPPFLAGS) $(CPPFLAGS)
               -o $@ -c $<
68
69 $(BUILD_DIR)/%.o: %.cxx
70         @printf "  CXX\t$<\n"
71         @mkdir -p $(dir $@)
72         $(Q)$(CXX) $(TGT_CXXFLAGS) $(CXXFLAGS) $(TGT_CPPFLAGS) $(
             CPPFLAGS) -o $@ -c $<
73
74 $(BUILD_DIR)/%.o: %.S
```

```
75          @printf "  AS\t$<\n"
76          @mkdir -p $(dir $@)
77          $(Q)$(CC) $(TGT_ASFLAGS) $(ASFLAGS) $(TGT_CPPFLAGS) $(
               CPPFLAGS) -o $@ -c $<
78
79  $(PROJECT).elf: $(OBJS) $(LDSCRIPT)
80          @printf "  LD\t$@\n"
81          $(Q)$(LD) $(TGT_LDFLAGS) $(LDFLAGS) $(OBJS) $(LDLIBS) -o $@
82
83  %.bin: %.elf
84          @printf "  OBJCOPY\t$@\n"
85          $(Q)$(OBJCOPY) -O binary $< $@
86
87  %.lss: %.elf
88          $(OBJDUMP) -h -S $< > $@
89
90  %.list: %.elf
91          $(OBJDUMP) -S $< > $@
92
93  flash: $(PROJECT).elf $(PROJECT).bin
94          st-flash --reset write $(PROJECT).bin 0x8000000
95
96  clean:
97          rm -rf $(BUILD_DIR) $(GENERATED_BINS)
98
99  %.flash: %.elf
100         @printf "  FLASH\t$<\n"
101
102 ifeq (,$(OOCD_FILE))
103         $(Q)(echo "halt; program $(realpath $(*).elf) verify reset" |
              nc -4 localhost 4444 2>/dev/null) || \
104               $(OOCD) -f interface/$(OOCD_INTERFACE).cfg \
105               -f target/$(OOCD_TARGET).cfg \
106               -c "program $(realpath $(*).elf) verify reset exit" \
107               $(NULL)
108 else
109         $(Q)(echo "halt; program $(realpath $(*).elf) verify reset" |
              nc -4 localhost 4444 2>/dev/null) || \
110               $(OOCD) -f $(OOCD_FILE) \
111               -c "program $(realpath $(*).elf) verify reset exit" \
112               $(NULL)
113 endif
114
115 .PHONY: all clean flash
116 -include $(OBJS:.o=.d)
```

```
117 include $(OPENCM3_DIR)/mk/genlink-rules.mk
```

Para la conexión MQTT se emplean varias bibliotecas en Python, incluyendo serial (comunicación serial), paho.mqtt.client (manejo de MQTT), json (estructuración de datos) y time (manejo de retardos), la configuración del puerto serial se realiza a través de `/dev/ttyACM1` a 115200 baudios para garantizar estabilidad en la transmisión.

Se configuran 5 funciones de callback en el cliente MQTT: `on_connect` que notifica el éxito o error en la conexión al broker; `on_message` que confirma la publicación exitosa de mensajes en el dashboard, `setup_mqtt_client` que configura y devuelve un cliente listo para usarse, `setup_serial_connection` que configura y devuelve una conexión serial, `parse_sensor_data` que compara los datos recibidos del puerto serie y devuelve un diccionario con los valores, se establece la conexión al servidor iot.eie.ucr.ac.cr en el puerto 1883 configurando credenciales de acceso y se implementa un bucle while para gestionar intentos de conexión y reconexión automática en caso de fallos temporales.

Los datos recibidos por el puerto serial se procesan y se separan en componentes como eje X, eje Y, eje Z y nivel de batería, se transforman según sea necesario y se publican en formato JSON en el tópico. Para establecer la conexión con ThingsBoard se ejecuta un comando cURL para registrar el dispositivo y habilitar la transmisión de datos realizando una solicitud POST al servidor con datos en formato JSON, luego se establece la conexión entre el STM32F429i y el dashboard, a continuación se presenta el código diseñado.

```
import paho.mqtt.client as mqtt
import json
import time
import serial
import sys

# Puerto serie desde los argumentos de la l nea de comandos
PORT = sys.argv[1]

# Configuraci n del broker MQTT
BROKER = "iot.eie.ucr.ac.cr"
PORT_MQTT = 1883
TOPIC_TELEMETRY = "v1/devices/me/telemetry"
TOPIC_REQUEST = "v1/devices/me/attributes/request/1/"
USERNAME = "urtcz88waqo9wjzbknpa"

# Diccionario para almacenar los datos
data_dict = {'temp': 0, 'x': 0, 'y': 0, 'z': 0, 'bat': 0, 'low_bat':
    "", "deg_alert": ""}

def on_connect(client, userdata, flags, rc):
    """Cliente se conecta al broker."""
    print(f"Conectado con c digo de resultado {rc}")
    client.subscribe(TOPIC_REQUEST)

```

```
25 def on_message(client, userdata, msg):
26     """Se recibe un mensaje en un t pico suscrito."""
27     print(f"Mensaje␣recibido␣en␣{msg.topic}:␣{msg.payload.decode()}")
28 
29 def setup_mqtt_client():
30     """Configura y devuelve un cliente MQTT listo para usarse."""
31     client = mqtt.Client()
32     client.username_pw_set(USERNAME)
33     client.on_connect = on_connect
34     client.on_message = on_message
35     client.connect(BROKER, PORT_MQTT, 60)
36     client.loop_start()
37     return client
38 
39 def setup_serial_connection(port):
40     """Configura y devuelve una conexi n serial con el puerto STM.
          """
41     try:
42         return serial.Serial(port, 115200, timeout=1)
43     except serial.SerialException as e:
44         print(f"Error␣al␣abrir␣el␣puerto␣serial:␣{e}")
45         sys.exit(1)
46 
47 def parse_sensor_data(data):
48     """Compara los datos recibidos del puerto serie y devuelve un
          diccionario con los valores."""
49     try:
50         dsplit = data.strip().split()
51         if len(dsplit) != 5:
52             return None
53 
54         x, y, z = map(float, dsplit[:3])
55         temp = int(dsplit[3])
56         batt = float(dsplit[4]) * 100 / 9
57         deg_alert = any(abs(axis) > 5 for axis in (x, y, z))
58 
59         return {
60             'x': x,
61             'y': y,
62             'z': z,
63             'temp': temp,
64             'bat': batt,
65             'low_bat': "Full␣Battery" if batt >= 90 else "Low␣Battery
                  " if batt < 80 else "Medium␣Battery",
66             'deg_alert': "Danger!" if deg_alert else "All␣good."
```

```
67          }
68      except (ValueError, IndexError):
69          return None
70
71  def main():
72      client = setup_mqtt_client()
73      ser = setup_serial_connection(PORT)
74
75      try:
76          while True:
77              if ser.in_waiting > 0:
78                  raw_data = ser.readline().decode('utf-8', errors='
                        ignore')
79                  sensor_data = parse_sensor_data(raw_data)
80
81                  if sensor_data:
82                      client.publish(TOPIC_TELEMETRY, json.dumps(
                            sensor_data))
83                      print(f"Datos␣enviados:␣{sensor_data}")
84
85                  time.sleep(0.1)  # Peque a pausa para evitar
                        sobrecarga
86
87      except KeyboardInterrupt:
88          print("\nSaliendo␣del␣programa.")
89
90      finally:
91          ser.close()
92          client.loop_stop()
93          client.disconnect()
94          print("Conexi n␣serial␣cerrada␣y␣desconectado␣del␣broker␣
                MQTT.")
95
96  if __name__ == "__main__":
97      main()
```

El firmware principal del microcontrolador está desarrollado en C e incluye diversas funciones para la configuración y control de periféricos, lectura de sensores, comunicación con dispositivos y actualización de la interfaz de usuario, la interfaz incorpora alertas para indicar el estado de la batería y advertencias ante cambios en la inclinación, también permite que el microcontrolador active el parpadeo de un LED en respuesta a eventos significativos, a continuación se presenta el código diseñado.

```
#include "labo.h"
int main(void)
{
    uint8_t temperature;
    double battery;
    bool USART_enable = true;
    data xyzData;

    INIT_ANGLE(xyzData.angle);
    INIT_SAMPLE_TIME(xyzData.lastTime);

    // Configuraci n de perif ricos
    clock_setup();
    console_setup(115200);
    mems_spi_init();
    sdram_init();
    lcd_spi_init();
    button_init();
    battery_init();
    gfx_init(lcd_draw_pixel, 240, 320);

    while (1)
    {
        // Toma de datos
        temperature = mems_temp();
        xyzData.reading = read_xyz();
        xyzData = integrate_xyz(xyzData);
        battery = get_battery(5);  // Canal 5 del ADC

        // Alertas
        five_degree_alert(xyzData);
        low_battery_alert(battery);

        // Env o de datos
        USART_enable = console_usart_enable(xyzData, temperature,
           battery, USART_enable);
        lcd_slope(temperature, xyzData.angle, battery, USART_enable);
    }
}
```

3.2. Pruebas de funcionamiento

Las pruebas realizadas al sistema incluyeron tanto la verificación de las conexiones físicas del circuito como del código implementado, se revisaron todas las conexiones para garantizar que no hubiera errores y posteriormente se cargó el programa en el STM32F429 Discovery Kit, uno de los aspectos destacados fue la correcta visualización de los parámetros del sistema en el display LCD.

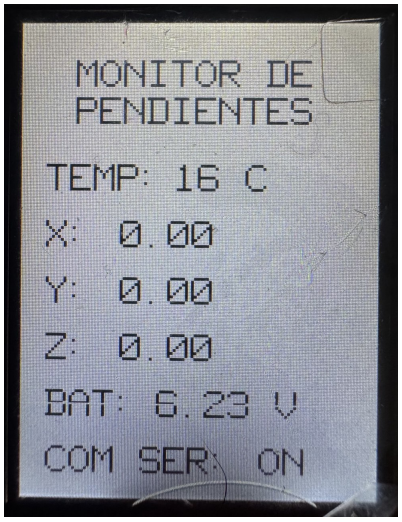

Figura 3: Sistema en estado OK.

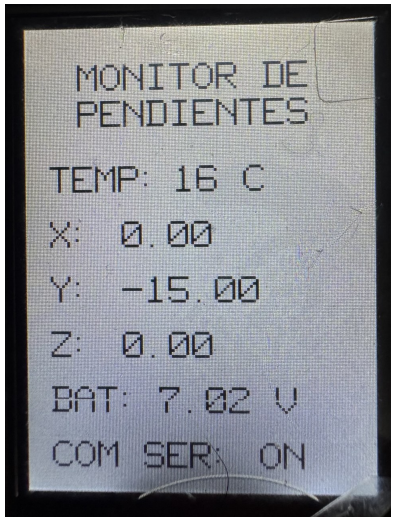

Figura 4: Sistema en estado de Alerta.

4. Conclusiones y Recomendaciones

Se desarrolló un firmware en C para el MCU STM32F429 logrando los objetivos del proyecto a pesar de los desafíos en la implementación de MQTT, se identificaron áreas de mejora como la conversión incorrecta del nivel de batería a un rango adecuado y la omisión de la lectura de temperatura en la comunicación MQTT por limitaciones de tiempo. Se recomienda corregir estos aspectos en futuras versiones para mejorar la precisión, robustez y organización del código, aun con estas dificultades el proyecto fue exitoso.

Referencias

[1] STMicroelectronics. (2024). *STM32F429xx 32b Arm Cortex-M4 MCU+FPU, 225DMIPS, up to 2MB Flash/256+4KB RAM, USB OTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 20 com. interfaces, camera & LCD-TFT Datasheet - production data*. Recuperado de https://www.st.com/resource/en/datasheet/stm32f427vg.pdf