Universidad de Costa Rica

Laboratorio 4
STM32: GPIO, ADC, comunicaciones, Iot

IE-0624 Laboratorio de Microcontroladores

Prof. MSc. Marco Villalta

Isaac Rojas Hernández
B76693
Amy Herrera Mora
B53473

23 de junio de 2023

Índice

1. Introducción

En el presente documento se describe todo el trabajado realizado para el cuarto laboratorio del curso Laboratorio de microcontroladores. En él se desea desarrollar un sismógrafo digital para registrar y estudiar las oscilaciones en el edificio de la escuela de ingeniería eléctrica. Este sismógrafo tiene alimentación por batería (cargadas por paneles solares) y poco ancho de banda para el envio de datos. Por lo tanto debe cumplir con los siguientes requisitos:

1. Debe utilizar una placa STM32F429 Discovery kit y la biblioteca libopencm3.
2. Leer los ejes del giroscopio (X,Y,Z).
3. Incluir un switch/botón para habilitar/deshabilitar comunicaciones por USART/USB.
4. Un led debe parpadear indicando la habilitación de la transmisión/recepción de datos por el periférico USART/USB.
5. Debe leer el nivel de la batería cuyo rango es de [0,9]V, en caso de estar cerca del limite mínimo de operación del microcontrolador(7 V) debería encender un led de alarma parpadeante y enviar la notificación de bateiia baja al dashboard de thingsboard.
6. Debe desplegar en la pantalla LCD el nivel de batería, los valores de los ejes X,Y,Z y si la comunicación serial/USB esta habilitada.
7. Crear un script de python que lea/escriba al puerto serial/USB y que envíe la información del giroscopio y nivel de batería para ser desplegados en un dashboard de una plataforma Iot thingsboard (Queda a criterio propio la utilización de los widgets para visualizar esta información)

Para esto, se utilizaron los ejemplos propios de la biblioteca libopencm3 para poder utilizar en conjunto en un archivo .c main el giroscopio, la pantalla LCD, los LEDs, el font y la consola. Además, se realiza el diseno de un ajuste de alimentación para la tarjeta con divisores de tensión como precaución Una vez compilado con make, se "flashea.[el] archivo binario al microcontrolador para verificar su funcionamiento. Adicionalmente, se realiza un código en Python que permite la comunicación con la plataforma IoT ThingsBoard para visualizar con Widgets los datos generados. Como principales conclusiones se obtuvieron que la familia de microcontroladores STM32 son perfectamente capaces de aplicaciones que tengan una exigencia superior a nivel de hardware, haciéndolos perfectos para usos más complejos y que implementar funcionalidades de IoT a las aplicaciones permite tener mayor control sobre las funciones deseadas así como una simplicidad para el usuario común.

2. Nota teórica

2.1. STM32

Los STM32 son una serie de circuitos integrados con microcontroladores como componente principal, son desarrollados por STMElectronics. La familia se divide por series, las cuales tienen siglas que las diferencian entre ellas y poseen procesadores ARM de 32-bits distintos dependiendo de a cual serie pertenecen, además una serie puede tener varias versiones con distinto procesador. En la siguiente imagen se pueden ver las divisiones de las mismas:

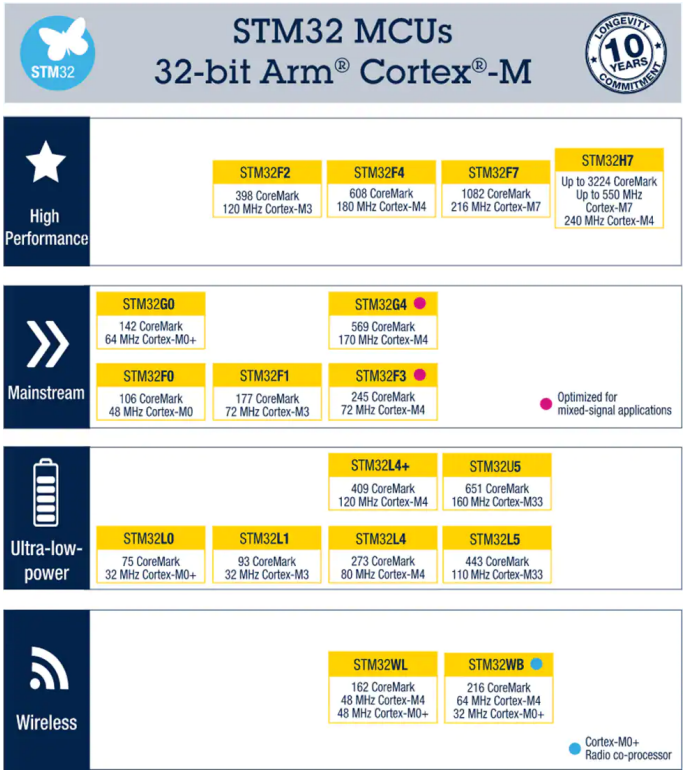


Figura 1: Distribución de la familia STM32 [1]

Para el desarrollo de este laboratorio se utilizó la serie F4, específicamente el modelo STM32F429i, basada en el ARM Cortex M4, este MCU posee un DSP y también instrucciones de punto flotante, las características específicas son las siguientes:

- Memoria: 2MB Memoria Flash, 256KB RAM.
- Procesador: ARM Cortex M4 32-bits con FPU a 180MHz.
- Pantalla: 2.4"QVGA TFT LCD.
- Sensor de movimiento L3GD20.
- Alimentación: USB o fuente externa de 3V o 5V.
- 17 timers: 12 timers de 16bit, 2 de 32bit de hasta 180MHz, c/u con 4IC/OC/PWM.
- Convertidor Analógico-Digital: 3x12bits.
- Convertidor Digital-Analógico: 2x12bits.
- Conectividad USB 2.0 Avanzada (Debug, COM virtual, almacenamiento, OTG).
- 21 Interfaces de comunicación (I2C, USART, SPI, SAI, CAN).
- Bajo consumo de potencia.

Las características eléctricas se encuentran descritas en la hoja de datos general para los modelos STM32F427xx STM32F429xx, se pueden ver las características de tensión en la figura (2) y las de corriente en la figura (3).

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 2: Características de voltaje del STM32F429i [3]

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 3: Características de corriente del STM32F429i [3]

El diagrama de pines del MCU se puede apreciar en la figura (4), mientras que el diagrama de bloques de la arquitectura del mismo se aprecia en la figura (4).

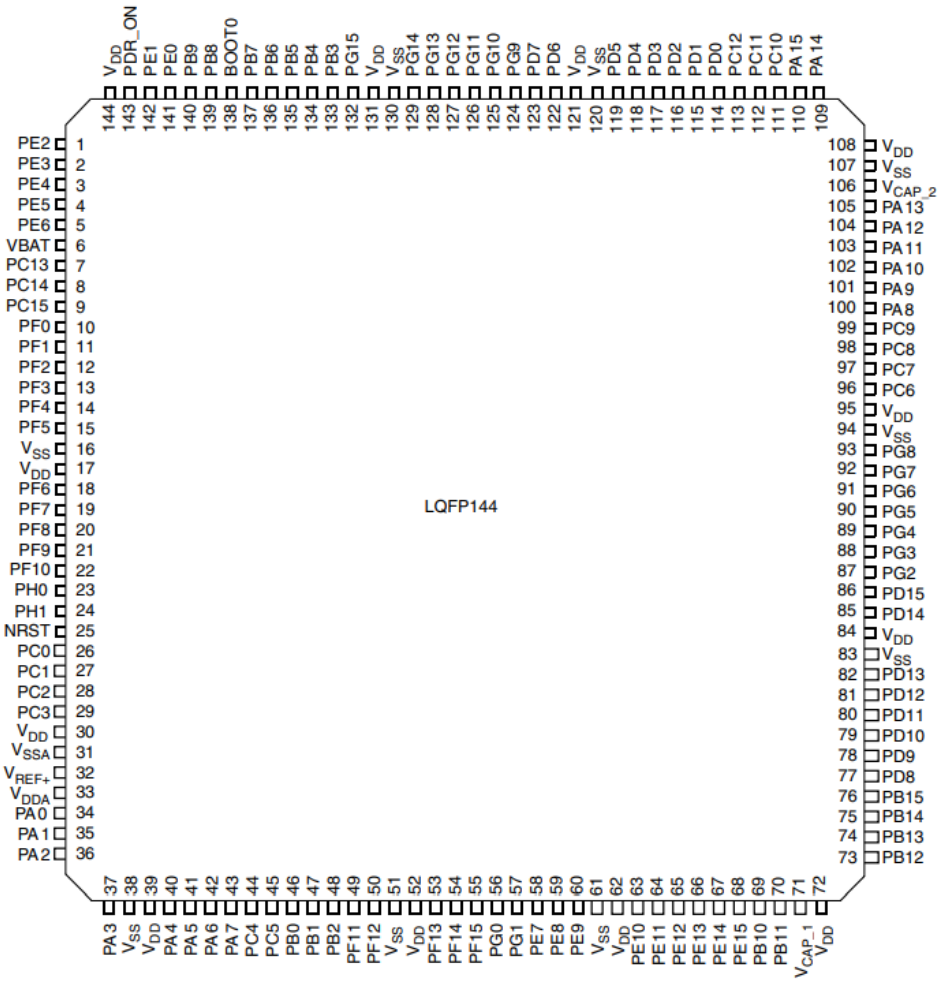


Figura 4: Pin Layout del STM32F429i [3]

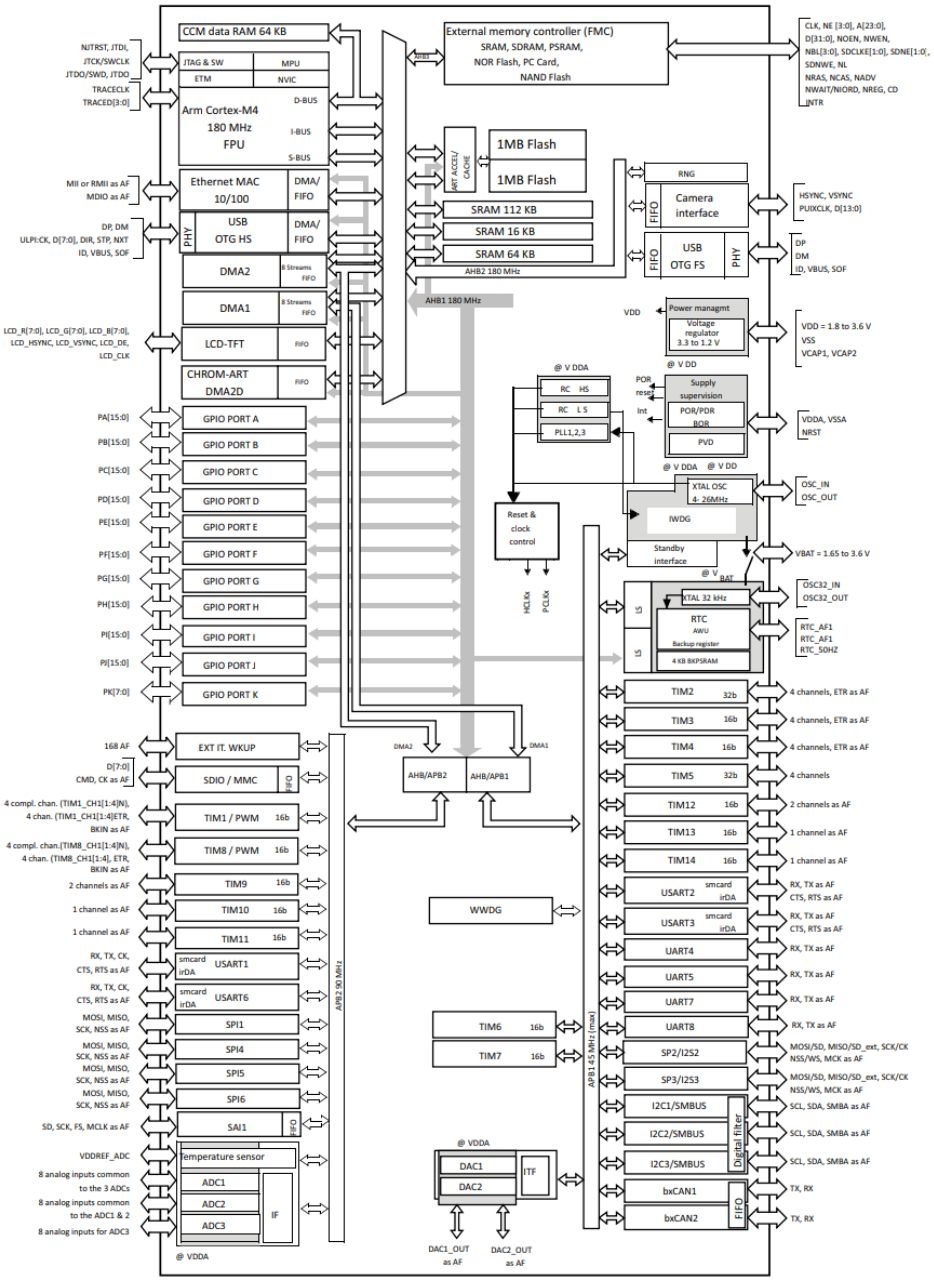


Figura 5: Diagrama de bloques de la arquitectura del STM32F249i [3]

2.1.1. Registros necesarios

El microcontrolador posee registros completamente independientes del giroscopio y la pantalla LCD, por lo que se describirán por separado a continuación. Primeramente, los registros con variables del giroscopio, habilitación de lectura de registros y el ajuste de variables del MCU se tienen los siguientes:

- GYR_OUT_X_L
- GYR_OUT_X_H

- GYR_OUT_Y_L
- GYR_OUT_Y_H
- GYR_OUT_Z_L
- GYR_OUT_Z_H
- CTRL_REG1
- CTRL_REG4

2.2. STM32 Discovery Kit

El kit Discovery STM32 aprovecha las capacidades del STM32F429, para permitir el desarrollo sencillo para usuarios comunes. Incluye una pantalla LCD, un botón de reset, un botón para el usuario, un sensor de giroscopio, 6 LEDs, puerto micro USB a USB A, además de permitir una facilidad de desarrollo de aplicaciones con interfaz gráfica [1].

2.2.1. Giroscopio

El kit con el que se realizó este laboratorio cuenta con un L3GD20 ST MEMS, que es un sensor de movimiento de 3 ejes de precisión. Es un sensor de velocidad angular el cual tiene bajo consumo de energía, utiliza una interfaz IC que le otorga la capacidad de brindar una tasa angular a través de la interfaz serial I2C/SPI y tiene escalas dinámicas que son ajustables por el usuario con los siguientes valores: ±250/500/2000 dps [3].

2.2.2. Pantalla LCD

El Kit discovery incluye una pantalla LCD de 2.4 pulgadas con tecnología LCD TFT (Thin film transistor with Liquid Crystal Display) y resolución QVGA de 240x320 pixeles. Opera con un voltaje de 2.8V ± 0.3V.

2.2.3. IoT

Internet of Things es un término muy popular y utilizado en medios de comunicación tanto digitales como analógicos. Los sensores electrónicos forman parten de muchísimos dispositivos que han desarrollado inteligencia y que se conectan a redes inalámbricas y a internet. Esto constituye una red global de conectividad total llamada Internet de las cosas. Como ejemplo, están los teléfonos inteligentes, los cuales poseen sensores tales como giróscopos y acelerómetros que logran su posicionamiento casi en tiempo real. Si a esto se le suma que estos dispositivos electrónicos han disminuido su tamaño y costo, y que pueden generar muchísima información, convierte al mundo actual conectado a cosas y objetos inteligentes. [4]

Los protocolos de comunicación permiten que se intercambie información entre aplicaciones y dispositivos, además de que los sensores ubicados en cualquier lugar puedan ser accedidos por los demás dispositivos y aplicaciones. Entonces, el Internet de las cosas consiste en un sistema tecnológico donde personas y objetos pueden conectarse a internet en cualquier momento y lugar para ganar inteligencia y que los objetos se comuniquen. Esto se hace posible porque el internet esta presente

al mismo tiempo en todas partes. Debido a este gran auge, Cisco, el cual es el primer fabricante de telecomunicaciones, proyectó que para el 2020 habrían cincuenta mil millones de dispositivos conectados en el mundo. [4]

La definición de este concepto tan importante incluye cuatro componentes fundamentales, los cuales son: la conexión que se refiere a los protocolos de comunicación que se estudian, las cosas que son los actuadores, sensores, controladores, la no operación por personas; y finalmente el internet que se refiere a la seguridad. [4]

2.3. Plataforma ThingsBoard

ThingsBoard es una plataforma de IoT (Internet of things) de código abierto. Esta hace posible un rápido desarrollo de proyectos relacionados con esta tecnología. Con esta plataforma se tiene acceso a una solución local o en la nube, que se podrá utilizar ya que habilita la infraestructura del lado del servidor para las aplicaciones pertinentes. [5]

Dicha plataforma presenta compatibilidad con protocolos de IoT estándar tales como: MQTT, CoAP y HTTP. Tiene la capacidad de combinar escalabilidad y tolerancia a los fallos, además de tener un buen rendimiento a la hora capturar los datos del dispositivo para poder procesarlos. Esto se logra porque dispone de un servidor de puerta de enlace encargado de la comunicación con los dispositivos conectados a la red. Con esto alcanza un buen funcionamiento y permanente actualización. [5]

2.4. Diseño del circuito

Para el diseño, se inició con una división de las diferentes partes que posee el circuito que realiza la función de sismógrafo, por ende el diseño que se pensó para el mismo se encuentra en la figura (6). Puesto que el MCU que se utilizó durante esta práctica cuenta con las facilidades de un giroscopio integrado, la pantalla, las interfaces de comunicación u otros periféricos, únicamente fue necesario el diseño del circuito de transformación de tensión para adecuar la batería de 9V DC a 5V DC, que es lo aceptado por la tarjeta.

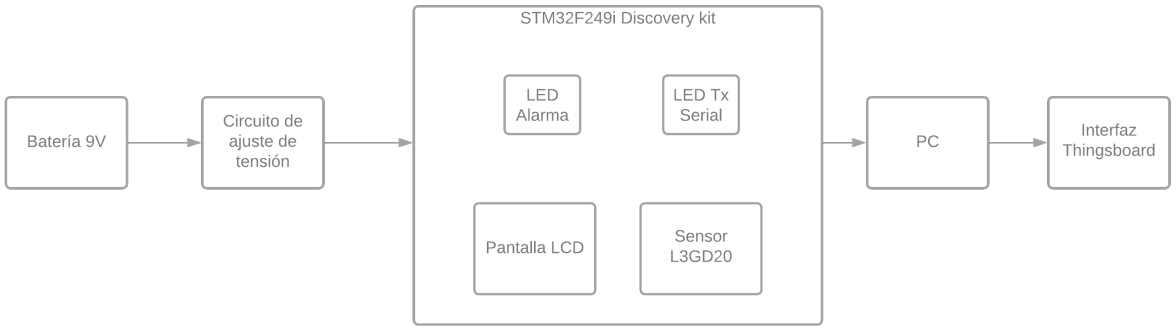


Figura 6: Diagrama de bloques del sismógrafo con STM32 (Creación propia)

2.5. Diseño del circuito de transformación de tensión 9V a 5V

Para este diseño, se utilizo un potenciómetro para poder regular la tensión de salida entre 7V y 9V. Con esta primera parte se pretendía enviar una señal de alarma que indicara que la batería estaba baja. Seguidamente, para adecuar una tensión de entrada para el microcontrolador se agregó un segundo divisor de tensión. El esquemático se puede apreciar en la figura (7).

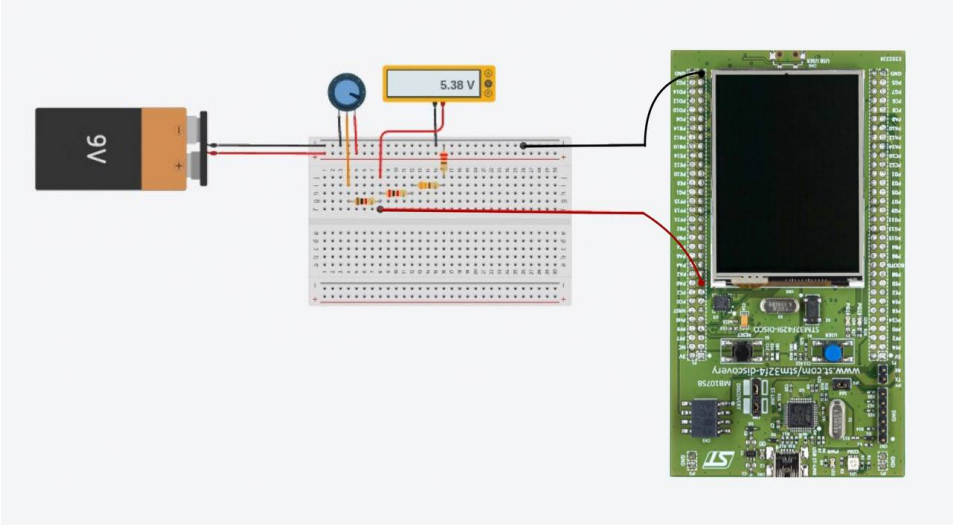


Figura 7: Esquemático para el diseño de la alimentación del MCU (Creación propia)

Para el valor del potenciómetro se escogió arbitrariamente que fuera de un valor de $5k\Omega$. Por otra parte, para escoger los valores del segundo divisor de tensión se utilizó la ecuación 1.

$$V_{out} = V_{in} * \frac{R_2}{R_1 + R_2} \tag{1}$$

Ahora, tomando como referencia una salida de 8.1V desde potenciómetro, este valor sera la entrada del segundo divisor. Se busca una tensión de entrada de mas de 5V por si la batería se descarga o por si ocurren caídas de tensión en los cables que alimentaran el microcontrolador. Entonces, para una tensión de salida de aproximadamente 5.8V y un valor de $1k\Omega$ para R_1, se realiza el cálculo de las ecuación 2.

$$5{,}8V = 8{,}1V * \frac{R_2}{1k\Omega + R_2} \tag{2}$$

Lo que resulta en la ecuación 3 al despejar para R_2.

$$R_2 = 2521{,}74\Omega \tag{3}$$

Esta resistencia puede reemplazarse por tres en serie cuyos valores teóricos son: $R_2 = 2k\Omega$, $R_3 = 330\Omega$ y $R_4 = 220\Omega$. Para la implementación en la protoboard se realizan las mediciones necesarias para los valores reales de estas que se obtuvieron como: $R_1 = 984\Omega$, $R_2 = 1{,}97k\Omega$, $R_3 = 326\Omega$ y $R_4 = 216\Omega$. Con esto se esperaba una tensión de ajuste que se muestra con la ecuación 4.

$$5{,}82 = 8{,}1 * \frac{2512\Omega}{984\Omega + 2512\Omega} \tag{4}$$

Componentes	Precio
Potenciometro	240 colones
2 Resistencias $1k\Omega$	150 colones
Resistencias 330Ω y 220Ω	120 colones
Discovery Kit	52 000 colones
Bateria 9V	900 colones
Total	**53 410 colones**

Tabla 1: Componentes necesarios para el desarrollo del proyecto

Al montar el circuito en la protoboard, se puede observar en la figura (8) que se obtiene un ajuste en la tensión de 5.34V.

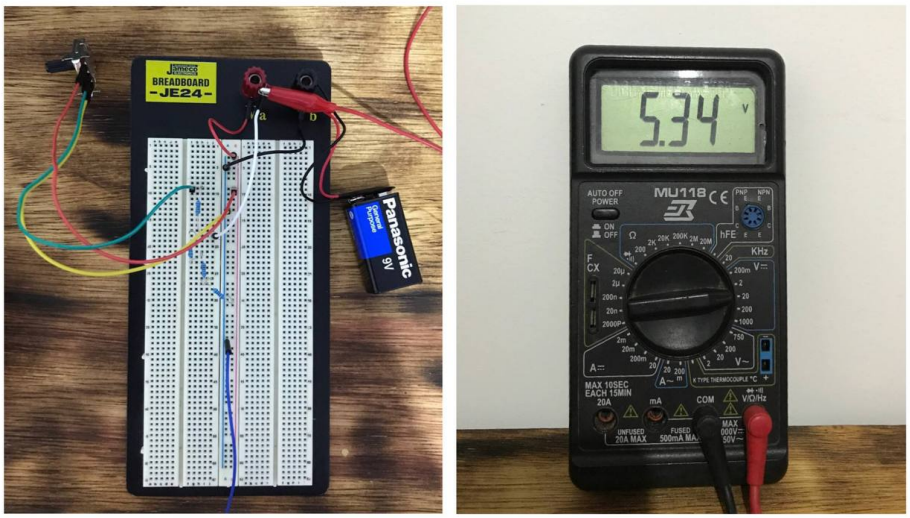


Figura 8: Prueba para el valor de ajuste de la tensión de entrada para el MCU (Creación propia)

2.6. Tabla de componentes para el desarrollo del proyecto

3. Desarrollo/Análisis de resultados

3.1. Desarrollo

Para poder desarrollar el programa que contenga todas las funcionalidades descritas en el enunciado del laboratorio, fue necesario seguir el diagrama de flujo mostrado en la figura (9).

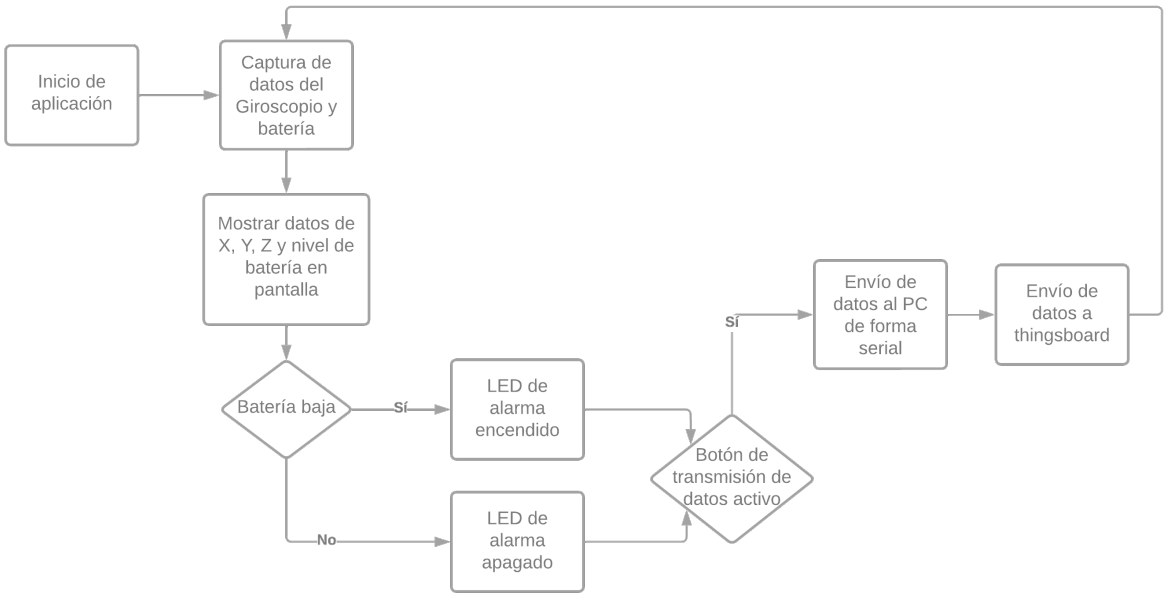


Figura 9: Diagrama de flujos del sismógrafo con STM32 (Creación propia)

El programa del sismógrafo sigue una línea algorítmica muy sencilla pero que debe ser implementada de forma cuidadosa. Es necesario destacar que la construcción del programa actual implementa código de los ejemplos de la librería libopencm3, pero con pequeñas modificaciones que lo adaptan para que realice lo solicitado por el enunciado. La combinación que se utilizó de los distintos ejemplos fue: adc-dac-printf, spi, miniblink y sdram. Se implementó un archivo principal que importó características de otros archivos dentro de la misma carpeta, así como archivos de la librería propiamente, el encabezado utilizado fue el siguiente:

```
#include <stdint.h>
#include <math.h>
#include <libopencm3/stm32/rcc.h>
#include <libopencm3/stm32/gpio.h>
#include <libopencm3/stm32/spi.h>
#include <libopencm3/stm32/adc.h>
#include "clock.h"
#include "console.h"
#include "sdram.h"
#include "lcd-spi.h"
#include "gfx.h"
#include <libopencm3/stm32/usart.h>
```

Por medio de estos imports se logró crear un código para que el microcontrolador actúe como sismógrafo y despliegue los datos en consola. Primeramente como con cualquier MCU, fue necesario realizar la configuración de los pines pertinentes para el laboratorio, que en este caso únicamente se utilizó el pin del positivo del circuito de la batería.

Además se modificaron los registros necesarios para que la pantalla y el giroscopio funcionasen de forma eficiente, así como el botón. Para el funcionamiento del botón incluído en el kit discovery, se pensó para que este activase y desactivase la transmisión de los datos capturados por medio de la transmisión serial, el código utilizado para activar el uso del botón fue el siguiente:

```
static void button_setup(void)
```

```
{
  /* Enable GPIOA clock. */
  rcc_periph_clock_enable(RCC_GPIOA);

  /* Set GPIO0 (in GPIO port A) to 'input open-drain'. */
  gpio_mode_setup(GPIOA, GPIO_MODE_INPUT, GPIO_PUPD_NONE, GPIO0);
  gpio_mode_setup(GPIOG, GPIO_MODE_OUTPUT, GPIO_PUPD_NONE, GPIO13 | GPIO14);
}
```

Pero la implementación en código del botón para saber si se encuentra activada la comunicación serial sería la siguiente:

```
if (gpio_get(GPIOA, GPIO0)) {
      if(com_en){
        com_en = 0;
      }
      else{
        com_en = 1;
      }
}
```

La captura de los datos se realiza por medio de los registros mencionados anteriormente en la nota teórica, específicamente los de GYR_OUT_EJE_L ó GYR_OUT_EJE_H. Estos registros contienen los valores altos y bajos capturados del sensor del giroscopio, por lo que fue necesario capturar inicialmente el valor bajo y luego el alto, que consiguientemente se promedian por medio de una operación binaria, como se muestra a continuación:

```
    gpio_clear(GPIOC, GPIO1);
    spi_send(SPI5, GYR_OUT_X_L | GYR_RNW);
    spi_read(SPI5);
    spi_send(SPI5, 0);
    gyr_x=spi_read(SPI5);
    gpio_set(GPIOC, GPIO1);

    gpio_clear(GPIOC, GPIO1);
    spi_send(SPI5, GYR_OUT_X_H | GYR_RNW);
    spi_read(SPI5);
    spi_send(SPI5, 0);
    gyr_x|=spi_read(SPI5) << 8;
    gpio_set(GPIOC, GPIO1);
```

El proceso de captura de datos se repite para los 3 ejes que posee el giroscopio, de forma que se pueda desplegar el movimiento completo que realiza el mismo. Con la captura de los datos realizada se implementó la sección de código que despliega los valores en pantalla en conjunto con la batería. Para la batería se implementó el código mostrado específicamente en el ejemplo de adc-dac-printf, de forma que obtuviese el valor que estaba siendo ingresado por medio del pin, para luego crear una función que utilice la captura de datos y realice la conversión necesaria para obtener el valor en voltz.

```
static uint16_t read_adc_naiive(uint8_t channel)
{
  uint8_t channel_array[16];
  channel_array[0] = channel;
  adc_set_regular_sequence(ADC1, 1, channel_array);
  adc_start_conversion_regular(ADC1);
  while (!adc_eoc(ADC1));
  uint16_t reg16 = adc_read_regular(ADC1);
  return reg16;
```

```
}
```

La función que realiza la conversión invocando la captura del dato ADC es la siguiente:

```
void adc_update(void){
  battery = (read_adc_naiive(1)*VOLTAGE_CONVERSION_FACTOR)/4095.0f;
}
```

Esta función se invoca cada vez que el while del programa principal se reinicia, de forma que se tiene una actualización en vivo del valor de la batería, ya que este también fue un requerimiento dado en el enunciado, así por medio del potenciómetro simular la descarga de la misma. Para el despliegue de los datos en pantalla, se modificó el código de uno de los ejemplos para que muestre continuamente esta información:

```
    sprintf(lcd_out, "%s", "X:");
    sprintf(int_to_str, "%d", gyr_x);
    strcat(lcd_out, int_to_str);

    gfx_fillScreen(LCD_BLACK);
    gfx_setCursor(15, 36);
    gfx_puts(lcd_out);

    sprintf(lcd_out, "%s", "Y:");
    sprintf(int_to_str, "%d", gyr_y);
    strcat(lcd_out, int_to_str);

    gfx_setCursor(15, 90);
    gfx_puts(lcd_out);

    sprintf(lcd_out, "%s", "Z:");
    sprintf(int_to_str, "%d", gyr_z);
    strcat(lcd_out, int_to_str);

    gfx_setCursor(15, 144);
    gfx_puts(lcd_out);

    sprintf(int_to_str, "%d", battery*100/9);
```

Como se mostró anteriormente la batería que se despliega en pantalla no es en voltaje, sino en porcentaje, ya que así se garantiza que el usuario promedio sea capaz de comprender el nivel de la misma. Por último la transmisión serial, se realiza por medio de las siguientes líneas de código:

```
    print_decimal(gyr_x); console_puts("\t");
    print_decimal(gyr_y); console_puts("\t");
    print_decimal(gyr_z); console_puts("\t");
    print_decimal(batt_alarm); console_puts("\t");
    //se realiza la conversi n para porcentaje de la bater a
    print_decimal(battery*100/9); console_puts("\n");
```

Con estas piezas de código siendo las mas importantes en conjunto con otros ajustes se completa el código del firmware para este laboratorio. Ahora para poder enviar datos al servidor de thingsboard fue necesario implementar un código aparte en Python para poder capturar los datos y enviarlos al servidor en formato JSON. El código de dicho script se puede apreciar a continuación:

```
#Import para poder enviar a thingsboard los datos
import paho.mqtt.client as mqtt
#Import para capturar los datos que llegan del puerto serial
import serial
```

```
#Import para el tiempo
import time
#import para manejar los datos
import json
import os

serial_port = serial.Serial(port="/dev/ttyACM0", baudrate="Ingresar el baudrate aqu
    ", timeout=1)
print("Conectado al microcontrolador")
data = []
data_header = ["x", "y", "z", "Battery"]
print(data_header)

def on_publish(client,userdata,result):             #create function for callback
    print("data published to thingsboard \n")
    pass

#Datos para la conexi n con el cliente:
broker = "iot.eie.ucr.ac.cr"
topic = "v1/devices/me/telemetry"
device= "Ingrese el token del device aqu "
port = 1883

#Establecemos los par metros de conexi n
client = mqtt.Client("Cliente aqu ")
client.on_publish = on_publish
client.username_pw_set(device)
client.connect(broker, port, keepalive=60)
dictionary = dict()


while True:
   data_captured = serial_port.readline().decode('utf-8').replace('\r', "").replace("
    \n", "")
   data_captured = data_captured.split('\t')
   if len(data_captured) >= 3:
        dictionary['x'] = data_captured[0]
        dictionary['y'] = data_captured[1]
        dictionary['z'] = data_captured[2]
        if data_captured[3] == '1':
             alarm = "Si"
        else:
             alarm = 'No'

        dictionary['Battery Low'] = alarm
        dictionary["Battery Level"] = data_captured[4]
        output = json.dumps(dictionary)
        print(output)
        client.publish(topic, output)
```

Donde algunas de las líneas de código deben sustituirse por los valores específicos de conexión del dispositivo creado dentro de la plataforma Thingsboard.

3.2. Compilación

Para poder realizar la compilación del programa presentado en este documento, se requiere poner la carpeta con el código dentro de la carpeta de los ejemplos:

- libopencm3-examples/examples/stm32/f4/stm32f429i-discovery

Luego únicamente se debe abrir la terminal dentro de la carpeta con el código fuente y usar el comando make para crear el archivo elf que luego deberá ser cambiado a un archivo hex por medio del siguiente comando:

- spi-mems.elf –reset write firmware.bin 0x8000000

3.3. Análisis de Resultados

Una vez que se logra flashear el archivo .bin al microcontrolador, se puede observar en la pantalla los valores de los ejes X, Y y Z del giroscopio al mover la tarjeta varias veces. Adicionalmente, también se muestra el nivel de batería al que se encuentra, y en este caso se indican 0V porque aun no se ha conectado a la alimentación. El funcionamiento de esto puede apreciarse en la figura (10).

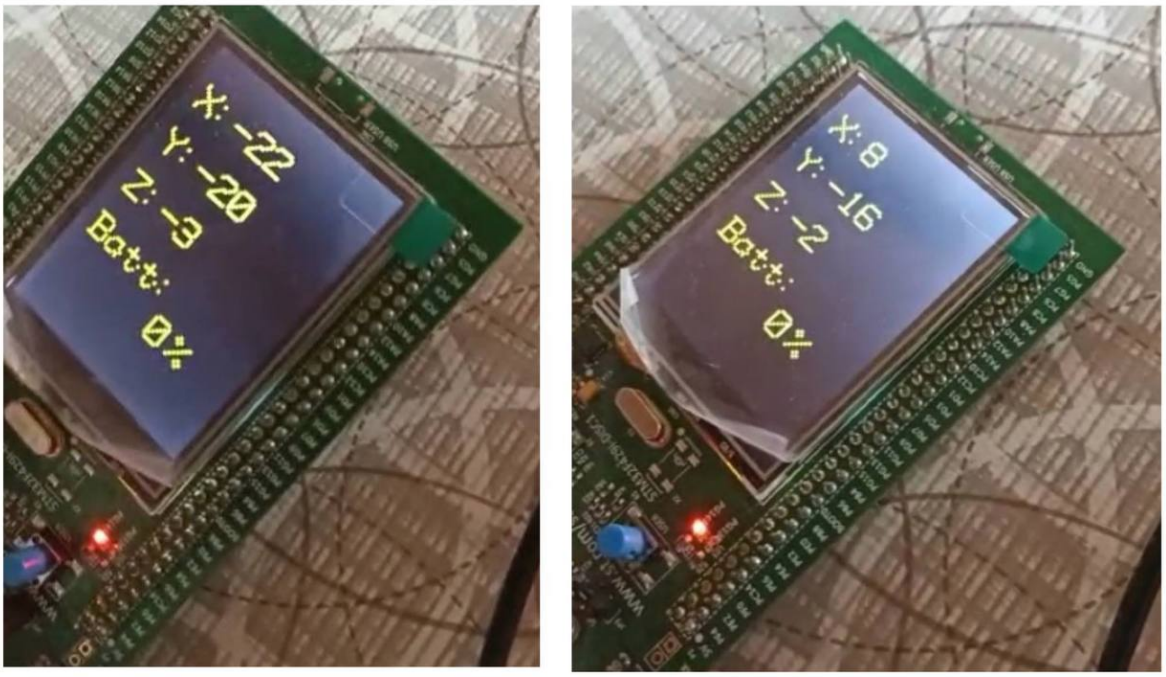


Figura 10: (Creación propia)

Luego, después de haber agregado un nuevo dispositivo a la plataforma Thingsboard, se procede a copiar el TOKEN. Esto porque el archivo .py debe configurarse para poder realizar bien la comunicaciones y exportar con esto los datos. Además, es necesario que el baudrate del programa del microcontrolador coincida con el baudrate que se le ajustara a este script en Python. Dicho valor es de 115200. Cabe destacar que antes de correr esto en terminal es necesario realizar por aparte una pruebas para verificar en la sección Telemetría que los datos están llegando bien. Dicho esto, al correr

finalmente este archivo, se puede observar en la figura (11) que los datos se exportan satisfactoriamente. Para este caso, se agregaron dos Widgets para ilustrar mejor este funcionamiento tanto con los datos en tiempo real como con gráficas. Se puede visualizar en un vídeo con el siguiente link: https://drive.google.com/drive/folders/1_2m_CpBfjC8vIHrsd_OBlq-wPm3ZjKCg?usp=sharing

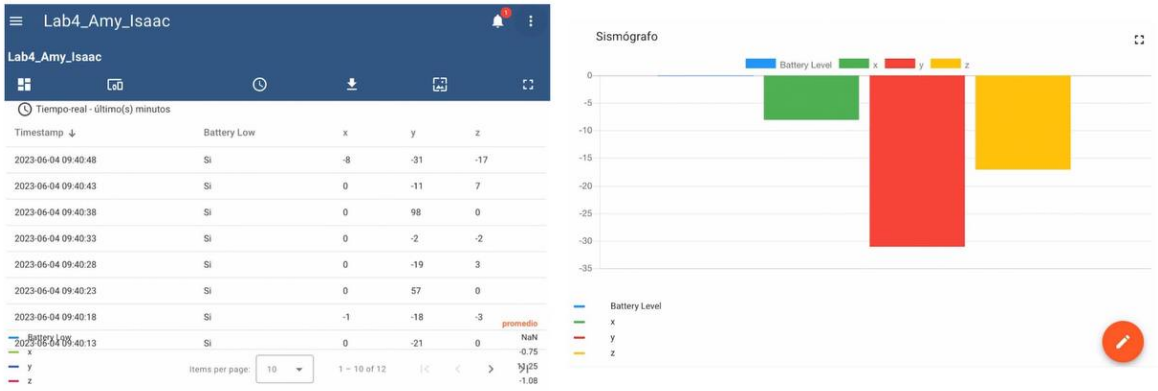


Figura 11: (Creación propia)

Seguidamente, se conecta la alimentación ajustada al microcontrolador y se varia el potenciómetro. Se aprecia en la figura (12) como la tensión de salida que alimenta al MCU comienza a disminuir, y con esto se evidencia el buen funcionamiento del circuito.

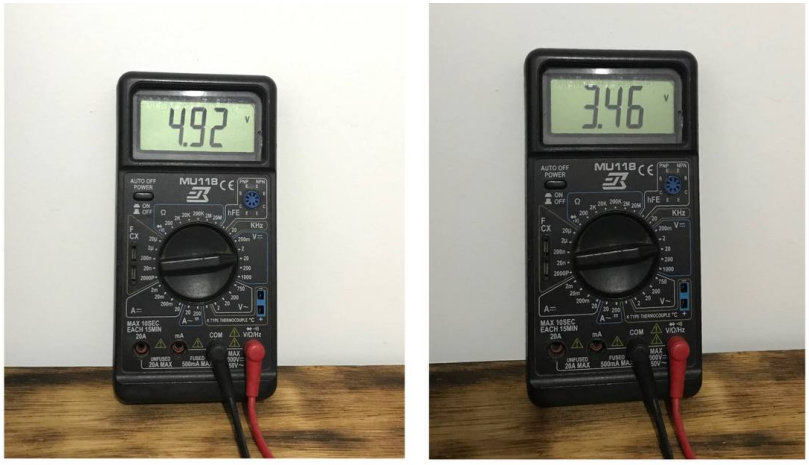


Figura 12: Variación del potenciómetro para verificar su correcto funcionamiento (Creación propia)

Por otra parte, el funcionamiento del botón para habilitar las comunicaciones USART/USB se muestra a continuación una vez que se vario casi por completo el potenciómetro. El Discovery Kit con el que se estaba trabajando integra dos botones. El primero es el de RESET, y el segundo el USER, el cual se utiliza para esta parte. Para esto, se tomaron pruebas en consola mientras se corría el archivo en Python. En la figura (13) se aprieta el botón para activar la transmisión de datos y se logra visualizar mediante un LED parpadeante de la misma tarjeta. Finalmente, en la figura (14), se aprieta el botón de nuevo hasta que el LED deje de encenderse y se aprecia en la terminal que no presenta datos porque su envío se encuentra deshabilitado.

```
{"x": "0", "y": "0", "z": "0", "Battery Low": "Si", "Battery Level": "1"}
data published to thingsboard

{"x": "0", "y": "0", "z": "0", "Battery Low": "Si", "Battery Level": "1"}
data published to thingsboard

{"x": "0", "y": "0", "z": "0", "Battery Low": "Si", "Battery Level": "1"}
data published to thingsboard

{"x": "0", "y": "0", "z": "0", "Battery Low": "Si", "Battery Level": "1"}
data published to thingsboard

{"x": "0", "y": "0", "z": "0", "Battery Low": "Si", "Battery Level": "1"}
data published to thingsboard

{"x": "0", "y": "0", "z": "0", "Battery Low": "Si", "Battery Level": "1"}
data published to thingsboard

{"x": "0", "y": "0", "z": "0", "Battery Low": "Si", "Battery Level": "1"}
data published to thingsboard
```

Figura 13: Comunicación activada (Creación propia)

```
{"x": "", "y": "", "z": "", "Battery Low": "No", "Battery Level": ""}
data published to thingsboard

{"x": "", "y": "", "z": "", "Battery Low": "No", "Battery Level": ""}
data published to thingsboard

{"x": "", "y": "", "z": "", "Battery Low": "No", "Battery Level": ""}
data published to thingsboard

{"x": "", "y": "", "z": "", "Battery Low": "No", "Battery Level": ""}
data published to thingsboard

{"x": "", "y": "", "z": "", "Battery Low": "No", "Battery Level": ""}
data published to thingsboard

{"x": "", "y": "", "z": "", "Battery Low": "No", "Battery Level": ""}
data published to thingsboard

{"x": "", "y": "", "z": "", "Battery Low": "No", "Battery Level": ""}
data published to thingsboard
```

Figura 14: Comunicación desactivada (Creación propia)

4. Conclusiones y recomendaciones

- La familia de microcontroladores STM32 son perfectamente capaces de aplicaciones que tengan una exigencia superior a nivel de hardware, haciéndolos perfectos para usos más complejos
- El kit discovery provee de periféricos y funcionalidades básicas y avanzadas para el desarrollo de aplicaciones, por lo que iniciar la creación de una aplicación con el mismo contiene una curva de dificultad menor a otros MCUs.
- Implementar funcionalidades de IoT a las aplicaciones permite tener mayor control sobre las funciones deseadas así como una simplicidad para el usuario común.

- La biblioteca de LibOpenCM3 junto con su repertorio de ejemplos hace que el proceso de desarrollo de aplicaciones tenga una curva de aprendizaje menor, debido a que muchas de las funcionalidades se encuentran demostradas en ellas.

5. GIT

El código fuente del presente laboratorio 2 se encuentra disponible en el siguiente enlace: https://github.com/mimiherrera/IE0624-Lab-de-Microcontroladores/tree/Lab4/

Bibliografía

1. STM Microelectronics, *STM32 32-bit ARM Cortex MCUs* [Online]. Disponible en: https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html [Accesado: Mayo 27, 2023]

2. M. M. V. Fallas, *STM32F429/LibOpenCM3*, IE-0624 Laboratorio de Microcontroladores [Accesado: Mayo 18, 2023].

3. STM Microelectronics, 2018. *STM32F427xx STM32F429xx*, DocID024030 Rev 10. [Accesado: Mayo 27, 2023]

4. Luis Joyanes Aguilar. *Internet de las cosas un futuro hiperconectado: 5G, Inteligencia artificial, Big Data, Cloud, Blockchain, Ciberseguridad*. RA-MA Editorial, 2018.

5. IberSontel. *ThingsBoard: plataforma para monitorización de IoT*. Disponible en: https://ibersontel.com/thingsboard-plataforma-para-monitorizacion-de-iot/4730. Accedido el 1 de junio de 2023.

6. Apéndices

Apéndice 1: Hoja de datos del microcontrolador

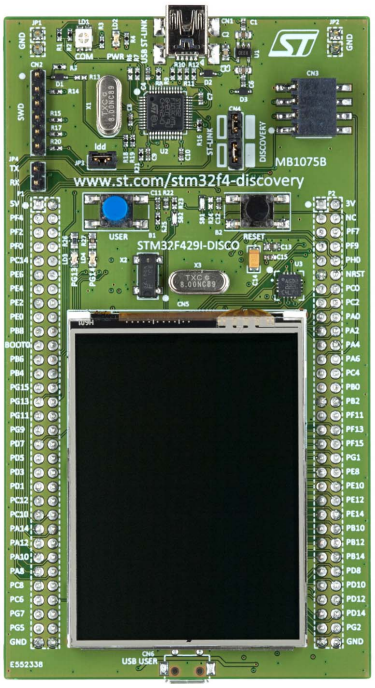
32F429IDISCOVERY

Discovery kit with STM32F429ZI MCU

Data brief

Features

- STM32F429ZIT6 microcontroller featuring 2 Mbytes of Flash memory, 256 Kbytes of RAM in an LQFP144 package
- On-board ST-LINK/V2 on STM32F429I-DISCO (old order code) or ST-LINK/V2-B on STM32F429I-DISC1 (new order code)
- mbed™-enabled (see http://mbed.org) with the new order code only
- USB functions:
  - Debug port
  - Virtual COM port (with new order code only)
  - Mass storage (with new order code only)
- Board power supply: through the USB bus or from an external 3 V or 5 V supply voltage
- 2.4" QVGA TFT LCD
- 64-Mbit SDRAM
- L3GD20, ST-MEMS motion sensor 3-axis digital output gyroscope
- Six LEDs:
  - LD1 (red/green) for USB communication
  - LD2 (red) for 3.3 V power-on
  - Two user LEDs: LD3 (green), LD4 (red)
  - Two USB OTG LEDs: LD5 (green) VBUS and LD6 (red) OC (over-current)
- Two push-buttons (user and reset)
- USB OTG with micro-AB connector
- Extension header for LQFP144 I/Os for a quick connection to the prototyping board and an easy probing
- Comprehensive free software including a variety of examples, part of STM32CubeF4 package or STSW-STM32138, for using legacy standard libraries



1. Picture is not contractual.

Description

The 32F429IDISCOVERY kit leverages the capabilities of the STM32F429 high-performance microcontrollers, to allow users to easily develop rich applications with advanced Graphic User interfaces.

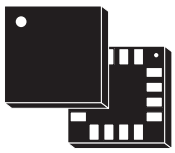
With the latest board enhancement, the new STM32F429I-DISC1 order code has replaced the old STM32F429I-DISCO order code.

Apéndice 2: Hoja de datos del Giroscopio

L3GD20

MEMS motion sensor: three-axis digital output gyroscope

Datasheet - production data



LGA-16 (4x4x1 mm)

Features

- Three selectable full scales (250/500/2000 dps)
- I^2C/SPI digital output interface
- 16 bit-rate value data output
- 8-bit temperature data output
- Two digital output lines (interrupt and data ready)
- Integrated low- and high-pass filters with user-selectable bandwidth
- Wide supply voltage: 2.4 V to 3.6 V
- Low voltage-compatible IOs (1.8 V)
- Embedded power-down and sleep mode
- Embedded temperature sensor
- Embedded FIFO
- High shock survivability
- Extended operating temperature range (-40 °C to +85 °C)
- ECOPACK® RoHS and “Green” compliant

Applications

- Gaming and virtual reality input devices
- Motion control with MMI (man-machine interface)
- GPS navigation systems
- Appliances and robotics

Description

The L3GD20 is a low-power three-axis angular rate sensor.

It includes a sensing element and an IC interface capable of providing the measured angular rate to the external world through a digital interface (I^2C/SPI).

The sensing element is manufactured using a dedicated micro-machining process developed by STMicroelectronics to produce inertial sensors and actuators on silicon wafers.

The IC interface is manufactured using a CMOS process that allows a high level of integration to design a dedicated circuit which is trimmed to better match the sensing element characteristics. The L3GD20 has a full scale of ±250/±500/ ±2000 dps and is capable of measuring rates with a user-selectable bandwidth.

The L3GD20 is available in a plastic land grid array (LGA) package and can operate within a temperature range of -40 °C to +85 °C.

Table 1. Device summary

Order code	Temperature range (°C)	Package	Packing
L3GD20	-40 to +85	LGA-16 (4x4x1 mm)	Tray
L3GD20TR	-40 to +85	LGA-16 (4x4x1 mm)	Tape and reel

Apéndice 3: Hoja de datos general de los STM32F4

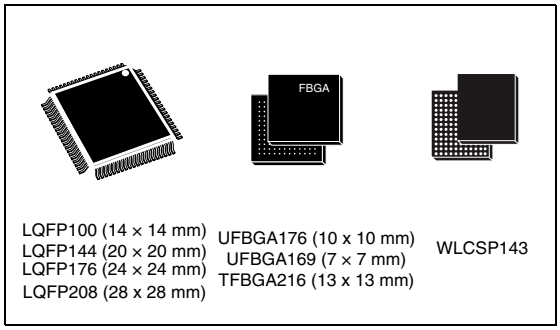
STM32F427xx STM32F429xx

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

Features

- Core: Arm® 32-bit Cortex®-M4 CPU with FPU, Adaptive real-time accelerator (ART Accelerator™) allowing 0-wait state execution from Flash memory, frequency up to 180 MHz, MPU, 225 DMIPS/1.25 DMIPS/MHz (Dhrystone 2.1), and DSP instructions
- Memories
  - Up to 2 MB of Flash memory organized into two banks allowing read-while-write
  - Up to 256+4 KB of SRAM including 64-KB of CCM (core coupled memory) data RAM
  - Flexible external memory controller with up to 32-bit data bus: SRAM, PSRAM, SDRAM/LPSDR SDRAM, Compact Flash/NOR/NAND memories
- LCD parallel interface, 8080/6800 modes
- LCD-TFT controller with fully programmable resolution (total width up to 4096 pixels, total height up to 2048 lines and pixel clock up to 83 MHz)
- Chrom-ART Accelerator™ for enhanced graphic content creation (DMA2D)
- Clock, reset and supply management
  - 1.7 V to 3.6 V application supply and I/Os
  - POR, PDR, PVD and BOR
  - 4-to-26 MHz crystal oscillator
  - Internal 16 MHz factory-trimmed RC (1% accuracy)
  - 32 kHz oscillator for RTC with calibration
  - Internal 32 kHz RC with calibration
- Low power
  - Sleep, Stop and Standby modes
  - V_{BAT} supply for RTC, 20×32 bit backup registers + optional 4 KB backup SRAM
- 3×12-bit, 2.4 MSPS ADC: up to 24 channels and 7.2 MSPS in triple interleaved mode
- 2×12-bit D/A converters
- General-purpose DMA: 16-stream DMA controller with FIFOs and burst support
- Up to 17 timers: up to twelve 16-bit and two 32-bit timers up to 180 MHz, each with up to 4 IC/OC/PWM or pulse counter and quadrature (incremental) encoder input



- Debug mode
  - SWD & JTAG interfaces
  - Cortex-M4 Trace Macrocell™
- Up to 168 I/O ports with interrupt capability
  - Up to 164 fast I/Os up to 90 MHz
  - Up to 166 5 V-tolerant I/Os
- Up to 21 communication interfaces
  - Up to 3 × I^2C interfaces (SMBus/PMBus)
  - Up to 4 USARTs/4 UARTs (11.25 Mbit/s, ISO7816 interface, LIN, IrDA, modem control)
  - Up to 6 SPIs (45 Mbits/s), 2 with muxed full-duplex I^2S for audio class accuracy via internal audio PLL or external clock
  - 1 x SAI (serial audio interface)
  - 2 × CAN (2.0B Active) and SDIO interface
- Advanced connectivity
  - USB 2.0 full-speed device/host/OTG controller with on-chip PHY
  - USB 2.0 high-speed/full-speed device/host/OTG controller with dedicated DMA, on-chip full-speed PHY and ULPI
  - 10/100 Ethernet MAC with dedicated DMA: supports IEEE 1588v2 hardware, MII/RMII
- 8- to 14-bit parallel camera interface up to 54 Mbytes/s
- True random number generator
- CRC calculation unit
- RTC: subsecond accuracy, hardware calendar
- 96-bit unique ID