Universidad de Costa Rica
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica
IE0624 – Laboratorio de Microcontroladores
II ciclo 2023

Laboratorio 4

STM32: GPIO, ADC, comunicaciones, Iot

Kenny Wu Wen C08592
Oscar Fallas B92861
Grupo 01

Profesor: MSc. Marco Villalta Fallas

28 de noviembre de 2023

Índice

Índice de figuras

1. Resumen

En este laboratorio se van a utilizar el microcontrolador STM32F429 Discovery Kit y la biblioteca libopencm3 para la creación de un sismógrafo digital a partir de configurar su giroscopio L3GD20 y se mostrada su información en la pantalla LCD/TFT ILI9341. Además, se medirá la salud de la batería y también se mostrara en la pantalla esa información y finalmente se transmitira esa información por UART/USB, que estará habilitada por medio de un boton/switch, el cuál se utilizara la plataforma ThingsBoard (IoT) para mostrar la información.

El repositorio para el laboratorio es el siguiente: GIT

2. Nota teórica

2.1. STM32F429 Discovery Kit

El STM32F429 Discovery Kit es un paquete con múltiples interfaces y periféricos disponibles para sacar le mejor proceso al STM32F429. Este tiene las siguientes características [1]:

- Microcontrolador STM32F429ZIT6
- Pantalla de 2.4 pulgadas QVGA TFT LCD
- Sensor de movimiento I3G4250D, Giroscopio ST MEMS de 3-ejes
- Dos botones, uno de reset y otro para el usuario
- 6 LEDs
- 64-Mbit SDRAM
- Alimentación por USB o fuente externa de 3-5V
- Procesador ARM 32 bits RISC Cortex-M4
- 2MB flash, 256 KB SRAM
- Consumo bajo de potencia
- Controlador LCD-TFT
- 3x12 bit ADC
- 2X12 convertidor D/A
- 17 timers: 12 timers de 16 bits, 2 de 32 bits
- Controladores DMA

En la figura 1, tenemos un diagrama de bloques de la placa. Luego, en la figura 2 y 3 las características electrónicas de la placa.

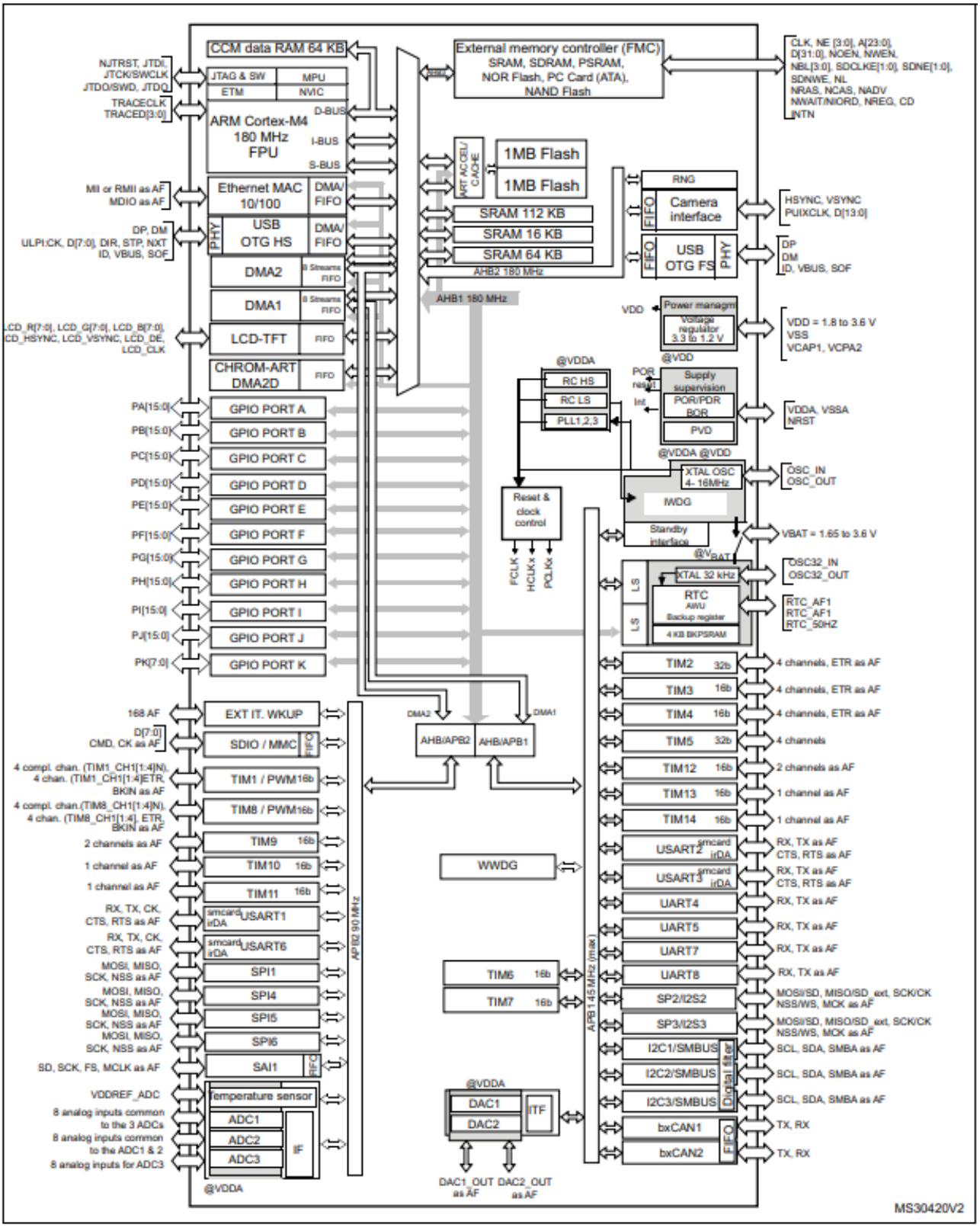

Figura 1: Diagrama de bloques del STM32F429ZIT6

En la figura 5 y 6, se muestra el pinmux de cada pin y su tensión de entrada [2].

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)*		

1. All main power (V_{DD}, V_{DDA}) and ground (V_{SS}, V_{SSA}) pins must always be connected to the external power supply, in the permitted range.
2. V_{IN} maximum value must always be respected. Refer to *Table 15* for the values of the maximum allowed injected current.

Figura 2: Características de tensión del STM32F429ZIT6

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 STM32F429ZIT6

2.2. Protocolos de comunicaciones

2.2.1. UART

Las siglas UART significan Receptor-Transmisor Asincronico Universal o en ingles *Universal Asyncronic Receptor-Transmitter*. Es un hardware que se encarga para la comunicación serial entre dos dispositivos UART significa Receptor/Transmisor Asíncrono Universal. Es un dispo-

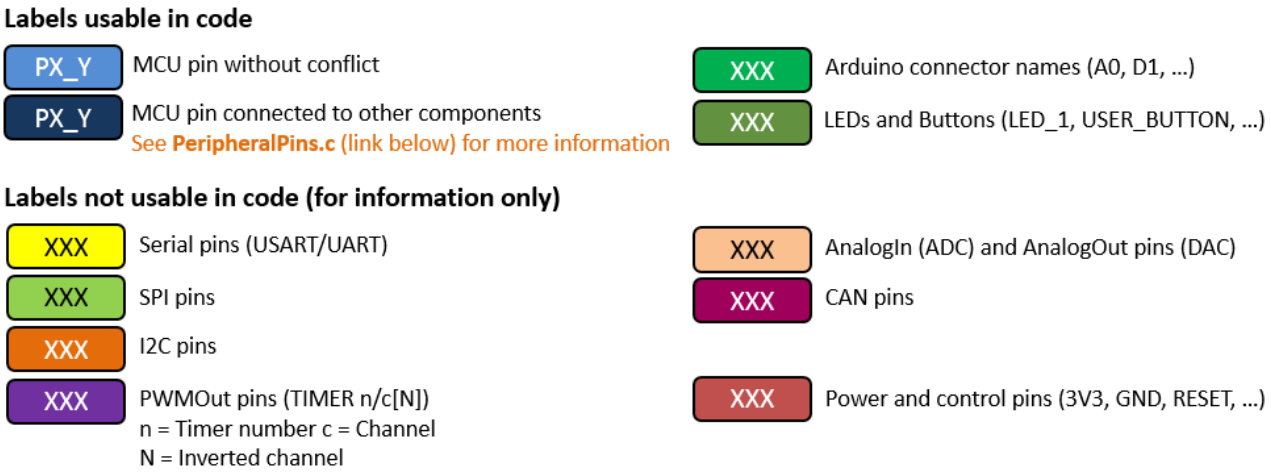

Figura 4: STM Pinmux etiquetas

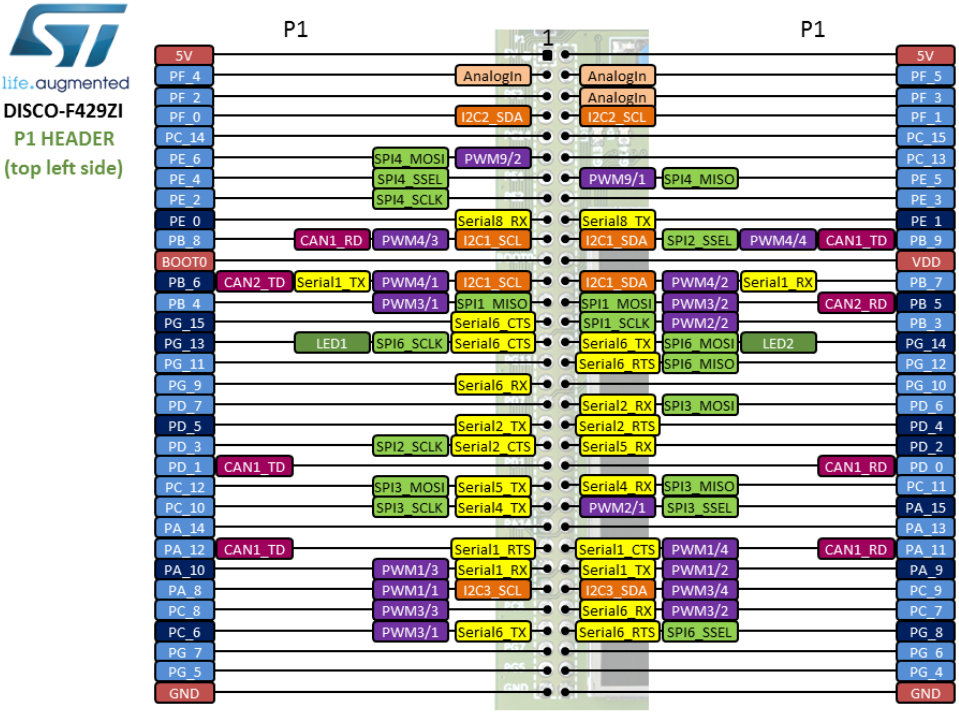

Figura 5: STM Pinmux parte 1

sitivo de hardware (o circuito) utilizado para la comunicación en serie entre dos dispositivos. En el Arduino tenemos un pin Tx y otro Rx que son para la comunicación UART [3].

El como funciona el protocolo de comunicación UART, es simple, se envían datos en serie y luego se reconstruyen. Estos datos en realidad son paquetes se dividen en bytes y luego a bits y los bytes son etiquetas que significan cuando empiezan, en donde se ubican los datos y donde termina el paquete de enviarse.

El UART se requiere que los dos dispositivos a comunicarse tengan los mismos parámetros de comunicación UART, como la tasa de baudios, la longitud de datos, el bit de paridad, el numero de bits de parada y el control de flujo [3].

- Tasa de baudios: numero de bits por segundo que un dispositivos logra transmitir/recibir, generalmente los valores utilizados son 1200, 2400, 4800, 9600, 19200, 38400, 57600 y

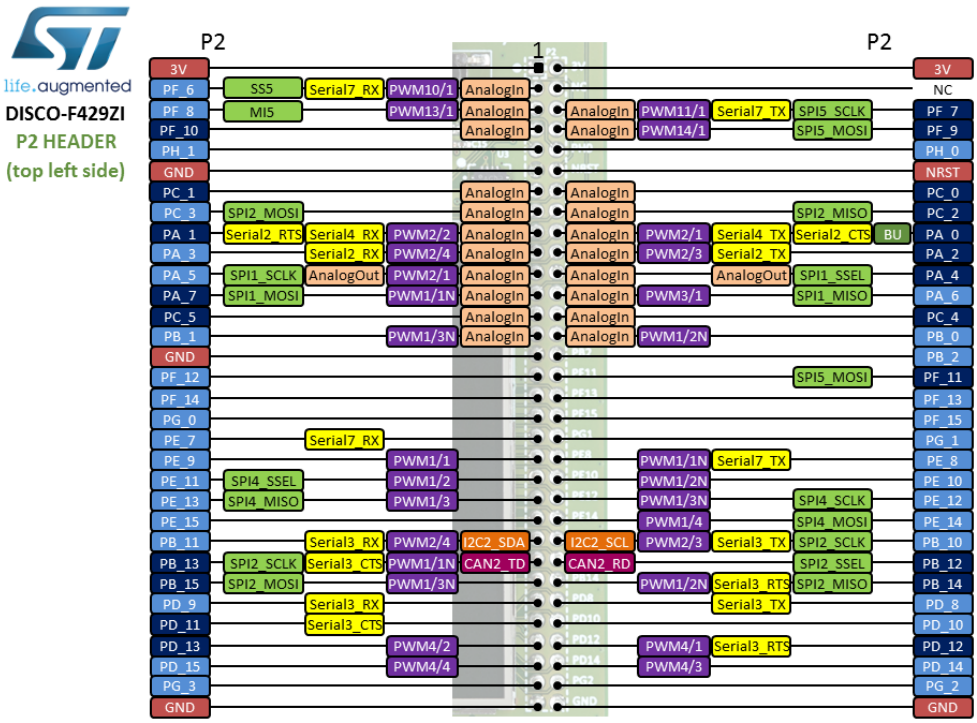

Figura 6: STM Pinmux parte 2

115200 bps.

- Longitud de datos: cantidad de bits por bytes de datos.
- Bit de paridad: bit agregado a los datos transmitidos, que permiten al receptor identificar la cantidad de 1's en los datos transmitidos es par o impar.
- Control de flujo: se utiliza para controlar la cantidad de datos enviados y recibidos con el fin evitar el riesgo a perder datos.

2.3. L3GD20

El L3GD20 es un sensor angular de baja potencia con tres ejes que viene integrada en el placa STM de este laboratorio. Algunas de sus características son [4]:

- Salida interfaz digital I2C/SPI
- Salida de datos de 16 bits
- Salida de temperatura de 8 bits
- Filtro pasa alto y pasa bajo configurable
- Sensor de temperatura incrustado
- Conectado en la STM32F429 Discovery Kit al SPI5

Algunas aplicaciones:

- Gaming y realidad aumentada

- GPS
- Aplicaciones de robots

Sus características mecánicas y eléctricas los podemos encontrar en las figuras 7 y 8, además se puede observar la tabla de registros del dispositivo en la figura 9

Symbol	Parameter	Test condition	Min.	Typ.(2)	Max.	Unit
FS	Measurement range	User-selectable		±250		dps
				±500		
				±2000		
So	Sensitivity	FS = 250 dps		8.75		mdps/digit
		FS = 500 dps		17.50		
		FS = 2000 dps		70		
SoDr	Sensitivity change vs. temperature	From -40 °C to +85 °C		±2		%
DVoff	Digital zero-rate level	FS = 250 dps		±10		dps
		FS = 500 dps		±15		
		FS = 2000 dps		±75		
OffDr	Zero-rate level change vs. temperature	FS = 250 dps		±0.03		dps/°C
		FS = 2000 dps		±0.04		dps/°C
NL	Non linearity	Best fit straight line		0.2		% FS
Rn	Rate noise density			0.03		dps/(√Hz)
ODR	Digital output data rate			95/190/ 380/760		Hz
Top	Operating temperature range		-40		+85	°C

Figura 7: Caracteristicas mecanicas del L3GD20

Symbol	Parameter	Test condition	Min.	Typ.(2)	Max.	Unit
Vdd	Supply voltage		2.4	3.0	3.6	V
Vdd_IO	I/O pins supply voltage(3)		1.71		Vdd+0.1	V
Idd	Supply current			6.1		mA
IddSL	Supply current in sleep mode(4)	Selectable by digital interface		2		mA
IddPdn	Supply current in power-down mode	Selectable by digital interface		5		µA
VIH	Digital high level input voltage		0.8*Vdd_IO			V
VIL	Digital low level input voltage				0.2*Vdd_IO	V
Top	Operating temperature range		-40		+85	°C

Figura 8: Características eléctricas del L3GD20

Name	Type	Register address		Default
		Hex	Binary	
Reserved	-	00-0E	-	-
WHO_AM_I	r	0F	000 1111	11010100
Reserved	-	10-1F	-	-
CTRL_REG1	rw	20	010 0000	00000111
CTRL_REG2	rw	21	010 0001	00000000
CTRL_REG3	rw	22	010 0010	00000000
CTRL_REG4	rw	23	010 0011	00000000
CTRL_REG5	rw	24	010 0100	00000000
REFERENCE	rw	25	010 0101	00000000
OUT_TEMP	r	26	010 0110	output
STATUS_REG	r	27	010 0111	output
OUT_X_L	r	28	010 1000	output
OUT_X_H	r	29	010 1001	output
OUT_Y_L	r	2A	010 1010	output
OUT_Y_H	r	2B	010 1011	output
OUT_Z_L	r	2C	010 1100	output
OUT_Z_H	r	2D	010 1101	output
FIFO_CTRL_REG	rw	2E	010 1110	00000000
FIFO_SRC_REG	r	2F	010 1111	output
INT1_CFG	rw	30	011 0000	00000000
INT1_SRC	r	31	011 0001	output
INT1_TSH_XH	rw	32	011 0010	00000000
INT1_TSH_XL	rw	33	011 0011	00000000
INT1_TSH_YH	rw	34	011 0100	00000000
INT1_TSH_YL	rw	35	011 0101	00000000
INT1_TSH_ZH	rw	36	011 0110	00000000
INT1_TSH_ZL	rw	37	011 0111	00000000
INT1_DURATION	rw	38	011 1000	00000000

Figura 9: Tabla de registros del L3GD20

2.4. LCD/TFT ILI9341

En el STM32F429 Discovery Kit, se tiene una pantalla LCD-TFT con las siguientes características [5]:

- Pantalla tactil a colores
- Resolución 240x320 píxeles
- Controlador gráfico ILI9341 y controlador tactil XPT2046
- Interfaz de comunicación I2C/SPI

En la figura 10 un diagrama de bloques del ILI9341.

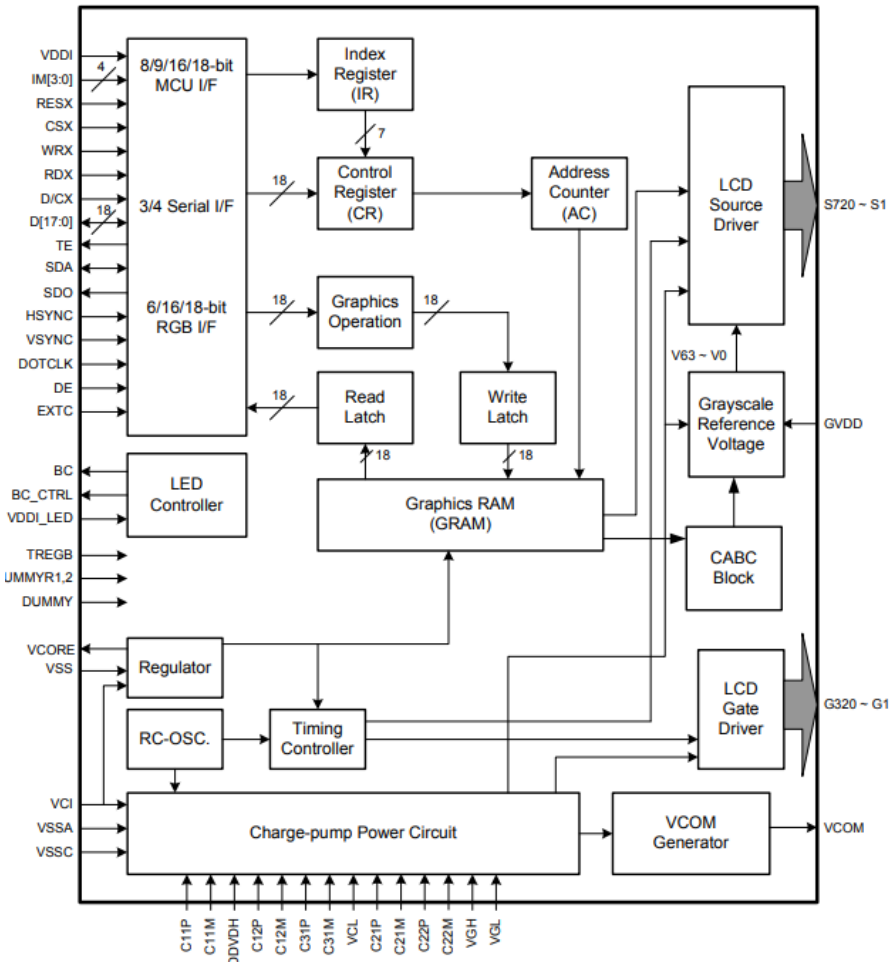

Figura 10: Diagrama de bloques del ILI9341

2.5. LibOpenCM3

Las librerías LibOpenCM3, también conocido como *libopenstm32*, es un proyecto open source de librerías de firmware para poder manejar/utilizar microcontroladores STM32, Toshiba TX03, ARM Cortex-M3, EFM32 y muchos otros [6]. Al utilizar esta librería nos ahorramos tener que escribir funciones de setup, escribir, leer y otros métodos que ocupemos a la hora de escribir firmware para un microcontrolador en nuestro caso el STM.

2.6. Thingsboard

Thingsboard es una plataforma de las cosas de Internet (IoT) open source que permite administrar, controlar y desarrollar proyectos de IoT, como crear aplicaciones que se controlen desde Internet y se comuniquen con un microcontrolador y mucho más [7].

Una forma de comunicarse con ThingsBoard es utilizando las librerias de MQTT con python, como se observa en la figura 11. MQTT es un protocolo de mensajeria estándar para IoT. Está diseñado para transportar mensajes livianos para hacer publish/suscribe a la plataforma IoT [8].

2.6.1. Thingsboard.py

Para este laboratorio usaremos las librerías MQTT con python enviar la información recibida por UART del STM, al Thingsboard de la escuela de ingeniaría eléctrica.

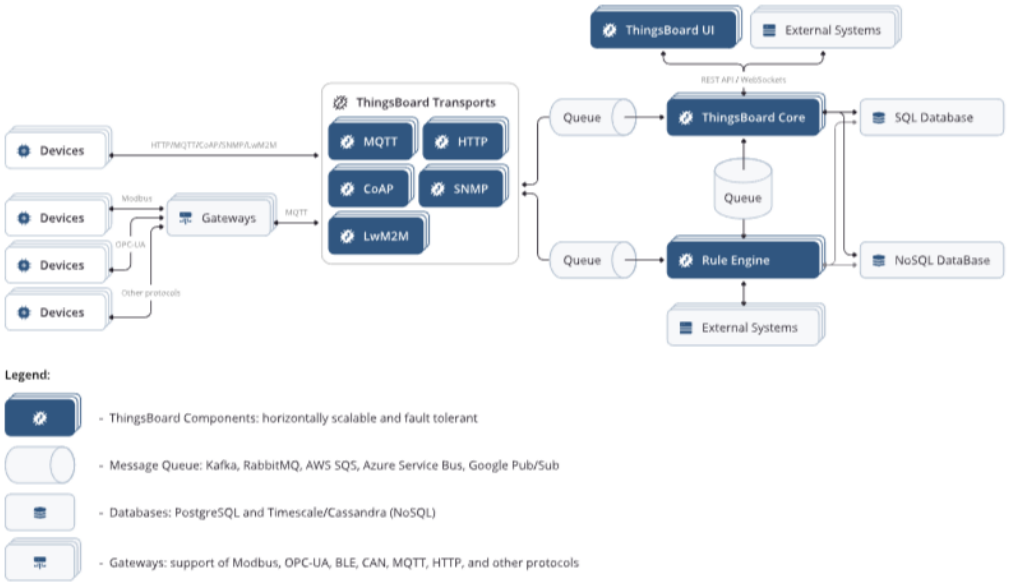

Figura 11: Diagrama de ThingsBoard

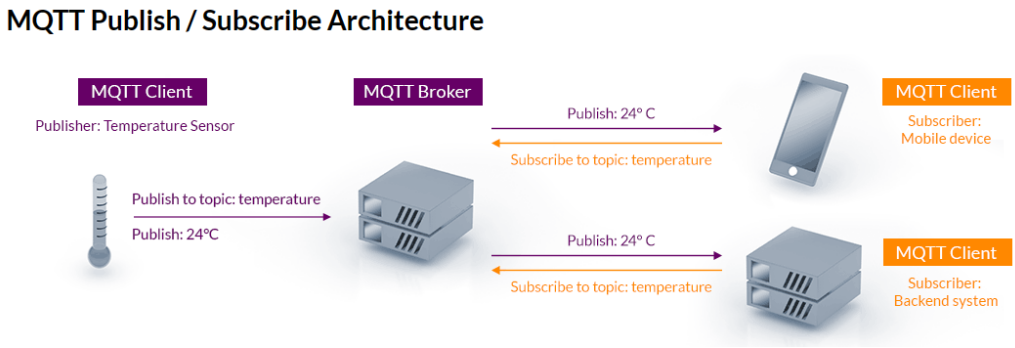

Figura 12: Diagrama de MQTT

En el apéndice [5] se puede ver el código completo, pero el algoritmo se puede resumir de la siguiente manera en base el diagrama de flujo de la figura 13:

1. Se importan las librerías necesarias
2. Se configura el puerto serial donde obtendremos los datos a enviar
3. Configuramos el cliente y sus métodos de on_connect, on_disconnect, on_publish y on_log.
4. Llamamos el método de connect con broker=ïot.eie.ucr.ac.crz port=1883
5. Esperamos que el cliente se conecte
6. Si se conecta continuamos, de lo contrario se sale del flujo y termina
7. Al conectarse, empezamos a enviar los datos recibidos por medio de la función publish con topic="v1/devices/me/telemetryz data=json

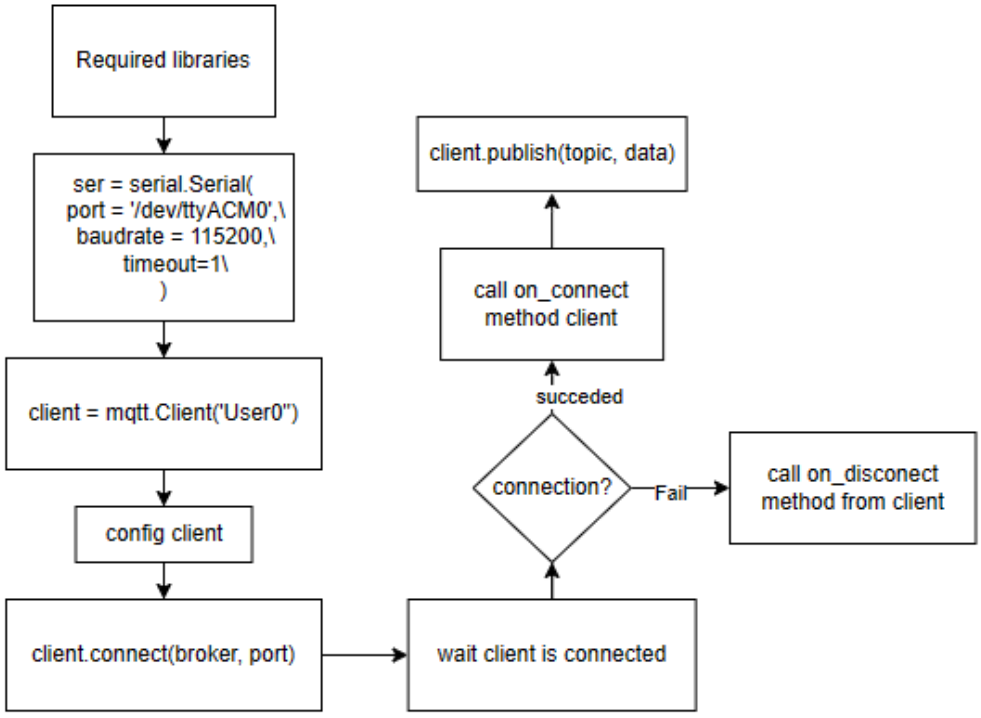

Figura 13: Diagrama de MQTT

2.7. Medición de salud de batería

Vamos a tener una batería de 9V pero la entrada del PA6 del STM32, como se observa en la figura 6, es solo de 3.33V, por lo que vamos a necesitar reducir con un circuito de división de tensión los 9V:

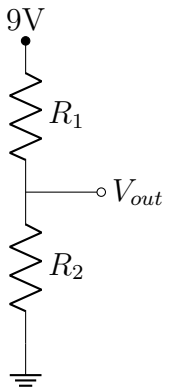

Figura 14: División de tensión 9V a 3.33V incógnitas

Suponiendo $R_1 = 2\text{k}\Omega$

$$V_{out} = \frac{R_2}{R_1 + R_2} \cdot 9$$
$$3.33 = \frac{R_2}{R_2 + 2000} \cdot 9$$

Despejando R_2:

$$R_2 = 1365\Omega \tag{1}$$

El circuito quedaría:

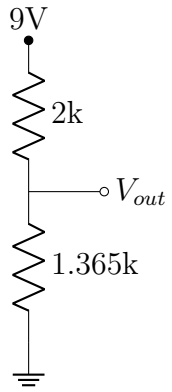

Figura 15: División de tensión 9V a 3.33V

2.8. Firmware

Para la construcción del Firmware se basó en el ejemplo la librería libopencm3 donde se utilizaron específicamente los ejemplos de spi, lcd-serial, adc-dac-printf, con el fin de realizar el setup del reloj, sdram, gpio y sintaxís de la librería. Esto permitió un implementación más segura, rápida y eficiente. A continuación en la figura 16, se muestra el flujo general del firmware

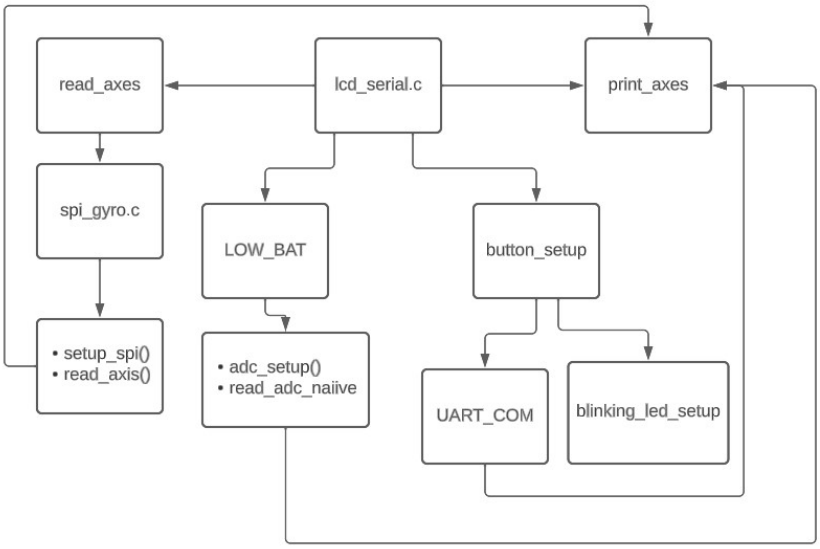

Figura 16: Diagrama de flujo del firmware

2.8.1. Lectura al giroscopio

Para esta función del firmware, la cuál se encuentra en el archivo spi-gyro.c se realiza incialmente del protocolo de comunicación serial, lo cuál realiza lo siguiente:

- Establece el baudrate.
- Asigna los pines F7, F8 y F9 como lecturas del giroscopio integrado
- Activa el reloj del periférico spi.

- Configura los parámetros de trasmisión de datos
- Activa la comunicación SPI

```
void setup_spi(){
    //Perifericos reloj
    rcc_periph_clock_enable(RCC_SPI5);
    rcc_periph_clock_enable(RCC_GPIOC);
        rcc_periph_clock_enable(RCC_GPIOF);


    //GPIO
        gpio_mode_setup(GPIOC, GPIO_MODE_OUTPUT, GPIO_PUPD_NONE, GPIO1);
    gpio_set(GPIOC, GPIO1);
    gpio_mode_setup(GPIOF, GPIO_MODE_AF, GPIO_PUPD_NONE, GPIO7 | GPIO8 |
       GPIO9);
        gpio_set_af(GPIOF, GPIO_AF5, GPIO7 | GPIO8 | GPIO9);

    //Set de SPI inicializacion
    spi_set_master_mode(SPI5);
        spi_set_baudrate_prescaler(SPI5, SPI_CR1_BR_FPCLK_DIV_64);
        spi_set_clock_polarity_0(SPI5);
        spi_set_clock_phase_0(SPI5);
        spi_set_full_duplex_mode(SPI5);
        spi_set_unidirectional_mode(SPI5);
        spi_enable_software_slave_management(SPI5);
        spi_send_msb_first(SPI5);
        spi_set_nss_high(SPI5);
    SPI_I2SCFGR(SPI5) &= ~SPI_I2SCFGR_I2SMOD;
        spi_enable(SPI5);
}
```

A su vez se realiza una función que envía la información necesaria y a su vez recibe los datos obtenidos del cada uno de los registros que almacenan las oscilaciones del giroscopio.

```
struct axis read_axis(){
        struct axis axes;
        uint8_t temp;

        gpio_clear(GPIOC, GPIO1);
        spi_send(SPI5, GYR_WHO_AM_I | GYR_RNW);
        spi_read(SPI5);
        spi_send(SPI5, 0);
        temp=spi_read(SPI5);
        gpio_set(GPIOC, GPIO1);

        gpio_clear(GPIOC, GPIO1);
        spi_send(SPI5, GYR_STATUS_REG | GYR_RNW);
        spi_read(SPI5);
        spi_send(SPI5, 0);
        temp=spi_read(SPI5);
        gpio_set(GPIOC, GPIO1);

        gpio_clear(GPIOC, GPIO1);
```

```
	spi_send(SPI5, GYR_OUT_TEMP | GYR_RNW);
	spi_read(SPI5);
	spi_send(SPI5, 0);
	temp=spi_read(SPI5);
	gpio_set(GPIOC, GPIO1);

	gpio_clear(GPIOC, GPIO1);
	spi_send(SPI5, GYR_OUT_X_L | GYR_RNW);
	spi_read(SPI5);
	spi_send(SPI5, 0);
	axes.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);
	axes.x|=spi_read(SPI5) << 8;
	gpio_set(GPIOC, GPIO1);

	gpio_clear(GPIOC, GPIO1);
	spi_send(SPI5, GYR_OUT_Y_L | GYR_RNW);
	spi_read(SPI5);
	spi_send(SPI5, 0);
	axes.y=spi_read(SPI5);
	gpio_set(GPIOC, GPIO1);

	gpio_clear(GPIOC, GPIO1);
	spi_send(SPI5, GYR_OUT_Y_H | GYR_RNW);
	spi_read(SPI5);
	spi_send(SPI5, 0);
	axes.y|=spi_read(SPI5) << 8;
	gpio_set(GPIOC, GPIO1);

	gpio_clear(GPIOC, GPIO1);
	spi_send(SPI5, GYR_OUT_Z_L | GYR_RNW);
	spi_read(SPI5);
	spi_send(SPI5, 0);
	axes.z=spi_read(SPI5);
	gpio_set(GPIOC, GPIO1);

	gpio_clear(GPIOC, GPIO1);
	spi_send(SPI5, GYR_OUT_Z_H | GYR_RNW);
	spi_read(SPI5);
	spi_send(SPI5, 0);
	axes.z|=spi_read(SPI5) << 8;
	gpio_set(GPIOC, GPIO1);

	axes.x = axes.x*L3GD20_SENSITIVITY_500DPS;
	axes.y = axes.y*L3GD20_SENSITIVITY_500DPS;
	axes.z = axes.z*L3GD20_SENSITIVITY_500DPS;

	return axes;
```

2.8.2. Funciones interactivas

Estas funciones se utilizan con el fin de permitir al usuario un control interactivo de diferentes situaciones pasando el sistema. La primera de ellas es la lectura de la salud de la batería mediante la función $LOW\_BAT()$, está función realiza lo siguiente:

- Configuración de la lectura de los pines analógicos para el ADC.
- Lectura analógica del pin A6 el cuál se realiza su conversión a digital con el fin de determinar el porcentaje real de la batería.
- Activa el LED G14 cuando el porcentaje es menor a 70 %.
- Envío de la lectura de la batería para su despliegue posterior

```
static void adc_setup(void)
{
        rcc_periph_clock_enable(RCC_ADC1);
        rcc_periph_clock_enable(RCC_GPIOA);
        gpio_mode_setup(GPIOA, GPIO_MODE_ANALOG, GPIO_PUPD_NONE, GPIO6);

        adc_power_off(ADC1);
        adc_disable_scan_mode(ADC1);
        adc_set_sample_time_on_all_channels(ADC1, ADC_SMPR_SMP_3CYC);

        adc_power_on(ADC1);

}

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;
}
void LOW_BAT(int* ptr_bat){
        uint16_t analog_read;
        char analog[20];
        analog_read = read_adc_naiive(6)*(9.00/4095);
        *ptr_bat = analog_read*(100/9.00);

        if(*ptr_bat <= 78){
                gpio_set(GPIOG, GPIO14);
                LOW_BATERY = 1;
        }else{
                gpio_clear(GPIOG, GPIO14);
                LOW_BATERY = 0;
        }
}
```

2.8.3. button_setup

Esta función se encarga de setear el puerto PA0 como entrada para el boton de manera que lo usaremos para activar o desactivar la comunicación UART.

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

        /* Ponemos el GPIO0 (pin PA0) del puerto A como input*/
        gpio_mode_setup(GPIOA, GPIO_MODE_INPUT, GPIO_PUPD_NONE, GPIO0);
}
```

2.8.4. blinkingLED_setup

Para poder utilizar los LEDs, se configurara con la función blinkingLED_setup, el cuál va a activar el GPIO13 Y GPIO14 (PG13 y PG14 LEDs) como salidas, de manera que podamos acitvar o desactivar los LEDs.

```
void blinkingLED_setup(void){
        /* Enable GPIOG clock. */
        rcc_periph_clock_enable(RCC_GPIOG);

        /* Set GPIO13 (in GPIO port G) to 'output push-pull' (pin PG13,
           el cual
        corresponde a la una LED) */
        gpio_mode_setup(GPIOG, GPIO_MODE_OUTPUT,
                        GPIO_PUPD_NONE, GPIO13);
        gpio_mode_setup(GPIOG, GPIO_MODE_OUTPUT,
                        GPIO_PUPD_NONE, GPIO14);
}
```

2.8.5. UART_COMM

Otra función, es la comunicación serial mediante protocolo UART para el envío de los datos obtenidos. Para esto se realizaron las siguientes configuraciones y tareas:

- Configuración y uso de console.c tomado del ejemplo de la librería.
- Conexión con la lectura de los datos del giroscopio y salud de la batería
- Despliegue de los datos en consola.
- Activa el LED PG13 de manera parpadeante una vez la comunicación se estableció.

```
void UART_COMM(struct axis axes, int batery) {
        //Conversion de int a str
        char X[20], Y[20], Z[20], batery_s[20], low_bat[20];

        sprintf(X, "%d", axes.x);
        sprintf(Y, "%d", axes.y);
        sprintf(Z, "%d", axes.z);
        sprintf(batery_s, "%d", batery);
        sprintf(low_bat, "%d", LOW_BATERY);
```

```
        gpio_toggle(GPIOG, GPIO13);

        console_puts(X);
        console_puts("\t");
        console_puts(Y);
        console_puts("\t");
        console_puts(Z);
        console_puts("\t");
        console_puts(batery_s);
        console_puts("\t");
        console_puts(low_bat);
        console_puts("\n");
        msleep(500);
}

void blinkingLED_setup(void){
        /* Enable GPIOG clock. */
        rcc_periph_clock_enable(RCC_GPIOG);

        /* Set GPIO13 (in GPIO port G) to 'output push-pull' (pin PG13,
           el cual
        corresponde a la una LED) */
        gpio_mode_setup(GPIOG, GPIO_MODE_OUTPUT,
                        GPIO_PUPD_NONE, GPIO13);
        gpio_mode_setup(GPIOG, GPIO_MODE_OUTPUT,
                        GPIO_PUPD_NONE, GPIO14);
}

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

        /* Ponemos el GPIO0 (pin PA0) del puerto A como input*/
        gpio_mode_setup(GPIOA, GPIO_MODE_INPUT, GPIO_PUPD_NONE, GPIO0);
```

2.8.6. Imprimir lecturas en LCD

La función *print_axes*() permite el despliegue de todos los datos obtenido mediante *read_axes*(), *LOW_BAT*() con el fin de desplegarlos en la pantalla integrada. Esta función hace uso de la configuración del ejemplo lcd-serial.c del ejemplo de la librería, donde se construyen las configuraciones necesarias para el envío de por medio del protocolo SPI a la pantalla. Esta función recibe cada uno de los datos, y los imprime de manera ordena configurando su posición, tamaño de letra y variable. Además, un punto importante es que la función gfx_puts únicamente puede recibir variables de tipo char por lo que esta función realiza para conversión de los datos.

```
void print_axes(struct axis axes, int COMMLEN, int batery){
        //Conversion de int a str
        char X[20], Y[20], Z[20], bat[20];

        sprintf(X, "%d", axes.x);
        sprintf(Y, "%d", axes.y);
        sprintf(Z, "%d", axes.z);
```

```
        sprintf(bat, "%d", batery);

        gfx_init(lcd_draw_pixel, 240, 320);
        gfx_fillScreen(LCD_GREY);
        gfx_fillRoundRect(10, 10, 220, 220, 5, LCD_WHITE);
        gfx_setCursor(15, 25);
        gfx_setTextSize(2);
        gfx_puts("LECTURAS:");
        gfx_setTextSize(2);
        gfx_setCursor(15, 85);
        gfx_puts("X::");
        gfx_setCursor(100, 85);
        gfx_puts(X);
        gfx_setTextSize(2);
        gfx_setCursor(15, 115);
        gfx_puts("Y::");
        gfx_setCursor(100, 115);
        gfx_puts(Y);
        gfx_setCursor(15, 145);
        gfx_puts("Z::");
        gfx_setCursor(100, 145);
        gfx_puts(Z);
        gfx_setCursor(15, 175);
        gfx_setTextSize(1);
        if (COMM_EN) {
                gfx_puts("SERIAL COMMS: ON");
        }else {
                gfx_puts("SERIAL COMMS: OFF");
        }
        gfx_setCursor(15, 205);
        gfx_setTextSize(1);
        gfx_puts("Batery:");
        gfx_puts(bat);
        lcd_show_frame();
}
```

2.8.7. Función Principal y configuración global

Dado que se requieren de datos archivos y diferentes funciones se encontró necesario realizar una función general que permita el llamado e inicialización de cada uno de los elementos necesarios para el sistema. Entre ellos se encuentran:

- Configuración del SPI
- Configuración de la lectura del botón
- Configuración de LEDS.
- Inicialización de protocolo SPI.

```
void setup(){
        setup_spi();
        button_setup();
        blinkingLED_setup();
```

```
        adc_setup();

        gpio_clear(GPIOC, GPIO1);
        spi_send(SPI5, GYR_CTRL_REG1);
        spi_read(SPI5);
        spi_send(SPI5, GYR_CTRL_REG1_PD | GYR_CTRL_REG1_XEN |
                        GYR_CTRL_REG1_YEN | GYR_CTRL_REG1_ZEN |
                        (3 << GYR_CTRL_REG1_BW_SHIFT));
        spi_read(SPI5);
        gpio_set(GPIOC, GPIO1);

    gpio_clear(GPIOC, GPIO1);
        spi_send(SPI5, GYR_CTRL_REG4);
        spi_read(SPI5);
        spi_send(SPI5, (1 << GYR_CTRL_REG4_FS_SHIFT));
        spi_read(SPI5);
        gpio_set(GPIOC, GPIO1);

}
```

Finalmente, se realiza la función principal que llama a la configuración de mayor jerarquía del sistema como lo son la configuración del reloj, la sdram, la consola, etc. Además realiza un mensaje de bienvenida que informa al usuario como se encuentra configurada la board. Por último, realiza un bucle infinito que hace lectura de los datos del giroscopio, y batería para enviarlos a la función de despliegue.

```
int main(void)
{
        clock_setup();
        console_setup(115200);
        sdram_init();
        setup();
        lcd_spi_init();
        init_message();
        struct axis lecturas;
        int COMM_EN = 0;
        int batery = 0;
        while(1){
                lecturas = read_axis();
                LOW_BAT(&batery);
                print_axes(lecturas, COMM_EN, batery);
                if (gpio_get(GPIOA, GPIO0)) {
                        COMM_EN = ~COMM_EN;
                }
                if (COMM_EN) {
                        UART_COMM(lecturas, batery);
                }
                gpio_clear(GPIOG, GPIO13);
        }


        return 0;
}
```

3. Desarrollo/Análisis de resultados

3.1. Análisis de funcionalidad electrónica

Como se mencionó anteriormente, la idea del laboratorio es implementar un sismógrafo haciendo uso del giroscopio integrado al kit de desarrollo, y desplegando sus valores utilizando la pantalla LED. A continuación se puede notar, el giroscopio en ceros ya que no se encuentra recibiendo ninguna oscilación por alguno de sus ejes

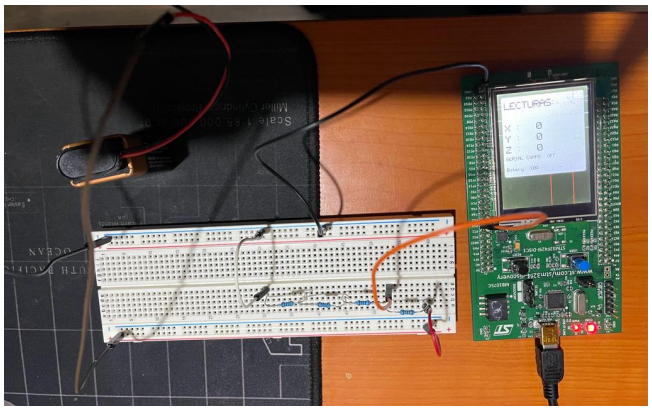

Figura 17: STM32f4-discovery giroscopio en estado de reposo

A su vez, se debe notar que también se encuentra la lectura de la salud de la batería, utilizando el divisor de tensión de 9V a 3.3V que permite la conexión al pin analógico de la tarjeta. La lectura, es del 100 % dado que se utilizó una bateria nueva, por otra parte notar que el LED PG14 no se encuentra encendido dando a entender que la batería no se encuentra en riesgo. A continuación, se adjunta el resultado de cuando el giroscopio recibe oscilaciones por sus ejes:

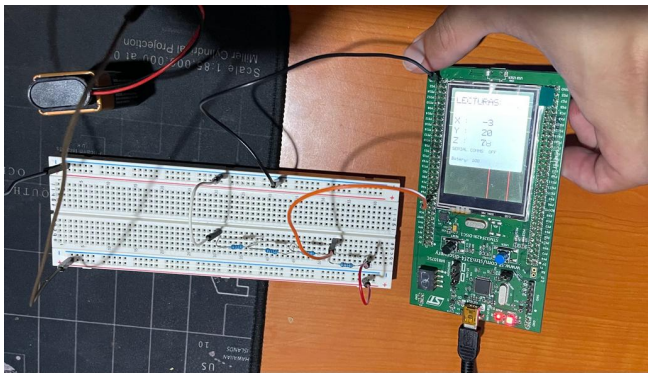

Figura 18: STM32f4-discovery giroscopio en estado de activo

Ahora, en la figura 19 se desconecta la batería, por lo tanto se ve que se realiza la lectura exitosa en la pantalla LED, y se enciende el LED PG14 indicando la alerta.

Finalmente, se configuró la comunicación serial de manera interactiva con el usuario, haciendo uso del botón se habilita la transferencia de información, se imprime en pantalla que la comunicación serial se encuentra active y se configura el LED PG13 parpadeante indicando la transmisión correcta. Mostrado en 20

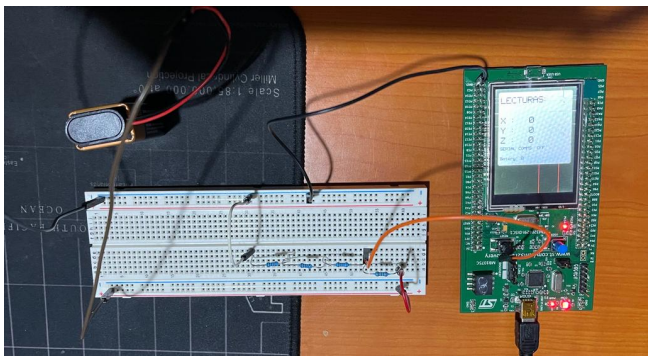

Figura 19: STM32f4-discovery LED de alerta de batería baja

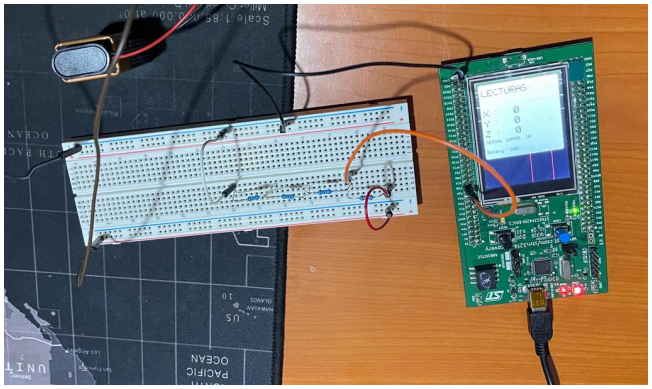

Figura 20: STM32f4-discovery comunicación serial

3.2. Análisis de la funcionalidad del programa

En las siguientes figuras observaremos el funcionamiento de código de Python enviando información a la plataforma IoT ThingsBoard. En la figura 22, se aprecia que al tener una bateria nueva, es 100 % la salud de este

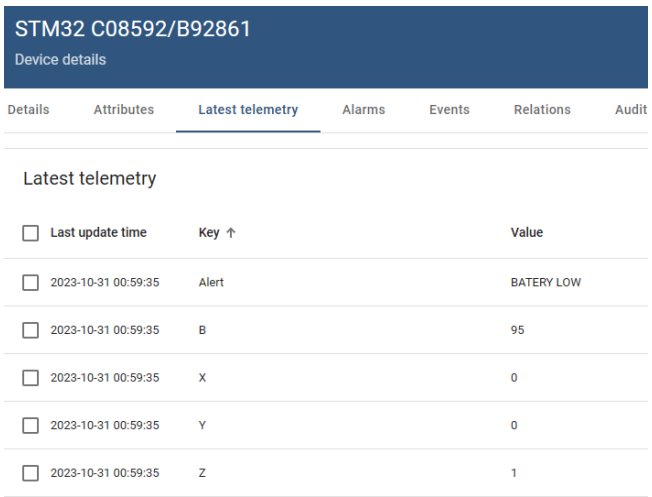

Figura 21: Plataforma Thingsboard EIE recibiendo datos del STM

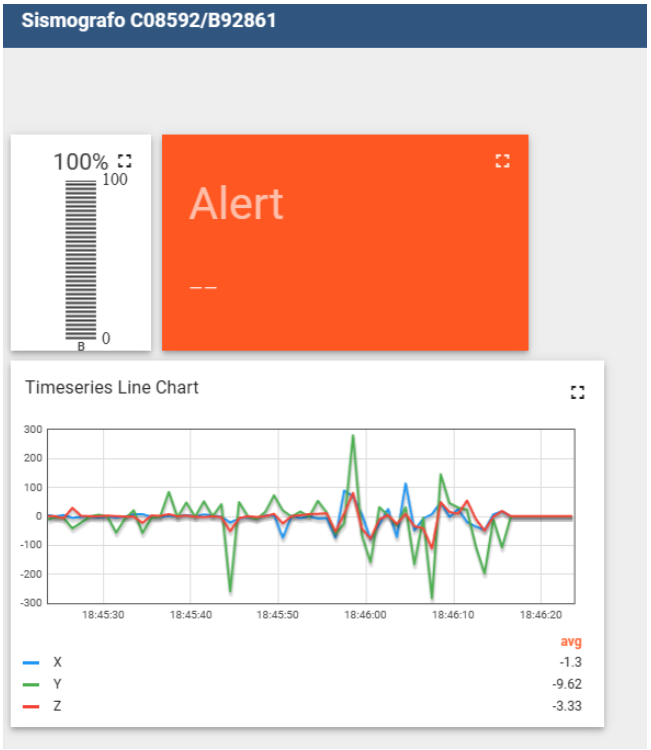

Figura 22: Plataforma Thingsboard EIE recibiendo datos del STM con widgets

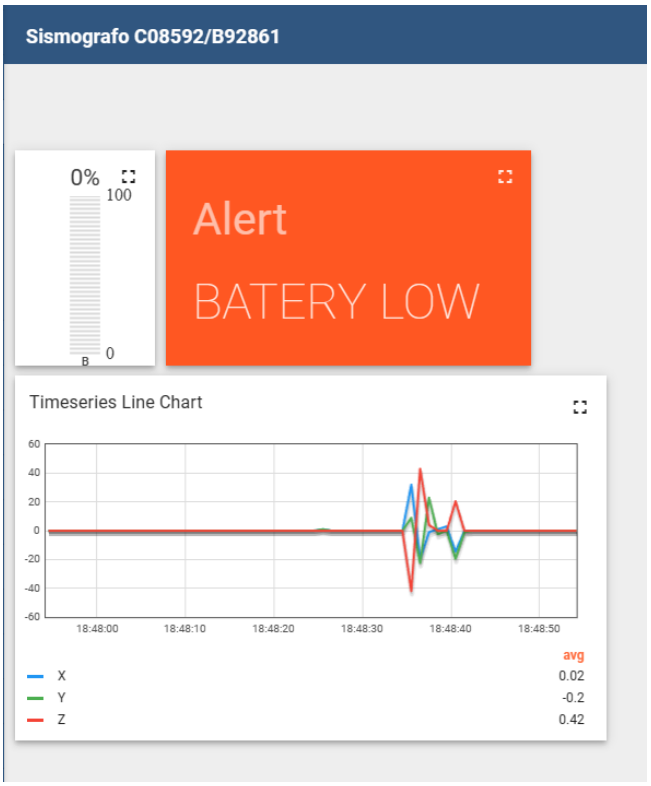

Figura 23: Plataforma Thingsboard EIE recibiendo datos del STM con widgets sin batería conectada

4. Conclusiones y recomendaciones

- Se logro crear un circuito para convertir las tensiones adecuadas para el STM32
- Se pudo comunicarse con el Thingsboard del EIE por medio de MQTT
- Se consiguió enviar los datos del STM32 al Thingsboard
- Hay que tener cuidado a la hora de usar los pines ADC, ya que al ser el microcontrolador muy desgastado los pines, algunos no funcionan correctamente
- Se configura correctamente el giroscopio para mostrar sus valores en la pantalla
- Se pudo mostrar los datos obtenidos en la pantalla LCD
- Se recomienda utilizar ejemplos y mucha documentación de internet para manejar la librería LibOpemCM3

Referencias

[1] STMicroelectronics. (2023) Stm32f427vg datasheet. Accedido el 30 de Octubre de 2023. [Online]. Available: https://www.st.com/resource/en/datasheet/stm32f427vg.pdf

[2] Mbed. (2023) St-discovery-f429zi - stm32f429 discovery kit. [Online]. Available: https://os.mbed.com/platforms/ST-Discovery-F429ZI/

[3] Arduino. (2023) Cómo configurar la comunicación uart en arduino. [Online]. Available: https://arduino.cl/como-configurar-la-comunicacion-uart-en-arduino/

[4] STMicroelectronics. (2023) L3gd20 - low power 3-axis gyroscope, i2c/spi digital output. [Online]. Available: https://www.st.com/en/mems-and-sensors/l3gd20.html

[5] ——. (2023) An4861: Lcd-tft display controller (ltdc) on stm32 mcus. [Online]. Available: https://www.st.com/resource/en/application_note/an4861-lcdtft-display-controller-ltdc-on-stm32-mcus-stmicroelectronics.pdf

[6] libopencm3. (2023) libopencm3: Open source cortex-m3 firmware library. [Online]. Available: https://libopencm3.org/

[7] ThingsBoard. (2023) What is thingsboard? — thingsboard community edition. [Online]. Available: https://thingsboard.io/docs/getting-started-guides/what-is-thingsboard/

[8] MQTT. (2023) Mqtt - a machine-to-machine (m2m)/ïnternet of thingsçonnectivity protocol. [Online]. Available: https://mqtt.org/

5. Apéndice

5.1. Precios de componentes

Cuadro 1: Precios de cada componente(Tienda)

Componente	Precio (Colones)	Cantidad
STM32F429I-DISC1	23000	1
1k ohm	300	2
1.5k ohm	300	4

5.2. Cliente de python para ThingsBoard

```
import csv, serial
import time, json
import paho.mqtt.client as mqtt

#CLIENT METHODS
def on_connect(client, userdata, flags, rc):
    if not rc:
        client.is_connected = True
        print('Connection established. Working')
    else:
        print('Connection Faild', rc)
        client.loop_stop()

def on_disconnect(client, userdata, rc):
    if(rc == 0):
        print("Client disconneted OK")
    else:
        print("System disconnected via code: ", rc)

def on_log(client, userdata, level, buf):
   print(buf)

def on_publish(client, userdata, mid):
    print("In on_pub callback mid= ", mid)

# Obtain data
ser = serial.Serial(
    port = '/dev/ttyACM0',\
    baudrate = 115200,\
    timeout=1\
)

#SETUP
client = mqtt.Client('B92861')
client.on_connect    = on_connect
client.on_disconnect = on_disconnect
client.on_publish    = on_publish
client.on_log        = on_log
client.is_connected  = False
port         =   1883
broker       =   "iot.eie.ucr.ac.cr"
topic        =   "v1/devices/me/telemetry"
username     =   'STM32 C08592/B92861'
password     =   'w9fjlckvd764vq8ydotj'
client.username_pw_set(password)
client.connect(broker, port)
while not client.is_connected:
    client.loop()
    time.sleep(0.5)

filename = open('data.csv','w')
```

```
output_file = csv.writer(filename)
data_saved = {'X':[], 'Y':[], 'Z':[], 'B':[], 'Alert':[]}
on_off = {1: "BATERY-LOW", 0: "---"}

print("Connection succeded")
while (1):
    data = ser.readline().decode('utf-8').replace("\n","").replace("\r","
       ").split("\t")
    data = [int(value) for value in data]
    data_saved['X'].append(data[0])
    data_saved['Y'].append(data[1])
    data_saved['Z'].append(data[2])
    data_saved['B'].append(data[3])
    data_saved['Alert'].append(data[4])
    if len(data)!=5:
        continue
    output_file.writerow(data)
    output = json.dumps({"X": data[0], "Y": data[1], "Z": data[2], "B":
       data[3], "Alert":on_off[data[4]]})
    print("Topic:", topic, "output=", output)
    pub = client.publish(topic, output)
    client.loop()
```

MQTT Client Error Codes

Error code (Decimal)	Error code (Hex)	Meaning
0	0x0	No Error
1	0x1	Connection Refused: Unacceptable protocol version
10	0xa	Timeout waiting for SUBACK
11	0xb	Timeout waiting for UNSUBACK
12	0xc	Timeout waiting for PINGRESP
13	0xd	Malformed Remaining Length
14	0xe	Problem with the underlying communication port
15	0xf	Address could not be parsed
16	0x10	Malformed received MQTT packet
17	0x11	Subscription failure
18	0x12	Payload decoding failure
19	0x13	Failed to compile a Decoder
2	0x2	Connection Refused: Identifier rejected
20	0x14	The received MQTT packet type is not supported on this client
21	0x15	Timeout waiting for PUBACK
22	0x16	Timeout waiting for PUBREC
23	0x17	Timeout waiting for PUBCOMP
3	0x3	Connection Refused: Server Unavailable
4	0x4	Connection Refused: Bad username or password
5	0x5	Connection Refused: Authorization error
6	0x6	Connection lost or bad
7	0x7	Timeout waiting for Length bytes
8	0x8	Timeout waiting for Payload
9	0x9	Timeout waiting for CONNACK

Figura 24: Tabla de errores de MQTT