UNIVERSIDAD TECNOLÓGICA NACIONAL

TÉCNICAS DIGITALES II - R4001 - 2022

BODE ANALYZER

Manual de Ingeniería

Domínguez Nimo, Alejo
adomnguez@frba.utn.edu.ar

Piñeiro, Joaquín Mariano
jpineiro@frba.utn.edu.ar

supervisado por
Romeo, Marcelo & Bacigalupo, Juan

24 de febrero de 2023

Índice

1. Introducción

Este trabajo detalla el desarrollo de un dispositivo capaz de medir la magnitud y la fase de la transferencia de un filtro pasivo de forma automatizada, a partir de un barrido en frecuencia acondicionado por un hardware externo y posterior procesamiento digital.

1.1. Estado del Arte

Actualmente, algunas de las empresas que se dedican a la fabricación de un Analizador de Bode son *Redpitaya* y *OMICRON LAB*. Por un lado, la primer empresa ofrece la serie de productos "STEMlab" [1], la cual tiene un rango de funcionamiento entre 1Hz y 20MHz con una resolución de 1Hz. Además ofrece un modo de ploteo lineal o logarítmico. Ofrece compatibilidad de software con MATLAB, Jupyter Notebook y LabView.

La segunda empresa, brinda al usuario la posibilidad de generar un análisis de respuesta en frecuencia con gráficos de bode a partir de un Analizador Vectorial de Redes, precisamente con el "Bode 100"[2]. El rango de funcionamiento va desde el 1Hz a 50MHz, con una amplitud a la salida desde los -30 dBm a 13 dBm. Ofrece 4 diagramas de bode distintos: impedancia, coeficiente de relfexión, admitancia, retardo de grupo, módulo y fase. Todas estas curvas pueden ser graficadas de forma lineal o logarítimica. Utiliza entradas superheterodinas con canales ADC de 24 bits.

1.2. Principio teórico de funcionamiento

Teniendo una señal cualquiera x(t) como entrada del sistema, el mismo responde con una determinada señal y(t). Al aplicar la transformada de Fourier tanto a x(t) como a y(t), podemos obtener X(f) e Y(f) y realizar el cociente para hallar H(f), valor complejo que contiene el módulo y la fase. Sin embargo, este proceso requiere del procesamiento de las señales temporales con la transformada discreta rápida de Fourier (FFT) para poder hacer el cociente.

Por esta razón la señal más conveniente para aplicar este análisis es una señal armónica sinusoidal (1). De esta manera, al excitar con un seno, de la ecuación (2) se deduce que la señal de salida también será un seno de la misma frecuencia pero con su amplitud y desfasaje afectados por $H_{(f)}$.

$$x_{(t)} = K \cdot e^{j \cdot w_0 t + \phi_0} \Rightarrow X_{(f)} = K \cdot \delta_{(f-f_0)} \cdot e^{j \cdot \phi_0} \tag{1}$$

$$Y_{(f)} = K \cdot \delta_{(f-f_0)} \cdot e^{j \cdot \phi_0} \cdot H_{(f)} = K \cdot e^{j \cdot \phi_0} \cdot H_{(f_0)} \tag{2}$$

$$Y_{(f)} = A \cdot e^{j \cdot \phi_1} = K \cdot e^{j \cdot \phi_0} \cdot H_{(f_0)} \tag{3}$$

$$Y_{(f)} = K \cdot e^{j \cdot \phi_0} \cdot |H_{(f_0)}| \cdot e^{j \phi_H} \tag{4}$$

Resultando en:

$$|H_{(f_0)}| = \frac{A}{K} \tag{5}$$

$$\phi_{H_{(f)}} = \phi_{1_{(f)}} - \phi_{0_{(f)}} \tag{6}$$

Entonces, midiendo la amplitud y el desfasaje existente entre x(t) e y(t) a una frecuencia f_0 se obtiene fácilmente el valor complejo de $H_{(f_0)}$, sin usar la transformada de Fourier.

1.3. Principio de Medición

Para obtener el módulo $|H_{(f)}|$ se acondicionan ambas señales senoidales (de entrada y salida) transformándolas en valores de tensión continua que sean iguales al valor máximo de cada una respectivamente. Estas señales continuas son leídas por dos canales del conversor digital analógico (ADC) del MCU para luego realizar un cociente.

Para la fase $\Phi_{H_{(f)}}$ se propone detectar los cruces por cero de ambas señales de forma coherente, es decir en ambas pendientes ascendentes o ambas descendentes. En este caso, con un comparador se generan señales cuadradas a partir de las senoidales, para que sea más fácil determinar los instantes de tiempo a evaluar de forma digital. De esta manera, el MCU deberá contar el tiempo entre flancos ascendentes o descendentes de ambas señales (se decidió utilizar los ascendentes). Sabiendo el desfasaje temporal entre las señales y la frecuencia siendo evaluada, se puede obtener la fase del filtro.

1.4. Metodología General

El proyecto consistió en la construcción de un circuito impreso, la programación de un software bajo la supervisión de un sistema operativo *FreeRTOS*, en conjunto con un Framework para manejo de interfaces gráficas *TouchGFX* [3]; por último se calibró el sistema de manera de cuantizar la incertidumbre de las mediciones. El circuito impreso fue diseñado con el programa Altium Designer 2022. Consta de 2 capas (bottom layer y top layer) y 130 vías, hecho con la técnica de planchado y acidado con percloruro férrico. Los componentes que posee la placa son resistencias y capacitores SMD 0805, y circuitos integrados DIP-8. Para la CPU se utilizó un cortex M4 ARM STM32F401RE [4] embebido en una placa de *STM NUCLEO-F401RE* [5] y para la interfaz gráfica una pantalla TFT con un controlador ILI9341 [6] en conjunto con un controlador touch embebido SPI XPT2046 **XPT2046**. La señal senoidal de excitación es producida por un generador de señales programable *AD9833* [7], que utiliza tecnología DDS *(Direct Digital Synthesis)*, la cual se basa en generar una señal digital variable en el tiempo y luego realizar una conversión digital a analógico de dicha señal.

2. HARDWARE

2.1. Diagrama en Bloques - Hardware

En la siguiente página se muestra un diagrama en bloques del dispositivo (Fig. 1).

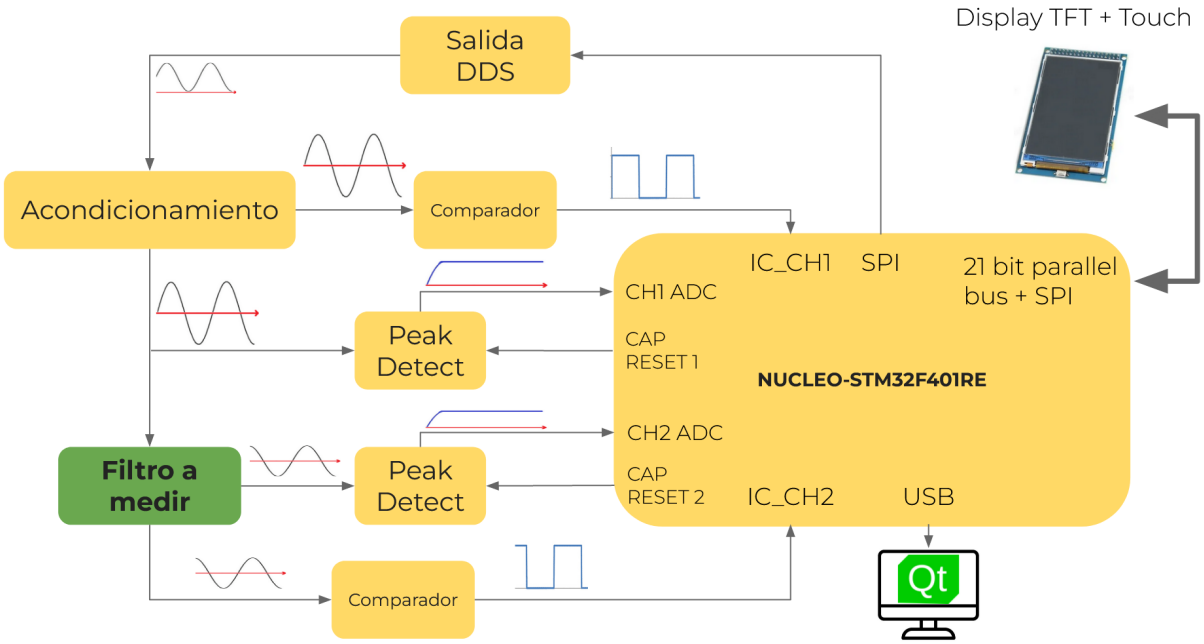

Figura 1: Diagrama en bloques del hardware

2.2. Tipos de comunicación empleadas

Se utilizan 3 tipos de comunicaciones dentro del sistema

1. **Serial Peripheal Interface SPI** para comandar el generador de señales AD9833, y la pantalla TFT específicamente su controlador touch.
2. **Paralelo-Paralelo 16 bit** para comandar el controlador ILI9341.
3. **Universal Serial Bus USB Full Speed**, para comunicar el MCU con la aplicación que se ejecuta en una computadora externa.

Se utiliza un único bus SPI por lo que la pantalla y el generador de señales comparten las mismas lineas de comunicación.

2.3. Acondicionamiento de Señales

La generación de una señal continua que representa el valor pico de ambas senoidales de entrada y salida se realiza a través de un circuito detector de pico (Fig. 2), constituido por un comparador **LM311** [8], un amplificador operacional **OPA604APG** [9], un diodo Schottky, un capacitor y un transistor MOSFET canal N.

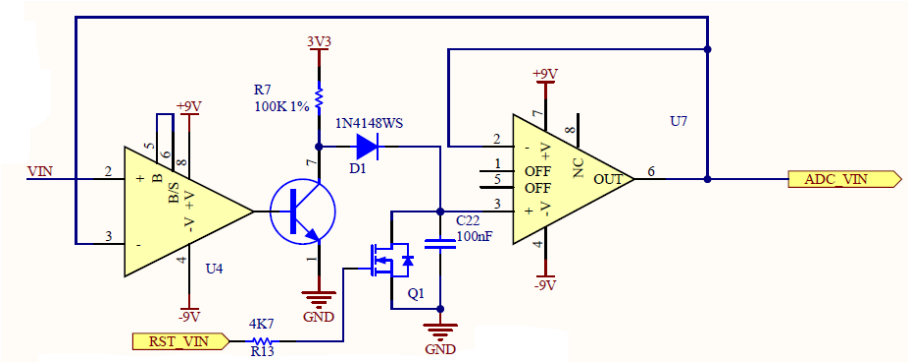

Figura 2: Circuito detector de pico

De forma breve, el funcionamiento se basa en la carga de un capacitor a través de un comparador que se activa cuando la tensión de entrada es mayor a la de salida del buffer (la cual es igual a la tensión del capacitor). Cuando dicha tensión de salida supera levemente la de entrada (VIN), el comparador se desactiva y el capacitor mantiene su tensión por un tiempo relativamente alto dado que la descarga es a través de la corriente en inversa del diodo y la de entrada del operacional.

Notar que el transistor presente se conecta en paralelo al capacitor y es controlado por una salida de la CPU. Al cambiar a otro punto de frecuencia, puede que el nuevo valor pico de salida sea menor que el medido anteriormente, por lo que el capacitor debe descargarse antes de volver a realizar una medición. Al activar el MOSFET, se cortocircuita el capacitor descargándolo rápidamente.

Por otro lado, la generación de las señales cuadradas para detectar el desfasaje se realiza con un comparador LM311 con un lazo de realimentación de histéresis (Fig. 3). De esta manera, ambas señales son más robustas al ruido que pueda haber en el sistema dado que impide que este genere falsos flancos a la salida de los comparadores cuando las señales senoidales se encuentran cerca del cero.

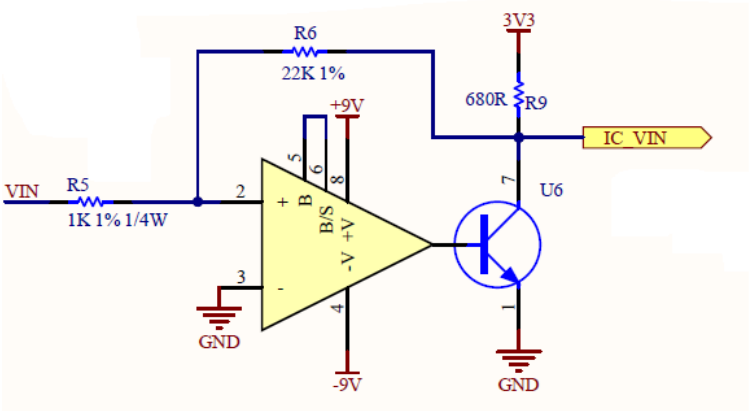

Figura 3: Circuito comparador con histéresis

El valor pico de la señal senoidal de excitación generada por el AD9833 es de apenas unas centenas de mV y además con un nivel de continua indeseado para nuestro análisis. Para aprovechar mejor el rango dinámico del ADC del microcontrolador, se filtra el nivel de continua con un filtro pasa-altos, y se amplifica la señal a una tensión pico de 3.3V (tensión de referencia del ADC) con un amplificador no inversor como el de la (Fig. 4) a partir de un OPA604APG. El propósito del jumper J5 y R4 será explicado con mas detalle en el apartado de calibración.

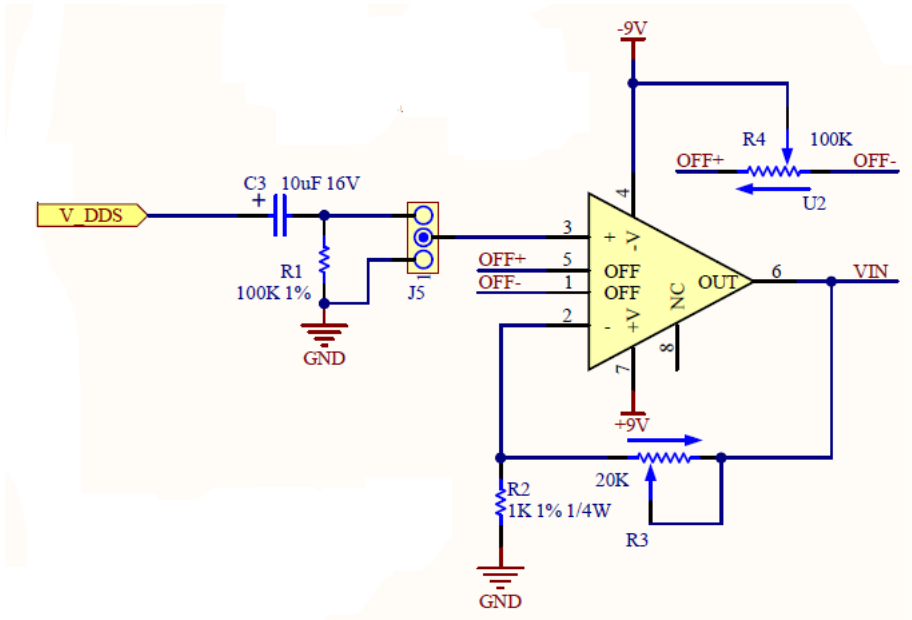

Figura 4: Acondicionamiento de señal generada: filtro pasa altos y amplificador no inversor

2.4. Distribución de GPIO NUCLEO

Se presenta en la figura 5 la distribución de pines elegida para interconectar señales del microcontrolador:

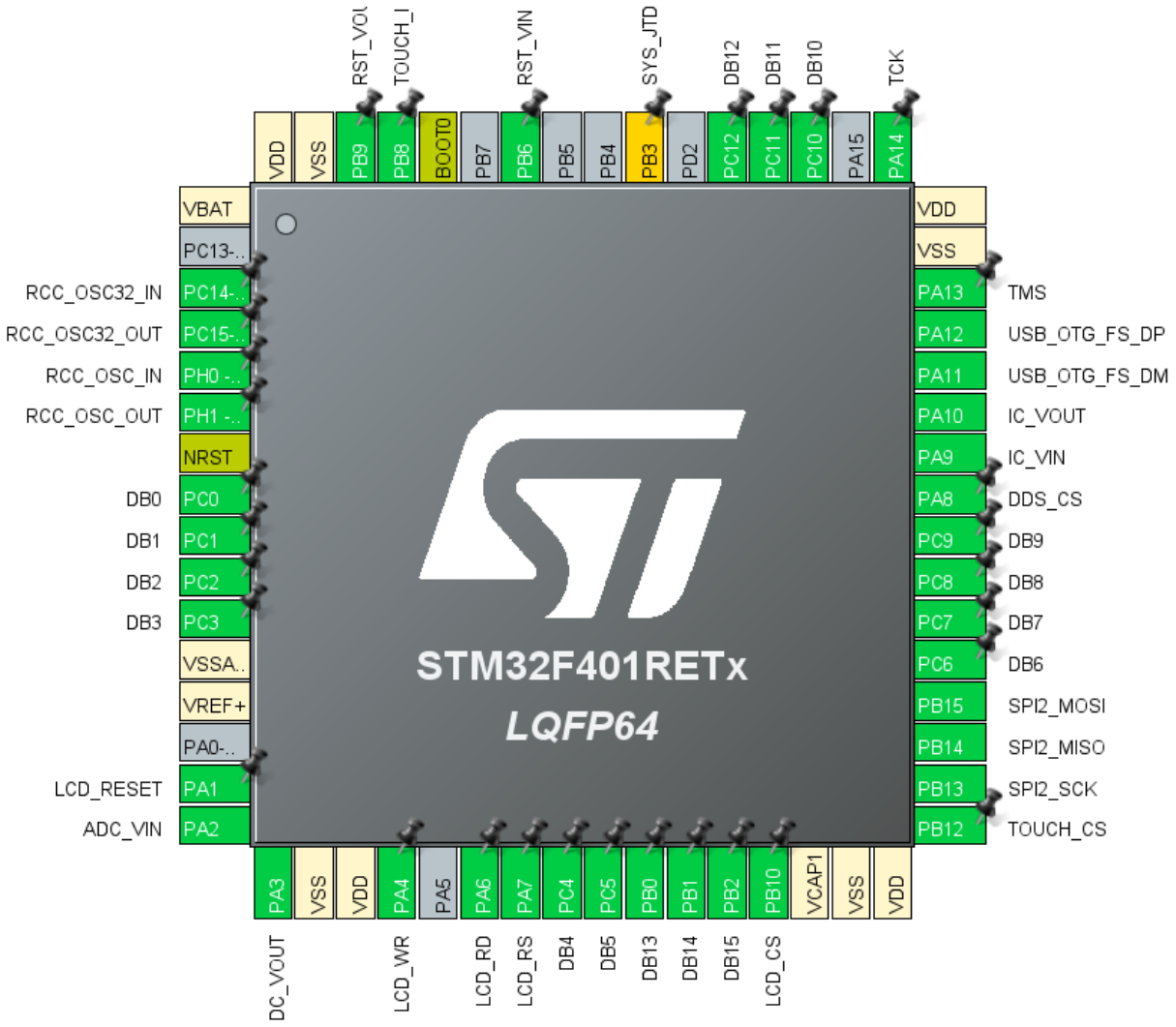

Figura 5: Diagrama de la GPIO dentro del STMCubeMX

2.5. Esquemáticos

A continuación se presentan los esquemáticos empleados para el diseño del PCB.

2.5.1. General

Esta hoja interconecta todos los puertos de salida y entrada entre las distintas hojas que existen en el proyecto de Altium (Fig.6).

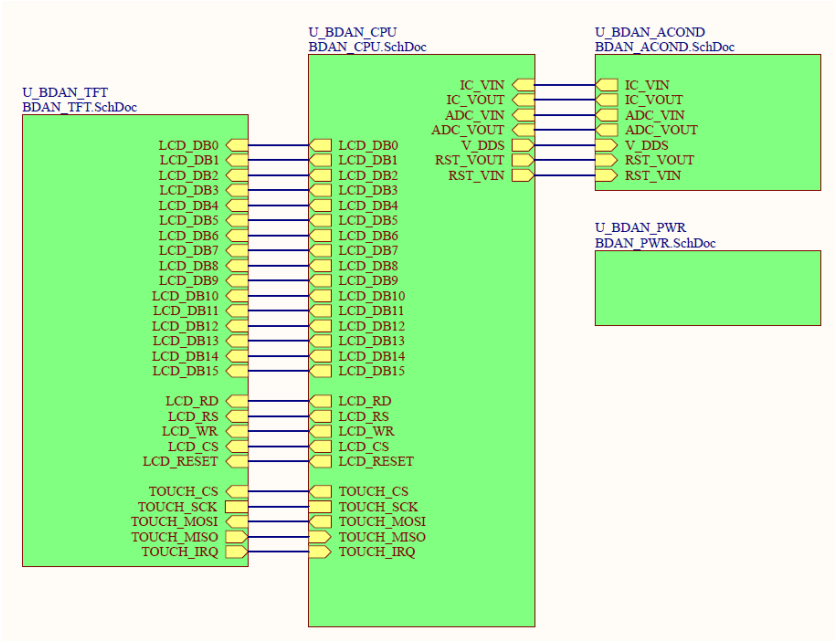

Figura 6: Interconexión de los distintos esquemáticos

2.5.2. Fuente de Alimentación

En esta hoja se encuentra la fuente de alimentación utilizada para todo el sistema(Fig.7).

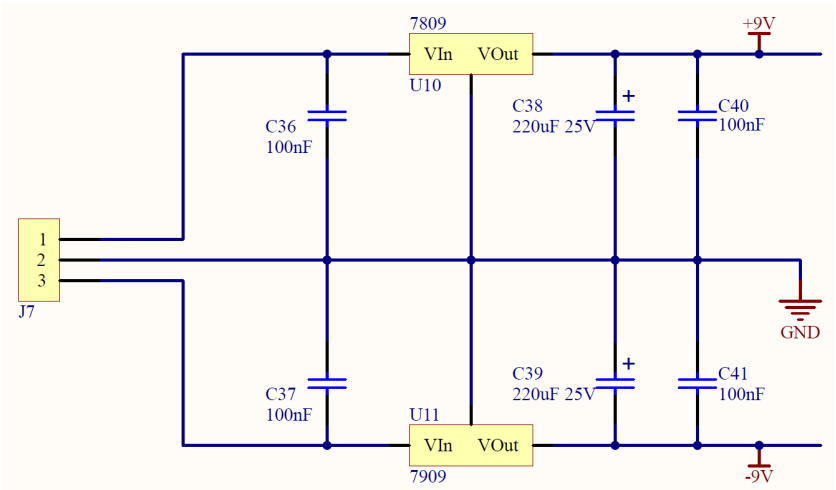

Figura 7: Fuente de alimentación ±9V 180mA

2.5.3. Acondicionamiento

En esta hoja se encuentran los circuitos de acondicionamiento de señal(Fig.8).

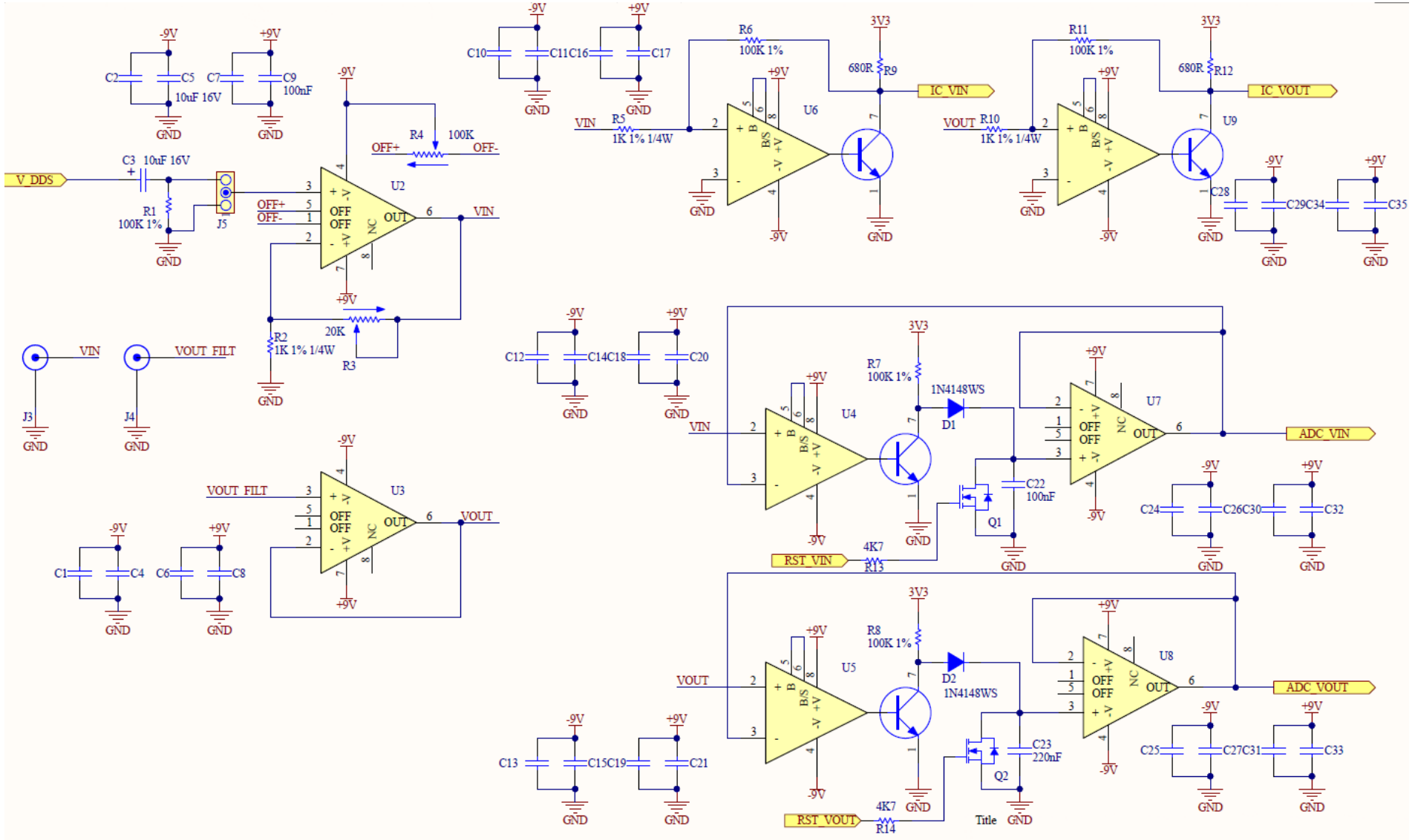

Figura 8: Circuitos de detección de pico y detección de cruce por cero + amplificación de entrada y filtrado DC.

2.5.4. CPU

En esta hoja se encuentra el interconexionado de la GPIO del MCU con los distintos elementos del sistema(Fig.9).

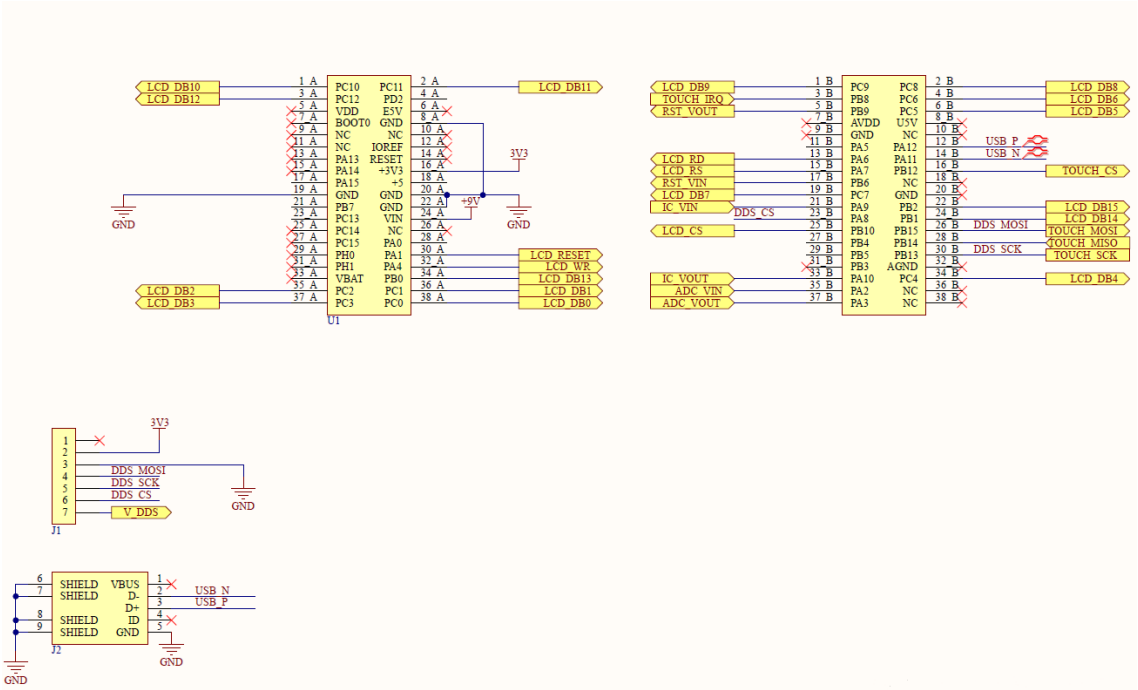

Figura 9: Conexión de la NUCLEO con la GPIO y conectores hembra + conector USB.

2.5.5. TFT

En esta hoja se encuentra el interconexionado de la interfaz de comunicación del display TFT con el MCU(Fig.10).

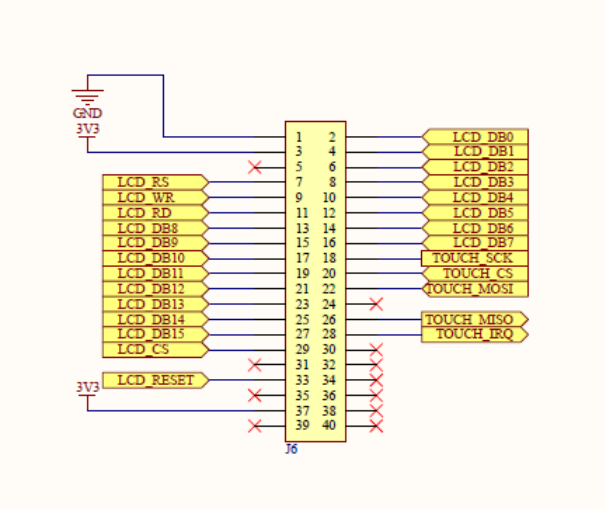

Figura 10: Interconexión de la comunicación paralelo-paralelo 16bit entre ILI9341 y el MCU, además del SPI utilizado por el XPT2046.

2.6. PCB

A continuación se muestra el circuito impreso diseñado, tanto como el lado *Top Layer* (Fig.11),el lado *Bottom Layer* (Fig.12) y el modelo 3D (Fig.13) (Fig.14).

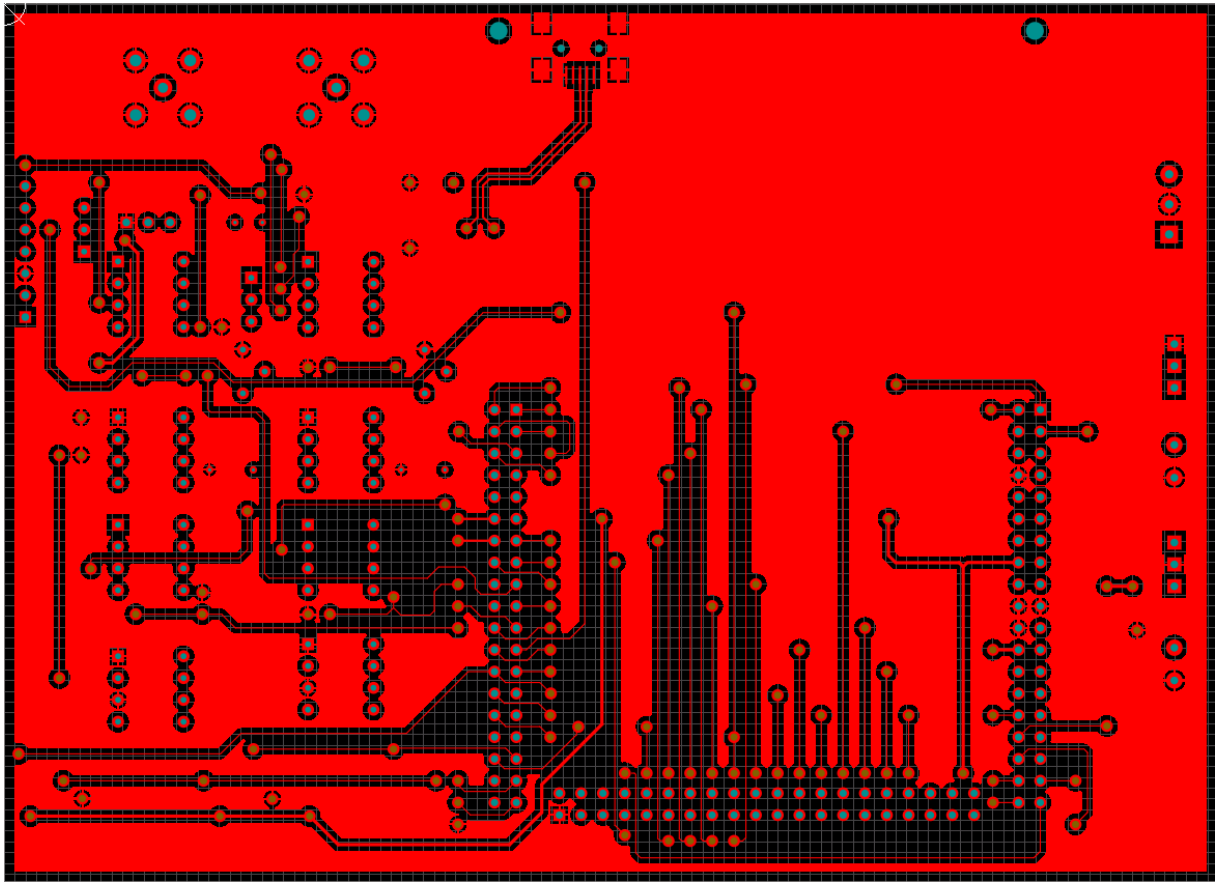

Figura 11: Top Layer

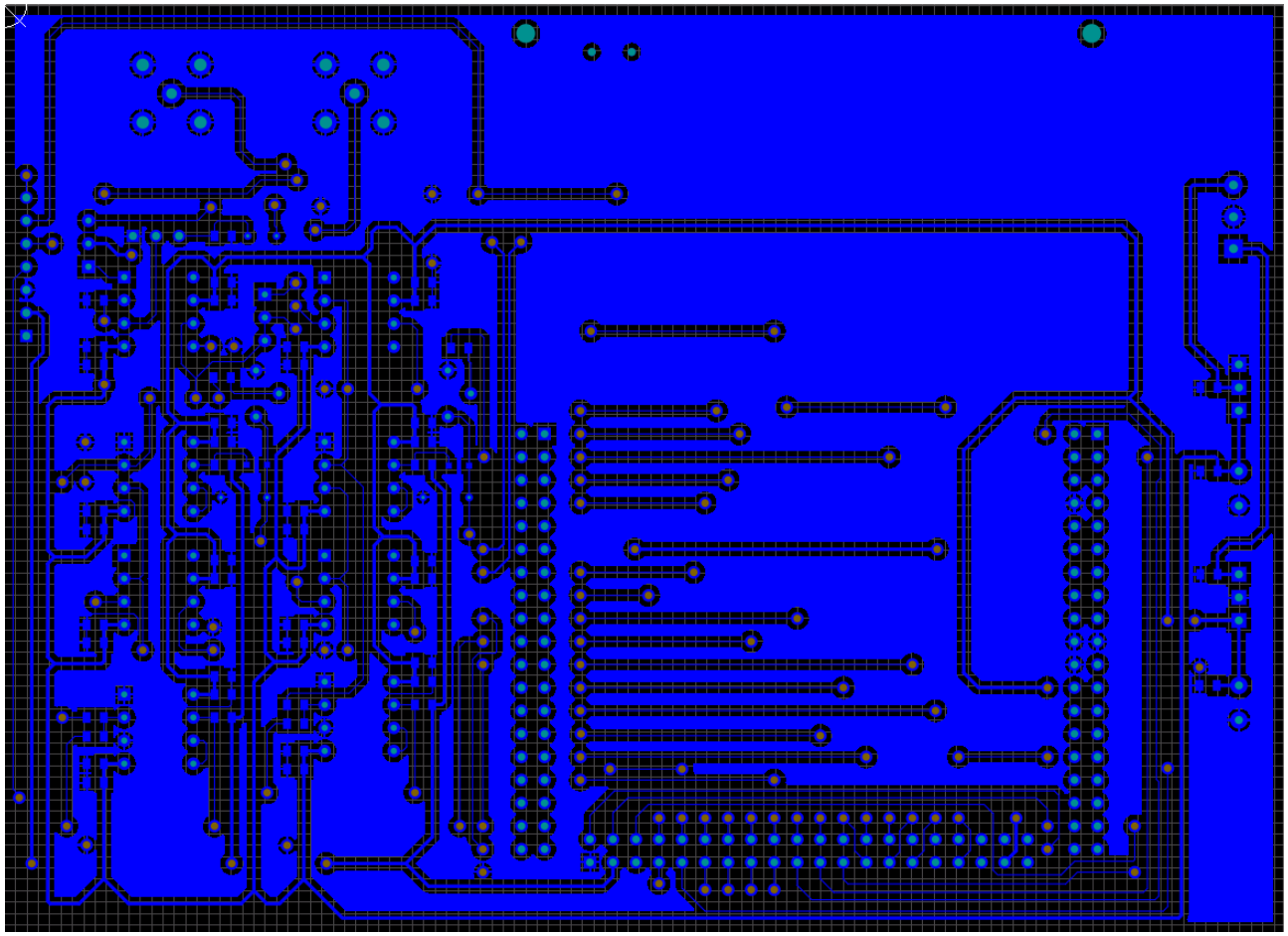

Figura 12: Bottom Layer

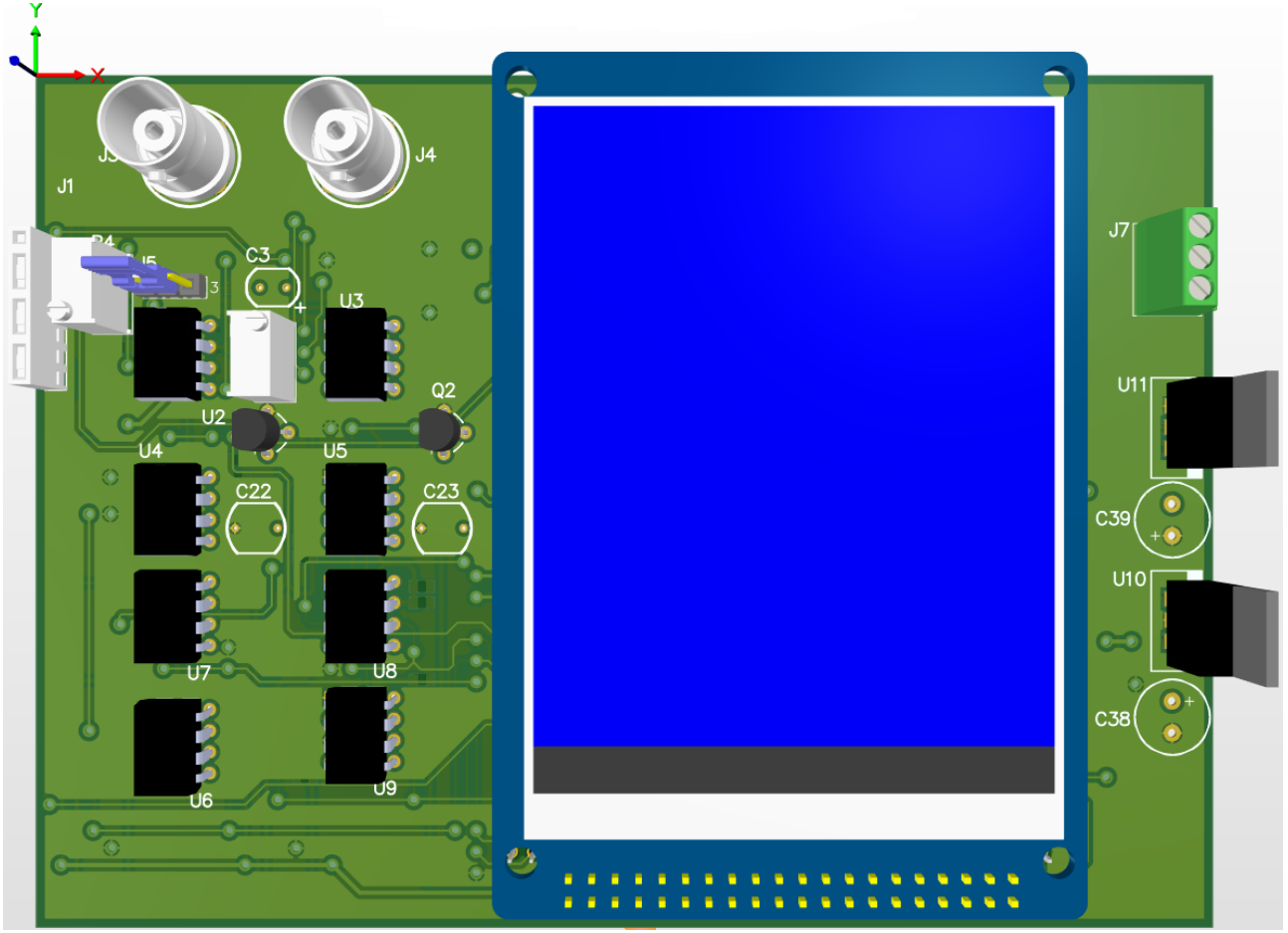

Figura 13: 3D Top

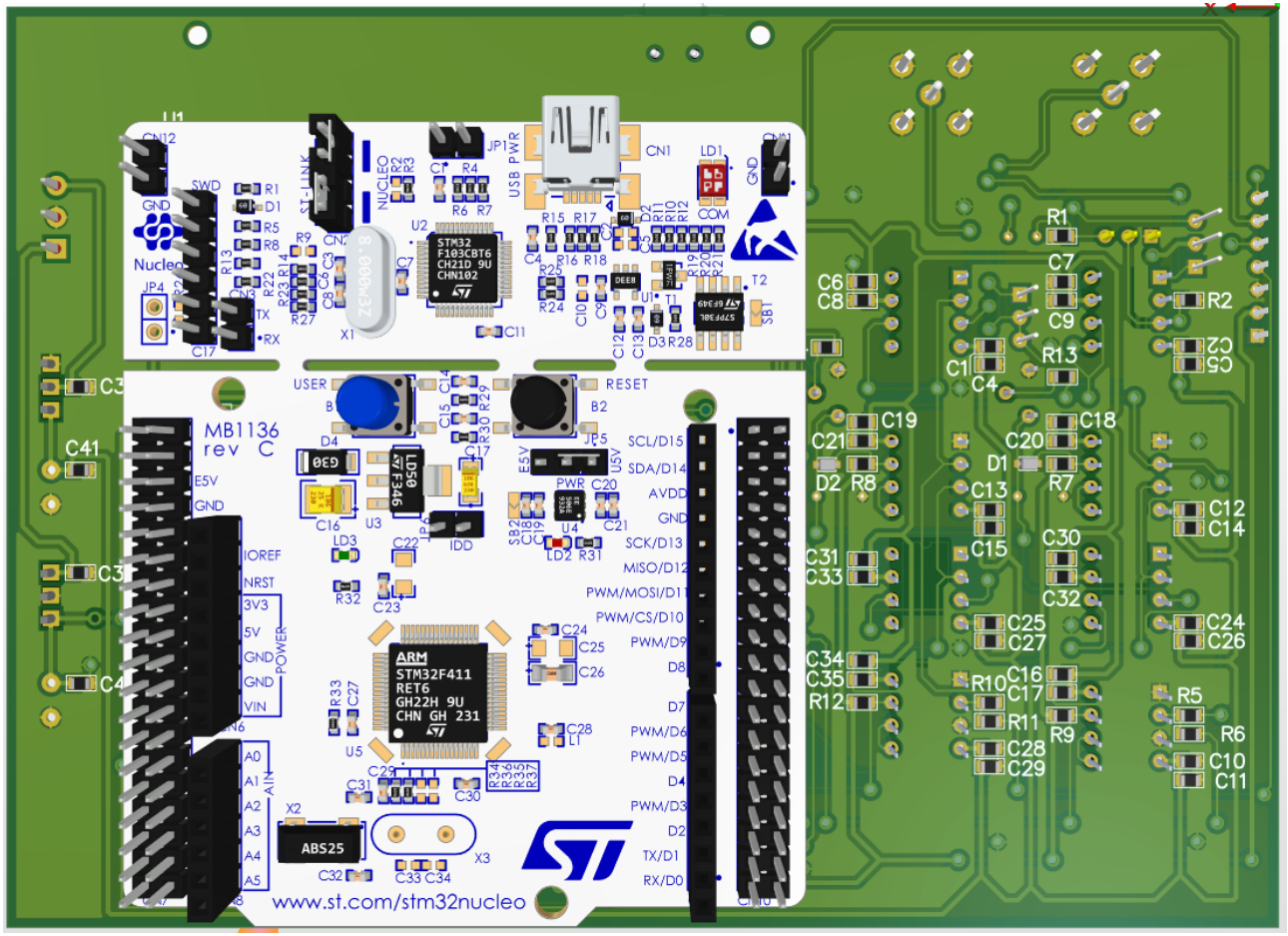

Figura 14: 3D Bottom

2.7. BOM

Se presenta la Bill of Materials utilizada para la fabricación del PCB (Fig.15).

Comment	Description	Designator	Footprint	LibRef	Quantity	Manufacturer	Manufacturer Part Number	Supplier 1	Supplier Part Number 1
100nF	CAP0805 100nF X7R	C1, C2, C8, C9, C10, C12, C13, C17, C20, C21, C24, C25, C28, C32, C33, C35, C36, C37, C40, C41	0805	CAP0805_100nF_X7R	20				
10uF 16V	CAP TANT 10UF 10% 16V RADIAL	C3	CAP318	CAP_TAN_10uF_16V_CAP318	1	KYOCERA AVX	TAP106K016CRS	Digi-Key	478-TAP106K016CRSCT-ND
10uF 16V	CAP0805 10uF 16V X5R	C4, C5, C6, C7, C11, C14, C15, C16, C18, C19, C26, C27, C29, C30, C31, C34	0805	CAP0805_10uF_16V_X5R	16	TDK Corporation	C2012X5R1C106K085AC	Digi-Key	445-7644-1-ND
220nF	CAP CER RAD 220NF 250V C0G 10%	C23	CAP508	CAP_CER_100nF_250V_CAP508	1			Digi-Key	
100nF	CAP CER RAD 100NF 250V C0G 10%	C22	CAP508	CAP_CER_100nF_250V_CAP508	1			Digi-Key	
220uF 25V	CAP ELECT 220uF 25V	C38, C39	CAP300RP	CAP_ELECT_220uF_25V	2	Nichicon	UVR1E221MPD1TA	Digi-Key	493-6097-3-ND
1N4148WS	DIODE 75V 150MA SOD323F	D1, D2	SOD323F	1N4148_SOD323F	2			Digi-Key	1N4148WSFSTR-ND
RECEPTACULO 7POS 0.1"	RECEPTACULO 1x7 POS 0.1"	J1	SIPF7	RECEPTACLE_7POS_0.1"	1			ElectroComponentes	
MINIUSB_FEMALE_SMD	CONECTOR MINIUSB FEMALE_SMD	J2	USB_MINI_FEMALE	CON_MINIUSB_FEMALE_SMD	1			ElectroComponentes	
BNC	CONN BNC RCPT STR 50 OHM PCB	J3, J4	BNC	BNC	2	Amphenol RF	112404	Digi-Key	ACX1051-ND
JUMPER3	TIRA DE PINES 1x3 0.1" + JUMPER	J5	JUMPER3	JUMPER3_TH	1	Harwin Inc.	M20-9990345	Digi-Key	952-2263-ND
ILI9341	ILI9341-3.2"	J6	ILI9341	ILI9341-3.2	1				
1984620	TERM BLK 3P SIDE ENT 3.5MM PCB	J7	BORNERA3_3.5MM	BORNERA_3_3.5MM_TORNILLO	1	Phoenix Contact	1984620	Digi-Key	277-1722-ND
100K 1%	RES0805 100K 1%	R1, R6, R7, R8, R11	0805	RES0805_100K_1%	3	Stackpole Electronics Inc	RMCF0805FG100K	Digi-Key	RMCF0805FG100KCT-ND
1K 1% 1/4W	RES SMD 1K OHM 1% 1/4W 0805	R2, R5, R10	0805	RES0805_1K_1%	3	Stackpole Electronics Inc	RNCP0805FTD1K00	Digi-Key	RNCP0805FTD1K00TR-ND
20K	3296 TRIMMER 20K OHM 0.5W PC PIN	R3	3296W-TOP	TRIMMER_3296W_20K	1	Bourns Inc.	3296W-1-203	Digi-Key	3296W-203-ND
100K	3296 TRIMMER 100K OHM 0.5W PC PIN	R4	3296W-TOP	TRIMMER_3296W_100K_MULTI	1	Bourns Inc.	3296W-1-104LF	Digi-Key	3296W-104LF-ND
680R	RES0805 680R 1%	R9, R12	0805	RES0805_680R_1%	2	Stackpole Electronics Inc	RMCF0805FT680R	Digi-Key	RMCF0805FT680RCT-ND
NUCLEO-F401RE	NUCLEO-F401RE	U1	NUCLEO-F401RE	NUCLEO-F401RE	1	STMicroelectronics	NUCLEO-F401RE	Digi-Key	
OPA604APG4	IC OPAMP GP 1 CIRCUIT 8DIP	U2, U3, U7, U8	DIP8	OPA604APG4	4	Texas Instruments	OPA604APG4	Digi-Key	OPA604APG4-ND
LM311M	IC VOLTAGE COMPARATOR 8-SOIC	U4, U5, U6, U9	DIP8	LM311N	4			Digi-Key	LM311MNS/NOPB-ND
7809	IC REG LDO 9V 1.5A TO220	U10	TO220-V	7809	1			ElectroComponentes	
7909	IC REG LDO -9V 1.5A TO220	U11	TO220-V	7909	1			ElectroComponentes	
2N7000_TO92	MOSFET N-CH 60V 200MA TO-92	Q1, Q2	TO92_DGS	MOSFET N	2				
4K7	RES0805 4K7 5%	R13, R14	0805	RES0805_4K7_5%	2	Stackpole Electronics Inc	RMCF0805JT4K70	Digi-Key	RMCF0805JT4K70TR-ND

Figura 15: Bill of Materials

3. MÉTODO DE MEDICIÓN

El método de medición se subdivide en 3 etapas:
1) Selección de rango de medición desde la UI.
2) Medición de amplitud.
3) Medición de fase.

1. El usuario controlará desde la UI el rango de frecuencias que desea medir y la cantidad de puntos por década que desee. Luego con el botón “Medir” comenzará la medición, donde el MCU controlará todo el procedimiento.

2. Con los rango obtenidos de la UI, el MCU comanda al generdor de señales para que genere una señal senoidal con la primer frecuencia del barrido. Se descargan los capacitores de los detectores de pico durante 1ms, y luego de un tiempo de espera para estabilizar las señales el ADC del MCU genera una conversión de 10 muestras para ambos canales (señal de entrada y salida). Estas son promediadas; con el valor obtenido se realiza el cociente de Vout y Vin para obtener $|H(f_0)|$.

3. Una vez realizada la medición de amplitud, el MCU comenzará la medición de fase si el módulo medido fue mayor a -30dB, y si la frecuencia es menor a 20kHz, explicado a continuación en la sección de *Rango de Medición*. Un timer dedicado del microcontrolador detectará un flanco ascendente de la señal de entrada, y cuando ocurra esto, almacenará el valor actual de la cuenta del mencionado timer. Al ocurrir luego un flanco ascendente pero esta vez de la señal de salida, nuevamente se almacena el valor de la cuenta. Con los valores obtenidos se calcula el desfasaje temporal, y posteriormente el desfasaje en grados sexagesimales.

Terminada la medición, el MCU cambia a la siguiente frecuencia (siguiente punto de medición) y se repite el procedimiento. Una vez terminado el barrido de frecuencia, en la UI se actualizan los gráficos de módulo y fase.

3.1. Rango de Medición

El rango de medición queda determinado por las características del filtro, y las limitaciones de hardware analógico y digital.

3.1.1. Magnitud

El límite superior es siempre de 0dB para toda frecuencia dado que se excita al filtro con una señal que tiene el pico igual a la tensión de referencia del ADC (de manera de aprovechar su rango dinámico), si el filtro es activo o tiene un realce eso no se podría medir teniendo en cuenta que saturan las cuentas del conversor. Para frecuencias menores a 20Hz el rango en la medición de magnitud queda limitado entre 0dB y -16dB debido a que el *ripple* por la descarga del capacitor del detector de pico a baja frecuencia es considerable a amplitudes bajas (menores a 200mV). Para frecuencias mayores a 20Hz y hasta 20kHz el rango está limitado entre 0dB y -44dB dado que el detector de pico conectado a la salida del filtro puede disminuir su valor sólo hasta 20mV.

3.1.2. Fase

La medición de fase está limitada por la histéresis y el umbral propio que tiene el comparador para discernir en un cruce por cero, y además por las limitaciones propias del timer de la CPU.

No es posible medir desfasajes de señales menores a 35mV debido a que están por debajo de los umbrales de histéresis de los comparadores. Por esta razón solo se mide fase si el módulo medido con anterioridad es mayor a -30dB (valor elegido con cierto margen de seguridad). Por otro lado, también hay limitaciones para el desfasaje a frecuencias altas (más de 25kHz) debido a que los canales de input

capture del timer del microcontrolador dejan de responder de forma rápido a los flancos ascendentes producidos por el comparador.

Por estas razones, la medición de fase está limitada a un rango entre 1Hz y 20KHz, cuando la magnitud es mayor a -30dB. Teniendo en cuenta estas restricciones, los valores de fase no tienen limitaciones en su medición: se muestran en grados sexagesimales desde -180° hasta 180°.

3.1.3. Impedancia del Filtro

La mínima impedancia de entrada que debe tener el filtro a medir está limitada por la máxima corriente que puede exigirse al operacional OPA604. La tabla de la figura 16 muestra dicho valor.

OUTPUT					
Voltage Output	R_L = 600Ω	±11	±12		V
Current Output	V_O = ±12V		±35		mA
Short Circuit Current			±40		mA
Output Resistance, Open-Loop			25		Ω

Figura 16: Tabla de valores de salida de OPA604

Dicha corriente máxima se puede exigir sólo en el peor caso de máximo consumo, cuando la tensión de salida del operacional es máxima, es decir de 3.3V. De esta forma:

$$Zin_{MAX} = \frac{3{,}3V}{40mA} = 82{,}5\Omega \tag{7}$$

Una impedancia menor a 82.5Ω, provocará que el sistema se comporte inadecuadamente dado que el operacional limitará la corriente a 40mA.

Por otro lado la máxima impedancia de salida que puede tener el filtro esta limitada por la impedancia de entrada del operacional OPA604, la cual es de 1TΩ. Esto quiere decir que la impedancia de salida del filtro puede ser de 10GΩ y de todas maneras el dispositivo recibe más del 99 % de la tensión de salida del filtro:

$$Vin^{dispositivo} = Vout^{filtro} \frac{1T\Omega}{1T\Omega \cdot 10G\Omega} = Vout^{filtro} \cdot 0{,}9901 \tag{8}$$

4. Firmware

El algoritmo está compuesto de un total de 6 tareas: inicialización del hardware *"StartHardwareTask"*; monitoreo de medición *"MeasureTask"*; medición de amplitud *"ModuleTask"*; medición de fase *"PhaseTask"*; envío de datos por USB "*USBTask*"; refresco de pantalla que corre dentro del motor gráfico, ejecutado periódicamente a través de un timer.

1. *"StartHardwareTask"* se encarga de inicializar el motor gráfico de TouchGFX y establecer la comunicación USB.

2. *"MeasureTask"* se encarga descargar los capacitores de los detectores de pico durante 1ms, recalcular y cambiar la frecuencia de la señal de entrada, generar una demora de 1s para que se estabilicen las señales, iniciar la medición de amplitud con *"ModuleTask"* y, si la frecuencia y la magnitud medida están dentro del rango permitido, procede a realizar la medición de fase con *"PhaseTask"*. Finalizadas las mediciones, se comunica con el motor gráfico para actualizar las pantallas con los valores de módulo y fase, y con *USBTask* para enviar los datos por USB, para mostrar los gráficos en la PC.

3. *"ModuleTask"* sincronizado con *"MeasureTask"*, genera 10 conversiones del ADC separadas 54.67μs entre sí de ambas señales pico de entrada y salida. Luego de la adquisición se promedia y se calcula la magnitud en decibeles.

4. *"PhaseTask"* sincronizado con *"MeasureTask"*, se activan las entradas que detectan flancos, y una vez sucedidos dos eventos de flancos ascendentes como se describió anteriormente, se calcula la diferencia temporal y posteriormente el desfasaje en grados sexagesimales. Obtenido este valor se analizan los casos en que la fase supera 360° para descartar una mala detección (posible a alta frecuencia), y el caso en que la fase supere -180°. Para el primero genera una nueva medición, y para el segundo se realiza un wrap de la fase para llevarla al primer cuadrante. Para este procedimiento, también se toman 10 muestras de la fase.

5. *"USBTask"* sincronizado con *"MeasureTask"*, envía por comunicación USB la medición de magnitud, fase y los valores de frecuencia.

4.1. Diagrama de Flujo

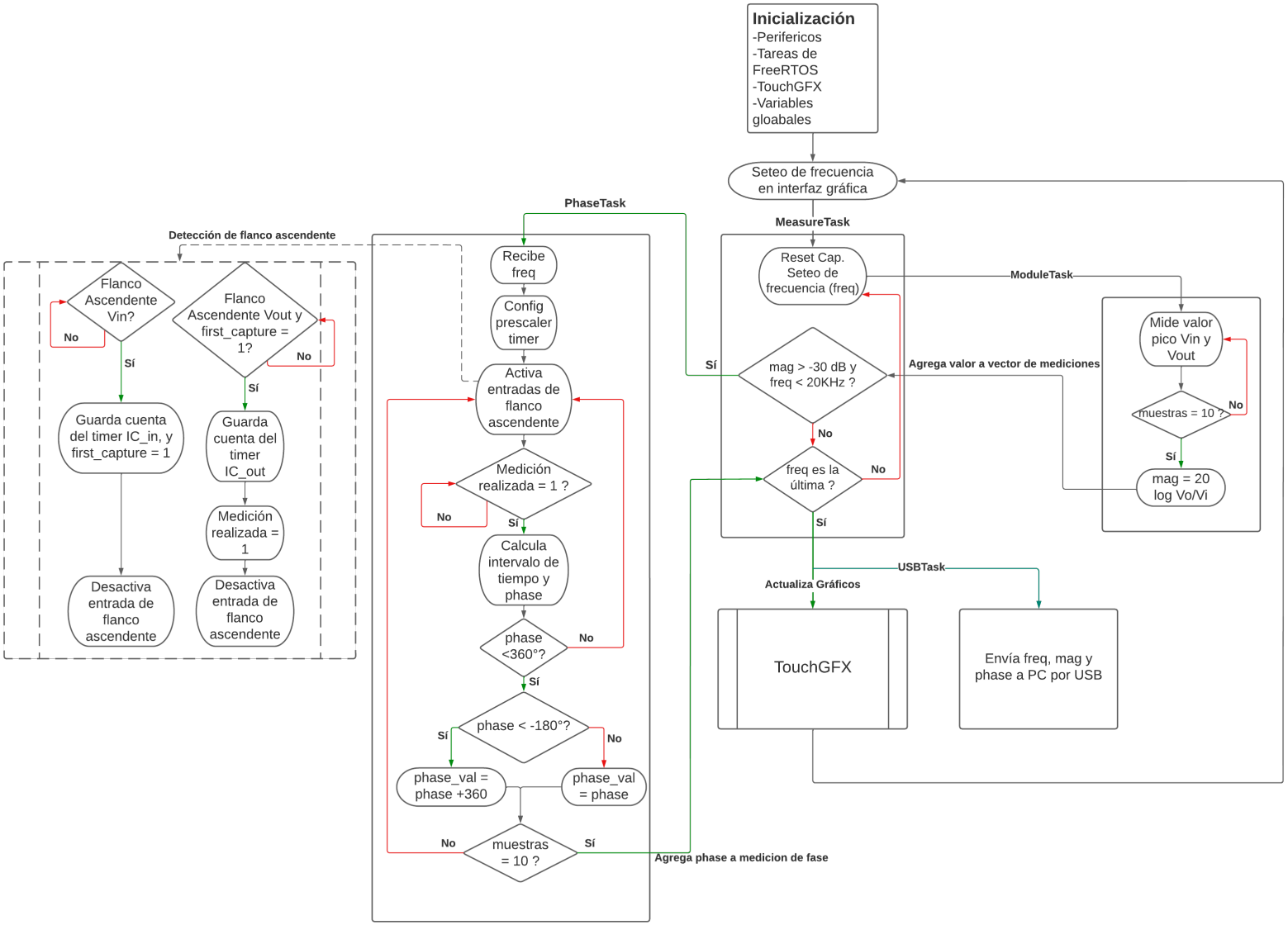

Figura 17: Diagrama de flujo del firmware

4.2. Código

En esta sección se mostrarán porciones del código trascendentes al funcionamiento del dispositivo.

En primer lugar, se tiene una pantalla diseñada con un middleware llamado ToughGFX la cual permite seleccionar los extremos y la resolución del barrido en frecuencia.

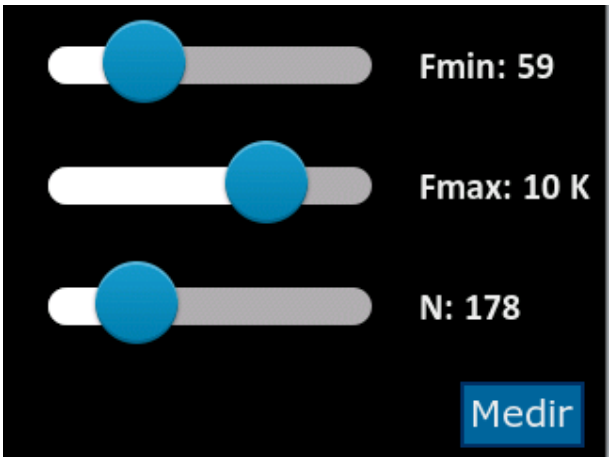

Figura 18: Pantalla inicial de configuración

Para obtener los datos de esta pantalla, el middleware TouchGFX utiliza una arquitectura llamada modelo-vista-presentador (MVP):

1. El modelo es una interfaz que define los datos a ser visualizados, y qué hacer con aquellos datos provenientes de la interfaz gráfica. Es el encargado de comunicarse con el back-end.
2. La vista es una interfaz pasiva que muestra en el display los datos provenientes del modelo, y envía los datos ingresados por el usuario en la interfaz gráfica.
3. El presentador actúa como intermediario entre el modelo y la vista.

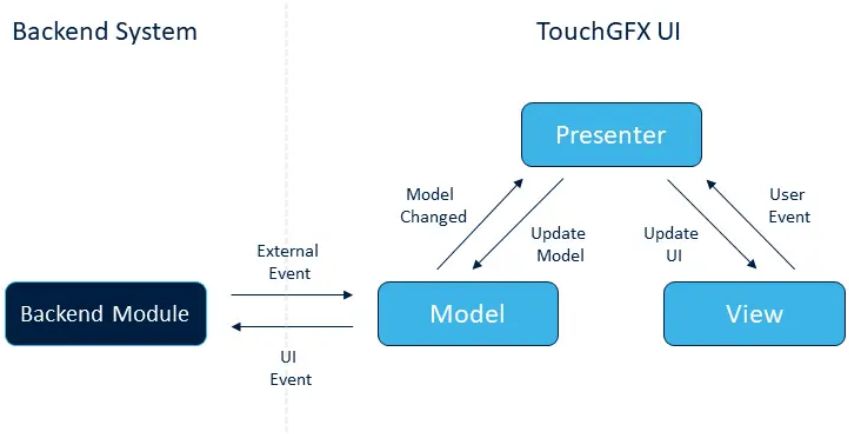

Figura 19: Pantalla inicial de configuración

Por esta razón, una vez que se pulsa el botón de “Medir” en la pantalla, los datos se trasladan desde la vista, al presentador, y luego al modelo, donde se encuentra el siguiente bloque de código:

```
22 void Model::Freq_Config(float freq_min, float freq_max, unsigned int
    ↪ points_decade)
23 {
24     unsigned int i = 1;     //indice para recorrer freq[]
25 
26     /*Se verifica que la cantidad de puntos totales que se
27     querrían computar no supere el tamaño máximo reservado para
28     el vector de frecuencia*/
29     if(log10(freq_max/freq_min) * points_decade > MAX_FREQ_POINTS)
30         points_decade = int(MAX_FREQ_POINTS/log10(freq_max/freq_min));
31 
32     float step = pow(10,1.0/points_decade);
33 
34     freq[0] = freq_min;
35 
36     while((freq[i-1] < freq_max) && (i < MAX_FREQ_POINTS))
37     {
38         freq[i] = freq[i-1] * step ;
39         i++;
40     }
41     total_points = i;
42 
43     xSemaphoreGive(sem_measure);     //Desbloqueo tarea de medición
44 }
```

Se encarga de generar un vector de frecuencias distanciadas exponencialmente respetando la cantidad de puntos por década hasta un límite determinado por el tamaño máximo del vector. Luego de esto libera el semáforo de la tarea “MeasureTask” para comenzar la medición:

```
703 void MeasureTask(void* pvParameters)
704 {
705     uint16_t i;
706     uint8_t phase_medida;
707 
708     while(1)
709     {
710         xSemaphoreTake(sem_measure, portMAX_DELAY);
711 
712         USBD_CUSTOM_HID_SendReport(&hUsbDeviceFS,(uint8_t*)&total_points,sizeof(total_points)
    ↪ //Envío la cantidad de puntos total
713 
714         vTaskDelay(pdMS_TO_TICKS(10));//Delay para evitar pérdida de paquetes
715 
716         USBD_CUSTOM_HID_SendReport(&hUsbDeviceFS,(uint8_t*)freq,total_points*4);     //Envío
    ↪ vector de frecuencias
717 
718         vTaskDelay(pdMS_TO_TICKS(100));
719 
```

```
720         //Reconfiguro el spi a lo que necesita el AD9833
721         touchEnabled = 0;    //Deshabilito el touch (ver
    ↪  TouchGFX/target/STM32TouchController.cpp)
722         SPI_ChangeParameters(&hspi2, SPI_DATASIZE_16BIT, SPI_POLARITY_HIGH);
723 
724         //Para cada punto de frecuencia
725         for(i = 0; i< total_points; i++)
726         {
727             //Reseteo detectores de pico (descargo capacitores)
728             HAL_GPIO_WritePin(RST_VIN_GPIO_Port, RST_VIN_Pin, GPIO_PIN_SET);
729             HAL_GPIO_WritePin(RST_VOUT_GPIO_Port, RST_VOUT_Pin, GPIO_PIN_SET);
730 
731             //Cambio de frecuencia (frecuencia redondeada a entero por el
    ↪  momento)
732             //Reescribo freq[i] con el valor que verdaderamente se genera (para
    ↪  realizar los calculos y graficar correctamente)
733             freq[i] = AD9833_SetFrequency(freq[i]);
734 
735             vTaskDelay(pdMS_TO_TICKS(1));
736 
737             //Desactivo mosfets que descargan capacitores
738             HAL_GPIO_WritePin(RST_VIN_GPIO_Port, RST_VIN_Pin, GPIO_PIN_RESET);
739             HAL_GPIO_WritePin(RST_VOUT_GPIO_Port, RST_VOUT_Pin, GPIO_PIN_RESET);
740 
741             if(i>0)
742                 xSemaphoreGive(sem_USB);
743 
744             vTaskDelay(pdMS_TO_TICKS(1000));
745 
746             //libero tarea de MEDICION DE MAGNITUD:
747             xSemaphoreGive(sem_mod);
748 
749             //Espero medicion de magnitud
750             xQueueReceive(queue_mod, &mag[i], portMAX_DELAY);
751 
752             //libero tarea de MEDICION DE FASE:
753 
754             if(mag[i]>-30 && freq[i] < 20000)
755             {
756                 xQueueSend(queue_freq_phase, &i, portMAX_DELAY);
757                 xQueueReceive(queue_phase, &phase[i], portMAX_DELAY);
758                 phase_medida = 1;
759             }
760             else if (i==0)
761             {
762                 phase[i] = 0;
763                 phase_medida = 0;
764             }
765 
766             else
767             {
768                 phase[i] = phase[i-1];
```

```
                phase_medida = 0;
            }

            xQueueSend(queue_hab_phase, &phase_medida,
    ↪ portMAX_DELAY);    //Informo a taskUSB si se realizó la medicion de fase o
    ↪ no

            //Actualizo la barra de progreso (0% a 100%)
            progress = ((i+1)*100) / total_points;
        }
        xSemaphoreGive(sem_USB);
        //Reconfiguro el spi a lo que necesita el touch
        SPI_ChangeParameters(&hspi2, SPI_DATASIZE_8BIT, SPI_POLARITY_LOW);

        touchEnabled = 1; //Habilito touch

        data_ready = 1;
    }
}
```

Esta tarea primero informa a la PC mediante USB la cantidad de puntos a medir, y luego envía el vector de frecuencias generado anteriormente.

Después se realiza el barrido en frecuencia con un for loop, donde se configura la frecuencia de la senoidal generada con el valor del vector, y luego se ejecutan secuencialmente las tareas de medición de magnitud y luego de fase.

Es importante destacar que debido a que la tarea que mide la fase tiene el riesgo de bloquear el progreso de la medición, para ejecutarla se debe cumplir que el módulo sea mayor a -30dB y que la frecuencia sea mayor a 20kHz, tal como se menciono en la sección 3.1 “Rango de Medición”.

Al terminar cada tarea, se guardan los valores medidos en vectores, y se actualiza la variable “progress” que es utilizada para la barra de progreso mostrada en pantalla al realizar la medición.

Luego de cada ciclo del for loop, en el que se analiza un único valor de frecuencia, se libera la tarea de USB “USBTask” que se encarga de enviar muestras de tensiones tomadas de la entrada y la salida, y las muestras de desfasajes temporales medidas.

A continuación se muestra el bloque de código encargado de la medición de la magnitud, compuesto por el callback del ADC ejecutado al completar una conversión de cada canal (entrada y salida) con DMA, y con la tarea “ModuleTask”:

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    static uint8_t mediciones = 0;

    BaseType_t xHigherPriorityTaskWoken = pdFALSE;

    mediciones++;

    if(mediciones == 10)
    {
        HAL_ADC_Stop_DMA(&hadc1);

        mediciones = 0;

```

```
801         xSemaphoreGiveFromISR(sem_ADC, &xHigherPriorityTaskWoken );
802 
803         portYIELD_FROM_ISR( xHigherPriorityTaskWoken );
804     }
805 }
806 
807 void ModuleTask(void *pvParameters)
808 {
809     float v_samples[MOD_SAMPLES];
810 
811     float v_in_acum = 0;
812     float v_out_acum = 0;
813 
814     float mag;
815 
816     while(1)
817     {
818         xSemaphoreTake(sem_mod, portMAX_DELAY);
819 
820         //Realizo MOD_SAMPLES mediciones: mitad de entrada y mitad de salida
    separadas 54.67us entre sí
821 
822         HAL_ADC_Start_DMA(&hadc1, ADC_buffer_DMA, MOD_SAMPLES);
823 
824         xSemaphoreTake(sem_ADC, portMAX_DELAY);
825 
826         v_in_acum = 0;
827         v_out_acum = 0;
828         //Obtengo tensiones a partir de cuentas con transferencia calibrada de
    ADC (realizando interpolación lineal)
829         for(uint8_t j=0; j<MOD_SAMPLES; j+=2)
830         {
831             v_samples[j] = ADCLinearInterpolation(ADC_buffer_DMA[j]);
832             v_in_acum += v_samples[j];
833             v_samples[j+1] = ADCLinearInterpolation(ADC_buffer_DMA[j+1]);
834             v_out_acum += v_samples[j+1];
835         }
836 
837         mag = 20*log10(v_out_acum/v_in_acum);
838 
839         for(uint8_t i=0; i<MOD_SAMPLES; i++)
840             xQueueSend(queue_USB_MOD, &v_samples[i], portMAX_DELAY);
841 
842         xQueueSend(queue_mod, &mag, portMAX_DELAY);
843     }
844 }
```

Al desbloquearse la tarea se inicia el ADC con DMA para realizar 20 mediciones, intercalando entrada y salida mientras se espera con el semáforo “sem_ADC”.

Luego se obtienen los niveles de tensión a partir de las cuentas usando valores obtenidos durante la calibración del ADC. Se realiza una interpolación lineal para aquellos valores de cuentas que están entre medio de valores de la tabla de calibración.

Se acumulan las 10 muestras tomadas de cada canal y se calcula el módulo realizando el cociente entre esos valores acumulados, lo cual da el mismo resultado que haciendo el cociente de los valores promediados.

Las 20 muestras de tensión se envían a “USBTask”, y el cociente en decibeles se envía a “MeasureTask”.

El bloque de código encargado de la medición de la fase esta compuesto por el callback del *input capture* del timer 1, y la tarea “PhaseTask”:

```
846 void HAL_TIM_IC_CaptureCallback(TIM_HandleTypeDef *htim)
847 {
848     static uint8_t first_capture = 0;
849 
850     if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_2)
851     {
852         if(first_capture == 0 && medicion_realizada == 0)     //Primer flanco
    ↪  ascendente
853         {
854             HAL_TIM_IC_Stop_IT(&htim1, TIM_CHANNEL_2);
855             IC_in = htim->Instance->CCR2;
856             first_capture = 1;
857         }
858     }
859     if(htim->Channel == HAL_TIM_ACTIVE_CHANNEL_3)
860     {
861         if(first_capture)     //Si ya se capturó el primer flanco en el canal 2
862         {
863             HAL_TIM_IC_Stop_IT(&htim1, TIM_CHANNEL_3);
864             medicion_realizada = 1;
865             IC_out = htim->Instance->CCR3;
866             first_capture = 0;
867         }
868     }
869 }
870 
871 void PhaseTask(void *pvParameters)
872 {
873     float time_diff[PHASE_SAMPLES];
874 
875     uint16_t index;
876 
877     float phase_met1;     //Valor auxiliar correspondiente a metodo 1 de medicion
    ↪  de fase
878 
879     float phase_val;     //Vector con PHASE_SAMPLES muestras de frecuencia
    ↪  puntual
880 
881     float phase_val_acum = 0;     //acumulador para hacer el promedio
882 
883     while(1)
884     {
885         phase_val_acum = 0;
```

```
886         xQueueReceive(queue_freq_phase, &index, portMAX_DELAY);
887 
888         //Calculo prescaler para que las cuentas del timer alcancen un período
    ↪ completo
889         if(freq[index] < 15.259)
890         {
891             TIM_ConfigPrescaler(ceil((long double)FREQ_TIM1/(65535*freq[index])
    ↪ - 1));
892             vTaskDelay(pdMS_TO_TICKS(1));
893         }
894         else if(htim1.Init.Prescaler != 71)    //Prescaler mínimo de 71 para que
    ↪ el timer funcione correctamente
895         {
896             TIM_ConfigPrescaler(71);
897             vTaskDelay(pdMS_TO_TICKS(1));
898         }
899 
900         for(uint8_t i = 0; i<PHASE_SAMPLES; i++)
901         {
902             HAL_TIM_IC_Start_IT(&htim1, TIM_CHANNEL_2);
903             HAL_TIM_IC_Start_IT(&htim1, TIM_CHANNEL_3);
904 
905             medicion_realizada = 0;    //Fuerzo a que vuelva a medir
906 
907             while(medicion_realizada == 0);
908 
909             if (IC_out >= IC_in)
910                 time_diff[i] = (float)(IC_out - IC_in) *
    ↪ (htim1.Init.Prescaler+1) / FREQ_TIM1;
911 
912             else if (IC_in > IC_out) //Entre IC_in e IC_out se llenó el registro
    ↪ de la cuenta
913                 time_diff[i] = (float)((65535 - IC_in) + IC_out) *
    ↪ (htim1.Init.Prescaler+1) / FREQ_TIM1;
914 
915             time_diff[i] =
    ↪ TimerLinearInterpolation_us(time_diff[i]*1000000)/1000000;
916 
917             //El metodo de medicion siempre es FaseOut - FaseIn (Método 1),
    ↪ luego
918             //matematicamente lo pasamos a FaseIn - FaseOut (Método 2) de ser
    ↪ necesario:
919 
920             phase_met1 = -2*180*freq[index]*time_diff[i];    //fase en grados
    ↪ sexagecimales (Metodo 1)
921 
922             if(abs(phase_met1) > 360) //Detecto si hay ruido de comparador (no
    ↪ puede haber desfasaje mayor a 360)
923                 i--;
924 
925             else
926             {
```

```
927                 if(phase_met1 < -180)    //Hay que pasar al metodo 2
928                     phase_val = phase_met1 + 360;
929 
930                 else
931                     phase_val = phase_met1; //Sigo con metodo 1
932 
933                 phase_val_acum += phase_val;
934             }
935             medicion_realizada = 0;
936 
937         }
938         phase_val_acum /= (float)PHASE_SAMPLES;
939 
940         for(uint8_t j = 0; j< PHASE_SAMPLES; j++)
941             xQueueSend(queue_USB_PHASE, &time_diff[j], portMAX_DELAY);
942 
943         xQueueSend(queue_phase, &phase_val_acum, portMAX_DELAY);
944     }
945 }
```

Con la cola que recibe *index*, la tarea se desbloquea y obtiene el índice del barrido en frecuencia (el for loop de "MeasureTask"). Con este índice se accede a la frecuencia siendo analizada:

- Si la misma es menor a 15.259Hz el prescaler del timer se actualiza automáticamente según la frecuencia para que las 65535 cuentas alcancen un período completo de la señal, es decir que en el peor de los casos puede medir un desfasaje de 360°.
- Si es mayor o igual a 15.259Hz el prescaler se mantiene en 71. No baja más que esto para que el timer funcione correctamente.

Luego comienza un for loop para obtener 10 muestras de desfasaje temporal. En cada ciclo se activan ambos canales del *input capture* y se espera que se realicen las 2 mediciones temporales con el *while* de la línea 907.

En el callback del *input capture* primero se almacena la cuenta del timer 1 en *IC_in* cuando se detecta un flanco ascendente en el canal 2 (entrada) exclusivamente, y posteriormente se almacena la cuenta en *IC_out* cuando se detecta un flanco ascendente en el canal 3 (salida).

Al detectar ambos flancos, se calcula el desfasaje temporal:

- Si $IC\_out \geq IC\_in$ es porque no hubo overflow del timer, y solo se deben restar las cuentas $IC\_out - IC\_in$ y luego multiplicar el resultado por la duración de cada cuenta.
- Si $IC\_in > IC\_out$ es porque hubo un overflow entre que se detectó el flanco ascendente de entrada y el de salida. En este caso la cantidad de cuentas que representan el desfasaje temporal esta compuesta por las cuentas entre *IC_in* y 65535 (máximo valor del timer) y las cuentas entre 0 e *IC_out*.

Luego de obtener el desfasaje temporal se corrige el valor a partir de los datos obtenidos de la calibración, realizando una interpolación lineal para aquellos valores que están entre medio de los tiempos de la tabla.

Posteriormente en la línea 920 se calcula la fase con lo que se denomina como "método 1", es decir siempre tomando en cuenta que el flanco ascendente de la señal de entrada ocurre primero que el de la salida, lo cual coincide con lo que efectivamente realiza el timer. Este método interpreta la fase del

filtro como negativa, lo cual es correcto siempre que el desfasaje sea menor a -180°, en cuyo caso se pasa al "método 2" donde se considera que el flanco ascendente de la señal de salida ocurre primero que el de la entrada. Para pasar al método 2 sólo se debe sumar 360°.

También se detectan errores en el *input capture* causados a frecuencias altas consultando si el valor absoluto de la fase calculada con el método 1 supera los 360°, caso en el que se vuelve a realizar la medición hasta obtener una fase válida.

Por último se envía el promedio de las 10 fases calculadas a "MeasureTask", y el vector de 10 desfasajes temporales a "USBTask".

El último bloque de código relevante al funcionamiento del dispositivo es la ya mencionada "USBTask":

```
947 void USBTask(void *pvParameters)
948 {
949     uint8_t phase_medida;
950 
951     float v_samples[MOD_SAMPLES];
952     float time_diff[PHASE_SAMPLES];
953 
954     while(1)
955     {
956         for(uint8_t i=0; i< MOD_SAMPLES; i++)
957             xQueueReceive(queue_USB_MOD, &v_samples[i], portMAX_DELAY);
958 
959         xQueueReceive(queue_hab_phase, &phase_medida, portMAX_DELAY);
960 
961         if(phase_medida)    //Si se realizó la medición de fase obtengo los
    ↪  valores, sino se mantienen
962         {
963             for(uint8_t i=0; i< PHASE_SAMPLES; i++)
964                 xQueueReceive(queue_USB_PHASE, &time_diff[i], portMAX_DELAY);
965         }
966 
967         xSemaphoreTake(sem_USB, portMAX_DELAY);
968 
969         USBD_CUSTOM_HID_SendReport(&hUsbDeviceFS,(uint8_t*)v_samples,MOD_SAMPLES
    ↪  * 4);    //Envío vector con muestras de tensiones
970 
971         vTaskDelay(pdMS_TO_TICKS(500));    //Delay para evitar pérdida de
    ↪  paquetes
972 
973         if(phase_medida)
974             USBD_CUSTOM_HID_SendReport(&hUsbDeviceFS,(uint8_t*)time_diff,PHASE_SAMPLES
    ↪  * 4);    //Envío vector con desfasajes temporales
975         else
976         {
977             time_diff[0] = 0;
978             USBD_CUSTOM_HID_SendReport(&hUsbDeviceFS,(uint8_t*)time_diff,1);    //Informo
    ↪  que no se realizó medición de fase a la PC
979         }
980     }
981 }
```

Con la cola *queue_USB_MOD* se reciben las muestras de tensión de la medición de amplitud. La cola *queue_hab_phase* le permite a la tarea conocer si se pudo realizar la medición de fase. Si se pudo hacer, con la cola *queue_USB_PHASE* se reciben las muestras de desfasaje temporal; sino en esta variable se guarda un único '0' en la primera posición.

Con *sem_usb*, "MeasureTask" informa a "USBTask" que la medición para una frecuencia terminó, y mientras el dispositivo espera 1s a que se estabilicen los detectores de pico para la próxima medición, ambos paquetes de fase y magnitud se mandan por USB en dos transferencias independientes con un delay de 0.5s entre ellas para evitar colisiones y pérdida de datos durante la comunicación.

5. Consumo eléctrico

El dispositivo se alimenta con fuente externa partida de ± 12V, y consume 180mA de la fuente positiva y 25mA de la fuente negativa.

Al tener reguladores de 9V se debe tener en cuenta la potencia disipada en cada uno:

$$P_{regulador+9V} = (12V - 9V) \cdot 180mA = 540mW \tag{9}$$

$$P_{regulador-9V} = (12V - 9V) \cdot 25mA = 75mW \tag{10}$$

Según la hoja de datos la resistencia térmica de juntura-ambiente θ_{ja} es de $65\frac{^oC}{W}$, por lo que se puede calcular una temperatura de juntura suponiendo una temperatura ambiente de 40^oC como extremo:

$$Regulador + 9V : T_J = 540mW \cdot 65\frac{^oC}{W} + 40^oC = 75{,}1^oC < 150^oC \tag{11}$$

$$Regulador - 9V : T_J = 75mW \cdot 65\frac{^oC}{W} + 40^oC = 44{,}875^oC < 150^oC \tag{12}$$

Sin colocar disipadores, la temperatura de juntura de ambos reguladores es menor a la máxima para el silicio (150^oC) con un gran margen. Aunque no sea necesario, se colocó un disipador al regulador positivo para que trabaje a una temperatura todavía menor a 75.1^oC.

Por otro lado, la fuente positiva debe alimentar la placa *NUCLEO-F401RE* a través del pin *Vin*. El esquemático de la alimentación de la placa es el siguiente:

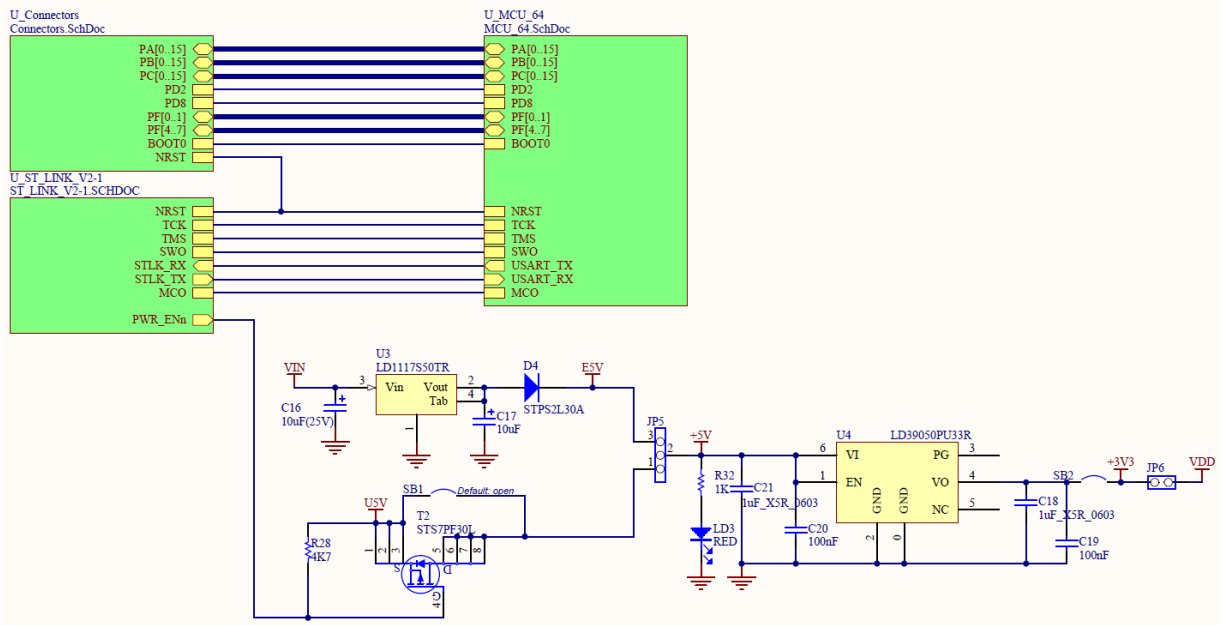

Figura 20: Esquemático de NUCLEO-F401RE

Se observa que el jumper en JP5 debe unir E5V con +5V, y que se tiene un regulador lineal U3 entre VIN y E5V. Para saber si es correcto alimentar con 9V a VIN se usó previamente el manual de la NUCLEO-F401RE:

Table 7. External power sources

Input power name	Connectors pins	Voltage range	Max current	Limitation
VIN	CN6 pin 8 CN7 pin 24	7 V to 12 V	800 mA	From 7 V to 12 V only and input current capability is linked to input voltage: 800 mA input current when Vin = 7 V 450 mA input current when 7 V < Vin <= 9 V 250 mA input current when 9 V < Vin <= 12 V
E5V	CN7 pin 6	4.75 V to 5.25 V	500 mA	-

Figura 21: Tabla de alimentaciones externas NUCLEO-F401RE

El rango de tensión de entrada al regulador interno de la placa es de 7V a 12V por lo que 9V es aceptable, y con 9V la corriente máxima permitida es de 450mA por lo que el consumo de 180mA de corriente está dentro del límite.

6. Planificación

6.1. Cronograma del proyecto

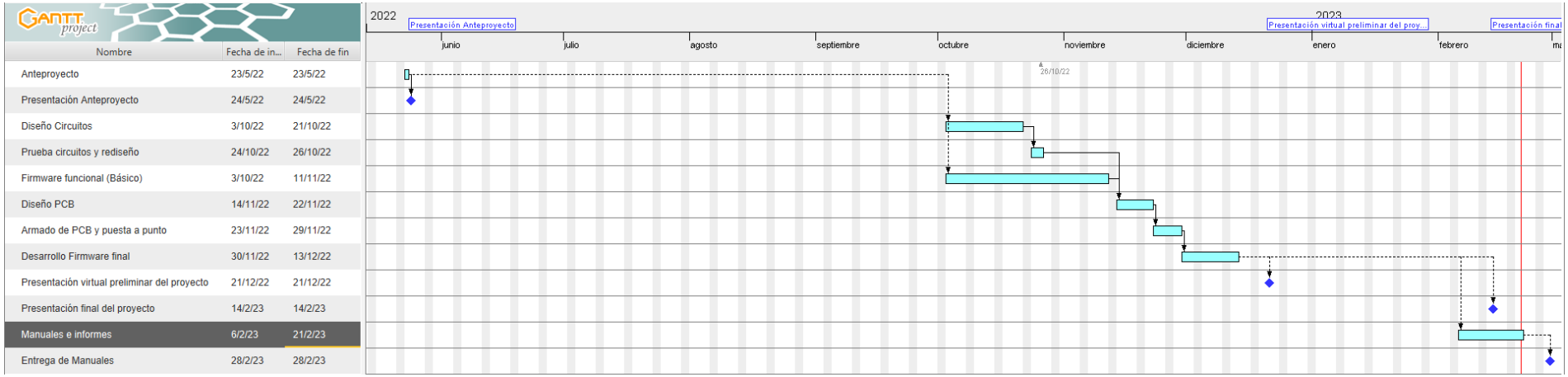

Figura 22: Diagrama de Gantt

6.2. Costos del proyecto

A continuación se detallan los costos del proyecto (Fig.23).

Costos	Cantidad	Precio	SubTotal
Pantalla TFT	1	11086	11086
Percloruro Ferrico	1	1192	1192
OPA604APG	8	436	3488
LM311	6	436.8	2620.8
CAP0805 100nF X7R	100	7.72	772
CAP0805 10uF 16V X5R	25	12.46	311.5
CAP ELECT 220uF 25V	10	29.66	296.6
IC REG 9V 1.5A TO220	1	190.91	190.91
IC REG -9V 1.5A TO220	1	241.46	241.46
CONECTOR MINIUSB FEMALE_SMD	1	152.59	152.59
POSTE RECTO 40x1 2.54MM	5	82.3	411.5
HE-SF VERT 10x1 2.54MM	5	60	300
HE-SF VERT 20x2 2.54MM	2	240.03	480.06
DIODE 75V 150MA SOD323F	100	8.19	819
3296 TRIMMER 100K OHM 0.5W PC PIN	1	270.15	270.15
RES0805 680R 1%	100	2.08	208
RES SMD 1K OHM 1% 1/4W 0805	100	2.08	208
RES0805 22K 1%	100	2.08	208
RES0805 100K 1%	100	2.08	208
RES0805 4K7 5%	100	1.67	167
MOSFET N-CH 60V 200MA TO-92	5	29.58	147.9
CONN BNC RCPT STR 50 OHM PCB	1	728.08	728.08
3296 TRIMMER 20K OHM 0.5W PC PIN	1	181.2	181.2
Horas hombre (CE)	190	5000	950000
Total			974688.75

Figura 23: Costos

7. CALIBRACIÓN

7.1. Calibración de ADC

Se obtiene la transferencia del ADC, y se almacena en la memoria FLASH del microcontrolador. Luego se realiza una interpolación lineal para cada cuenta obtenida del ADC, de manera de obtener la tensión de entrada aproximada que la generó.

La calibración consta de un generador de señales arbitrario Rigol DG5070 [10] controlado con una computadora mediante USB, que genera una rampa escalonada de 1024 niveles de tensión de 0V a 3.3V mantenidas 4ms cada una. El microcontrolador realiza 3 lecturas separadas 1ms entre sí comandado por la misma PC que controla al generador, y devuelve el resultado de la medición a la PC.

Se promedian los valores de tensión correspondientes a un mismo código, y luego se hace un ajuste lineal de la transferencia (Fig. 24). Con los valores de la transferencia ajustada a una recta se genera una matriz de 1024 filas (puntos de medición), y dos columnas siendo la primera de cuentas y la segunda de la tensión de entrada en función de la cuenta obtenida.

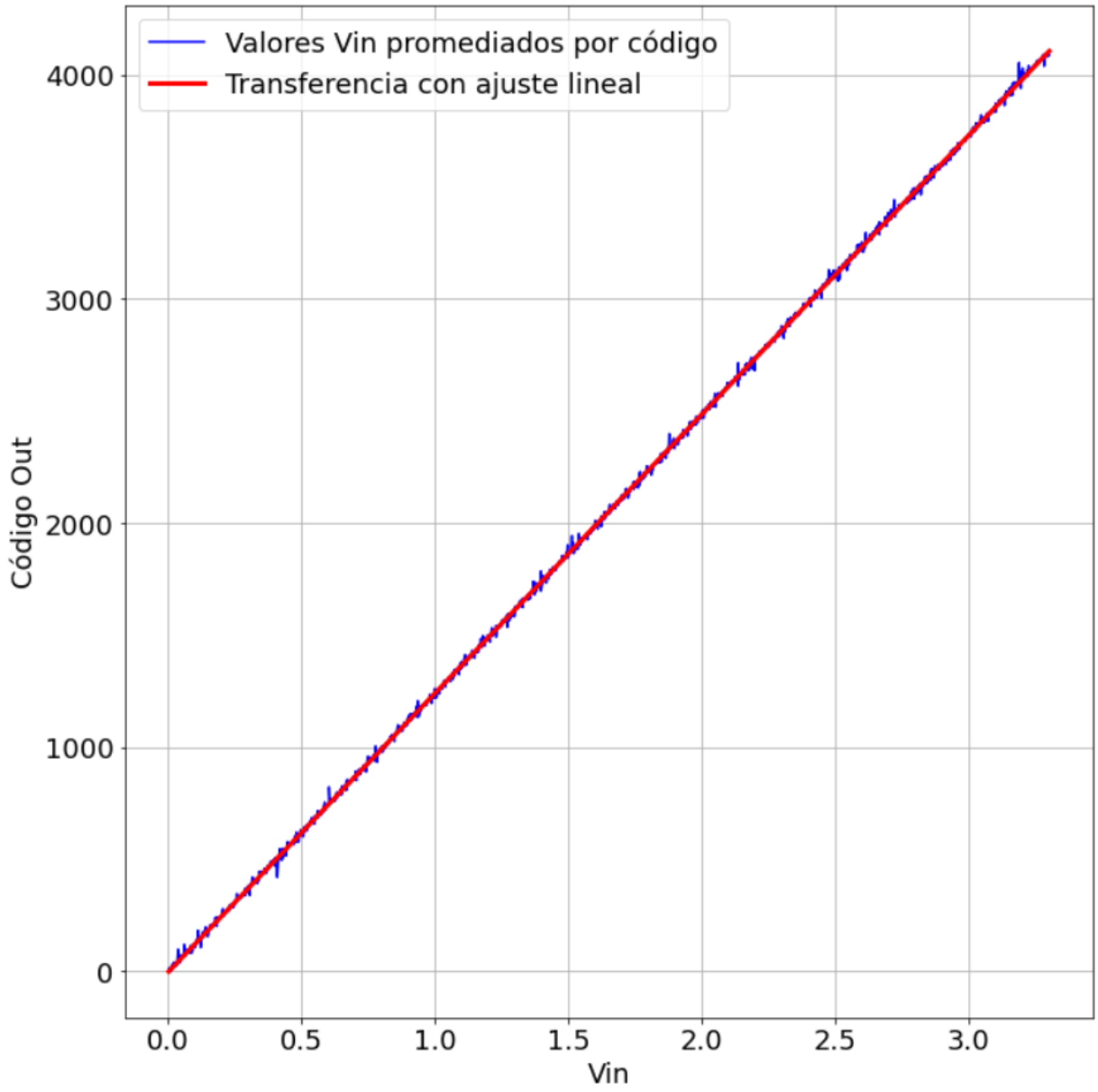

Figura 24: Transferencia de ADC

Teniendo esta matriz en la flash del microcontrolador (const float transf_ADC[PUNTOS]), cada vez que se obtenga una cuenta (C) con el conversor se debe encontrar la cuenta de la matriz que sea inmediatamente menor a la cuenta obtenida (C_{INF}), la cual tendrá su respectiva tensión de entrada (V_{INF}), y la cuenta que sea inmediatamente mayor (C_{SUP}) también con su respectiva tensión analógica (V_{SUP}). Con estos valores se realiza el siguiente cálculo:

$$V_{IN} = \frac{V_{INF}(C_{SUP} - C) + V_{SUP}(C - C_{INF})}{C_{SUP} - C_{INF}} \tag{13}$$

El cual se deduce de averiguar la ecuación de la recta que pasa por los puntos (C_{INF}, V_{INF}) y (C_{SUP}, V_{SUP}). En otras palabras, una interpolación lineal.

7.2. Calibración de timer con Input Capture

Se utiliza como patrón un frecuencímetro Protek U2000A [11] utilizado como medidor de intervalo de tiempo promediado por 100 mediciones. Con el generador de señales Agilent 33500 [12] se generan dos señales cuadradas a modo de evitar error de trigger del frecuencímetro, y se mide el desfasaje temporal de las mismas. La medición se realiza desde 1us hasta 1s con el microcontrolador y con el frecuencímetro. En este ensayo también se obtiene una matriz de dos columnas (valor del timer y valor de patrón) y 25 filas (puntos de medición). Las mediciones se encuentran separadas exponencialmente debido a que es más probable que se produzcan intervalos de tiempo de microsegundos hasta las decenas milisegundos, que a del orden de las décimas segundo en adelante (dado que esto solo ocurre con valores de fase grandes en baja frecuencia) (Fig. 25). Luego esta matriz es almacenada en la FLASH y utilizada por el MCU en cada medición de tiempo, para corregir el resultado utilizando una interpolación lineal idéntica a la utilizada para calibrar el ADC.

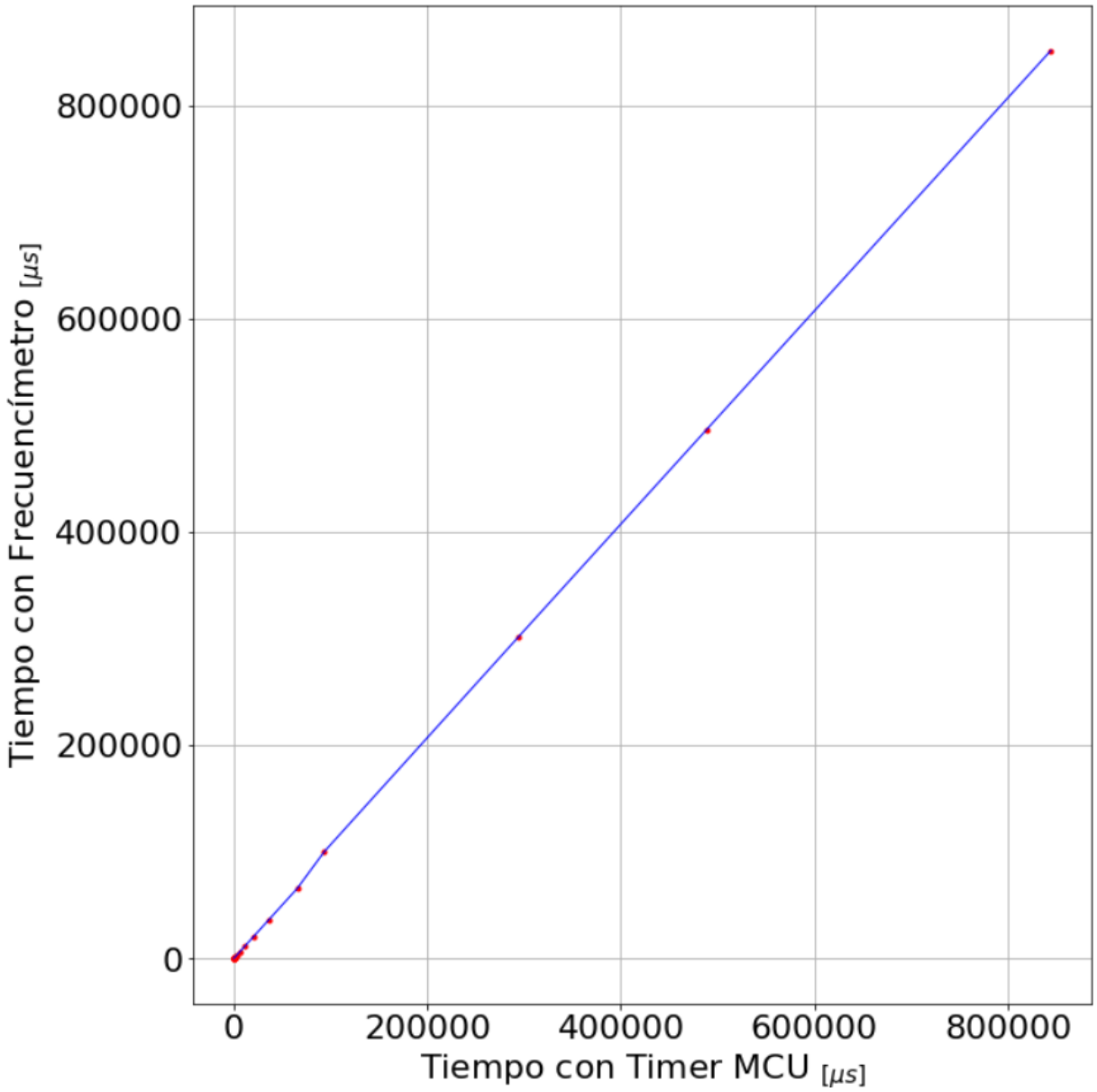

Figura 25: Calibración de medición de intervalo de tiempo

8. INCERTIDUMBRE

Se tiene incertidumbre tipo A en los valores de tensión medidos con el ADC, y en los valores de intervalo de tiempo medidos con el timer, dado que se toman 10 muestras de la tensión de entrada al filtro, 10 muestras en la tensión de salida y 10 muestras de intervalo de tiempo respectivamente.

Estas muestras son almacenadas por el MCU, y en cada punto de medición de frecuencia se realiza una transferencia por USB con una aplicación que se ejecuta en una computadora aparte.

La aplicación calcula los valores medios y las incertidumbres TIPO A para cada punto de frecuencia de: la tensión de entrada al filtro, la tensión de salida y el intervalo de tiempo.

Con los datos aportados por los manuales del generador arbitrario utilizado en la calibración del ADC y el manual del frecuencímetro utilizado en la calibración del input capture, se calcula la incertidumbre tipo B de las mismas magnitudes mencionadas en el párrafo anterior, y con ellas las combinadas.

Para obtener las incertidumbres combinadas del módulo y de la fase, se aplica la teoría de propagación de incertidumbre, calculando las derivadas parciales que correspondan.

Luego de varias pruebas se determina que tanto las 10 mediciones de las cuentas devueltas por el timer del input capture como las 10 mediciones de las cuentas entregadas por el ADC, resultan precisas, es decir, que prácticamente no hay incertidumbre en el estudio TIPO A, por lo que se considera que la mayor parte del error proviene del estudio TIPO B. Entonces, todas las incertidumbres combinadas poseen distribución uniforme y se expanden al 95 % con $k = \sqrt{3}$.

La incertidumbre en cada magnitud medida se expresa de forma gráfica con una banda del intervalo de confianza sobre el mismo gráfico que muestra magnitud y fase por separado.

Luego de un estudio de incertidumbre, en las figuras 26, 27, 28, 29 se muestran como ejemplo las mediciones de un filtro pasa-bajos RC y un filtro Notch pasivo, con sus respectivas bandas de incertidumbre para la fase y módulo.

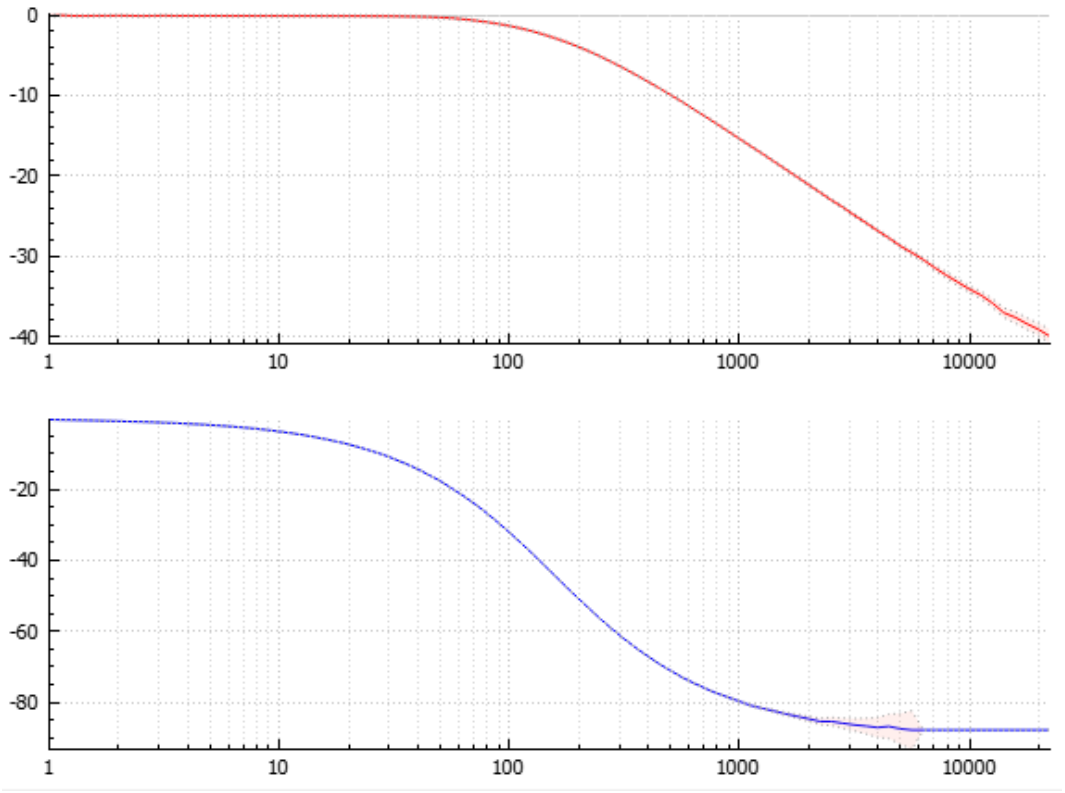

Figura 26: Filtro pasa-bajos RC

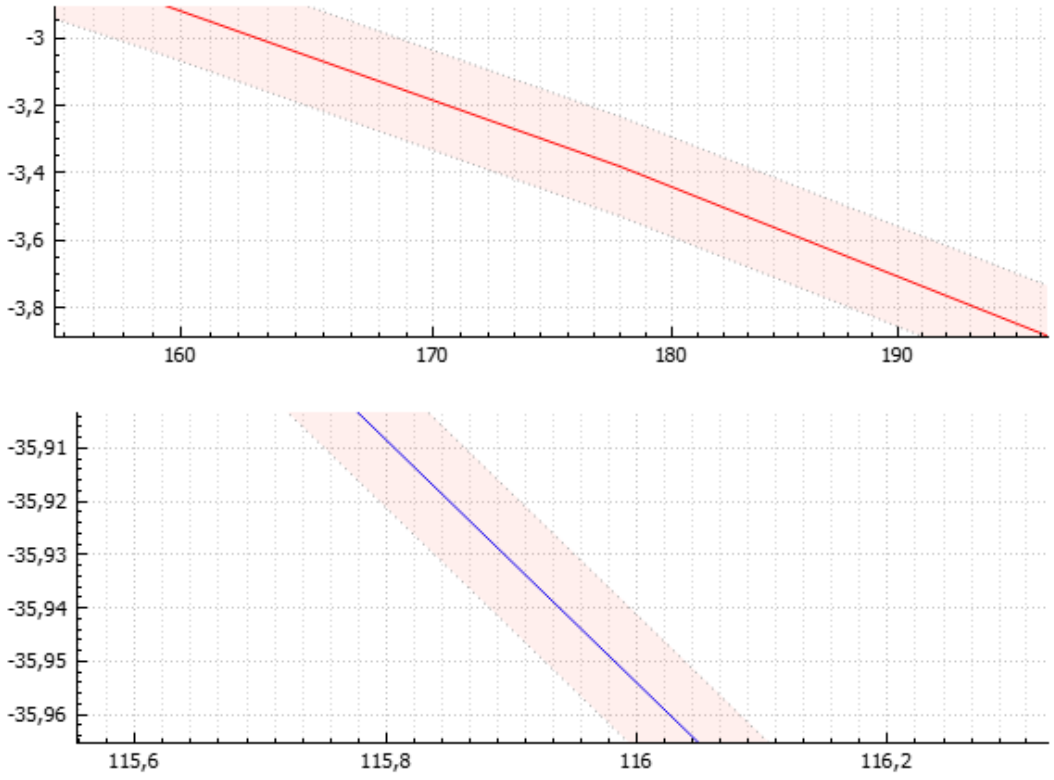

Figura 27: Ampliación de gráficos Filtro pasa-bajos RC

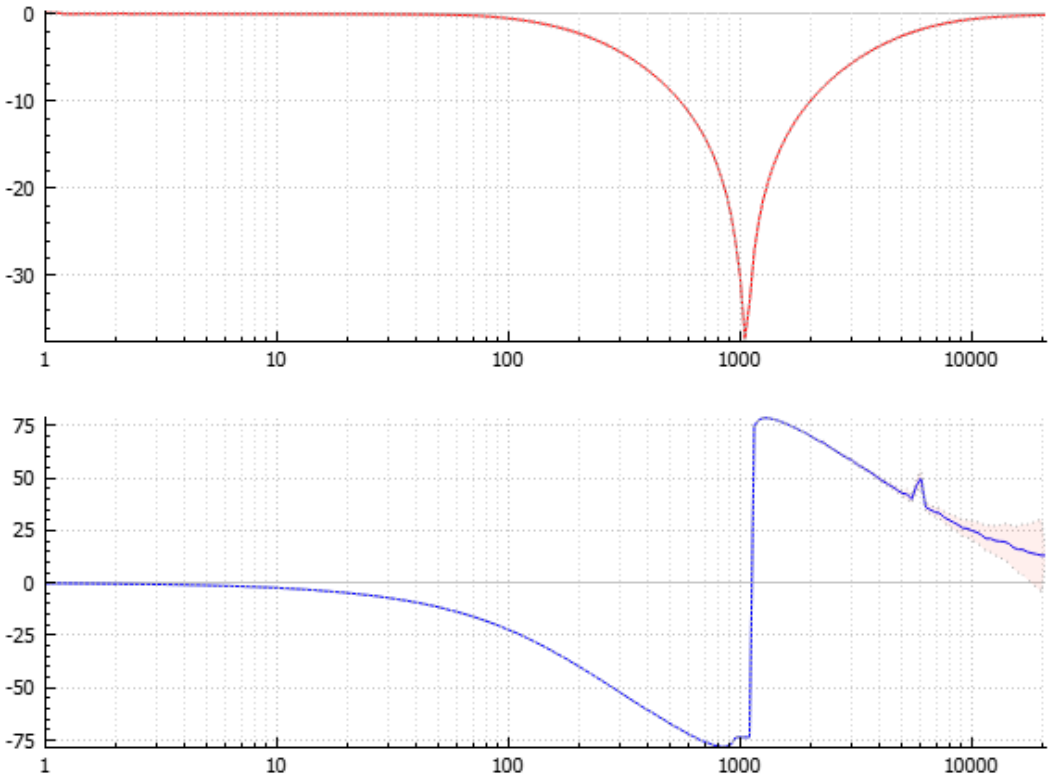

Figura 28: Filtro Notch pasivo

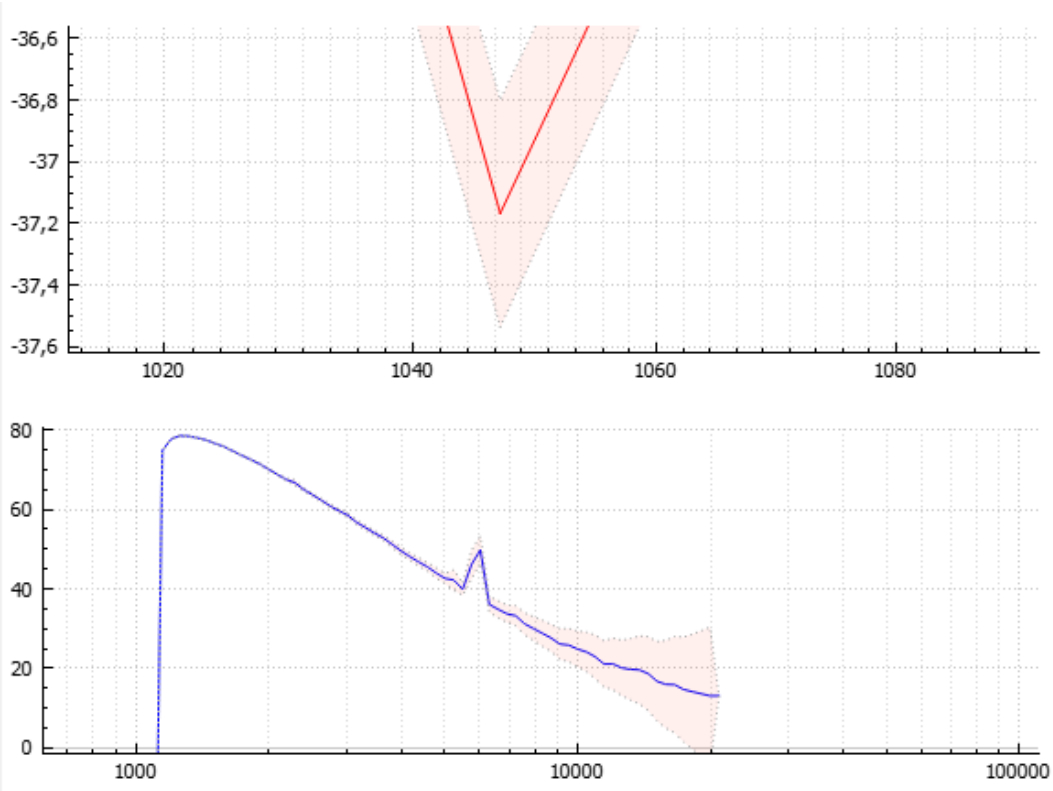

Figura 29: Ampliación de gráficos filtro Notch pasivo

9. Conclusiones

9.1. Resultados

Los resultados obtenidos del cálculo de incertidumbre reflejan que el dispositivo es suficientemente preciso y que no hay casi dispersiones en las curvas medidas. Sin embargo, a frecuencias más altas cercanas al límite del dispositivo, se puede observar como aumenta la incertidumbre en la fase debido a que la calibración tiene una limitación en la resolución del intervalo de tiempo, la cual es de 1ns. El estudio tipo A hecho mostró que la incertidumbre es casi despreciable en comparación a la que introduce el frecuencímetro utilizado a esas frecuencias.

Para la incertidumbre que muestra el módulo, ocurre algo similar a la fase en cuanto a que aumenta la incertidumbre pero cuando la señal a medir es pequeña. En este caso la incertidumbre la aporta casi toda la tipo A debido a que el dispositivo intenta medir señales que están cercanas al piso de ruido del sistema, por lo que aumenta la variación en las cuentas que arroja el ADC del MCU debido a este efecto; por lo tanto aumenta la incertidumbre tipo A.

9.2. Limitaciones

El ancho de banda resultó ser mucho menor al esperado antes de realizar el proyecto. Por un lado porque para la medición de módulo se optó por un circuito detector de pico con bajo nivel de ruido y que mida tensiones bajas en el orden de los milivolts a otro con buena respuesta en frecuencia pero con mayor nivel de ruido y por ende menor precisión. Por otro lado, al medir la fase, si bien los comparadores funcionan perfectamente hasta 1MHz, el input capture deja de detectar los flancos de las señales cuadradas una vez superados aproximadamente los 25kHz (por eso el límite a 20kHz dejando un margen de seguridad).

Otro problema, mencionado en la sección de rangos, tiene que ver con la medición de amplitud a baja frecuencia y baja amplitud debido que el ripple de descarga de los capacitores de los detectores de pico se vuelve considerable.

La medición se torna ciertamente lenta, ya que le toma 1 segundo a los capacitores de los detectores de pico para descargarse y cargarse nuevamente a la tensión correspondiente, por lo que para una medición con mucha resolución en frecuencia (muchos puntos), puede tardar hasta casi 10 minutos en medir.

9.3. Posibles Mejoras

Una forma de mejorar sustancialmente el sistema es utilizando un microcontrolador con un ADC más rápido para poder muestrear la señal, y con mayor frecuencia de clock para poder realizar un algoritmo de Goertzel, el cual se puede tomar como una transformada de rápida Fourier pero que sólo devuelve un número complejo correspondiente al bin de una frecuencia específica, la cual configuraríamos para que sea la frecuencia que generamos con el DDS. De esta manera se puede reducir el hardware al máximo, retirando los detectores de pico y los comparadores, y no depender del input capture del timer interno del microcontrolador. Esto tiene como consecuencia una mejora en la exactitud de la medición (al evitar errores y limitaciones del hardware adicional), en el ancho de banda, ya que solo depende de la velocidad del ADC y de procesamiento del microcontrolador, y en el tiempo de medición dado que no hay que esperar a que se descarguen y se carguen los capacitores de los detectores de pico.

Una mejora a este último sistema mencionado sería realizar un barrido en frecuencia casi instantáneo componiendo un espectro de agujas, centradas en las frecuencias de interés, y luego obtener una señal asociada este espectro mediante una transformada rápida de Fourier inversa. Esta sería la señal que excitaría al filtro a analizar. Luego a la señal filtrada se le aplica una transformada rápida de Fourier para analizar de forma simultánea como se modificaron las agujas en el espectro de salida.

9.4. Desafíos y problemáticas

1. PCB
   a) Por una cuestión de dinero y tiempo, la placa de dos capas fue hecha a mano, y hubo un ligero desfasaje entre las mismas que generó que algunos pads no tengan la corona completa, y la soldadura fue más difícil. Se deberían haber agrandado las coronas de los pads para asegurarse de tener suficiente cobre a pesar de que ocurra desfasaje.
   b) Las vías con las que el PCB traslada señales de una capa a otra, están hechas con pines macho, por lo cual se tuvo que cortar y soldar uno por uno para cada vía. Esto produjo en muchos casos lagunas de estaño por mala soldadura que terminaban en cortocircuitos.
   c) Deterioro de los pads de muchos agujeros de componentes THT. Al tener que agujerear a mano todos ellos, la corona que inicialmente estaba pensada para un PCB hecho profesionalmente con agujereadora de una máquina automática, era destruida en muchos casos, haciendo mucho más complicada la soldadura de componentes.
   d) En muchas partes de la placa se produjeron cortocircuitos durante el proceso de traspaso del PCB de la hoja al cobre.
   e) Las prueba iniciales sobre protoboard no permitían sacar resultados correctos para tomar decisiones sobre el hardware debido a un alto nivel de ruido existente, por lo que se tomó el riesgo de proseguir con el hardware propuesto, y desacoplar el ruido lo máximo posible con capacitores de acoplamiento en todas las alimentaciones del circuito. Al probar el PCB, el ruido quedó muy atenuado. Todos estos problemas fueron solucionados ya que la placa anduvo al 100 %
2. MCU
   a) Encontrar la frecuencia máxima en la que el timer utilizado para el Input Capture deja de funcionar. Al principio no se sabía si eran errores de lógica de software o mala configuración de periféricos. Finalmente se estableció que la frecuencia máxima con la que podía trabajar el timer era 1MHz (cuentas de $1\mu s$).
   b) Algunos canales de los timers funcionaban incorrectamente como input capture. Se probaron uno a uno todos los canales en busca de todos los que funcionan correctamente, para después elegir los más cómodos para el diseño del PCB.
   c) Exceso de Flash al usar TouchGFX para generar las pantallas. En el diseño de las pantallas se usaron los widgets que menos memoria consumen, y la cantidad mínima posible.
   d) Problemas para hacer mediciones inicialmente con el ADC debido a que se usaron canales compartidos con la UART. Se desoldaron unos resistores de 0Ω que desconectaban la UART, tal como especificaba el manual de la NUCLEO.
   e) El SPI compartido entre el touch y el AD9833 utiliza dos configuraciones distintas tanto de polarización y fase, como de tamaño de trama. Por lo que se tuvo que encontrar una forma a través del HAL para reconfigurar el SPI cuando se utilizaba para dos comunicaciones distintas
   f) El USB utilizado no permite grandes volúmenes de transferencias sucesivas, es decir sucesivos llamados a función para realizar transferencias. Por lo que se tuvo que mandar tramas separadas por delays.
   g) Las funciones de USB al trabajar con uint8, y la información que se envía a la PC del tipo float, requirió realizar un casteo de floats como uint8 y verificar que el endianess de la PC y el microcontrolador sea el mismo.

3. IDE

   a) El generador de código (STM32CubeMX) de versión 6.7.0 funcionaba mal y recortaba código escrito en regiones donde no debería borrar nada. Cuando se identificó el problema, se restauró el código con github y se pasó a una versión del software anterior (6.6.1).

   b) El explorador de variables no funciona correctamente, al modificar variables en el código el explorador mostraba que se modificaban con valores erróneos, e incluso se modificaban otras variables en simultáneo. Esto hizo que se llegue a una primera conclusión errada de que había corrupción de memoria. Una vez identificada la verdadera causa se optó por utilizar el segundo explorador de variables que aparece como ventana emergente.

Referencias

[1] REDPITAYA, *Bode Analyzer*, Disponible en https://redpitaya.com/applications-measurement-tool/bode-analyzer/.

[2] O. LAB, *Bode 100*, Disponible en https://www.omicron-lab.com/products/vector-network-analysis/bode-100#.

[3] STM, *TouchGFX Documentation*, Disponible en https://support.touchgfx.com/4.21/docs/introduction/welcome.

[4] STM, *STM32F401xB/C and STM32F401xD/E advanced Arm®-based 32-bit MCUs*, Disponible en https://www.st.com/resource/en/reference_manual/rm0368-stm32f401xbc-and-stm32f401xde-advanced-armbased-32bit-mcus-stmicroelectronics.pdf (19/12/18).

[5] STM, *STM32 Nucleo-64 boards (MB1136)*, Disponible en https://www.st.com/resource/en/user_manual/um1724-stm32-nucleo64-boards-mb1136-stmicroelectronics.pdf (20/08/20).

[6] ILITEK, *a-Si TFT LCD Single Chip Driver 240RGBx320 Resolution and 262K color*, Disponible en https://cdn-shop.adafruit.com/datasheets/ILI9341.pdf.

[7] A. Devices, *Low Power, 12.65 mW, 2.3 V to 5.5 V, Programmable Waveform Generator*, Disponible en https://www.analog.com/media/en/technical-documentation/data-sheets/ad9833.pdf (04/19).

[8] T. Instruments, *LM111, LM211, LM311 Differential Comparators*, Disponible en https://www.ti.com/lit/ds/symlink/lm311.pdf (03/17).

[9] T. Instruments, *FET-Input, Low Distortion OPERATIONAL AMPLIFIER*, Disponible en https://www.ti.com/general/docs/suppproductinfo.tsp?distId=26&gotoUrl=https://www.ti.com/lit/gpn/opa604 (09/03).

[10] RIGOL, *Arbitrary Waveform Generator Specifications*, Disponible en https://beyondmeasure.rigoltech.com/acton/attachment/1579/f-0051/0/-/-/-/-/file.pdf.

[11] PROTEK, *Frequency Counter Specifications*, Disponible en http://www.arivigevano.net/files/pk9100.pdf.

[12] AGILENT, *Waveform Generator Specifications*, Disponible en https://electricayelectronica.uniandes.edu.co/sites/default/files/laboratorios/manual-33500-Op-and-Svc-US.pdf.