Universidad Nacional de La Plata

Facultad de Ingeniería

Sistemas Embebidos

Informe Trabajo especial 2
Analizador de espectro

Alumna:
Paloma Domínguez Estrada
72331/4

Profesores:
Federico Guerrero
Marcelo Haberman

1 de agosto de 2024

Índice

1. Introducción

En el presente informe se desarrolla la implementación de un analizador de espectro utilizando el sistema operativo en tiempo real FreeRTOS. Este sistema tiene la capacidad de capturar señales a una tasa de muestreo de 1 kHz mediante un ADC, y realizar una Transformada Rápida de Fourier (FFT) de 256 puntos. La visualización de los datos se realiza en una pantalla, mostrando las frecuencias positivas de la transformada en los primeros 128 puntos.

Adicionalmente, el sistema incluye un generador de señales senoidales mediante PWM para excitar el analizador en el rango de frecuencias que es capaz de medir. Se implementa una interfaz de usuario con un encoder rotatorio y dos pulsadores, permitiendo la selección y configuración de diferentes parámetros del sistema y la visualización de la FFT con un cursor interactivo.

El objetivo principal es diseñar y desarrollar un sistema responsivo, capaz de gestionar múltiples tareas en paralelo utilizando FreeRTOS. Para ello, se implementan diversos mecanismos de sincronización entre tareas e interrupciones, asegurando una operación coherente y estable del sistema. Además, se propone un esquema de tareas, que se desarollan en el infore, que incluye la toma de muestras, procesamiento de datos, monitoreo de entradas y visualización en pantalla.
ANexo codigo de las tareas, cambiar la primera parte de software, agregar periodo de envio por uart. Explicar las prioridades.

2. Diagrama esquemático

El diagrama del circuito esquemático del sistema se muestra a continuación:

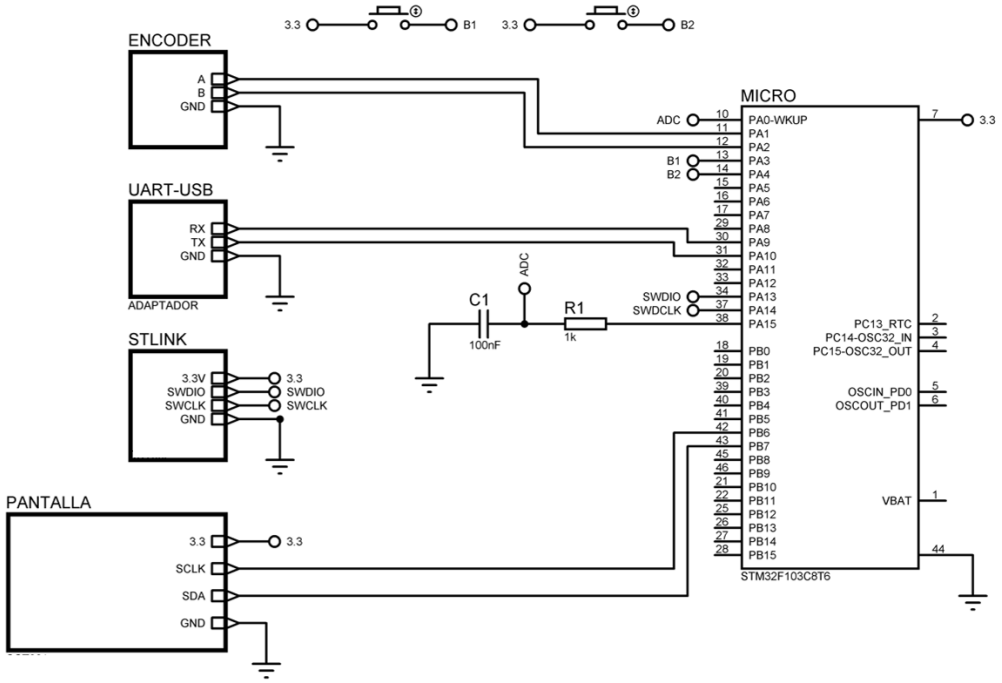

Figura 1: Diagrama esquemático de los distintos componentes que conforman el analizador de espectro

Por un lado, las entradas al sistema son: los dos pulsadores, el encoder y la señal analógica que desea medir. Por otro lado, las salidas del sistema son la señal PWM, la cual se genera en el Pin PA15, las líneas de comunicación por UART para transmitir los datos a la computadora y las líneas de comunicación por I2C para comandar la pantalla LED. Toda la alimentación es brindada por el microcontrolador, que a su vez este está conectado a la computadora por medio del programador STLINK.

Por otra parte, en la Tabla 1 se resume los recursos y periféricos que se utilizan para la implementación del analizador de espectro.

Recurso/Periférico	Funcionalidad
USART1	Transferir datos de la FFT a la computadora
I2C1	Comunicación con la pantalla LED
ADC1	Un canal para adquirir las 256 muestras
TIMER 1	Genera interrupciones cada 1ms, establece el tiempo de muestreo
TIMER 2	Genera la señal PWM
TIMER 3	Determina cada cuanto el DMA realiza una transferencia de datos
DMA	Transfiere desde memoria hacia el contador del TIMER 2

Tabla 1: Resumen de los periféricos utilizados

3. Interfaz de ususario

El analizador de espectro cuenta con tres pantallas, cada una con una funcionalidad específica. El usuario pasa de una pantalla a la siguiente oprimiendo el pulsador número 1.

3.1. Pantalla de Configuración

La primera pantalla permite configurar la funcionalidad del encoder. El usuario se desplaza por las opciones mediante el encoder y una vez que desea seleccionar debe oprimir el pulsador 2. Las opciones disponibles son:

- **Cursor:** Al seleccionar esta opción, cuando el usuario esté en la pantalla de visualización de la FFT (Pantalla 3) y mueva el encoder, el cursor se desplazará en la gráfica. Simultáneamente, se mostrarán los valores de amplitud y frecuencia correspondientes a la posición del cursor.
- **Frecuencia:** Si se elige esta opción, el movimiento del encoder en la pantalla de visualización de la FFT (Pantalla 3) modificará la frecuencia de la señal generada. Esto se reflejará en la gráfica como un desplazamiento del pico de amplitud en el espectro.
- **Amplitud:** Seleccionando esta opción, el movimiento del encoder en la pantalla de visualización de la FFT (Pantalla 3) ajustará la amplitud de la señal generada, permitiendo observar los cambios en tiempo real en la gráfica.
- **Impresión por UART:** Al seleccionar esta opción, los últimos valores de la transformada de Fourier de la señal, tanto en su parte real como imaginaria, se transmitirán a través del protocolo UART. Esto permite replicar la gráfica mostrada en la pantalla en una computadora.

3.2. Pantalla de Información

La segunda pantalla muestra la amplitud y frecuencia actuales de la señal senoidal que se está generando. Además, indica la funcionalidad actual del encoder configurada en la Pantalla 1, que puede ser çursor", "modificar frecuencia.º "modificar amplitud".

3.3. Pantalla de Visualización de la FFT

La tercera pantalla permite visualizar la gráfica de la transformada de Fourier junto con los valores de frecuencia y amplitud en la posición del cursor. Esta pantalla interactúa con las configuraciones seleccionadas en la Pantalla 1, permitiendo el ajuste dinámico de la señal generada y la visualización en tiempo real del espectro.

4. Tareas e interrupciones

Las tareas e interrupciones programadas para el correcto funcionamiento del analizador de espectro se explican a continuación:

Tarea Entradas

En esta tarea se implementa una máquina de estados, cuyos eventos están determinados por los dos pulsadores y la rotación del encoder. El esquema de la máquina de estados es el siguiente:

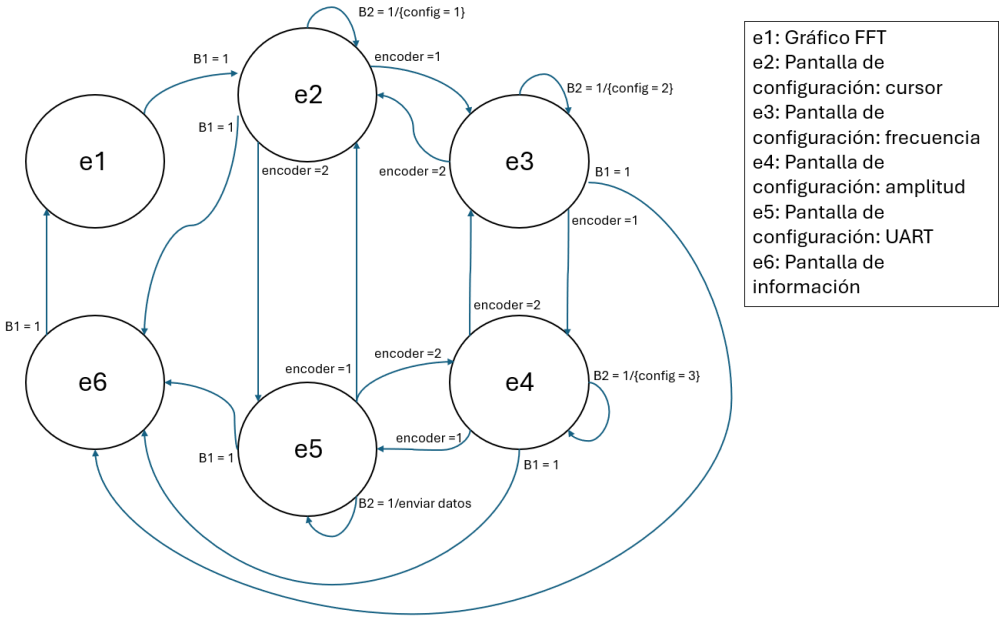

Figura 2: Máquina de estados implementada para administrar las entradas y lo que se imprime en pantalla

Cada uno de los estados se corresponde con lo que se imprime en pantalla. Para esto, se utiliza una cola (Queue) como elemento de sincronización entre esta tarea y la tarea que se encarga de la pantalla. Cada vez que se pasa de un estado a otro, se guarda una variable entera en la Queue que será interpretada por la otra tarea.

Los estados $e2$, $e3$ y $e4$ corresponden a las pantallas de configuración del encoder. Cuando se selecciona alguna de estas opciones, se actualiza una variable global `config`, que luego será utilizada por otras tareas. Específicamente, si `config` vale 1, se actualizará la posición del cursor que se imprime en pantalla. En cambio, si `config` vale 2 o 3, se actualizará el arreglo que almacena los valores de la señal con las frecuencias o amplitudes que se vayan modificando en la Tarea Seno.

Además, se utiliza un semáforo como elemento de sincronización entre esta tarea y la Tarea Procesamiento. De esta manera, cuando se pasa del estado $e1$ al $e2$, primero se finaliza el procesamiento de las muestras, asegurando que estos datos queden almacenados en memoria.

Tarea Pantalla

Esta tarea se encarga de gestionar la pantalla OLED SSD1306 y actualizar su contenido basado en las entradas del sistema y los estados internos. La tarea maneja varias pantallas que muestran diferentes tipos de información y configuraciones.

Inicialmente, se verifica si hay mensajes en la cola `myQueue01Handle`. Si hay mensajes, se obtiene el valor y se almacena en una variable denominada `flag_pantalla` y se limpia la pantalla. Si `flag_pantalla` vale 1, se adquiere un mutex para asegurar la exclusión mutua con la

Tarea Procesamiento al acceder a los datos. Luego, se actualiza la pantalla con la gráfica de la FFT, los valores de frecuencia y amplitud, y el cursor. Después, se libera el mutex y se activan las interrupciones del timer 1 para adquirir nuevas muestras. Si `flag_pantalla` vale de 2 a 6, se muestran las opciones de configuración o la información de la señal, dependiendo de su valor.

Tarea Encoder

En esta tarea se emplea una máquina de estados que permite detectar si el encoder fue rotado una posición. Para esto, se actualiza el valor de una variable global `encoder`, que por defecto vale 0. Si el encoder es rotado una posición en sentido horario, `encoder` se actualiza a 1; si es rotado en sentido antihorario, `encoder` se actualiza a 2. Dependiendo del estado en el que se encuentre el sistema, la rotación del encoder puede modificar las opciones de la pantalla de configuración o los distintos parámetros cuando se grafica la FFT.

Tarea Seno

Esta tarea solo funciona cuando se requiere modificar la amplitud o frecuencia de la señal senoidal. Para esto, se detiene la generación de la señal PWM con el timer 3 y el DMA, se actualiza el arreglo que almacena la señal senoidal con los valores de frecuencia o amplitud establecidos, y se vuelve a habilitar el timer y el DMA para generar la señal PWM y transferir los datos al comparador.

Tarea Procesamiento

Esta tarea funciona en el estado en que se calcula e imprime la FFT. Una vez que se obtienen las 256 muestras mediante el ADC, se toma un mutex para asegurar la exclusión mutua con la Tarea Pantalla y evitar conflictos con los datos. Luego, se obtiene la transformada de Fourier de la señal y se calcula su módulo. Al finalizar, se libera el mutex para que la otra tarea pueda realizar la gráfica requerida.

Interrupción por Timer 1

Se utiliza la interrupción de este timer para cumplir con el tiempo de muestreo. Cada vez que se genera esta interrupción, se habilita la interrupción del ADC para tomar una muestra. La interrupción por timer comienza a funcionar cuando se pasa al estado $e1$ de la máquina de la Tarea Entradas y se deshabilita cuando se toman las 256 muestras, permitiendo que estas se procesen y se impriman. Luego de la impresión, la interrupción se vuelve a habilitar mientras el sistema esté en la pantalla de impresión del espectro. Cuando se sale de esa pantalla, las interrupciones se deshabilitan.

Interrupción por ADC

Esta interrupción es activada por la interrupción del timer 1. Se toman las muestras del ADC y se almacenan en una estructura para luego calcular la FFT. Al tomar 256 muestras, se deshabilitan las interrupciones por timer 1.

4.1. Prioridades

Todas las tareas del sistema operativo tienen la misma prioridad, la cual se estableció en Normal. Esta asignación de prioridades se debe a la utilización del mecanismo de time slicing. Este mecanismo funciona de la siguiente manera: el scheduler asigna un tiempo específico a

cada tarea. Cuando una tarea consume su tiempo de ejecución asignado, el scheduler selecciona la próxima tarea de una lista de tareas en estado READYz le asigna una nueva cantidad de tiempo. Si no hay ninguna tarea lista, se ejecuta la tarea ÏDLE". Este mecanismo asegura que todas las tareas de igual prioridad reciban tiempo de CPU de manera justa y equitativa, previniendo que alguna tarea se ejecute de manera continua sin permitir la ejecución de otras tareas.

Las interrupciones por hardware, específicamente las del timer 1 y del ADC tienen prioridades que se relacionan con las de las tareas del sistema operativo, ya que el cambio de contexto del sistema operativo se realiza a partir de interrupciones. Para garantizar el muestreo uniforme de la señal senoidal de entrada, se estableció que la interrupción del timer 1 tenga una prioridad mayor que las interrupciones de cambio de contexto del sistema operativo. Mientras que la interrupción del ADC tiene la misma prioridad que las interrupciones del sistema operativo. Esto es posible porque el tiempo de muestreo de 1 ms es lo suficientemente largo para que el esquema de las tareas continúe sin necesidad de interrupciones adicionales.

5. Depuración del Stack

A continuación se muestra en la Tabla 2 el stack asignado a cada tarea en palabras de 4 bytes. En la Tabla 3 se encuentran los datos del Heap del sistema operativo.

Tarea	Stack	Stack libre	Stack utilizado
Pantalla	180	33	147
Entradas	180	52	128
Encoder	128	93	93
Seno	128	66	66
Procesamiento	128	68	68

Tabla 2: Datos sobre el stack de cada tarea.

	Heap	Heap libre	Heap utilizado
Sistema operativo	1250	112	1138

Tabla 3: Datos del Heap del sistema operativo

En la Figura 3 se observan los datos que fueron brindados en las tablas. Los mismos se obtuvieron con las funciones $osThreadGetStackSpace()$ y $xPortGetFreeHeapSize()$ durante la ejecución del programa.

Por otra parte, el entorno de desarrollo provee una estimación del Heap utilizado y cuántos bytes quedan disponibles. Como se observa en la Figura 4, la estimación es que se ocupan 4486 bytes mientras que al momento de ejecución se determinó que se ocupan 4552 bytes (resultado obtenido de multiplicar 1138 palabras por 4). Por lo tanto, se determina que la estimación realizada no es precisa.

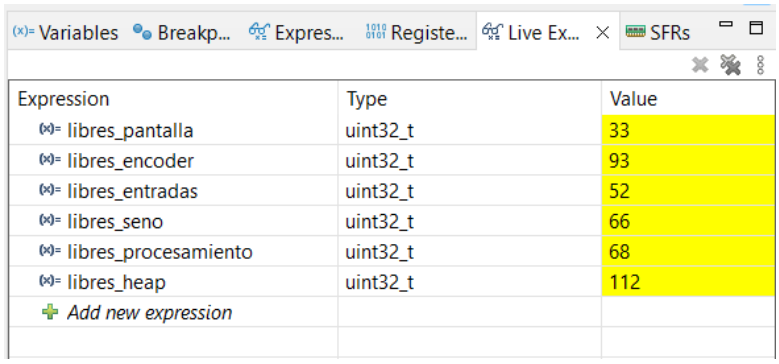

Figura 3: Datos del heap y stack de cada tarea obtenidos durante la ejecución del programa

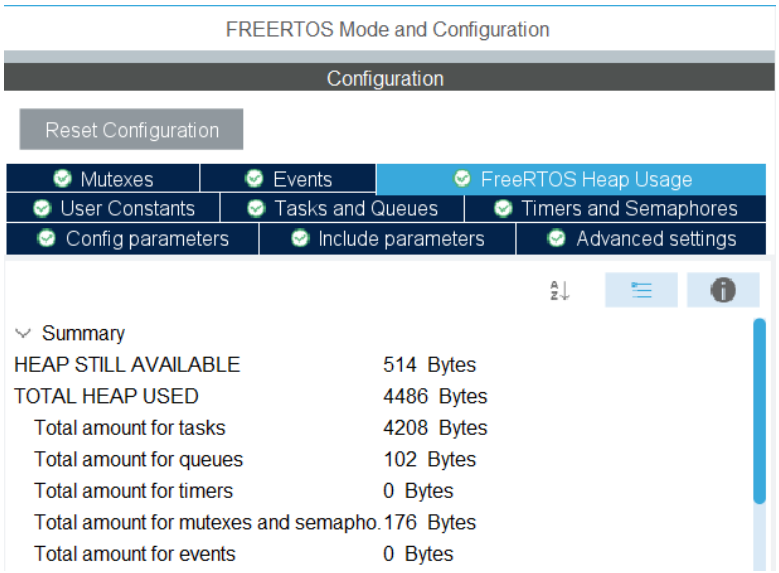

Figura 4: Estimación sobre los valores del Heap que brinda el entorno de desarrollo

Además, es posible determinar la cantidad de memoria que ocupa el sistema. Al analizar los resultados mostrados en la Figura 5, se observa que el uso de memoria alcanza el 99,8 % de la capacidad total disponible. Este elevado uso de memoria es una consecuencia directa de la implementación del analizador de espectro, que requiere múltiples tareas para su correcto funcionamiento.

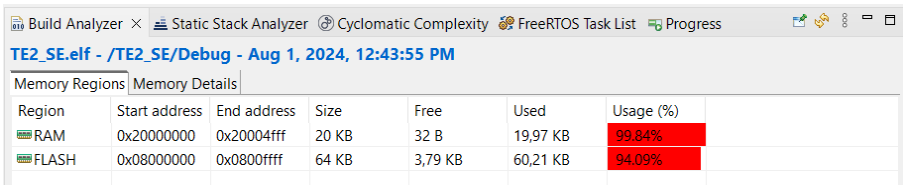

Figura 5: Espacio ocupado en memoria

6. Análisis en tiempo de ejecución

Se procede a calcular el factor de utilización de cada tarea y del sistema completo, para eso, se mide el tiempo de ejecución de cada tarea en el peor caso y cada cuanto se repiten. Dicho factor se calcula de la siguiente manera:

$$U = \frac{e_1}{p_1} + \frac{e_2}{p_2} + \frac{e_3}{p_3} + \frac{e_4}{p_4} + \frac{e_5}{p_5} \tag{1}$$

Donde el subíndice 1 corresponde a la Tarea Pantalla, 2 a la Tarea Entradas, 3 a la Tarea Encoder, 4 a la Tarea Seno y 5 a la Tarea Procesamiento. Reemplazando con los valores medidos, se tiene:

$$U = \frac{88{,}5ms}{427{,}16ms} + \frac{20\mu s}{5ms} + \frac{20\mu s}{5ms} + \frac{72{,}77ms}{217ms} + \frac{31,45ms}{431{,}2ms} = 0{,}6234 \tag{2}$$

Se obtuvo un buen resultado en el factor de utilización de acuerdo al algoritmo Rate Monotonic. Sin embargo, no se consideró en el cálculo la condición en la que se envían los datos de la FFT por UART, ya que no se conoce de forma precisa la frecuencia con la que esta acción se repite, aunque se estima que ocurre cada pocos segundos. En este caso, se consideró un período de 5 segundos para la tarea que realiza la transmisión hacia la computadora. Realizando nuevamente el cálculo, teniendo en cuenta el período estimado y la duración del envío de datos, el nuevo factor de utilización queda:

$$U = \frac{88{,}5ms}{427{,}16ms} + \frac{20\mu s}{5ms} + \frac{662,55ms}{5s} + \frac{72{,}77ms}{217ms} + \frac{31,45ms}{431{,}2ms} = 0{,}755 \tag{3}$$

A continuación se adjuntan las siguientes imágenes obtenidas en la medición de los tiempos, los cuales fueron necesarios para calcular el factor de utilización. Para la correcta interpretación de los datos, se debe aclarar que el canal 0 corresponde con el muestreo de la señal, y cada canal corresponde con la tarea cuyo subíndice ya fue utilizado en la ecuación 1.

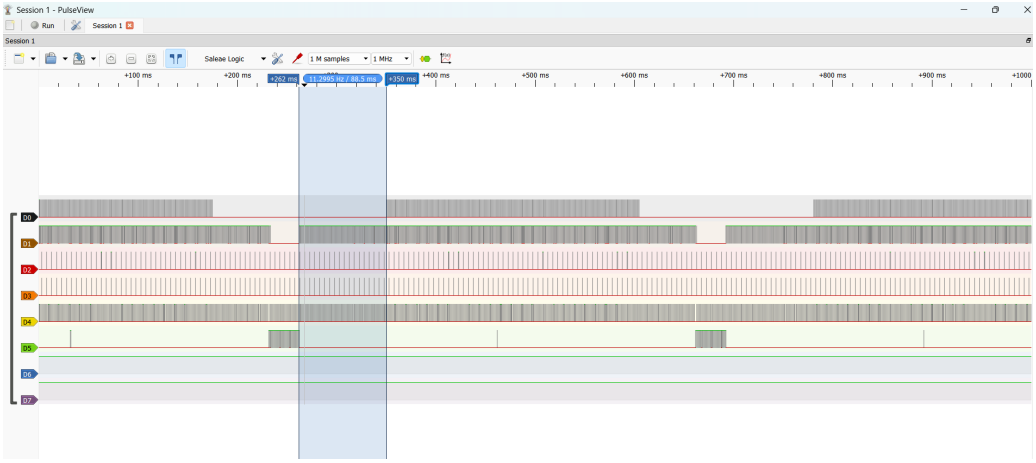

Figura 6: Tiempo de ejecución que le toma a la Tarea Pantalla imprimir la FFT, canal 1

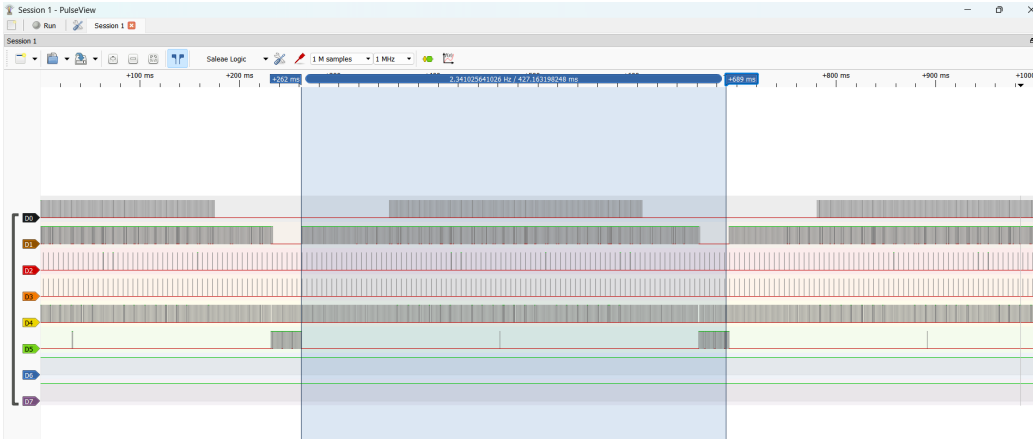

Figura 7: Período de la Tarea Pantalla, canal 1

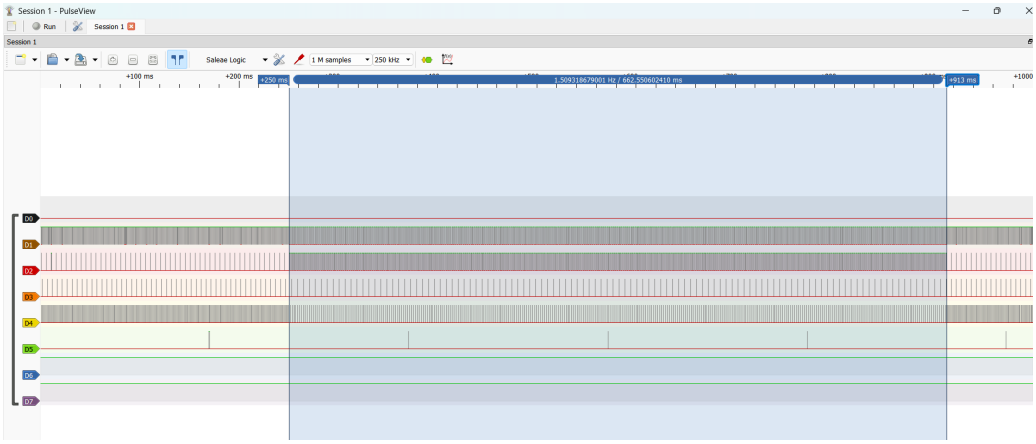

Figura 8: Tiempo de ejecución de la Tarea Entradas cuando se transmiten los datos por UART, canal 2

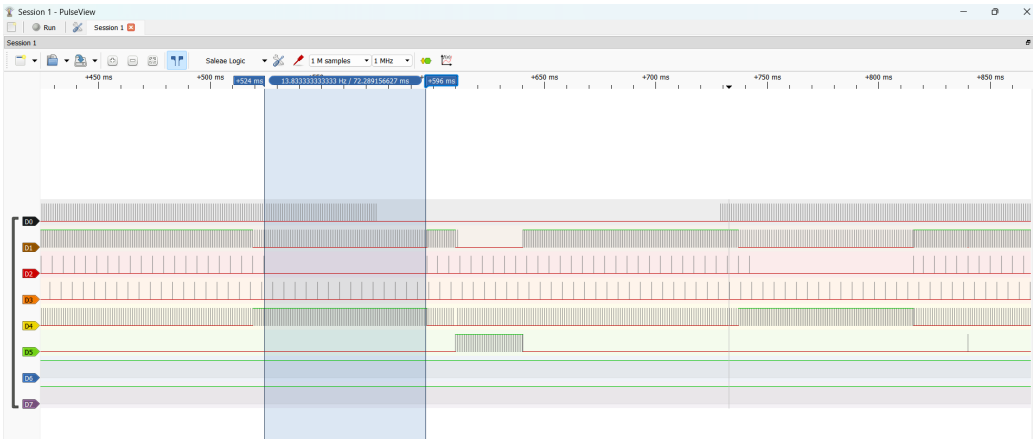

Figura 9: Tiempo de ejecución de la Tarea Seno, canal 4

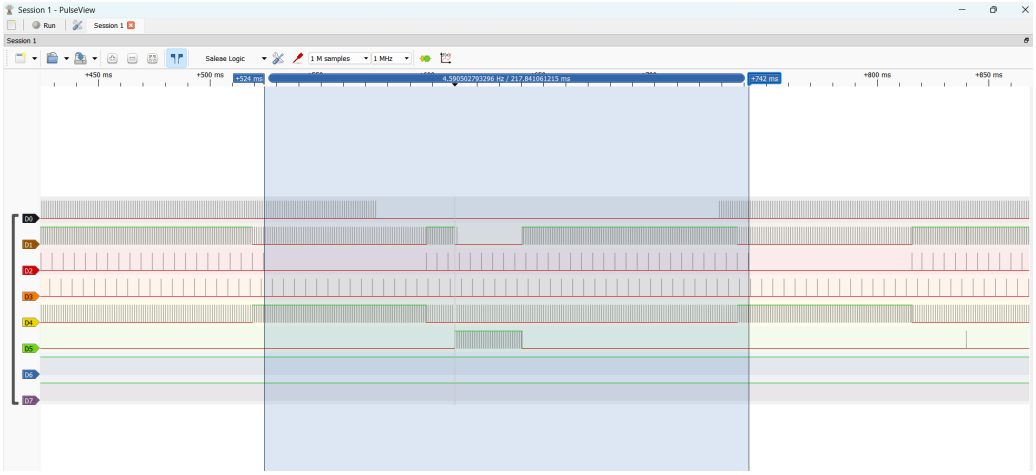

Figura 10: Período de la Tarea Seno, canal 4

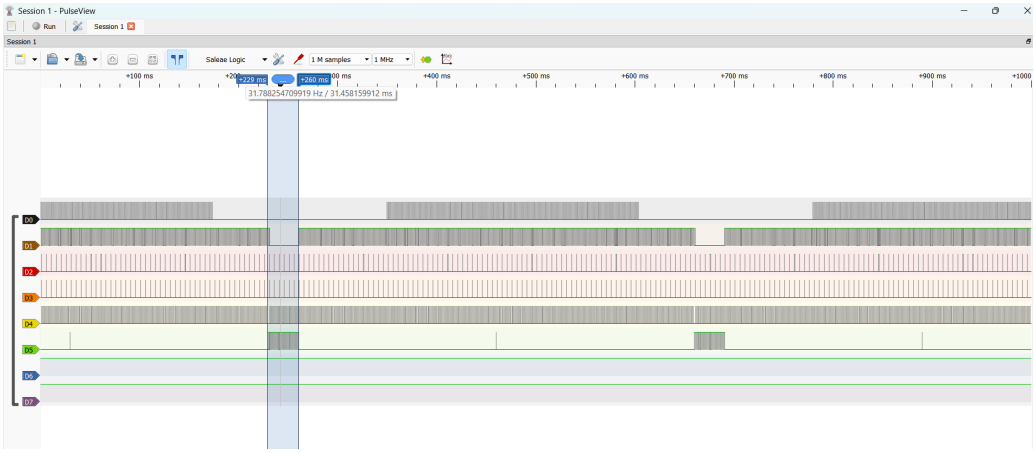

Figura 11: Tiempo de ejecución de la Tarea Procesamiento, canal 5

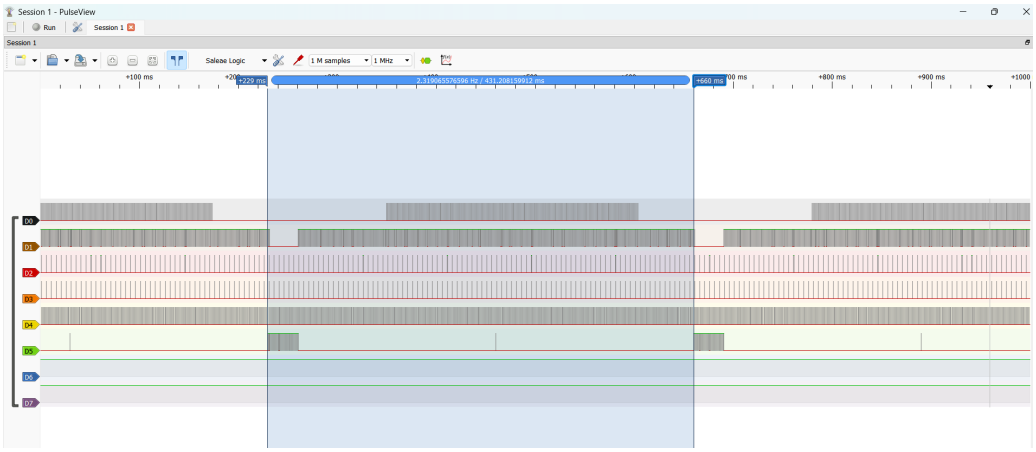

Figura 12: Período de la Tarea Procesamiento, canal 5

7. Conclusión

En este informe, desarrolló e implementó un analizador de espectro utilizando el sistema operativo en tiempo real FreeRTOS. Se logró verificar que este sistema es capaz de capturar señales a una tasa de muestreo de 1 kHz y realizar una Transformada Rápida de Fourier (FFT) de 256 puntos, presentando los resultados en la pantalla. La implementación de diversas tareas e interrupciones bajo FreeRTOS, con mecanismos de sincronización como colas, mutex y semáforos, permitieron el correcto funcionamiento del sistema.

A. Anexo: Código de todas las tareas

Listing 1: Variables globales

```
typedef enum {
    ENC_IDLE,
    ENC_CW01, ENC_CW00, ENC_CW10,
    ENC_CCW10, ENC_CCW00, ENC_CCW01,
} enc_state_t;

typedef enum {ENC_00, ENC_01, ENC_10, ENC_11} enc_input_t;

enc_state_t enc_state = ENC_IDLE;
enc_input_t input;

typedef enum {btn_soltando, btn_reposo} estado_btn;
estado_btn estado_btn1 = btn_reposo;
estado_btn estado_btn2 = btn_reposo;

int NSEN = 100;
int VMAX = 255;

uint8_t onda[1000] = {0};
int idx = 0;

int flag_amp = 0;
int flag_frec = 0;
int flag_tf = 0;
int flag_datos = 0;
int encoder = 0;
int cont = 0;
int frec_seno = 100;
int config = 1;
int cursor = 0;
float amp = 3.3;
int cont_inc = 0;
```

```
int cont_dec = 0;
struct cmpx tf[256] = {0};
float result[256] = {0};
```

Listing 2: Callback de conversión completa del ADC

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef *hadc) {
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, SET);
    float val;
    val = HAL_ADC_GetValue(&hadc1)*3.3/3969;
    (tf[cont]).real = (float) val;
    cont++;
    if (cont == 255){
        HAL_TIM_Base_Stop_IT(&htim1);
    }
    HAL_GPIO_WritePin(GPIOB, GPIO_PIN_3, RESET);
}
```

Listing 3: Tarea Pantalla

```
void entryTareaPantalla(void *argument)
{
  /* USER CODE BEGIN 5 */
    vTaskSetApplicationTaskTag( NULL, (void*) TAG_TASK_PANTALLA);
    SSD1306_Init();
    SSD1306_Clear();
    SSD1306_UpdateScreen();
    uint8_t flag_pantalla = 2;
    int cuentas = 0;
    char amplitud[5];
    char frecuencia[5];
    uint8_t x = 0;
    float y = 0;
    float y_ant = 0;
    int frec_aux = 0;
    float amp_aux = 0;
  /* Infinite loop */
  for(;;)
  {
      cuentas = osMessageQueueGetCount(myQueue01Handle);
      if (cuentas >= 1){
          osMessageQueueGet(myQueue01Handle, &flag_pantalla, 0, 0);
          SSD1306_Clear();
          SSD1306_UpdateScreen();
      }

      if (flag_pantalla == 1){
```

```
28              osMutexAcquire(mutexDatosHandle, 1000);
29              if (flag_tf == 1){
30                  flag_tf = 0;
31                  SSD1306_Clear();
32                  SSD1306_UpdateScreen();
33                  SSD1306_DrawLine(cursor, 0, cursor, 64, 1);
34                  frec_aux = cursor * ((float)500/128);
35                  amp_aux = result[cursor]*100;
36                  itoa(frec_aux, frecuencia, 10);
37                  itoa((int)amp_aux, amplitud, 10);
38                  SSD1306_GotoXY(90, 1);
39                  SSD1306_Puts(frecuencia, &Font_7x10, 1);
40                  SSD1306_Puts("Hz", &Font_7x10, 1);
41                  SSD1306_GotoXY(90, 15);
42                  if (amp_aux < 100){
43                      SSD1306_Puts("0.", &Font_7x10, 1);
44                      SSD1306_Puts(amplitud, &Font_7x10, 1);
45                      SSD1306_Puts("V", &Font_7x10, 1);
46                  }
47                  else {
48                      amp_aux = ((int)amp_aux) % 100;
49                      itoa((int)amp_aux, amplitud, 10);
50                      SSD1306_Puts("1.", &Font_7x10, 1);
51                      SSD1306_Puts(amplitud, &Font_7x10, 1);
52                      SSD1306_Puts("V", &Font_7x10, 1);
53                  }
54                  x = 1;
55                  y_ant = (uint16_t) 64 - (result[0]*64/1.65);
56                  for (int i = 1; i < 128; i++){
57                      y = 64 - (result[i]*64/1.65);
58                      SSD1306_DrawLine(x-1, (int)y_ant, x, (int)y, 1);
59                      y_ant = y;
60                      x++;
61                      result[i] = 0;
62                  }
63                  SSD1306_UpdateScreen();
64                  Reset_tf();
65                  HAL_TIM_Base_Start_IT(&htim1);
66              }
67              osMutexRelease(mutexDatosHandle);
68          }
69
70          if (flag_pantalla == 2) {
71              SSD1306_GotoXY(1, 1);
72              SSD1306_Puts("Cursor", &Font_7x10, 0);
```

```
73              SSD1306_GotoXY(1, 15);
74              SSD1306_Puts("Frecuencia", &Font_7x10, 1);
75              SSD1306_GotoXY(1, 30);
76              SSD1306_Puts("Amplitud", &Font_7x10, 1);
77              SSD1306_GotoXY(1, 45);
78              SSD1306_Puts("Impresion por UART", &Font_7x10, 1);
79              SSD1306_UpdateScreen();
80          }
81
82          if (flag_pantalla == 3) {
83              SSD1306_GotoXY(1, 1);
84              SSD1306_Puts("Cursor", &Font_7x10, 1);
85              SSD1306_GotoXY(1, 15);
86              SSD1306_Puts("Frecuencia", &Font_7x10, 0);
87              SSD1306_GotoXY(1, 30);
88              SSD1306_Puts("Amplitud", &Font_7x10, 1);
89              SSD1306_GotoXY(1, 45);
90              SSD1306_Puts("Impresion por UART", &Font_7x10, 1);
91              SSD1306_UpdateScreen();
92          }
93
94          if (flag_pantalla == 4) {
95              SSD1306_GotoXY(1, 1);
96              SSD1306_Puts("Cursor", &Font_7x10, 1);
97              SSD1306_GotoXY(1, 15);
98              SSD1306_Puts("Frecuencia", &Font_7x10, 1);
99              SSD1306_GotoXY(1, 30);
100             SSD1306_Puts("Amplitud", &Font_7x10, 0);
101             SSD1306_GotoXY(1, 45);
102             SSD1306_Puts("Impresion por UART", &Font_7x10, 1);
103             SSD1306_UpdateScreen();
104         }
105
106         if (flag_pantalla == 5) {
107             SSD1306_GotoXY(1, 1);
108             SSD1306_Puts("Cursor", &Font_7x10, 1);
109             SSD1306_GotoXY(1, 15);
110             SSD1306_Puts("Frecuencia", &Font_7x10, 1);
111             SSD1306_GotoXY(1, 30);
112             SSD1306_Puts("Amplitud", &Font_7x10, 1);
113             SSD1306_GotoXY(1, 45);
114             SSD1306_Puts("Impresion por UART", &Font_7x10, 0);
115             SSD1306_UpdateScreen();
116         }
117
```

```
118         if (flag_pantalla == 6) {
119             int aux = amp * 10;
120             aux = aux % 10;
121             sprintf(amplitud, "%d.%dV", (int)amp, aux);
122             itoa(frec_seno, frecuencia, 10);
123             SSD1306_GotoXY(1, 1);
124             SSD1306_Puts("Informacion seno", &Font_7x10, 0);
125             SSD1306_GotoXY(1, 12);
126             SSD1306_Puts("Frecuencia: ", &Font_7x10, 1);
127             SSD1306_Puts(frecuencia, &Font_7x10, 1);
128             SSD1306_Puts("Hz", &Font_7x10, 1);
129             SSD1306_GotoXY(1, 24);
130             SSD1306_Puts("Amplitud: ", &Font_7x10, 1);
131             SSD1306_Puts(amplitud, &Font_7x10, 1);
132             SSD1306_GotoXY(1, 36);
133             if (config == 1){
134                 SSD1306_Puts("Cursor", &Font_7x10, 1);
135             }
136             if (config == 2){
137                 SSD1306_Puts("Modificar frec", &Font_7x10, 1);
138             }
139             if (config == 3){
140                 SSD1306_Puts("Modificar Amp", &Font_7x10, 1);
141             }
142             SSD1306_GotoXY(1, 48);
143             SSD1306_Puts("Cursor:", &Font_7x10, 1);
144             frec_aux = cursor * ((float)500/128);
145             amp_aux = result[cursor]*100;
146             itoa(frec_aux, frecuencia, 10);
147             itoa((int)amp_aux, amplitud, 10);
148             SSD1306_Puts(frecuencia, &Font_7x10, 1);
149             SSD1306_Puts("Hz ", &Font_7x10, 1);
150             if (amp_aux < 100){
151                 SSD1306_Puts("0.", &Font_7x10, 1);
152                 SSD1306_Puts(amplitud, &Font_7x10, 1);
153                 SSD1306_Puts("V", &Font_7x10, 1);
154             }
155             else {
156                 amp_aux = ((int)amp_aux) % 100;
157                 itoa((int)amp_aux, amplitud, 10);
158                 SSD1306_Puts("1.", &Font_7x10, 1);
159                 SSD1306_Puts(amplitud, &Font_7x10, 1);
160                 SSD1306_Puts("V", &Font_7x10, 1);
161             }
162             SSD1306_UpdateScreen();
```

```
163         }
164
165         osDelay(1);
166     }
167     /* USER CODE END 5 */
168 }
```

Listing 4: Tarea Entradas

```
1  void entryTareaEntradas(void *argument)
2  {
3    /* USER CODE BEGIN entryTareaEntradas */
4      vTaskSetApplicationTaskTag(NULL, (void*) TAG_TASK_ENTRADAS);
5      typedef enum {e1, e2, e3, e4, e5, e6} state_t;
6      state_t estado = e2;  // Estado inicial
7      int B1 = 0;  // Estado del boton 1
8      int B2 = 0;  // Estado del boton 2
9      int flag = 0;  // Bandera para la pantalla
10     float tf_real_dec = 0;
11     float tf_real_aux = 0;
12     float tf_imag_aux = 0, tf_imag_dec = 0;
13
14   /* Infinite loop */
15   for(;;)
16   {
17       B1 = pulsadores(GPIOA, GPIO_PIN_3, &estado_btn1);
18       B2 = pulsadores(GPIOA, GPIO_PIN_4, &estado_btn2);
19       switch(estado){
20       case e1:
21           osMutexAcquire(mutexPantallaHandle, 1000);
22           if (B1 == 1){
23               flag_datos = 1;
24               osSemaphoreAcquire(SemTFHandle, 1000);
25               estado = e2;
26               flag = 2;
27               osMessageQueuePut(myQueue01Handle, &flag, 0, 0);
28               HAL_TIM_Base_Stop_IT(&htim1);
29               cont = 0;
30           }
31           if (config == 1){
32               if (encoder == 1 && cursor < 128){
33                   encoder = 0;
34                   cursor = cursor +1;
35               }
36               if (encoder == 2 && cursor > 0){
37                   encoder = 0;
38                   cursor = cursor -1;
```

```
39                  }
40              }
41              if (config == 2){
42                  if (cont_inc > 0){
43                      encoder = 0;
44                      flag_frec = 1;
45                      frec_seno = frec_seno + cont_inc;
46                      cont_inc = 0;
47                      if (frec_seno > 500){
48                          frec_seno = 500;
49                      }
50                  }
51                  if (cont_dec > 0){
52                      encoder = 0;
53                      flag_frec = 1;
54                      frec_seno = frec_seno - cont_dec;
55                      cont_dec = 0;
56                      if (frec_seno < 0){
57                          frec_seno = 0;
58                      }
59                  }
60              }
61              if (config == 3){
62                  if (cont_inc > 0){
63                      encoder = 0;
64                      flag_amp = 1;
65                      amp = amp + 0.1*cont_inc;
66                      cont_inc = 0;
67                      if (amp > 3.3){
68                          amp = 3.3;
69                      }
70                  }
71                  if (cont_dec > 0){
72                      encoder = 0;
73                      flag_amp = 1;
74                      amp = amp - 0.1*cont_dec;
75                      cont_dec = 0;
76                      if (amp < 0){
77                          amp = 0;
78                      }
79                  }
80              }
81              osMutexRelease(mutexPantallaHandle);
82
83              break;
```

```
84
85          case e2:
86              if (encoder == 1){
87                  encoder = 0;
88                  estado = e3;
89                  flag = 3;
90                  osMessageQueuePut(myQueue01Handle, &flag, 0, 0);
91              }
92              if (encoder == 2){
93                  encoder = 0;
94                  estado = e5;
95                  flag = 5;
96                  osMessageQueuePut(myQueue01Handle, &flag, 0, 0);
97              }
98              if (B2 == 1){
99                  config = 1;
100             }
101             if (B1 == 1){
102                 estado = e6;
103                 flag = 6;
104                 osMessageQueuePut(myQueue01Handle, &flag, 0, 0);
105             }
106             break;
107
108         case e3:
109             if (encoder == 1){
110                 encoder = 0;
111                 estado = e4;
112                 flag = 4;
113                 osMessageQueuePut(myQueue01Handle, &flag, 0, 0);
114             }
115             if (B2 == 1){
116                 config = 2;
117             }
118             if (encoder == 2){
119                 encoder = 0;
120                 estado = e2;
121                 flag = 2;
122                 osMessageQueuePut(myQueue01Handle, &flag, 0, 0);
123             }
124             if (B1 == 1){
125                 estado = e6;
126                 flag = 6;
127                 osMessageQueuePut(myQueue01Handle, &flag, 0, 0);
128             }
```

```
129             break;
130
131         case e4:
132             if (B1 == 1){
133                 estado = e6;
134                 flag = 6;
135                 osMessageQueuePut(myQueue01Handle, &flag, 0, 0);
136             }
137             if (B2 == 1){
138                 config = 3;
139             }
140             if (encoder == 1){
141                 encoder = 0;
142                 estado = e5;
143                 flag = 5;
144                 osMessageQueuePut(myQueue01Handle, &flag, 0, 0);
145             }
146             if (encoder == 2){
147                 encoder = 0;
148                 estado = e3;
149                 flag = 3;
150                 osMessageQueuePut(myQueue01Handle, &flag, 0, 0);
151             }
152             break;
153
154         case e5:
155             if (B2 == 1){
156                 char str_real[20];
157                 char str_imag[20];
158                 char mensaje[] = "Envio de datos\n";
159                 HAL_UART_Transmit(&huart1, (uint8_t *)mensaje, strlen(mensa
160                 for (int i=0; i<256; i++){
161                     tf_real_aux = tf[i].real*1.65/425;
162                     tf_imag_aux = tf[i].imag*1.65/425;
163                     tf_real_dec = (tf_real_aux - (int)tf_real_aux)*100000;
164                     tf_imag_dec = (tf_imag_aux - (int)tf_imag_aux)*100000;
165
166                     if (tf[i].real >= 0){
167                         sprintf(str_real, "%d.%05d", (int)tf_real_aux, (int
168                     }
169                     if (tf[i].real < 0){
170                         sprintf(str_real, "-%d.%05d", (int)-tf_real_aux, (i
171                     }
172                     if (tf[i].imag >= 0){
173                         sprintf(str_imag, "+%d.%05d*i, ", (int)tf_imag_aux,
```

```
174                     }
175                     if (tf[i].imag < 0){
176                         sprintf(str_imag, "-%d.%05d*i, ", (int)-tf_imag_aux
177                     }
178 
179                     HAL_UART_Transmit(&huart1, (uint8_t *)str_real, strlen(
180                     HAL_UART_Transmit(&huart1, (uint8_t *)str_imag, strlen(
181                 }
182                 HAL_UART_Transmit(&huart1, (uint8_t *)"\n", 1, HAL_MAX_DELA
183             }
184             if (B1 == 1){
185                 estado = e6;
186                 flag = 6;
187                 osMessageQueuePut(myQueue01Handle, &flag, 0, 0);
188             }
189             if (encoder == 1){
190                 encoder = 0;
191                 estado = e2;
192                 flag = 2;
193                 osMessageQueuePut(myQueue01Handle, &flag, 0, 0);
194             }
195             if (encoder == 2){
196                 encoder = 0;
197                 estado = e4;
198                 flag = 4;
199                 osMessageQueuePut(myQueue01Handle, &flag, 0, 0);
200             }
201             break;
202 
203         case e6:
204             if (B1 == 1){
205                 estado = e1;
206                 flag = 1;
207                 osMessageQueuePut(myQueue01Handle, &flag, 0, 0);
208                 HAL_TIM_Base_Start_IT(&htim1);
209             }
210             break;
211         }
212       osDelay(5);
213     }
214     /* USER CODE END entryTareaEntradas */
215 }
```

Listing 5: Tarea Encoder

```
1 void entryTareaEncoder(void *argument)
2 {
```

```
3   /* USER CODE BEGIN entryTareaEncoder */
4     vTaskSetApplicationTaskTag(NULL, (void*) TAG_TASK_ENCODER);
5     int A, B;
6
7   /* Infinite loop */
8   for(;;)
9   {
10      //osMutexAcquire(mutexPantallaHandle, 1000);
11      A = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_1);
12      B = HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_2);
13
14      if(A == 0 || B == 0){
15        if (A == 1 && B == 1){
16            input = ENC_11;
17        }
18        if (A == 1 && B == 0){
19            input = ENC_10;
20        }
21        if (A == 0 && B == 0){
22            input = ENC_00;
23        }
24        if (A == 0 && B == 1){
25            input = ENC_01;
26        }
27      }
28      else {
29          input = ENC_11;
30      }
31
32    switch (enc_state){
33    case ENC_IDLE:
34        switch(input){
35        case ENC_01: enc_state = ENC_CW01;  break;
36        case ENC_10: enc_state = ENC_CCW10; break;
37        default:     enc_state = ENC_IDLE;  break;
38        } break;
39
40    case ENC_CW01:
41        switch(input){
42        case ENC_11: enc_state = ENC_IDLE; break;
43        case ENC_00: enc_state = ENC_CW00; break;
44        default:     enc_state = ENC_CW01; break;
45        } break;
46
47    case ENC_CW00:
```

```
48          switch(input){
49          case ENC_11: enc_state = ENC_IDLE; break;
50          case ENC_01: enc_state = ENC_CW01; break;
51          case ENC_10: enc_state = ENC_CW10; break;
52          default:     enc_state = ENC_CW00; break;
53          } break;
54
55      case ENC_CW10:
56          switch(input){
57          case ENC_11:
58              enc_state = ENC_IDLE;
59              encoder = 1;
60              if (config == 2 || config == 3) {cont_inc++;}
61              break;
62          case ENC_00: enc_state = ENC_CW00; break;
63          default:     enc_state = ENC_CW10; break;
64          } break;
65
66      case ENC_CCW10:
67          switch(input){
68          case ENC_11: enc_state = ENC_IDLE;  break;
69          case ENC_00: enc_state = ENC_CCW00; break;
70          default:     enc_state = ENC_CCW10; break;
71          } break;
72
73      case ENC_CCW00:
74          switch(input){
75          case ENC_11: enc_state = ENC_IDLE;  break;
76          case ENC_01: enc_state = ENC_CCW01; break;
77          case ENC_10: enc_state = ENC_CCW10; break;
78          default:     enc_state = ENC_CCW00; break;
79          } break;
80
81      case ENC_CCW01:
82          switch(input){
83          case ENC_00: enc_state = ENC_CCW00; break;
84          case ENC_11:
85              encoder = 2;
86              enc_state = ENC_IDLE;
87              if (config == 2 || config == 3) {cont_dec++;}
88              break;
89          default:     enc_state = ENC_CCW01; break;
90          } break;
91      }
92
```

```
    //osMutexRelease(mutexPantallaHandle);
    osDelay(5);
  }
  /* USER CODE END entryTareaEncoder */
}
```

Listing 6: Tarea Seno

```
void entryTareaSeno(void *argument)
{
  /* USER CODE BEGIN entryTareaSeno */
    vTaskSetApplicationTaskTag( NULL, (void*) TAG_TASK_SENO);
  /* Infinite loop */
  for(;;)
  {
      if (flag_amp == 1){
          osMutexAcquire(mutexPantallaHandle, 1000);
          flag_amp = 0;
          VMAX = (int) (amp * 255 / 3.3);
          HAL_TIM_PWM_Stop(&htim2, TIM_CHANNEL_1);
          for (idx = 0; idx < 1000 ; idx++){
              onda[idx] = (uint8_t)((float)VMAX/2.0*(sin(2.0*M_PI*idx/(fl
          }
          HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
          HAL_DMA_Start(&hdma_tim3_ch4_up, (uint32_t) onda, (uint32_t)&(T
          __HAL_TIM_ENABLE_DMA(&htim3, TIM_DMA_UPDATE);
          osMutexRelease(mutexPantallaHandle);

      }
      if (flag_frec == 1){
          osMutexAcquire(mutexPantallaHandle, 1000);
          flag_frec = 0;
          NSEN = 10000/frec_seno;
          HAL_TIM_PWM_Stop(&htim2, TIM_CHANNEL_1);
          for (idx = 0; idx < 1000 ; idx++){
              onda[idx] = (uint8_t)((float)VMAX/2.0*(sin(2.0*M_PI*idx/(fl
          }
          HAL_TIM_PWM_Start(&htim2, TIM_CHANNEL_1);
          HAL_DMA_Start(&hdma_tim3_ch4_up, (uint32_t) onda, (uint32_t)&(T
          __HAL_TIM_ENABLE_DMA(&htim3, TIM_DMA_UPDATE);
          osMutexRelease(mutexPantallaHandle);

      }
    osDelay(1);
  }
  /* USER CODE END entryTareaSeno */
}
```

Listing 7: Tarea Procesamiento

```
void entryTareaProcesamiento(void *argument)
{
  /* USER CODE BEGIN entryTareaProcesamiento */
    vTaskSetApplicationTaskTag( NULL, (void*) TAG_TASK_PROCESAMIENTO);
  /* Infinite loop */
  for(;;)
  {
      osMutexAcquire(mutexDatosHandle, 1000);
        if (cont >= 255) {
            cont = 0;
            FFT(&tf[0], 256);
            for(int i=0; i<256; i++){
                result[i] = sqrt(tf[i].real*tf[i].real+tf[i].imag*tf[i].i
            }
             flag_tf = 1;
             if (flag_datos == 1){
                 flag_datos = 0;
                 osSemaphoreRelease(SemTFHandle);
             }
        }

        osMutexRelease(mutexDatosHandle);
    osDelay(200);
  }
  /* USER CODE END entryTareaProcesamiento */
}
```