

Trabajo Especial 2

Sistemas embebidos

Juan Felipe Perdomo Rojas

2 de agosto de 2024



Índice

1. Resumen:	3
2. Desarrollo:	3
2.1. Adquisición de datos	3
2.2. Generación de señal	3
2.3. Cálculo de la FFT y Renderizacion en tiempo Real	4
2.4. Interfaz de Usuario (UI)	4
3. Hooks y verificación de implementación:	6
4. Análisis del Stack y RAM:	10
5. ANEXOS DE CODIGO:	11
5.1. TAREA UI:	11
5.2. Tarea Generación de señal	14
5.3. Tarea monitoreo	14

1. Resumen:

El presente informe tiene como objetivo mostrar el procedimiento detallado de la implementación de un analizador de espectro utilizando la plataforma Blue Pill y el sistema operativo FreeRTOS, desarrollado en el entorno CubeIDE32. El proyecto consiste en analizar un rango específico de frecuencias del espectro, calculando la FFT y visualizándola en una pantalla OLED.

Además, se implementó una interfaz de usuario mediante un encoder que permite diversas opciones de configuración. Para la generación de señales, se construyó un DAC casero utilizando el timer de la Blue Pill para generar una señal PWM de alta frecuencia, que luego se filtra mediante un filtro pasabajos para obtener una onda senoidal. Esta señal es adquirida a una frecuencia de 1 kHz utilizando el ADC de la Blue Pill.

El informe también incluye imágenes que aportan fidelidad a los resultados obtenidos, asegurando una representación precisa de las características de las señales analizadas.

2. Desarrollo:

Para el desarrollo de este trabajo, fue fundamental descomponer el sistema en bloques de problemas específicos, resolviendo cada uno por separado. A continuación, se detallan los procedimientos para cada una de las tareas:

2.1. Adquisición de datos

El primer paso fue asegurar un muestreo uniforme a 1 kHz mediante el ADC. Para lograr esto, se configuró el Timer3 para que genere una interrupción cada milisegundo. Esta configuración se realizó mediante el IOC, estableciendo un prescaler de 71 y un período de contador de 999.

Una vez generada la interrupción, se utilizó como disparador para que el ADC generara una interrupción al adquirir una muestra, almacenándolas en un arreglo de 256 elementos. Al completarse el arreglo, se desactiva la interrupción del ADC, lo que detiene temporalmente el muestreo. Posteriormente, tras procesar los datos y vaciar el arreglo, se vuelve a habilitar la interrupción del ADC para continuar con el muestreo, manteniendo así un ciclo de adquisición y procesamiento de datos.

2.2. Generación de señal

Para la generación de la señal analógica final, se emplea un filtro pasabajos analógico compuesto por un capacitor y una resistencia. El objetivo es obtener una onda

senoidal a la salida, lo que requiere una señal de excitación con un rango de frecuencias significativamente mayor. Para ello, se utiliza una señal PWM generada por la Blue Pill a una frecuencia de 282 kHz.

Se configuraron los Timers 1 y 2 junto con el DMA para este propósito. El Timer 2 se estableció como el canal de salida de la PWM a 282 kHz, configurando el prescaler en 0 y el período de contador en 255, lo que determina el valor máximo de ARR que la PWM puede alcanzar. Este valor limita el rango del CCR, ya que no puede ser superior al ARR. Esto es crucial, ya que los valores de una senoide se cargan en un arreglo para modelarlos mediante la PWM, lo que afecta directamente la amplitud de la señal generada.

Por otro lado, el Timer 1 se configuró para operar a 1 kHz, generando una solicitud de DMA que permite la transferencia de los valores de la senoide a la memoria RAM sin consumir tiempo de procesamiento. Todas estas configuraciones se realizaron en el IOC.

2.3. Cálculo de la FFT y Renderización en tiempo Real

Una vez obtenidas las muestras, el siguiente paso fue procesarlas adecuadamente y mostrar el resultado en una pantalla OLED en tiempo real.

Para mantener la pantalla responsiva, se estableció un objetivo de actualización de al menos cada 100 ms. Fue fundamental sincronizar la adquisición de muestras con la renderización en pantalla, ya que sin muestras no se podría graficar correctamente.

Se utilizaron semáforos para gestionar esta sincronización. El proceso funciona de la siguiente manera: si la tarea que maneja la interfaz desea graficar, solicita un semáforo, el cual solo está disponible una vez que las muestras han sido procesadas. Esto se implementó utilizando las funciones `:osSemaphoreAcquire(myBinarySem01Handle)` y `osSemaphoreRelease(myBinarySem01Handle)`.

El cálculo de la FFT se llevó a cabo utilizando una biblioteca proporcionada por la cátedra. La función de la FFT toma como entrada las muestras y un arreglo, donde los datos deben ser escalados correctamente. La parte real del arreglo de entrada se escala por un factor de $(3.3/4096)$ donde 3.3 V es el valor máximo y 4096 corresponde a la resolución de 12 bits del ADC. Además, el resultado de la FFT se divide por el número de puntos (256) y se escala por 2 para asegurar una visualización adecuada en la pantalla OLED de la interfaz de usuario.

2.4. Interfaz de Usuario (UI)

En esta tarea, se debía implementar la configuración de tres pantallas: una que muestra la FFT con un cursor, otra con opciones de configuración, y una tercera

al stack de las tareas, superando el valor habitual de 128 bytes. En este caso, aumentamos el tamaño del heap a 3500 bytes desde el IOC.

Finalmente, asignamos un esquema de prioridades utilizando el algoritmo de Rate Monotonic, que sugiere otorgar mayor prioridad a las tareas con menores períodos de ejecución, dado que nuestro factor de utilización no supera la cota de 0.77. Así, el esquema de prioridades quedó establecido de la siguiente manera, de mayor a menor:

- ✓ Tarea de monitoreo de entradas (cada 1 ms)
- ✓ Tarea de generación de señal (cada 10 ms)
- ✓ Tarea de interfaz de usuario (UI) (cada 100 ms)

3. Hooks y verificación de implementación:

Gracias a la herramienta de depuración proporcionada por el STM32, y con la activación de los hooks, podemos observar el tiempo de ejecución de cada una de nuestras tareas y estimar el factor de utilización. A continuación, se adjuntan imágenes que muestran la comprobación de los hooks y se especifica qué señal corresponde a cada tarea y su tiempo de ejecución:

Estimamos el factor de utilización aprovechando la tarea IDLE como $\frac{T_{TOT}-T_{IDLE}}{T_{TOT}} \approx 0,44$

Figura 2. Señal Violeta: Generación de señal: PWM

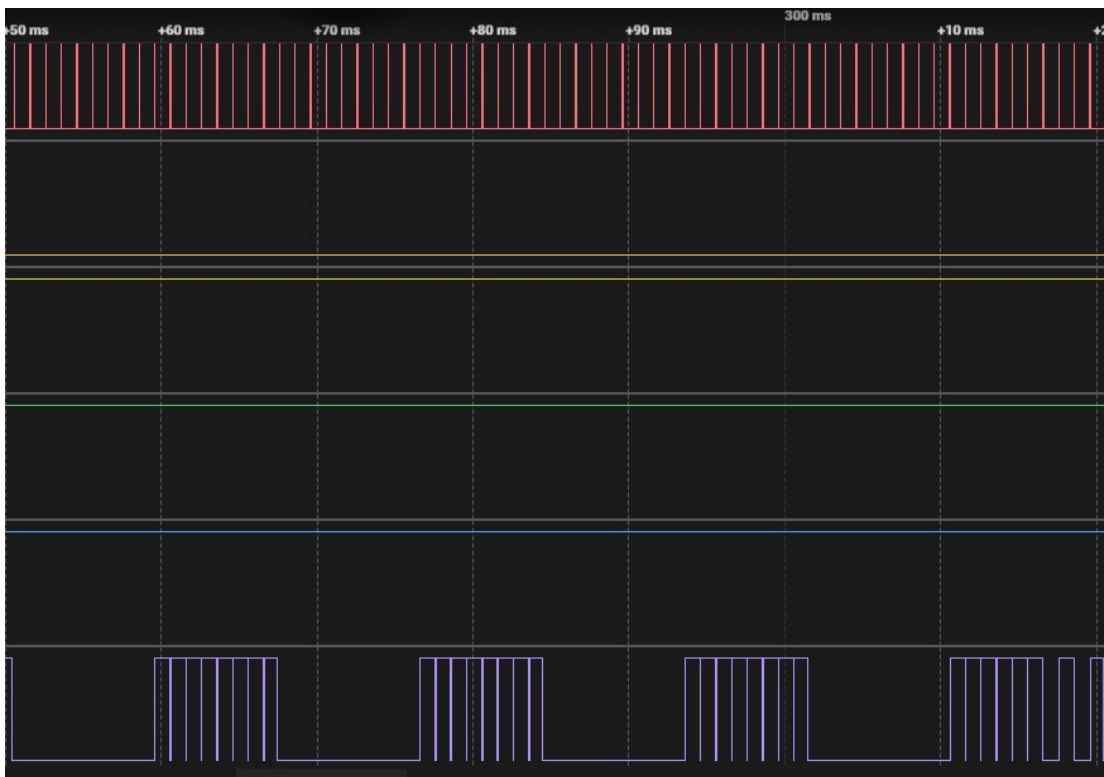


Figura 3. Señal Blanca: Interfaz de Usuario

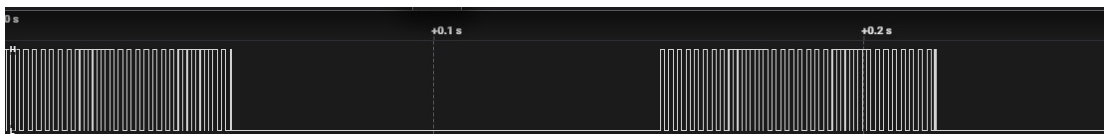


Figura 4. Señal marrón: Idle

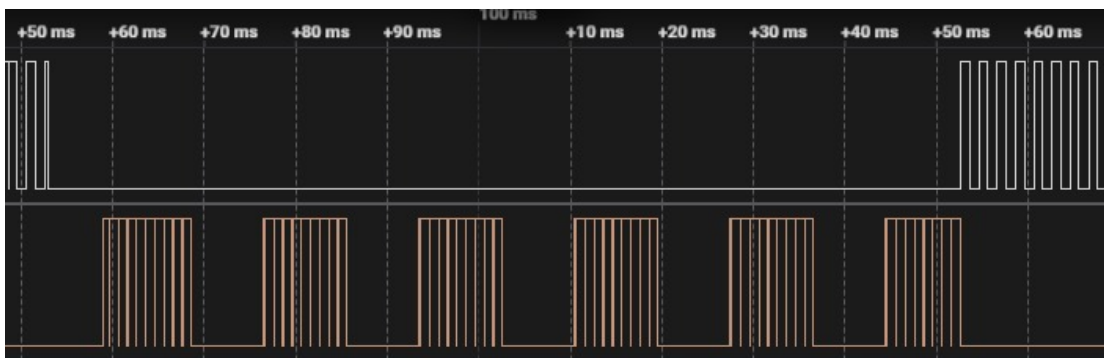


Figura 5. Señal Naranja: Muestreo interrumpido



Figura 6. Señal Naranja: Muestreo ininterrumpido



Figura 7. Señal Verde: Monitoreo de entradas






















4. Análisis del Stack y RAM:

Finalmente en esta última sección se chequea que no hayan problemas por overflow, en el caso del stack. y que la Ram no esté tan exigida.

Figura 8. Uso de la memoria

Memory Regions		Memory Details				
Region	Start address	End address	Size	Free	Used	Usage (%)
 RAM	0x20000000	0x20004fff	20 KB	2,16 KB	17,84 KB	89.22%
 FLASH	0x08000000	0x0800ffff	64 KB	6,04 KB	57,96 KB	90.57%

Figura 9. Análisis estimativo del stack

List	Call graph				Search...
Function	Depth	Max cost	Local cost	Type	
 CAN1_RX1_IRQHandler	?	?	0		
>  Interfaz_de_usuario	9	596	176	STATIC	
>  Reset_Handler	12	444	0		
>  prvTimerTask	8	352	24	STATIC	
>  ADC1_2_IRQHandler	7	240	8	STATIC	
>  Mi_pwm	8	176	40	STATIC	
>  prvIdleTask	6	168	16	STATIC_I...	
>  Manejar_Entradas	5	168	32	STATIC	
>  SysTick_Handler	2	72	16	STATIC_I...	
>  PendSV_Handler	3	56	0	STATIC_I...	
>  TIM3_IRQHandler	3	44	8	STATIC	
>  TIM4_IRQHandler	3	44	8	STATIC	
>  DMA1_Channel5_IRQHandler	1	32	8	STATIC	
 NMI_Handler	0	4	4	STATIC	
 DebugMon_Handler	0	4	4	STATIC	
 UsageFault_Handler	0	4	4	STATIC	
 BusFault_Handler	0	4	4	STATIC	
 HardFault_Handler	0	4	4	STATIC	
 MemManage_Handler	0	4	4	STATIC	

Tal y como se puede ver en [Figura 10](#) y [Figura 9](#) no tenemos problema con el stack asignado, pero estamos un poco al limite con la memoria RAM lo cual puede ser peligroso ya que puede sobrescribir variables. Sin embargo en nuestro código final esto no sucede y se renderiza todo correctamente.

Figura 10. Stack en tiempo de ejecución

(x)= FreeHeap	uint16_t	688
(x)= Freeproces	uint16_t	256
(x)= FreeDefault	uint16_t	384
(x)= FreeMonitoreo	uint16_t	424

5. ANEXOS DE CODIGO:

5.1. TAREA UI:

```

1 void StartTaskUI(void *argument)
2 {
3     /* USER CODE BEGIN StartTaskUI */
4     vTaskSetApplicationTaskTag(NULL, (void*) TAG_TASK_UI);
5     char frecuencia_display[4];
6     char amplitud_display[6];
7     char fft_buffer_display[50];
8     char amplitud_pantalla[6];
9     char frec_pantalla[6];
10    int integerPart_A = 0;
11    int decimalPart_A = 0;
12    int integerPart_F = 0;
13    int decimalPart_F = 0;
14    int integerPart;
15    int decimalPart;
16
17    /* Infinite loop */
18    for(;;)
19    {
20        SSD1306_Fill(0);
21        flag_enc = 0;
22        flag_uart = 0;
23        flag_frec = 0;
24        flag_amp = 0;
25        integerPart = (amplitud * 330 / 255) / 100;
26        decimalPart = (amplitud * 330 / 255) % 100;
27        itoa(frecuencia * 10, frecuencia_display, 10);
28        switch(Screen)
29        {
30            case 0:
31                if (osSemaphoreAcquire(Semaforo1Handle, 1000) == osOK) {
32                    for (int i = 0; i < BUFFER_SIZE; i++) {
33                        fft[i].real = adc_buffer[i] * 3.3 / 4096.0;
34                        fft[i].imag = 0;
35                    }
36                    FFT(fft, BUFFER_SIZE);
37                    for (int i = 0; i < 128; i++) {

```

```

38         if (i != 0 && i != 128) {
39             fft_R[i] = (sqrt(fft[i].real * fft[i].real + fft[i]
40                 ↪ ].imag * fft[i].imag) / 256) * 2;
41         } else {
42             fft_R[i] = (sqrt(fft[i].real * fft[i].real + fft[i]
43                 ↪ ].imag * fft[i].imag) / 256);
44         }
45         f[i] = 10000 * i / 255;
46     }
47     DibujarFFT(fft_R);
48     HAL_ADC_Start_IT(&hadc1);
49     osSemaphoreRelease(Semaforo1Handle);
50 }
51 //Muestra en pantalla la configuraci n del cursor
52 integerPart_A = abs(fft_R[encoder_position] * 1000) /
53     ↪ 1000;
54 decimalPart_A = abs(fft_R[encoder_position] * 1000) %
55     ↪ 1000;
56 snprintf(amplitud_pantalla, sizeof(amplitud_pantalla), "%
57     ↪ d.%03d", integerPart_A, decimalPart_A);
58 integerPart_F = f[encoder_position] / 10;
59 decimalPart_F = f[encoder_position] % 10;
60 snprintf(frec_pantalla, sizeof(frec_pantalla), "%03d.%01d
61     ↪ ", integerPart_F, decimalPart_F);
62 if (enc_graf == 1) {
63     SSD1306_DrawLine(encoder_position, 0, encoder_position,
64         ↪ 64, SSD1306_COLOR_WHITE);
65 }
66 SSD1306_GotoXY(70, 0);
67 SSD1306_Puts(amplitud_pantalla, &Font_7x10, 1);
68 SSD1306_GotoXY(70, 10);
69 SSD1306_Puts(frec_pantalla, &Font_7x10, 1);
70 SSD1306_GotoXY(110, 0);
71 SSD1306_Puts("V", &Font_7x10, 1);
72 SSD1306_GotoXY(110, 10);
73 SSD1306_Puts("Hz", &Font_7x10, 1);
74 if (ConfSub == 1) {
75     flag_frec = 1;
76 }
77 if (ConfSub == 2) {
78     flag_amp = 1;
79 }
80 if (Boton2 == 0 && flag_salto_screen == 1) {
81     Screen = 1;
82     flag_salto_screen = 0;
83     flag_cursor_quieto = 0;
84 }
85 break;
86 case 1:
87     SSD1306_GotoXY(0, 0);
88     SSD1306_Puts("Configuracion", &Font_7x10, 1);
89     SSD1306_GotoXY(0, 20);

```

```

83 SSD1306_Puts("Cursor", &Font_7x10, 1);
84 SSD1306_GotoXY(0, 30);
85 SSD1306_Puts("Frecuencia:", &Font_7x10, 1);
86 SSD1306_GotoXY(0, 40);
87 SSD1306_Puts("Amplitud:", &Font_7x10, 1);
88 SSD1306_GotoXY(0, 50);
89 SSD1306_Puts("Imp por UART:", &Font_7x10, 1);
90 //SWITCH DE LA PSEUDOMAQUINA DEL SCREEN1
91 switch (ConfSub)
92 {
93     case 0:
94         flag_enc = 1;
95         flag_uart = 0;
96         flag_frec = 0;
97         flag_amp = 0;
98         SSD1306_FillArea(0, 20, SSD1306_WIDTH, 10, 1);
99         SSD1306_GotoXY(0, 20);
100        SSD1306_Puts("Cursor", &Font_7x10, 0);
101        SSD1306_GotoXY(70, 20);
102        if (encoder_c == 1) {
103            SSD1306_Puts("si", &Font_7x10, 0);
104            enc_graf = 1;
105            flag_salto_screen = 1;
106            flag_cursor_quieto = 0;
107        }
108        else if (encoder_c == 0) {
109            SSD1306_Puts("no", &Font_7x10, 0);
110            enc_graf = 0;
111            flag_salto_screen = 0;
112            flag_cursor_quieto = 1;
113        }
114        if (Boton2 == 1 && flag_salto_screen == 1) {
115            Screen = 0;
116        }
117        break;
118        case 1:
119            flag_enc = 0;
120            flag_uart = 0;
121            flag_frec = 1;
122            flag_amp = 0;
123            flag_cursor_quieto = 1;
124            SSD1306_FillArea(0, 30, SSD1306_WIDTH, 10, 1);
125            SSD1306_GotoXY(0, 30);
126            SSD1306_Puts("Frecuencia:", &Font_7x10, 0);
127            SSD1306_GotoXY(90, 30);
128            SSD1306_Puts(frecuencia_display, &Font_7x10, 0);
129            flag_salto_screen = 1;
130            if (Boton2 == 1 && flag_salto_screen == 1) {
131                Screen = 0;
132            }
133            break;
134            case 2:

```

```

135         flag_enc = 0;
136         flag_uart = 0;
137         flag_freq = 0;
138         flag_amp = 1;
139         flag_cursor_quieto = 1;
140         SSD1306_FillArea(0, 40, SSD1306_WIDTH, 10, 1);
141         SSD1306_GotoXY(0, 40);
142         SSD1306_Puts("Amplitud:", &Font_7x10, 0);
143         SSD1306_GotoXY(90, 40);
144         snprintf(amplitud_display, sizeof(amplitud_display),
145                 ↪ "%d.%02d", integerPart, decimalPart);
146         SSD1306_Puts(amplitud_display

```

5.2. Tarea Generación de señal

```

1     void StartDAC(void *argument)
2     {
3         /* USER CODE BEGIN StartDAC */
4         vTaskSetApplicationTaskTag(NULL, (void*) TAG_TASK_DAC);
5         /* Infinite loop */
6         for(;;)
7         {
8             for(int i = 0; i < NSEN ; i++){
9                 sine_wave[i] = (uint8_t)((float)amplitud/2*(sin(2.0*PI*i*
10                    ↪ frecuencia/(float)NSEN)+1.0));
11             }
12             FreeDAC = 4 * osThreadGetStackSpace(DACHandle);
13             FreeHeap = xPortGetFreeHeapSize();
14             osDelay(10);
15         }
16         /* USER CODE END StartDAC */
17     }

```

5.3. Tarea monitoreo

```

1     void StartBorneraTask(void *argument)
2     {
3         /* USER CODE BEGIN 5 */
4         vTaskSetApplicationTaskTag(NULL, (void*) TAG_TASK_BORNERA);
5         /* Infinite loop */
6         for(;;)
7         {
8             Encoder_UpdatePosition();
9             // LOGICA BOTON ENCODER
10            static uint32_t last_screen_change_time = 0;
11            uint32_t current_time_screen = HAL_GetTick();
12            if ((current_time_screen - last_screen_change_time) > 200){
13                if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_3) == GPIO_PIN_RESET){
14                    Screen = (Screen + 1) % 3;

```

```
15     last_screen_change_time = current_time_screen;
16 }
17 }
18 // LOGICA BOTON CONFSUB
19 static uint32_t last_conf_change_time = 0;
20 uint32_t current_time_conf = HAL_GetTick();
21 if ((current_time_conf - last_conf_change_time) > 200){
22     if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_2) == GPIO_PIN_RESET)
23     {
24         ConfSub = (ConfSub + 1) % 4;
25         last_conf_change_time = current_time_conf;
26     }
27 }
28 static uint32_t last_conf2_change_time = 0;
29 uint32_t current_time_conf2 = HAL_GetTick();
30 if ((current_time_conf2 - last_conf2_change_time) > 200){
31     if (HAL_GPIO_ReadPin(GPIOA, GPIO_PIN_4) == GPIO_PIN_RESET){
32         Boton2 = (Boton2 + 1) % 2;
33         last_conf2_change_time = current_time_conf2;
34     }
35 }
36 FreeBornera = 4 * osThreadGetStackSize(BorneraHandle);
37 osDelay(1);
38 }
39 /* USER CODE END 5 */
40 }
41 \end{verbatim}
```