

## Problema especial 2

UNLP - Facultad de Ingenieria  
Sistemas embebidos - E1504  
Potente Guido - 73230/5

### Planteo de la resolucioin

#### Consigna base

En primer lugar se busco un diseño del sistema para cumplir con las consignas de modo que se cumpla:

1. Captura del **adc** a  $1kHz$  mostrando en pantalla la **FFT** cada 256 muestras.
2. Uso de un cursor que permita ver el valor actual de tension y frecuencia en ese punto.
3. Creacion de una **PWM** con el **dma** para pasarla por un pasabajos y obtener una sinusoidal a muestrear.
4. Encoder con el que mover el cursor.
5. Botones para seleccionar cosas en el menu y cambiar entre pantallas.
6. Una pantalla de configuracion, una para la **FFT** y una para ver los valores actuales.
7. Uso de 3 tareas y algun metodo de sincronizacion.

Ademas, en la pantalla de configuracion se deben tener los siguientes botones:

- Cambio de amplitud de la **PWM**.
- Cambio de la frecuencia de la **PWM**.
- Impresion por **UART** de los valores de los valores actuales medidos.
- Cambio entre el encoder como cursor o cambiando las variables de la **PWM**.

Con esto en mente se procedio con un esquematico basico de las interfaces esperadas.

#### Idea basica del esquema

Para tener un esquema de lo buscado con las pantallas se dibujo en papel algunos ejemplos hasta llegar al resultado deseado. Las siguientes figuras muestran los esquemas finales con los que se ideo el resto.

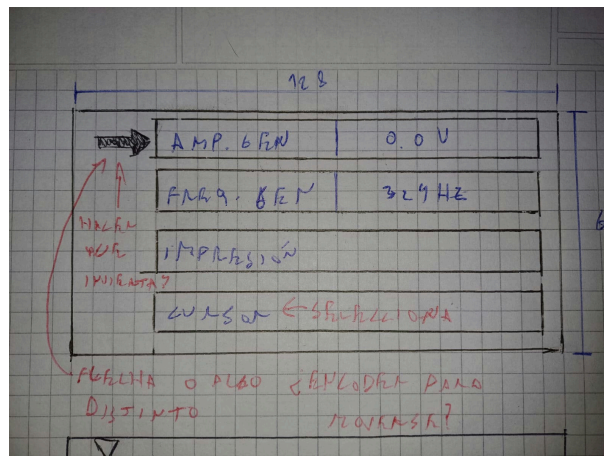


Figure 1: Diagrama basico de la configuracion

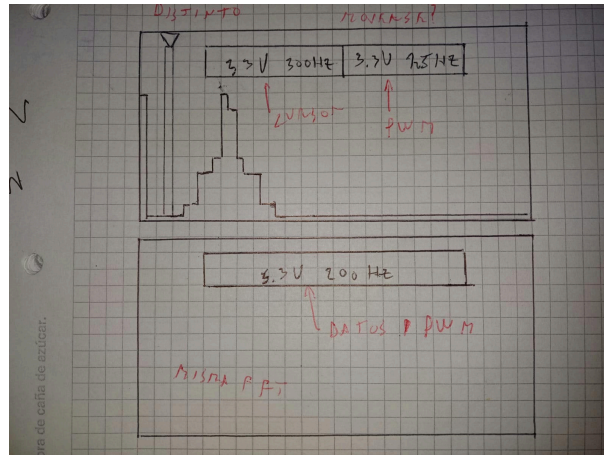


Figure 2: Diagrama basico de la fft en modo cursor y con la pwm

Como se puede ver se removio la pantalla de visualizacion de los valores actuales porque la **FFT** con el cursor ya tienen esta informacion. Ademas, la funcionalidad del puntero en la configuracion lo que hace es seleccionar la opcion y con el encoder modificarla de modo que aumente o disminuya lo que se esta haciendo.

## Toma de datos

En cuanto a los datos que deben ser tomados se decidio usar interrupciones para todos con la finalidad de poder ahorrar la creacion de una tarea que tenga que estar constantemente sampleando todas las entradas. Para esto se tendran que incorporar por interrupcion estas cosas:

- **adc** con un timer cada 1ms.
- **PWM** con el **DMA** para no tener que estar manualmente actualizando.
- Encoder con interrupcion por **GPIO**.
- Botones con timer para usar estrategia antirebote.

Por lo tanto solo seran necesarias dos tareas para hacer todo el proceso.

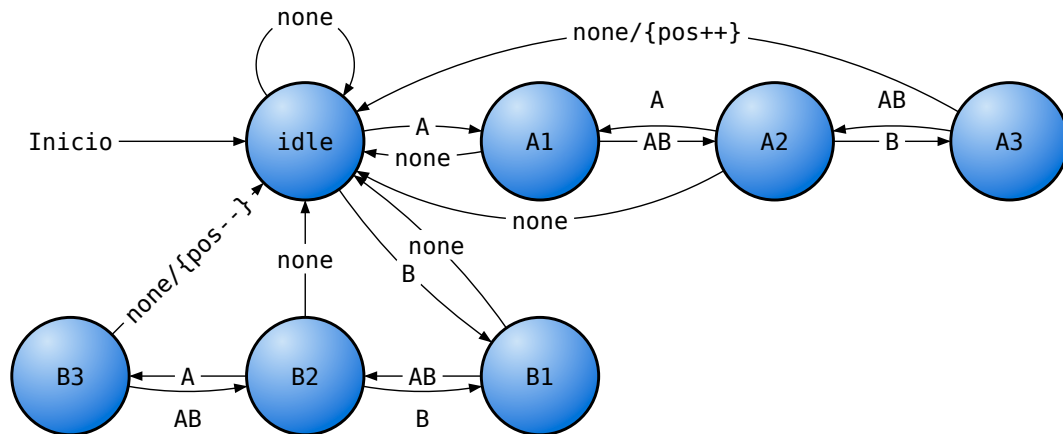
## Implementacion de funciones

### PWM

Para poder hacer esta onda especifica se tomo lo dado en clase generando los valores del duty cycle para cada ciclo de la **PWM** y despues enviar esto con el **DMA**. Esto permite precision en la onda final, pero genera problemas a hora de hacer frecuencias muy bajas respecto a la **PWM** pues aumenta la cantidad de puntos a almacenar. Por lo tanto, con esto se planteo usar una funcion para setear una frecuencia y amplitud.

### Encoder

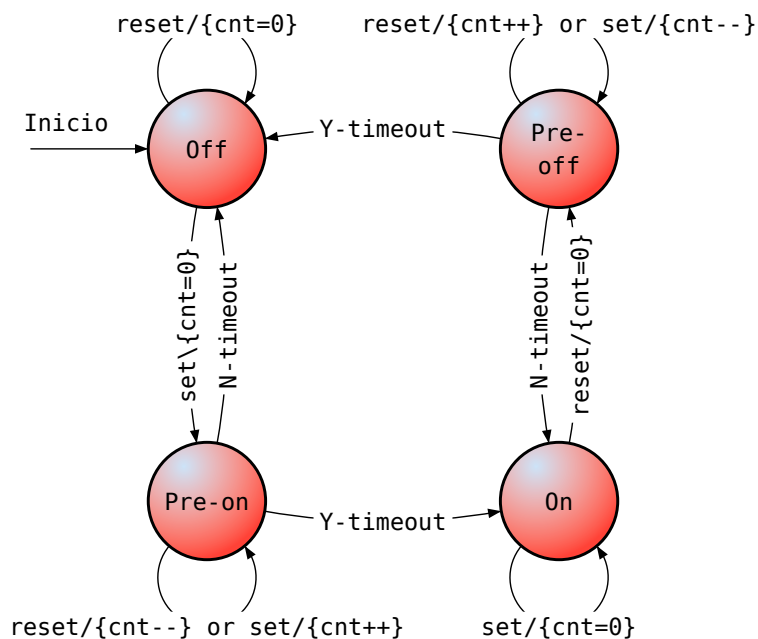
El encoder tiene la caracteristica de que es ruidoso y muchas veces se puede volver dificil detectar un cambio correcto sin una estrategia antirebote. Entonces, se tomo la maquina de estados dada en clase que setee la posicion respecto a un offset.



Como se puede ver esta maquina avanza en los estados hasta llegar a comprobar que es correcta realmente se pudo hacer un paso del encoder. Cabe aclarar que los eventos son dependiendo de que pin este en bajo, como por ejemplo el **AB** representa cuando ambos pines estan activos.

## Botones

En este caso tambien se implemento una estrategia antirebote por lo que necesariamente se debe hacer un muestreo constante a estos. La maquina de estados de estos tambien se dio en clase por lo que nuevamente se hace lo mismo.



Por lo tanto con los botones usando un timer y el encoder con las interrupciones por **GPIO** se puede no usar una tarea para leer variables. Falta el **adc** pero este simplemente lee en el valor y lo guarda en un arreglo con la interrupcion de un timer.

## UART

Para pasar los datos se puede hacer bloqueante, pero es conveniente que no tarde mucho por lo que se impone la maxima frecuencia asincronica que soporta el controlador de **UART** y la *Blue Pill* (921600 *bps*). Ademas cree un arreglo con las strings para los 128 valores a pasar de modo que cada vez que se escribe unicamente se necesita escribir este los valores y pasarlo.

## Interrupciones y tareas

## Interrupciones

Como ya se menciono, se tienen las siguientes interrupciones configuradas:

- Timer para el **adc** con prioridad **5** que es la mayor posible con la configuracion de **FreeRTOS** actual.
- **GPIO** interrupt en bajada y subida para el encoder con prioridad **6** por lo que es critico como el **adc** y no es tan importante la temporizacion pues simplemente se debe usar para detectar si se movio o no.
- Timer para los botones integrado con el mismo del **SysTick** que tiene la menor prioridad (**15**) posible. Nuevamente la temporizacion no es importante pues lo unico que puede cambiar es que se modifique el tiempo del contador anti-rebote, pero no es algo grave a su funcionamiento.

## Tareas

Como no se debe usar la tarea del monitoreo solamente nos queda una que actualiza pantalla y otra que procesa los datos para la **FFT**. Teniendo en cuenta que el proceso de la pantalla suele tomar  $30ms$  mientras que el procesamiento solo se hace en una pantalla y cada  $256ms$  es logico asignar mayor prioridad a esta pues es la que menor tiempo consume. Mas aun, la actualizacion de pantalla puede retrasarse cerca de  $70ms$  pues eso le daría cerca de  $100ms$  de respuesta.

## Solucion

### Sincronizacion

Para sincronizar las tareas se usaron dos mutex ya que uno habilita la tarea de procesamiento y el otro es para bloquear cuando se esta procesando. Esto es porque mientras no este en la pantalla de la **FFT** no necesito calcular nada, pero cuando sea necesario es importante no actualizar la pantalla hasta que esten los datos porque el buffer de datos es uno solo lo que podria ocasionar problemas de lectura.

Por lo tanto los mutex son:

- **adcSave** → Bloquea el procesamiento hasta estar en la pantalla de la **FFT**.
- **fftRead** → Bloquea la tarea de pantallas hasta que se termine el calculo de la **FFT**.

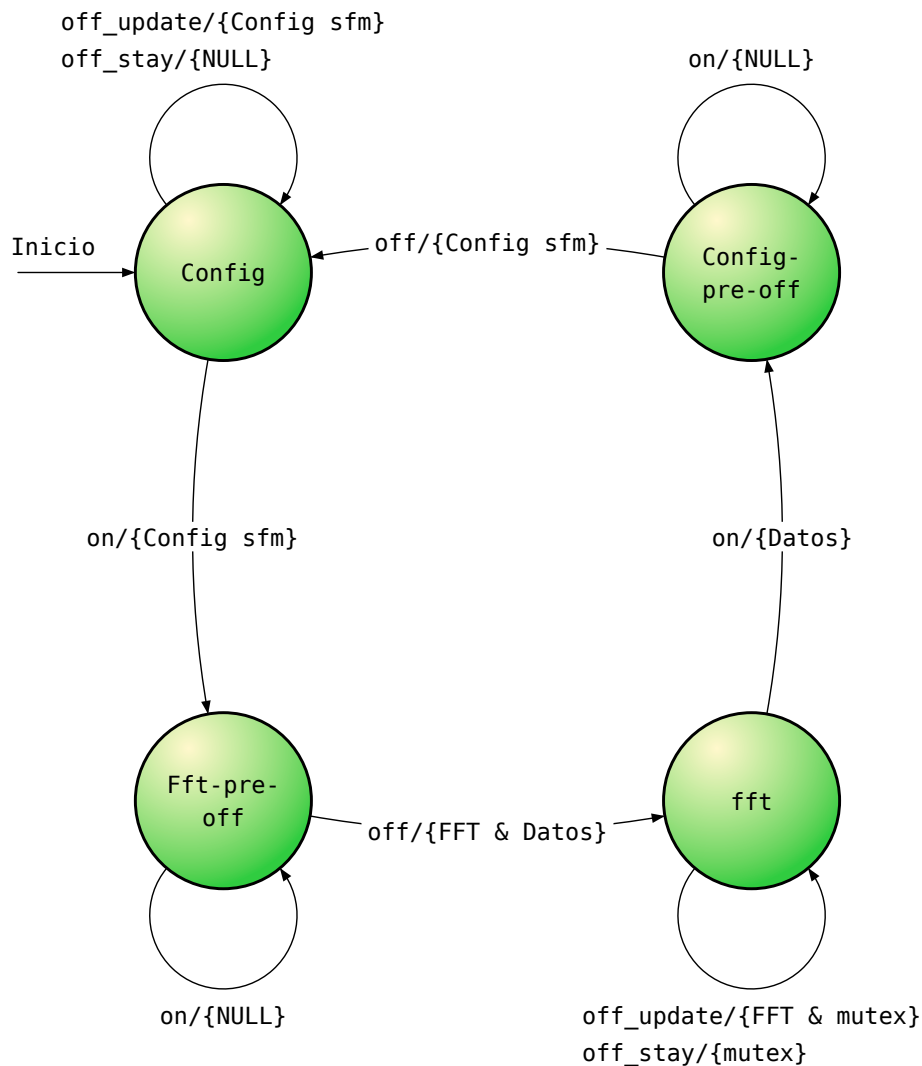
### Modo de operacion ideado

Teniendo en cuenta los puntos marcados se decidio que finalmente la idea es:

- **Pantalla configuracion**
  - 4 opciones que se recorren con un cursor que se mueve a partir del encoder.
  - Si el encoder se apreta una vez selecciona el item y no se mueve.
  - Los valores de tension/frecuencia y la funcion del encoder se modifican girando el encoder.
  - Los valores finales se ejecutan unicamente cuando se aprete nuevamente el encoder.
- **Pantalla FFT**
  - Con un segundo boton se cambia entre esta y la configuracion.
  - En caso de ser con cursor el movimiento del encoder mueve de a 1 Hz.
  - El muestreo solo para cuando se calcula y actualiza la pantalla.
  - En caso de ser con **PWM** el encoder mueve la frecuencia o amplitud.

### Maquina de estados

Puesto que la maquina de estados se volvio muy complicada de implementar completa en una sola se plantearon dos maquinas iguales pero que ejecutan distintas acciones y donde una esta dentro de la otra. Esto refiere a que hay una maquina para los displays y dependiendo del estado y evento de esta, se ejecuta otra maquina identica pero de la pantalla de configuracion.



Los eventos estan determinados por las siguientes condiciones:

- **off\_stay** → No hay actualizacion de botones, encoder o pedido de **FFT** y el boton 2 esta apagado.
- **off\_update** → Hay alguna actualizacion y el boton 2 esta apagado.
- **on\_stay** → No hay actualizacion de botones, encoder o pedido de **FFT** y el boton 2 esta prendido.
- **off\_update** → Hay alguna actualizacion y el boton 2 esta prendido.

Cabe aclarar que los **on** o los **off** significan que sea con o sin actualizacion se ejecuta eso.

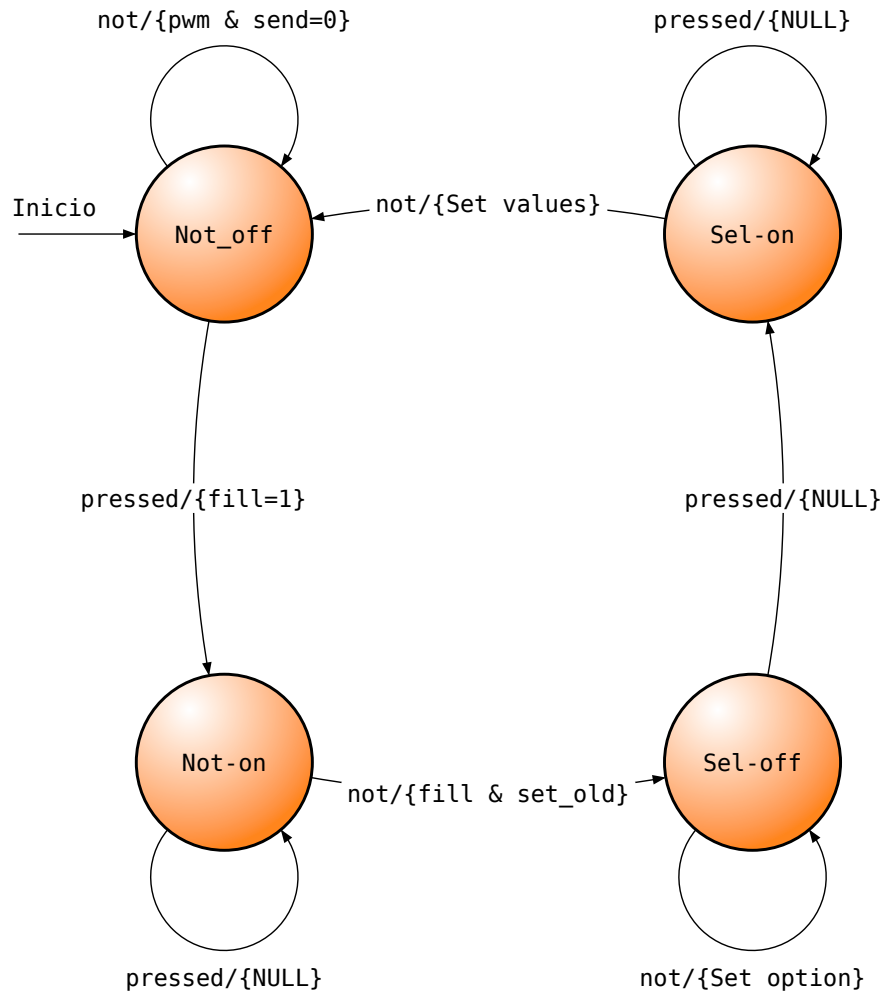
Por otro lado las acciones son:

- **update** → En cualquier caso que haya un evento con *update* se actualiza la pantalla.
- **Config sfm** → Llamado a la maquina de la pantalla de configuracion.
- **FFT** → Grafica la **FFT**.
- **Datos** → Libera o toma el mutex que permite a la tarea de procesamiento funcionar.
- **mutex** → Intenta tomar el mutex que calcula la **FFT** de modo que espere en caso de no estar disponible.

Aunque la maquina de estados de la pantalla de configuracion es identica, toma acciones distintas en cada caso, pero para resumir su uso lo que se hace es tener un **struct** accesible solo a la libreria que tiene banderas que pueden llamarse con funciones **getter** y ejecutar desde el **main**.

La razon de esta ejecucion fuera de la maquina es porque los botones se guardan en una cola que se debe leer en la tarea y pasar todos los valores a la maquina antes de obtener el estado actual. Esto significa que no tiene sentido actualizar la pantalla con un valor de la cola que instantaneamente se va a modificar.

En definitiva la maquina de la pantalla **config** es:



Los eventos estan determinados por las siguientes condiciones:

- **not** → Boton del encoder apagado.
- **on** → Boton del encoder prendido.

Por otro lado las acciones son:

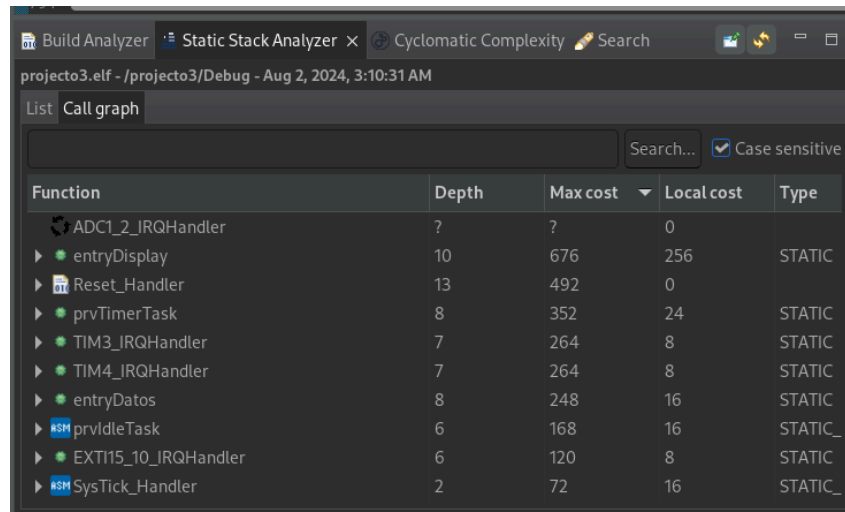
- **pwm** → Indica que se actualiza bandera para actualizar pwm.
- **fill** → Indica si la flecha del menu se debe rellenar o no.
- **send** → Especifica si mandar o no por uart.
- **set\_old** → Guarda el valor actual de la configuracion de la pantalla.
- **Set option** → Selecciona en base a la posicion del encoder una de las 4 opciones en pantalla y actualiza sus variables en caso de ser necesario.
- **Set values** → Activa flag para actualizar **PWM**, para cambiar el relleno de la flecha y dependiendo si se habia seteado para mandar por uart envia.

Por otro lado, siempre que entra a **off** guarda el ultimo valor del encoder y en **on** guarda el valor de **set\_old** en el actual. Esto es para evitar problemas en caso de que se actualize la variable por alguna otra razon.

Finalmente uno llama desde el **main** a una funcion de la libreria que en base a estas banderas dibuja la pantalla. Sin embargo todo lo que pueda tardar varios ms se pasa con in **getter** al main y de ahi se actualiza (lo unico que no cumple con esto es la **UART**).

## Uso de memoria

Observando las tareas durante la compilacion se obtuvieron los siguientes resultados:

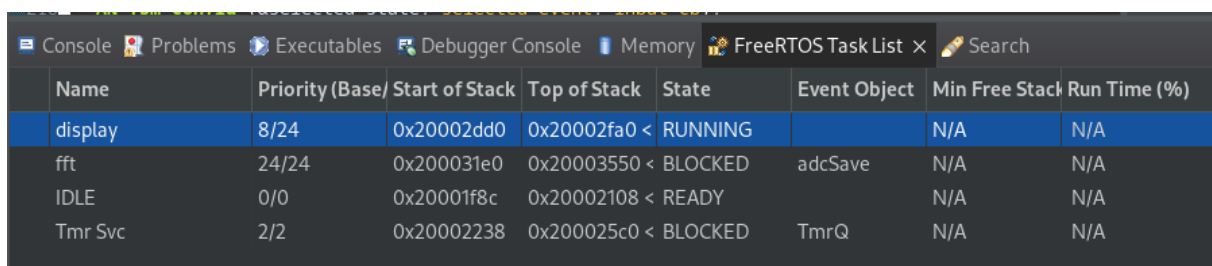


Function	Depth	Max cost	Local cost	Type
ADC1_2_IRQHandler	?	?	0	
entryDisplay	10	676	256	STATIC
Reset_Handler	13	492	0	
prvTimerTask	8	352	24	STATIC
TIM3_IRQHandler	7	264	8	STATIC
TIM4_IRQHandler	7	264	8	STATIC
entryDatos	8	248	16	STATIC
prvIdleTask	6	168	16	STATIC
EXTI15_10_IRQHandler	6	120	8	STATIC
SysTick_Handler	2	72	16	STATIC

Figure 3: Depuracion en la compilacion

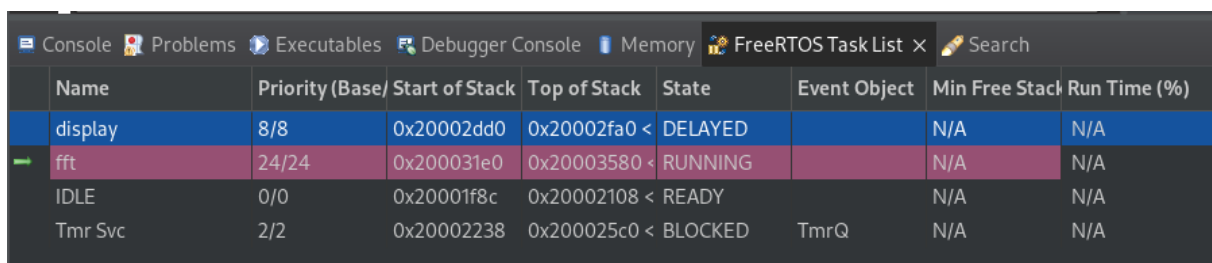
Como se puede ver, las tareas **entryDisplay** que procesa las pantallas consumira aproximadamente 676 bytes o 169 palabras. Por otro lado **entryDatos** consume 248 bytes o 71 palabras.

Por otro lado, usando el debugger para ver los stacks en distintos puntos de cada tarea se pudo obtener estos valores maximos:



Name	Priority (Base/Start of Stack	Top of Stack	State	Event Object	Min Free Stack	Run Time (%)
display	8/24	0x20002dd0	0x20002fa0 < RUNNING		N/A	N/A
fft	24/24	0x200031e0	0x20003550 < BLOCKED	adcSave	N/A	N/A
IDLE	0/0	0x20001f8c	0x20002108 < READY		N/A	N/A
Tmr Svc	2/2	0x20002238	0x200025c0 < BLOCKED	TmrQ	N/A	N/A

Figure 4: Maximo consumo encontrado en la tarea de los displays



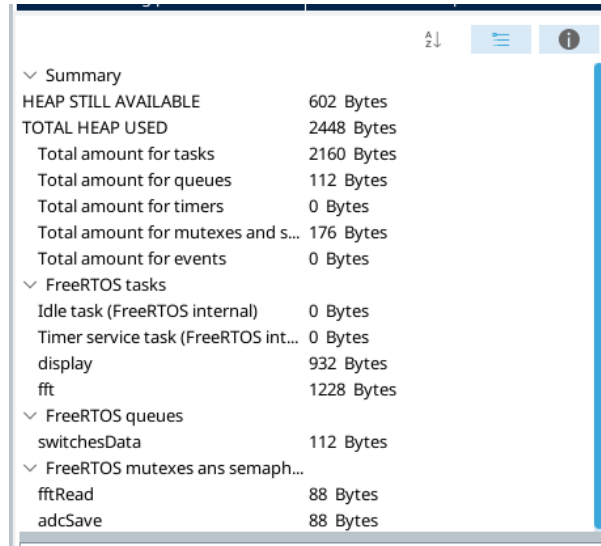
Name	Priority (Base/Start of Stack	Top of Stack	State	Event Object	Min Free Stack	Run Time (%)
display	8/8	0x20002dd0	0x20002fa0 < DELAYED		N/A	N/A
fft	24/24	0x200031e0	0x20003580 < RUNNING		N/A	N/A
IDLE	0/0	0x20001f8c	0x20002108 < READY		N/A	N/A
Tmr Svc	2/2	0x20002238	0x200025c0 < BLOCKED	TmrQ	N/A	N/A

Figure 5: Maximo consumo encontrado en la tarea que procesa los datos

Traduciendo a bytes/palabras se tiene que:

- **display** → 464 bytes / 116 palabras (menor a lo visto en la compilacion).
- **fft** → 924 bytes / 231 palabras (mayor a lo visto en compilacion).

Tomando el peor caso de cada uno y agregando un extra de 20% se tiene que **display** tendra 203 palabras y **fft** tendra 277. Ademas, esto nos determina en gran parte cuanto stack necesitamos para el **FreeRTOS** de modo que observando el heap usado:



Esta utilizando 2448 bytes por lo que nuevamente poniendo un margen del 20% se puede dejar en 2937 bytes de modo que finalmente el uso de RAM y Flash quede:

Memory Regions		Memory Details				
Region	Start address	End address	Size	Free	Used	Usage (%)
RAM	0x20000000	0x20004fff	20 KB	4.01 KB	15.99 KB	79.96%
FLASH	0x08000000	0x0800ffff	64 KB	4.88 KB	59.12 KB	92.37%

## Temporizacion de las tareas

Para observar el factor de uso se dispuso de distintas situaciones:

### 1. Pantalla configuracion en idle

- En este caso se deja la pantalla de configuracion quieta sin tocar botones o el encoder de modo que se ejecuta solo la tarea de los diplays con esta tasa:



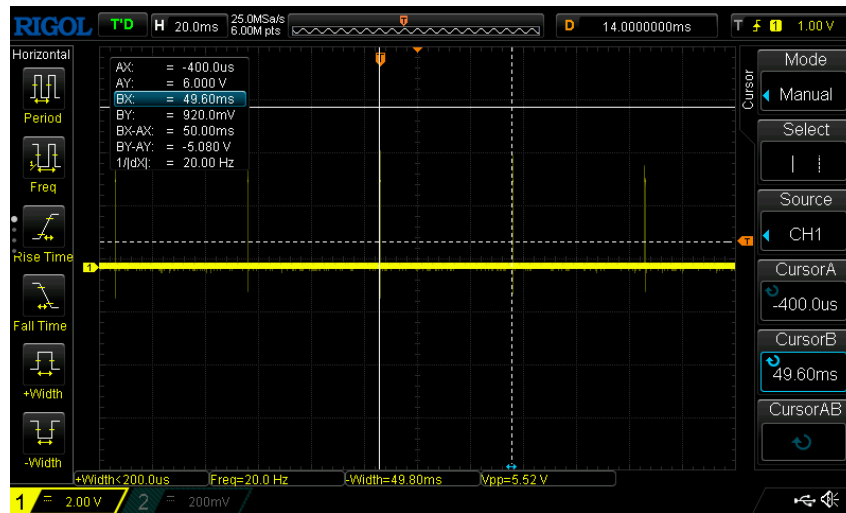


Figure 7: Hook de la tarea de display sin interactuar

Como se puede ver se llama cada 50 ms y como no ocurren nada sale en  $26\mu s$ , pero también hay que tener en cuenta el llamado a la interrupción del botón cada  $1ms$ , lo que tarda  $7.6\mu s$ .

$$\frac{26\mu s + 50 \times 7.6\mu s}{50ms + 26\mu s} \times 100\% \approx 0.81\%$$

## 2. Moviendo constantemente

- Si fuese posible mover constantemente el encoder o algún botón se obtendría la mayor tasa de uso en la pantalla de configuración. Esto es debido a que en todos los llamados a la tarea se tendrá que actualizar la pantalla, que tarda cerca de  $30ms$ .

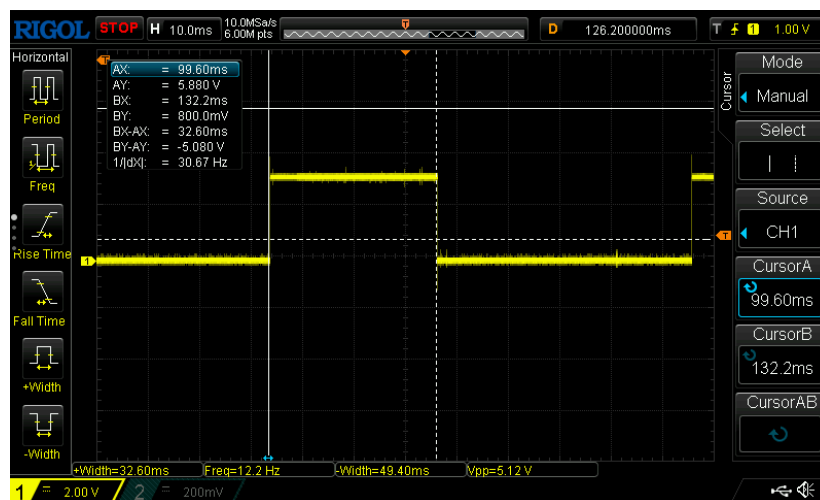


Figure 8: Hook de la tarea display cuando se debe actualizar

Con esto ya se puede calcular teniendo en cuenta nuevamente la interrupción:

$$\frac{32.6ms + 50 \times 7.6\mu s}{50ms + 32.6ms} \times 100\% \approx 39.92\%$$

## 3. Mostrando FFT

- En caso de que se muestre la **FFT** el uso aumenta pues la otra también está funcionando, como también la interrupción del adc. Esto implica que es más simple calcular en base a lo observando en la idle y midiendo las interrupciones.

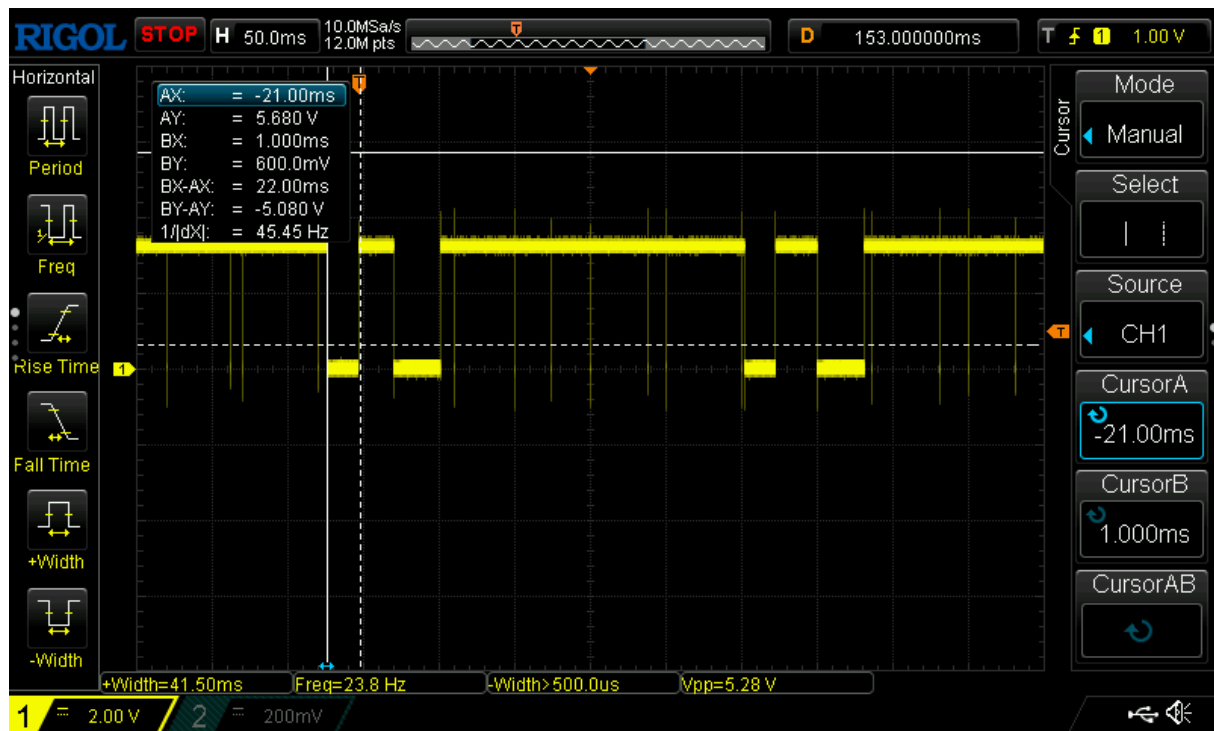


Figure 9: Tarea Idle durante la pantalla de la FFT

Entonces los datos no conocidos que se midieron son el calculo de la **FFT** (22ms), el adc que tarda  $21.2\mu s$  y el tiempo que esta en alto el idle (223ms).

$$\frac{22ms + 32.6ms + 21.2\mu s \times 256 + 7.6\mu s \times 277}{223 + 22 + 32.6} \times 100\% \approx 22.38\%$$

Claramente si uno mueve el cursor aumentara el uso pero solo momentaneamente porque no es posible mover constantemente el encoder.