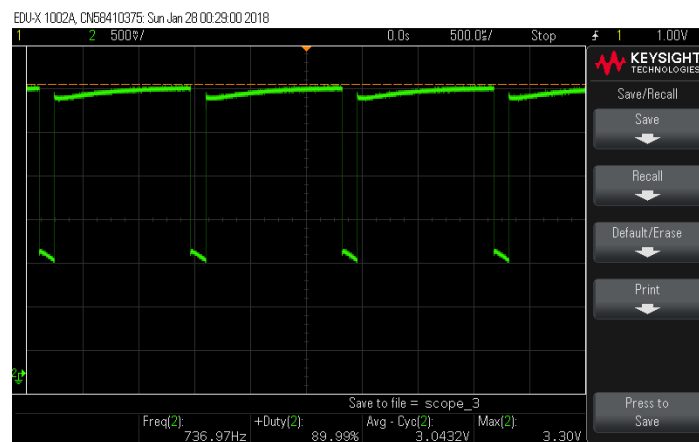
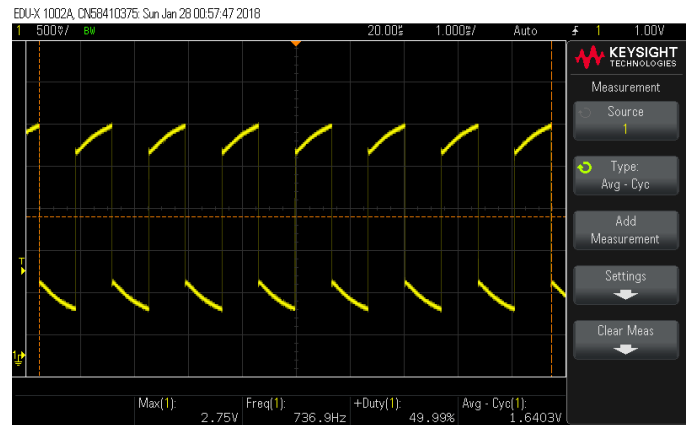
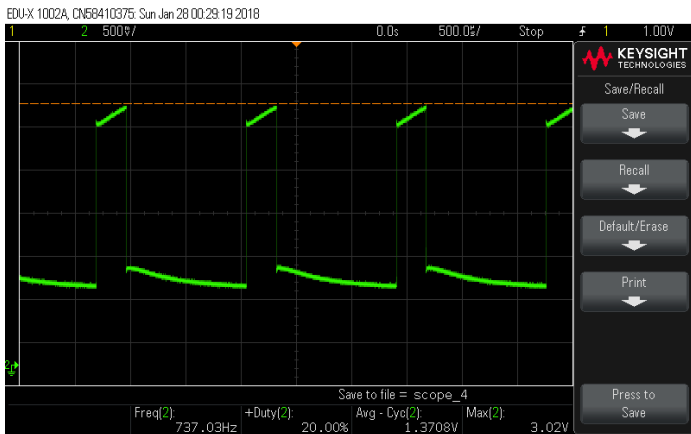


# Trabajo Final dsPIC33CH512MP508

## 1. Generación

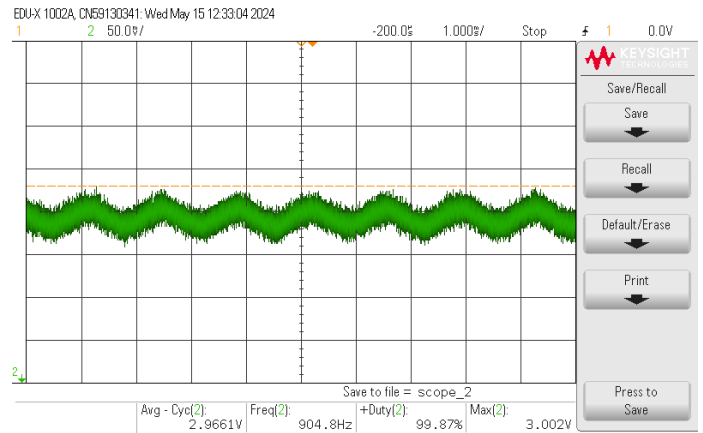
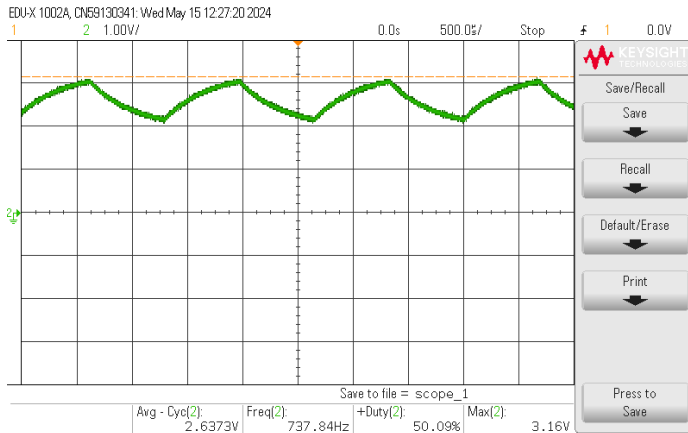
Se utiliza una señal de partida PWM de 737Hz generada con el módulo PWM Generator 2 de alta resolución, a través del pin RB13. Se utiliza una rutina de interrupción a través del pulsador RE9 para cambiar el ciclo de trabajo de la señal, pudiendo tomar valores desde 0.1 a 0.9.



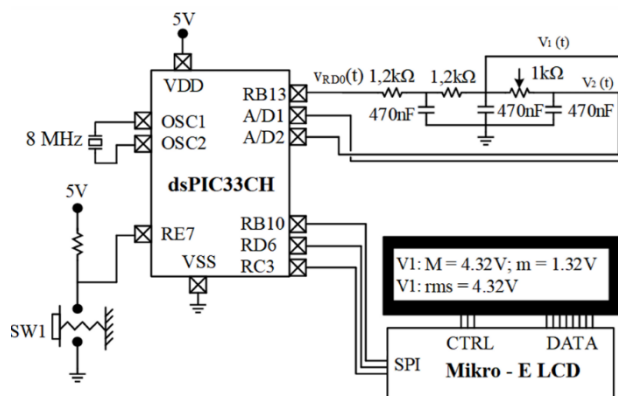
Al ir variando el ciclo de trabajo se modifica el valor medio, cambiando así el nivel de continua de la señal. Cuanto mayor duty se tenga más grande será el nivel de continua.

La amplitud de pico a pico no se ve afectada debido a que no se ha modificado la amplitud de la PWM. Sin embargo, el valor de pico absoluto sí que varía con el ciclo de trabajo.

La señal PWM se pasa a través de un filtro pasivo obteniendo dos señales, una senoidal y otra triangular. De las cuales se han medido sus valores medios, máximos y eficaces. Dichos valores se muestran en el LCD a través de los pulsadores RE7 (señal triangular) y RE8 (senoidal).



El esquema de conexiones muestra con mayor claridad las conexiones al dsPIC:



- Pulsador para cambiar el DUTY en pin RE9.
- Señal PWM generada a través de pin RB13.
- Canales de lectura de la conversión pines RC1 y RC2.
- Pulsadores RE7 y RE8 para mostrar medias en LCD.

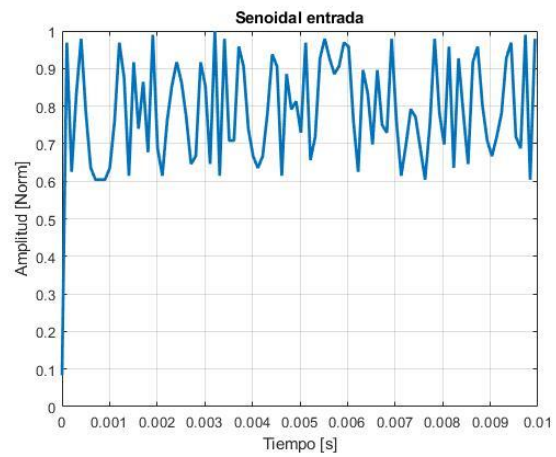
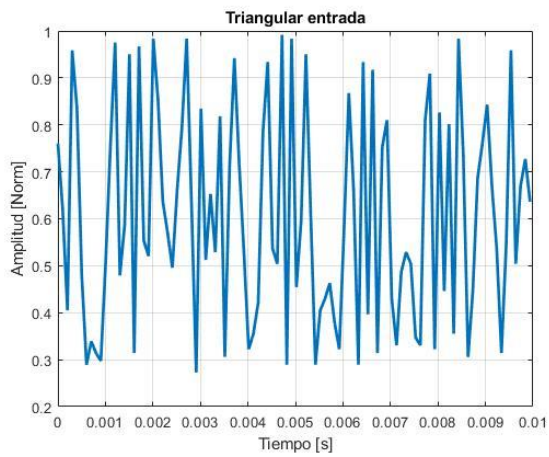
## 2. Muestreo

Ambas señales serán muestreadas mediante el CAD, definiendo dos canales desde donde se lee la conversión en los pines RC1 y RC2. Se utiliza un módulo SCCP independiente del PWM con un ciclo de trabajo constante del 50% y del mismo periodo que el PWM.

La interrupción del módulo SCCP activa el CAD el cual realiza el muestreo de las señales triangular y senoidal.

Para la elección de la frecuencia de muestreo se ha tenido en cuenta el teorema de Nyquist escogiéndose una frecuencia igual a  $f_s = 12529\text{Hz}$ .

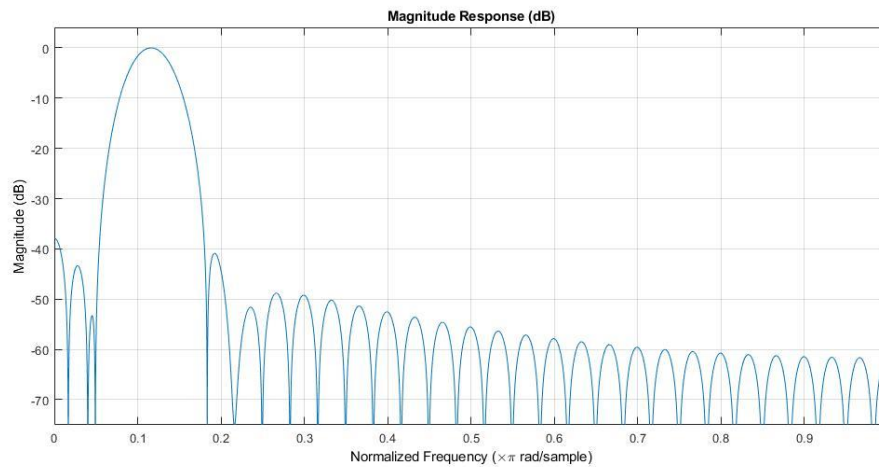
Cada  $79.82\mu\text{s}$  se captura una muestra de ambas señales almacenándose en dos vectores de 256 bytes para su posterior procesamiento.

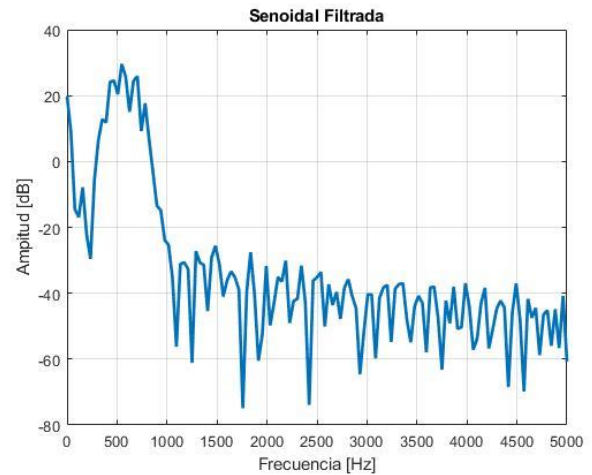
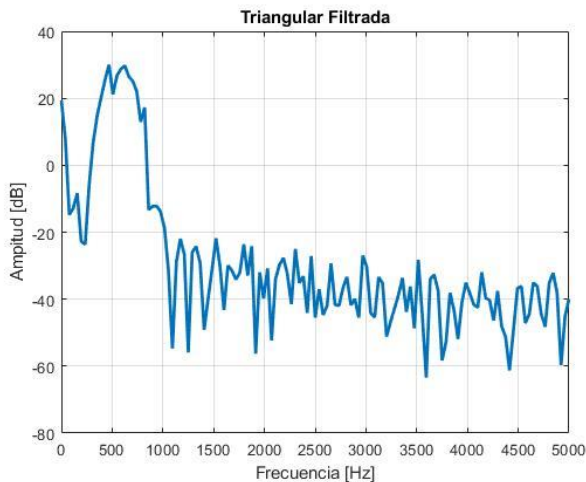


### 3. Filtrado

#### Filtro FIR

- Tipo de respuesta: Paso Banda
- Orden del filtro:  $N=60$
- Tipo de ventana: Hanning
- Frecuencias de corte:  $fc1=710\text{Hz}$ ;  $fc2=750\text{Hz}$ ;
- Frecuencia de muestreo:  $fs=12529\text{Hz}$

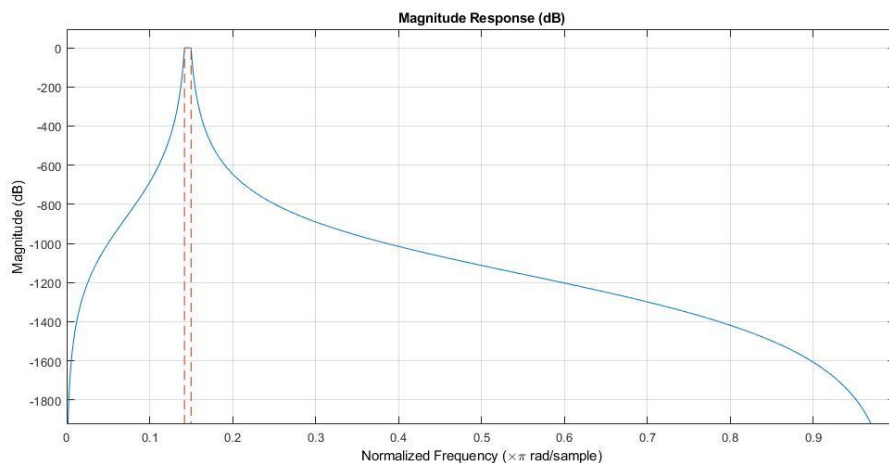




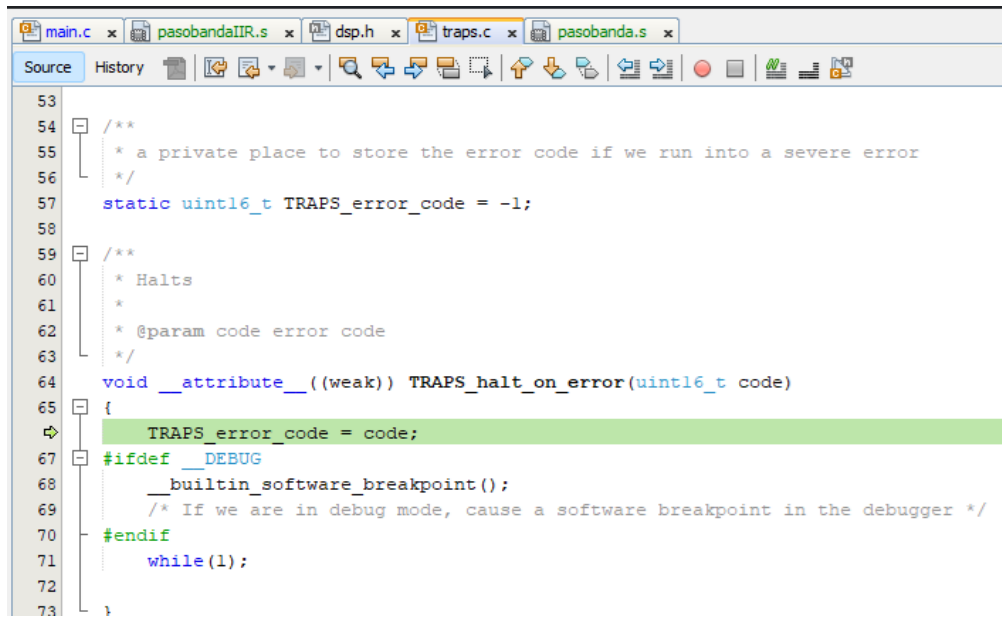
A pesar de que las frecuencias de corte están establecidas correctamente y el orden es el suficiente el filtro no atenúa la componente de baja frecuencia de ambas señales muestreadas. Esto puede deberse a efectos de borde que causan distorsiones en las primeras y últimas muestras de la señal filtrada. Una solución podría ser ignorar dichas muestras al analizar la señal. También existen componentes frecuenciales que al estar muy próximas a la banda de paso son difíciles de filtrar. Una medida podría ser aumentar el orden del filtro haciéndolo más selectivo mejorando así su capacidad para atenuar las frecuencias fuera de la banda de paso.

#### Filtro IIR

- Tipo de respuesta: Paso Banda
- Orden del filtro:  $N=60$
- Tipo de ventana: BUTterworth
- Frecuencias de corte:  $fc1=710\text{Hz}$ ;  $fc2=750\text{Hz}$ ;
- Frecuencia de muestreo:  $fs=12529\text{Hz}$



Una vez realizado el filtro con la herramienta Fiter Designer de Matlab se construyó el script (genera\_coeff\_IIR.m) para generar el archivo (pasobanda\_IIR.s) y pasarlo al dsPIC. Sin embargo, arroja un error que no he podido solucionar.



The screenshot shows a code editor with several tabs: main.c, pasobandaIIR.s, dsp.h, traps.c (selected), and pasobanda.s. The editor displays the source code for traps.c, which includes a static variable TRAPS\_error\_code and a function TRAPS\_halt\_on\_error. The function sets the error code and, if in debug mode, triggers a software breakpoint and enters an infinite loop.

```
53
54 /**
55  * a private place to store the error code if we run into a severe error
56  */
57 static uint16_t TRAPS_error_code = -1;
58
59 /**
60  * Halts
61  *
62  * @param code error code
63  */
64 void __attribute__((weak)) TRAPS_halt_on_error(uint16_t code)
65 {
66     TRAPS_error_code = code;
67     #ifdef __DEBUG
68         __builtin_software_breakpoint();
69         /* If we are in debug mode, cause a software breakpoint in the debugger */
70     #endif
71     while(1);
72 }
73
```