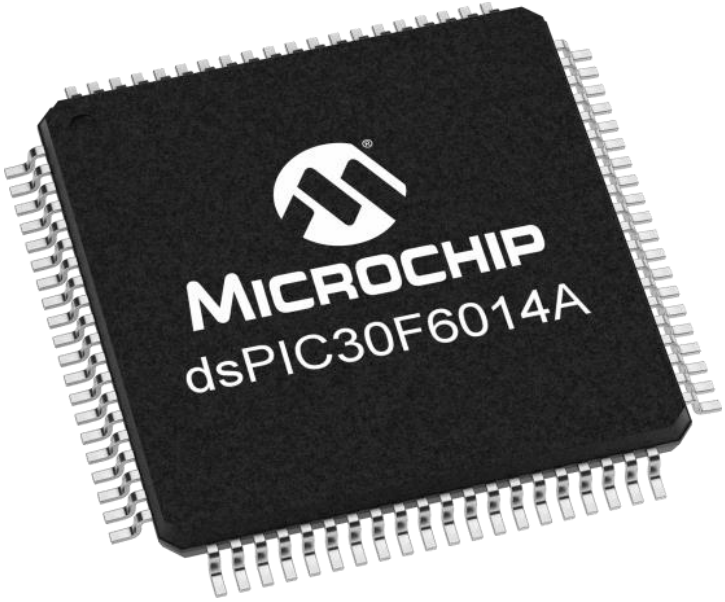
MEMORIA DEL TRABAJO DSPIC

DANIEL GONZÁLEZ DEL COZ

UO258128



El trabajo consistirá en realizar un muestreo sobre una señal *Pulse Width Modulation (PWM).* Filtrando esta señal vamos a obtener dos señales, una senoidal y una triangular.

La frecuencia de la señal PWM es de 737Hz, generada en el PIN D0, y las entradas de la señal triangular y senoidal serán los pines RB10 y RB11 respectivamente.

Apoyándonos en el LCD mostraremos por pantalla información relativa a las señales, todo ello programado en el entorno MPLAB X IDE y, con la ayuda de MATLAB, diseñaremos filtros específicos para las señales que con un código específico importaremos al MPLAB.

EJERCICIO 1 – FUNCIÓN BÁSICA

Generaremos una señal de onda cuadrada con un ciclo de trabajo (Duty) de 0.5, a una frecuencia de 737MHz.

Más o menos las tres señales serían:

$$\boldsymbol{PWM(t) = 5 * \prod_{0}^{\infty} \left(\frac{t - n * Ts}{d * Ts}\right)}$$

$$\boldsymbol{TIANGULAR(t) = V_{RB10pico} * \bigwedge_{0}^{\infty} \left(\frac{t - n * Ts}{Ts/2}\right)}$$

$$\boldsymbol{SENOIDAL(t) = \overline{V_{RB11}} + \frac{V_{RB11pico}}{2} * \left(-\cos(2 * \pi * \frac{1}{Ts} * t)\right)}$$

La frecuencia de muestreo elegida, cumpliendo *Nyquist*, es de 3000 Hz, para poder tomar aproximadamente cuatro muestras por periodo y no saturar nuestro controlador. El Ts o tiempo de sampling tendrá un valor de 333,3 µs.

Los valores de la señal se cogerán del buffer del CAD, almacenándolos en arrays y variables flotantes para sacar por pantalla valores de señal. Con el TIMER1 se controlarán los valores por pantalla, el TIMER2 se usará para generar la señal PWM, y el TIMER3 es para el CAD.

Al muestrear RB10 y RB11, y con los canales multiplexados AN10 y AN11, las variables se almacenarán en el ADCBUFA y en el ADCBUFB, respectivamente.

Lo primero a mostrar son los valores máximos, medio y rms de las señales triangulares y senoidales. Usando una función llamada MostrarVoltajes logramos tener una salida en la pantalla que se muestra en la siguiente imagen siguiente.

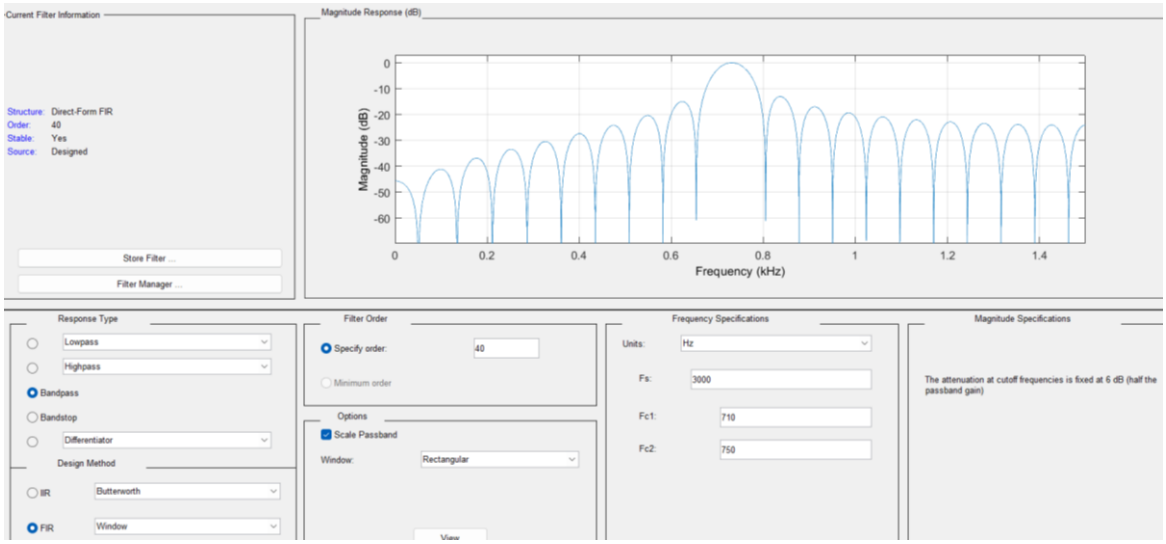
Todo se ha configurado mediante interrupciones:

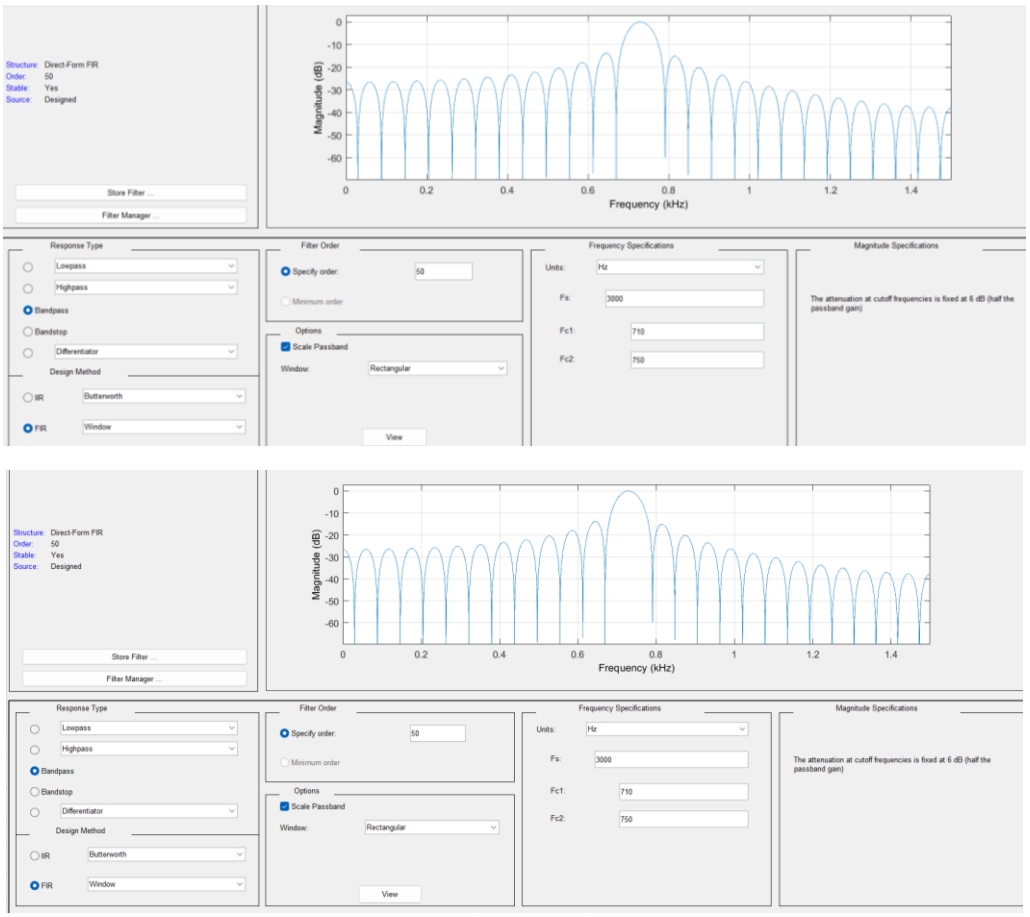
El refresco de pantalla del LCD cada 0,5 segundos con T1Interrupt.

El almacenamiento de las conversiones con el ADCINTERRUPT.

Para el diseño de filtro FIR (Finite Impulse Response) se hará uso de la herramienta Filter Designer de MATLAB. Generaremos un archivo .s con el que podremos filtrar la señal del dsPIC.

Para la elección del orden, se ha hecho un barrido entre varios para ver en cual obtenemos una buena respuesta y un orden no más alto del necesario para no sobrecargar código.

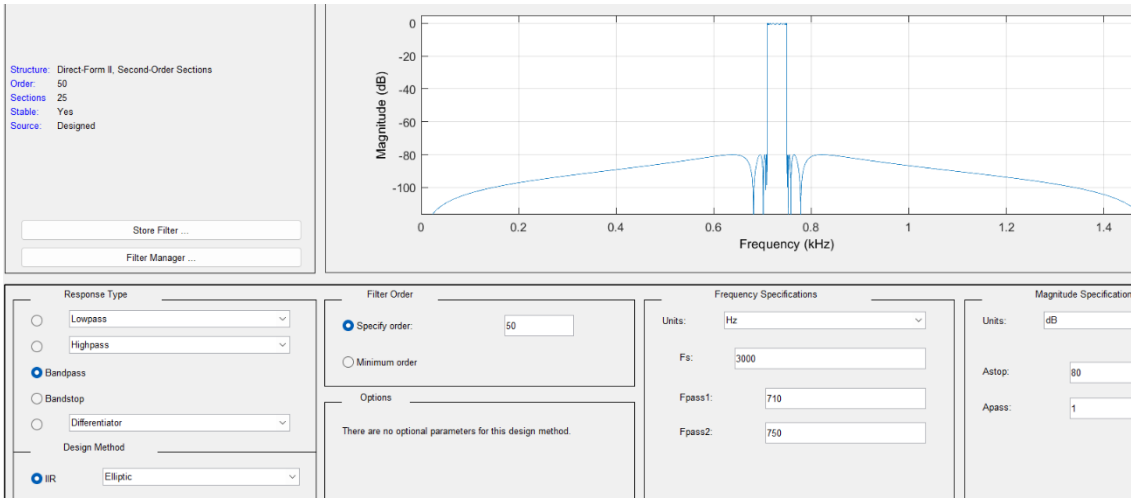




Se ve claro como cuanto más subimos el orden mejor respuesta tenemos. Con un orden igual a 50, creo que la respuesta ya es buena y el orden ya es alto. La frecuencia Fs será la que escogí antes, de 3000 Hz, y las frecuencias de corte inferior y superior serán 710 y 750 Hz respectivamente.

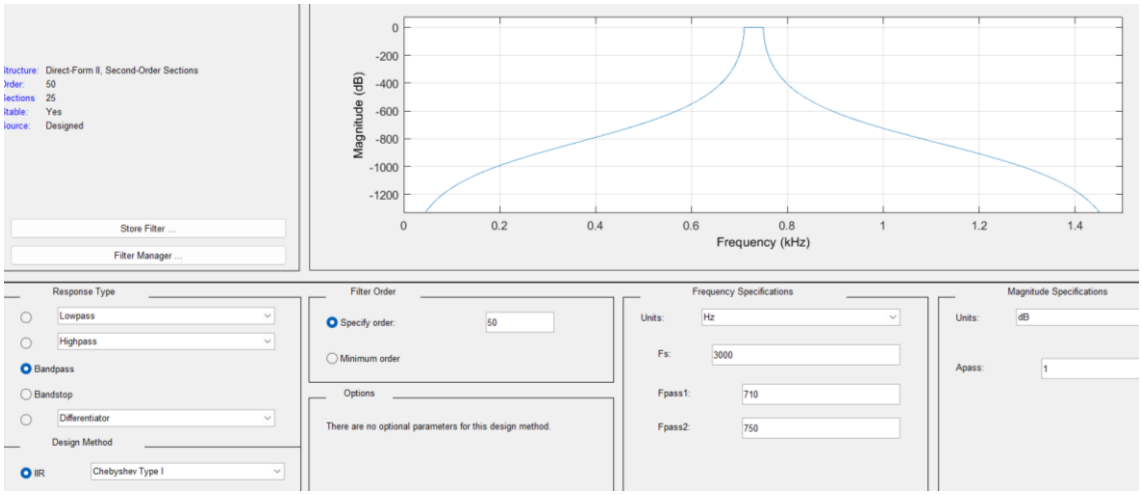
Un Filtro IIR (Infinite Impulse Response) para un orden similar, logra una respuesta más aguda que el FIR, así que se va a diseñar un filtro FIR con las mismas características para ver cómo sería su respuesta.

Con un elíptico, por ejemplo, se tendría:



Vemos como la respuesta ya es mucho más aguda. No obstante, entre los filtros IIR, los que mejor pendiente tienen son los elípticos. Su caída es muy abrupta, pero nos aparecen oscilaciones en cero decibelios, no tienen respuesta plana.

Por ejemplo, para no tener esas oscilaciones, escogería un Chebyshev de tipo I, con los mismos datos que arriba. Butterworth también tienen esa respuesta plana



GENERACIÓN DE CÓDIGO

1. MATLAB:

   Ayudándonos del archivo *genera_coeff.m* generaremos un archivo .s que contendrá la información del filtro para poder pasársela al dsPIC.

2. MPLAB:

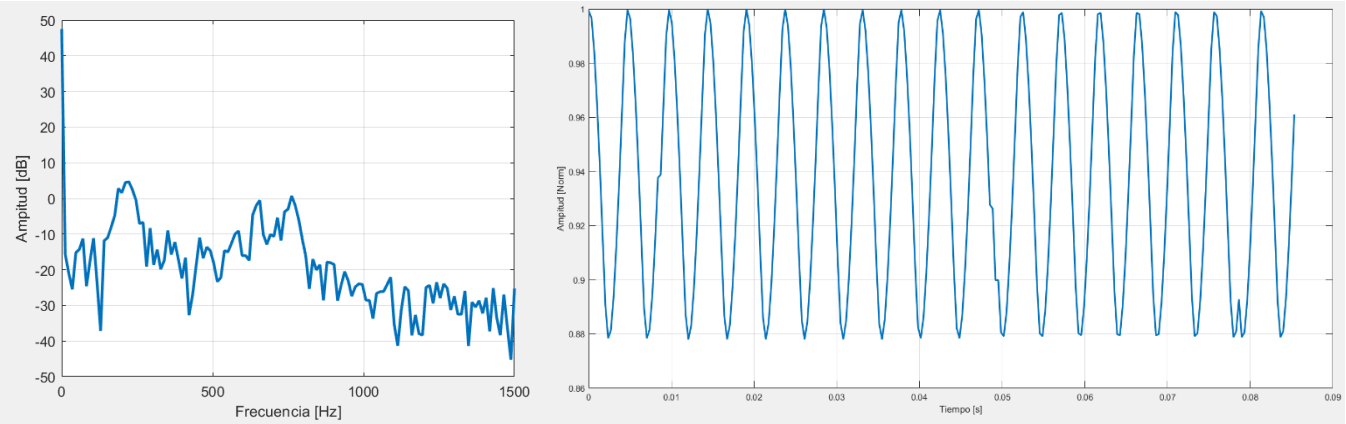
   Lo primero que habrá que hacer será tener las variables en formato *fractional,* como en el ejemplo de clase. Definiremos una estructura dónde cargaremos el filtro y en la propia interrupción del CAD llamaremos a la rutina del filtro, ya que es dónde se me llenan las variables, y tengo una variable llamada *posición* que se aumenta hasta 256, que es con la que voy llenando los arrays.

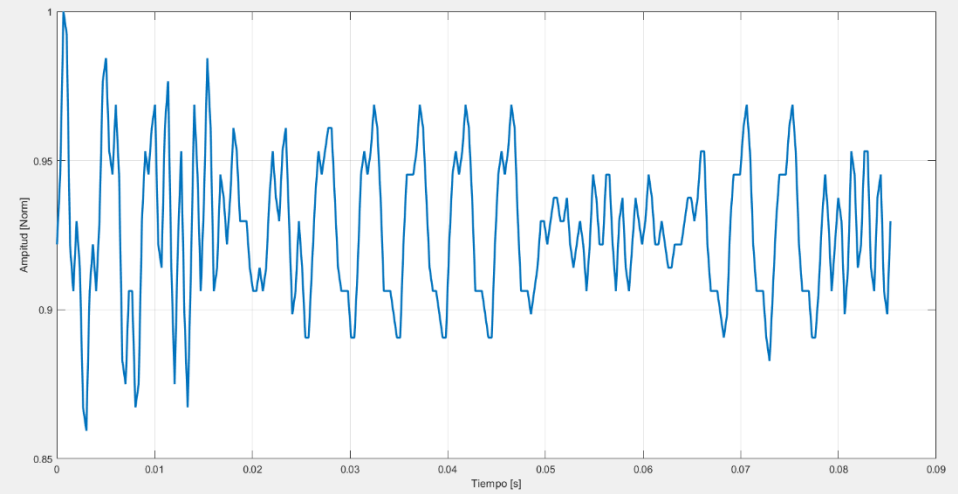
   El filtro lo inicializo en el main, como los timers, o el CAD.

AMPLIACIÓN – IMPLEMENTACIÓN DEL FILTRO FIR

Lo primero que hacemos es un Clean and Build for Debugging, luego exportamos los datos de los watches y los cargamos en Matlab para comprobar que realmente ha filtrado las señales.

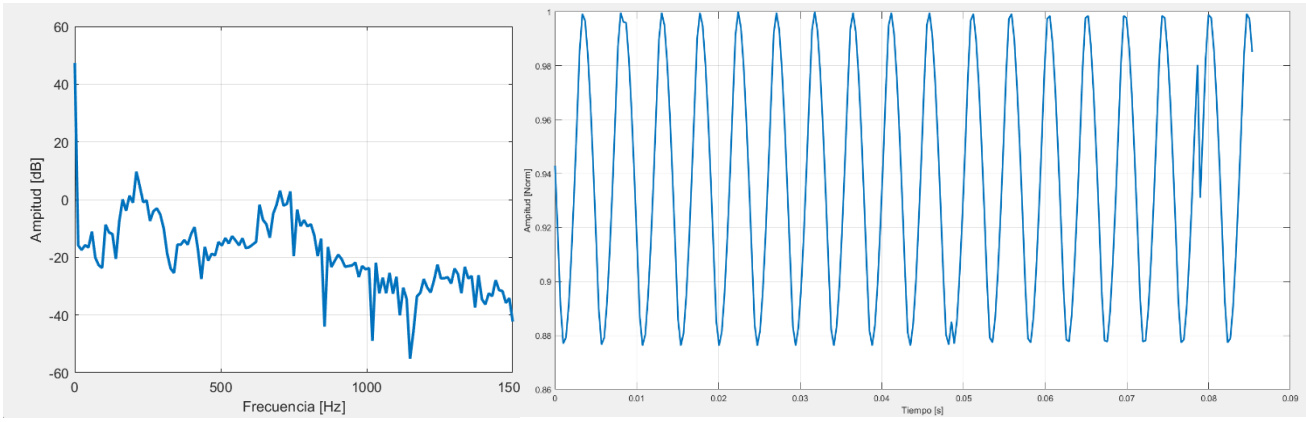
Con el dibuja.m obtenemos de la señal triangular:

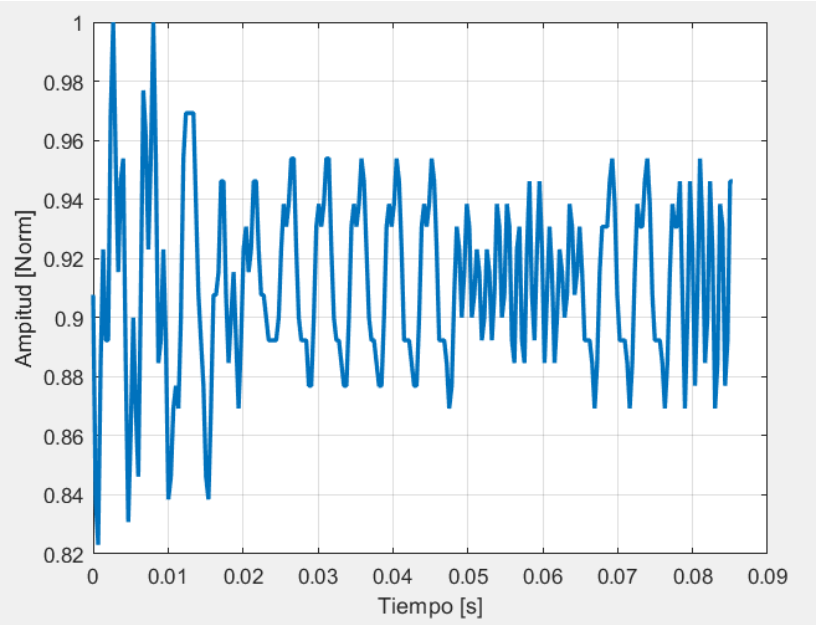




Las dos gráficas de arriba muestran la señal de entrada al filtro tanto en el dominio de la frecuencia como en el tiempo. Si nos fijamos, la señal de entrada tiene un nivel de continua que se ve reflejado en la delta a frecuencia 0 en el espectro.

Para la señal senoidal obtenemos:





Todos los archivos que se han extraído están en la carpeta del proyecto.

AMPLIACIÓN – DUTY

Se han creado dos funciones para modificar el ciclo de trabajo que inicialmente es 0.5. Las rutinas de interrupción lo incrementarán o decrementarán 0.1 unidades.

Se han usado los switches 1 y 2 para modificarlo, y se establece como valor máximo y mínimo 0.9 y 0.1. Cuando el duty alcance uno de esos dos valores se reseteará y volverá a valer 0.5.

Se ha propuesto esta idea de tener un switch para que suba y otro para que baje con la utilidad de poder ajustarlo a un número determinado. Si quieres 0.7 y te pasas a 0.8, para que puedas volver a atrás y no tengas que clicar en el switch siete veces para poder volver a ese valor.

Para los temas de prioridad, he considerado que lo más importante es el CAD, seguido del TMR1. Las interrupciones de los SW1 y SW2, las he considerado las de menos prioridad. Tanto el CAD como el TMR1 son los que realmente requieren tener más alto el tercer flag.

```
// Interrución CAD

IFS0bits.ADIF = 0;              // Clear ADC flag
IEC0bits.ADIE = 1;              // Enable ADC mask
IPC2bits.ADIP = 5;              // ADC priority level

//Interrupción TMR1

IFS0bits.T1IF = 0;              // Clear TMR1 flag
IEC0bits.T1IE = 1;              // Enable TMR1 mask
IPC0bits.T1IP = 4;              // TMR1 priority level

//Interrupción del SW1

IFS1bits.INT1IF=0;              // Clear SW1 flag
IEC1bits.INT1IE=1;              // Enable SW1 mask
IPC4bits.INT1IP=3;              // SW1 priority level

//Interrupción del SW2

IFS1bits.INT2IF=0;              // Clear SW2 flag
IEC1bits.INT2IE=1;              // Enable SW2 mask
IPC5bits.INT2IP=2;              // SW1 priority level

SET_CPU_IPL(1);                 // Set CPU priority level to a
```