Proyecto N°2: Convertidor puente completo con IGBT y transformador, pulso SPWM obtenido mediante microcontrolador PIC18F4550 que interactúa con interfaz gráfica en Matlab (enero 2021).

Jocelyn Matus y Rafael Burgos, Ingeniería Civil Electrónica, Asignatura Sistemas Electrónicos, 547307-1, Departamento de Ingeniería Eléctrica, Universidad de Concepción, Chile. Profesor Lautaro Salazar, Ayudante Felipe Pineda y Esteban Badilla.

Abstract – **En este informe se va a estudiar, investigar y emular un circuito de potencia llamado inversor, el cual convierte voltaje o corriente directa (DC) a una alterna (AC), a través de un tipo de modulación llamado modulación por ancho de pulsos sinusoidales (SPWM), el cual va a ser controlado con el microcontrolador PIC18F4550 a través de una interfaz gráfica.**

INTRODUCCIÓN

Los inversores es un tipo de convertidor estático de potencia que convierten voltaje o corriente DC a voltaje o corriente AC. Esto se realiza a través de conmutadores que, a través de un control, se abren y se cierran para crear la señal alterna.

La importancia de estos circuitos electrónicos de potencia proviene en la cantidad de uso que se le puede designar, como poder controlar la velocidad de un motor de inductancia a entregar energía a la red eléctrica a través de paneles solares. Por esto, se va a realizar, en nuestro caso, el estudio de un convertidor de puente completo, con switchs que son IGBTs, y transformador.

TEORÍA

Convertidor puente completo:

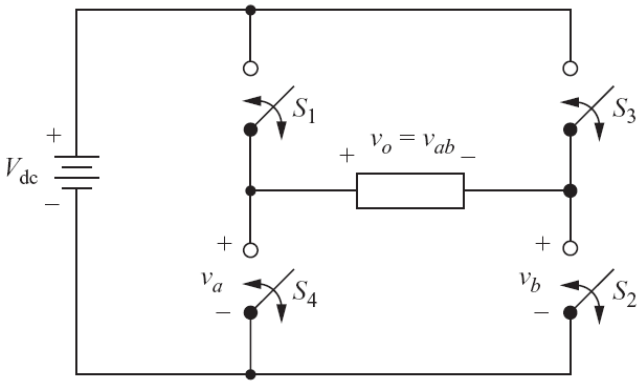

Figura 1. Convertidor Puente Completo

En primera instancia, un convertidor puente completo es un circuito de potencia que trabaja con 4 conmutadores o switch, los cuales se abren y se cierran de manera que el switch S1 y S2 están abierto y los switch S3 y S4 estén cerrados, y de manera inversa [1].

De esta manera, se le entrega a la carga una señal que, la cual filtrada por un filtro pasa bajo, entrega una señal deseada.

MODULACIÓN SPWM

Para poder generar cierta señal en la salida, se tienen que abrir y cerrar los switches a ciertos tiempos; es decir, con cierto duty y a cierta frecuencia. Para esto, que se llama modulación, se va a realizar un método: SPWM (Sine Pulse Width Modulation), modulación por ancho de pulsos sinusoidal, la cual la señal que modulada proviene de la comparación de una señal triangular con una señal de referencia.

Existen 2 tipos de modulación SPWM, las cuales se diferencian entre dos tipos: bipolar y unipolar.

En el método bipolar, el voltaje de salida cambia de Vcc a -Vcc, mientras que en el método unipolar el voltaje de salida cambia, en la primera mitad del ciclo, de Vcc a 0, y en el segundo ciclo de 0 a -Vcc. En nuestro caso, se va a utilizar el método bipolar ya que es mucho más fácil de implementar en el microcontrolador PIC, por ciertas funciones ya implementadas en estas, las cuales se van a detallar más adelante.

En lo que se especifica en la modulación bipolar, se tiene lo siguiente:

$$v_o = +V_{\text{dc}} \quad \text{for } v_{\text{sine}} > v_{\text{tri}}$$

$$v_o = -V_{\text{dc}} \quad \text{for } v_{\text{sine}} < v_{\text{tri}}$$

Figura 2. Voltaje de salida según comparación.

S_1 and S_2 are on when $v_{\text{sine}} > v_{\text{tri}} \quad (v_{\text{o}} = +V_{\text{dc}})$

S_3 and S_4 are on when $v_{\text{sine}} < v_{\text{tri.}} \quad (v_{\text{o}} = -V_{\text{dc}})$

Figura 3. Abrir y cerrar de switches.

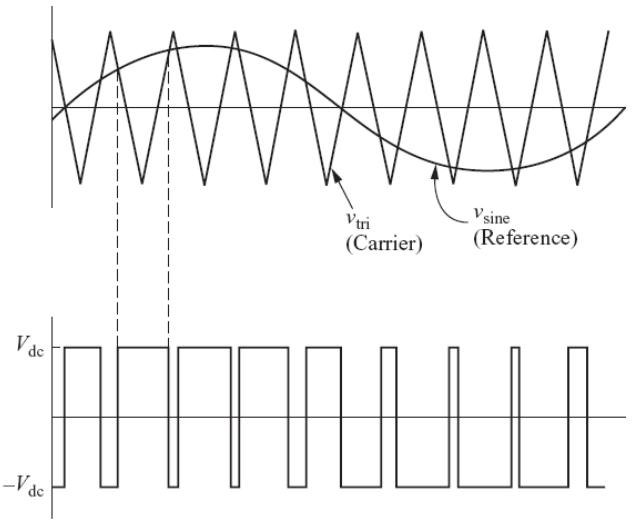

Figura 4. Operación grafica de SPWM bipolar.

Para poder realizar este tipo de modulación, como, por ejemplo, controlar la magnitud de la señal de referencia, y la frecuencia de la portadora, se tienen los siguientes parámetros:

El índice de modulación de frecuencia es la razón entre la frecuencia portadora y la frecuencia de referencia:

$$m_f = \frac{f_{portadora}}{f_{referencia}} = \frac{f_{triangular}}{f_{senoidal}}$$

Este índice indica, multiplicado con la frecuencia de la portadora, a que frecuencia se encuentran los armónicos, y múltiplos de estos, cuando la conmutación es bipolar. Los armónicos que aparecen se pueden ver en la siguiente grafica normalizada:

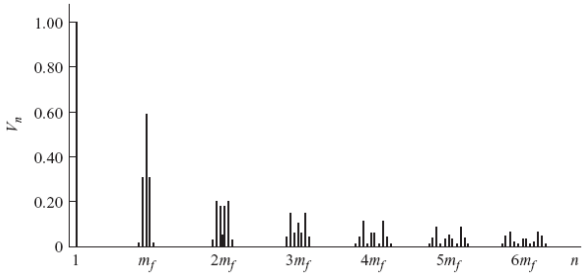

Figura 5. Armónicos de modulación bipolar.

El índice de modulación de amplitud es la razón entre el voltaje de referencia y el voltaje de la portadora:

$$m_a = \frac{V_{m,referencia}}{V_{m,portadora}} = \frac{V_{m,senoidal}}{V_{m,triangular}}$$

Este índice ayuda a tener una relación entre el voltaje de entrada y el voltaje de salida. En este caso, con un circuito de puente completo, y con $m_a \leq 1$, se tiene la siguiente relación lineal:

$$V_1 = m_a \cdot V_{dc}$$

$$V_1 = V_0 \cdot \sqrt{2}$$

En este proyecto vamos a trabajar siempre con $m_a \leq 1$, que significa que no hay una sobre modulación, y, por lo tanto, hay una relación lineal entre el voltaje de carga y de salida. Si este no fuera el caso, no habría una relación lineal, y se tendría la siguiente relación gráfica:

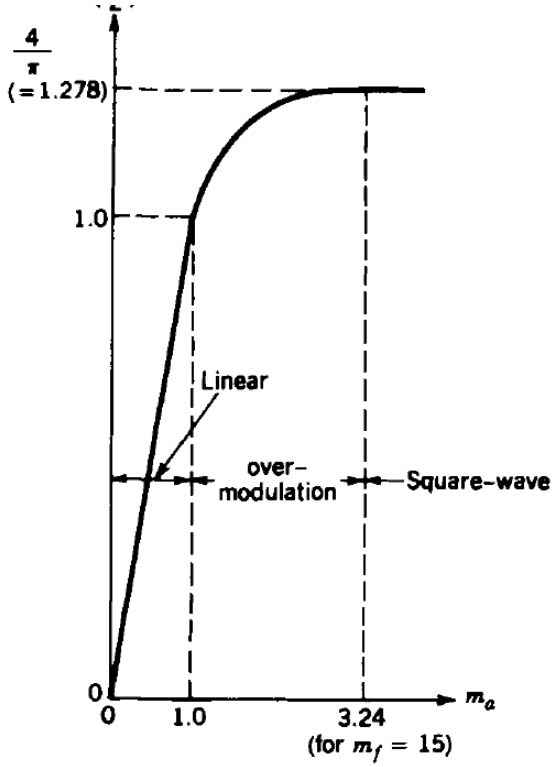

Figura 6. Rango de modulación, sobre modulación.

Ahora, con los requerimientos ya entregados, e impuestos por nosotros, los cuales son los siguientes:

Circuito de potencia	Puente completo con transformador
Transistor	IGBT
Frecuencia portadora	25 [kHz]
Carga	Resistiva
Potencia Carga	100 [W]
THDv	<5%
THDi	<10%

Se tiene los siguientes índices de modulación de frecuencia y de amplitud:

$$m_f = \frac{f_{triangular}}{f_{senoidal}} = \frac{25000\ [Hz]}{50\ [Hz]} = 500$$

$$m_a = \frac{V_{tr}}{V_{cc}} = \frac{23\ [V]}{23\ [V]} = 1$$

Entonces, se espera que la primera armónica se encuentre en la armónica de $500 \cdot 50\ [Hz] = 25000$ [Hz], además de los que se aparecen en su alrededor.

FILTRO

Para que no aparezcan esta armónicos, y sus múltiplos, se calcula un filtro para que solamente pase la armónica fundamental de la señal de salida, la cual va a ser sinusoidal, o muy cercana a esta. Para esto, se tiene que considerar en el filtro la resistencia de carga, con lo que se tiene la siguiente imagen:

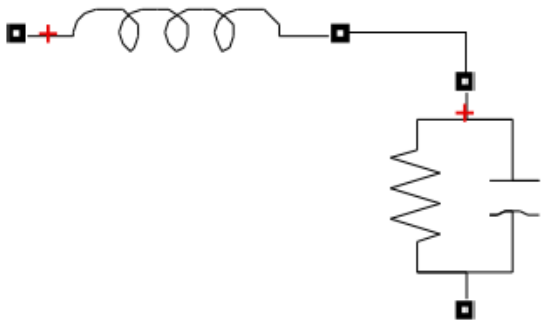

Figura 7. Filtro LC con carga resistiva.

Se considero, por simpleza de diseño, un filtro LC pasa bajo. La función de transferencia de esta, incluida la carga, es la siguiente:

$$F_{filtro}(s) = \frac{\frac{1}{LC}}{s^2 + \frac{s}{RC} + \frac{1}{LC}}$$

Ahora, para poder calcular correctamente los valores del filtro, y de la resistencia de carga, con las especificaciones del proyecto, se tiene que considerar ciertas cosas:

- Que el filtro corte las armónicas superiores a la frecuencia fundamental (50 [Hz])
- Que el filtro no afecte la ganancia de la frecuencia fundamental.

Entonces, tomando esto en consideración, se tiene, además, lo siguiente: la frecuencia natural del filtro debería estar alejada a lo menos una década antes de la frecuencia que se quiere atenuar. En este caso, la frecuencia que queremos atenuar es de 25000 [Hz]. Entonces, la frecuencia natural ser la siguiente:

$$f_{natural} \leq \frac{f_{tri}}{10} = \frac{25\ [kHz]}{10} = 2.5\ [kHz]$$

Ahora, según la función de transferencia, que ya está normalizada, se puede encontrar una equivalencia entre los parámetros del filtro (C, L y la resistencia de carga), y el factor Q y la frecuencia natural, las cuales son las siguientes:

$$2 \cdot \pi \cdot f_{natural} = \omega_0 = \frac{1}{\sqrt{LC}}$$

$$Q = R\sqrt{\frac{C}{L}}$$

Ahora, para poder obtener el valor de R, se tiene lo siguiente:

$$R = \frac{V_{out}^2}{P}$$

Ahora, para que nosotros tengamos una salida de 220 [V], tenemos que utilizar un transformador. Para nuestro proyecto, no ahondamos tan profundamente en esto, en referente a la simulación, más que nada la razón de vueltas que debe tener el transformador, y verificar que trabaje a 50 [Hz].

En este caso, la razón de vueltas que se consigue es la siguiente:

$$n = \frac{V_{out}}{V_{cc}} = \frac{220 \cdot \sqrt{2}\ [V]}{23\ [V]} = 11.1117 \approx 11$$

Este valor de numero de vueltas va a ser usado más adelante.

Ya que queremos que en la salida haya 220 [Vrms], con una potencia de P = 100 [W], se tiene el siguiente valor de resistencia:

$$R = \frac{(220\ [V])^2}{100\ [W]} = 484\ [\Omega]$$

Entonces, como se quiere que en la frecuencia de corte baje 3[dB], entonces se tiene que:

$$Q = \frac{1}{\sqrt{2}}$$

Con esto, se puede calcular los valores de L y C resolviendo el siguiente sistema de ecuaciones:

$$2 \cdot \pi \cdot f_{natural} = \frac{1}{\sqrt{LC}}$$

$$Q = R\sqrt{\frac{C}{L}}$$

Lo cual, resolviéndolo de manera simbólica, entrega lo siguiente:

$$C = \frac{Q}{R \cdot 2 \cdot \pi \cdot f_{natural}}$$

$$L = \frac{R}{Q \cdot 2 \cdot \pi \cdot f_{natural}}$$

Reemplazando valores, se obtiene lo siguiente:

$$C = 98\ [nF]$$

$$L = 43.6\ [mH]$$

Y, a través del programa MATLAB, se puede ver el bode de la función de transferencia:

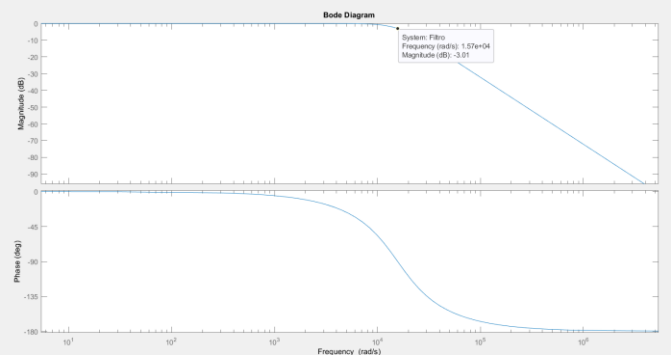

Figura 8. Bode filtro.

THD

Ahora, para poder verificar que, de manera teórica, sí funciona el filtro, vamos a calcular el THDv y el THDi de la señal de salida.

Se define, en nuestro caso, el THDv de la siguiente manera (de manera general):

$$\mathrm{THD_F} = \frac{\sqrt{V_2^2 + V_3^2 + V_4^2 + \cdots}}{V_1}$$

Figura 9. Formula THD.

Con V1 siendo el valor del voltaje en la frecuencia fundamental, y V2, V3, V4, …, los valores de los voltajes de los armónicos de la fundamental.

En nuestro caso, para el THDv, vamos a calcular el THDv hasta la primera armónica, ya que las demás armónicas, en magnitud, son más pequeñas, y considerando solamente la primera armónica, y las armónicas alrededor de esta, se puede considerar un cálculo aproximado aceptable.

A partir de esto, se necesita conseguir los valores de los voltajes a la primera armónica, que sería en una frecuencia de 250000 [Hz]. Esto se puede conseguir a través de la función de transferencia, y con la ayuda de la siguiente tabla [1]:

Table 8-3 Normalized Fourier Coefficients V_n/V_{dc} for Bipolar PWM

	m_a=1	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
n=1	1.00	0.90	0.80	0.70	0.60	0.50	0.40	0.30	0.20	0.10
n=m_f	0.60	0.71	0.82	0.92	1.01	1.08	1.15	1.20	1.24	1.27
n=mf±2	0.32	0.27	0.22	0.17	0.13	0.09	0.06	0.03	0.02	0.00

Figura 10. Magnitud de armónicos de los armónicos.

Lo cual sería equivalente a lo siguiente:

$$THD_v = \frac{\sqrt{\left(\frac{V_{498}}{\sqrt{2}}\right)^2 + \left(\frac{V_{500}}{\sqrt{2}}\right)^2 + \left(\frac{V_{502}}{\sqrt{2}}\right)^2}}{V_1}$$

Para poder calcular esto, se tiene que saber la ganancia de la función de transferencia del filtro. Para esto, se hace lo siguiente:

$$|F_{filtro}(s = j\omega)| = \left|\frac{\frac{1}{LC}}{(j\omega)^2 + \frac{j\omega}{RC} + \frac{1}{LC}}\right| \Longrightarrow$$

$$G(\omega) = |F_{filtro}(j\omega)| = \frac{R}{\sqrt{R^2(C \cdot L \cdot \omega^2 - 1)^2 + L^2 \cdot \omega^2}}$$

Entonces, con esto ya calculado, se calcula el THDv, tomando en cuenta el valor de n, que es la razón de vueltas del transformador, para el cálculo:

$$THDv = \frac{\sqrt{\left(\frac{0.32 \cdot G(2 \cdot \pi \cdot 50 \cdot 498) \cdot n \cdot V_{cc}}{\sqrt{2}}\right)^2 + \left(\frac{0.6 \cdot G(2 \cdot \pi \cdot 50 \cdot 500) \cdot n \cdot V_{cc}}{\sqrt{2}}\right)^2 + \left(\frac{0.32 \cdot G(2 \cdot \pi \cdot 50 \cdot 502) \cdot n \cdot V_{cc}}{\sqrt{2}}\right)^2}}{\frac{n \cdot V_{cc}}{\sqrt{2}}}$$

$$\rightarrow THD_v = 0.007515 = 0.7515\%$$

Lo cual cumple con el requerimiento del proyecto.

Para poder calcular el THDi, se tiene que calcular la corriente que pasa en la carga, la cual tiene distintos valores según el valor de frecuencia que se quiera tener. Pero para poder calcular esto, hay que primero calcular la corriente total que pasa en el filtro y en la resistencia de carga, para poder realizar un divisor de corriente y saber la corriente de la carga. A partir de esto, se puede obtener el THDi correspondiente.

Entonces, con el divisor de corriente, se tiene lo siguiente:

$$I_{carga} = \frac{Z_{total}}{Z_{carga}} \cdot I_{total}$$

Y se tiene que:

$$I_{total} = \frac{V_n}{Z_{total}}$$

Con Vn el voltaje de salida a cierta frecuencia. Entonces, con esto en cuenta, y además considerando que no necesitamos la fase de la corriente, solamente su magnitud, se tiene que la corriente de carga es:

$$I_{carga} = \left|\frac{V_n}{Z_{carga}}\right|, \qquad Z_{carga} = R$$

Entonces, calculando los valores necesarios, se tiene lo siguiente:

$$I_1 = \frac{n \cdot V_{cc}}{R} = 0.\overline{63}\ [A] \approx 0.63636\ [A]$$

$$I_{498} = \frac{0.32 \cdot G(2 \cdot \pi \cdot 50 \cdot 498) \cdot n \cdot V_{cc}}{R} \approx 0.00205\ [A]$$

$$I_{500} = \frac{0.6 \cdot G(2 \cdot \pi \cdot 50 \cdot 500) \cdot n \cdot V_{cc}}{R} \approx 0.00382\ [A]$$

$$I_{502} = \frac{0.32 \cdot G(2 \cdot \pi \cdot 50 \cdot 502) \cdot n \cdot V_{cc}}{R} \approx 0.00202\ [A]$$

Entonces, con los valores ya calculados, se procede a calcular el THDi:

$$THD_i = \frac{\sqrt{\left(\frac{I_{498}}{\sqrt{2}}\right)^2 + \left(\frac{I_{500}}{\sqrt{2}}\right)^2 + \left(\frac{I_{502}}{\sqrt{2}}\right)^2}}{\frac{I_1}{\sqrt{2}}}$$

$$\rightarrow THD_i = 0.007515 = 0.7515\%$$

Entonces, con el valor calculado de THDi, se cumple la especificación del proyecto, la cual era un THDi menor a 10%.

SIMULACIÓN CIRCUITO

Se realizo la simulación del circuito de potencia en el programa Simulink, más que nada por su simpleza en el análisis para el THD.

Se realizo de la siguiente manera el circuito:

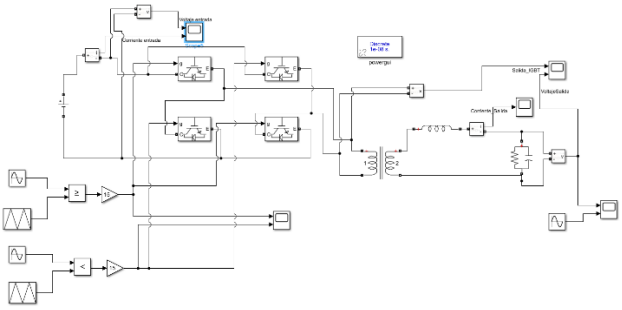

Figura 11. Circuito de simulación en Simulink.

En la parte de abajo, la cual es esta:

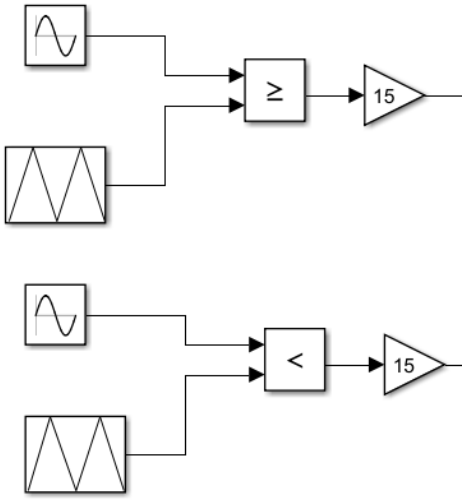

Figura 12. Circuito control de IGBT.

Es la señal que controla los gates del IGBT, a través de un control SPWM, como se explicó antes. En este caso , el primer comparador va a ser la señal la cual va los gates de los switches S1 y S2, mientras que el segundo comparador, el cual es el complemento del primer comparador, es la señal la cual va los gates de los switches S3 y S4. Estas señales están multiplicadas por una ganancia de 15 para poder activar correctamente los gates.

Lo demás del circuito de potencia no cambia de lo típico, además de que, en la carga, está el filtro y la resistencia de carga, con los valores que fueron calculados anteriormente, además de la razón de vueltas del transformador. Los valores de los componentes del circuito son los siguientes:

IGBT:

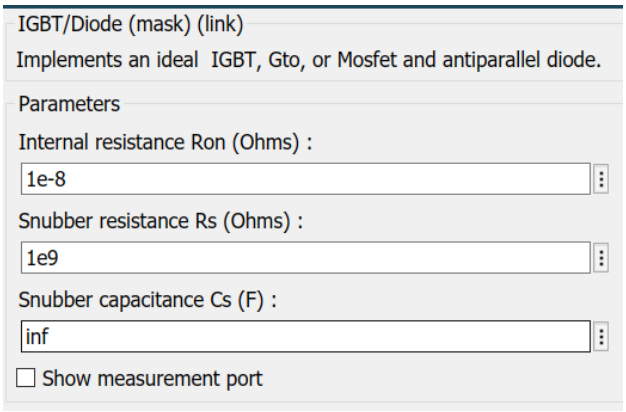

Figura 13. Parámetros IGBT.

Transformador:

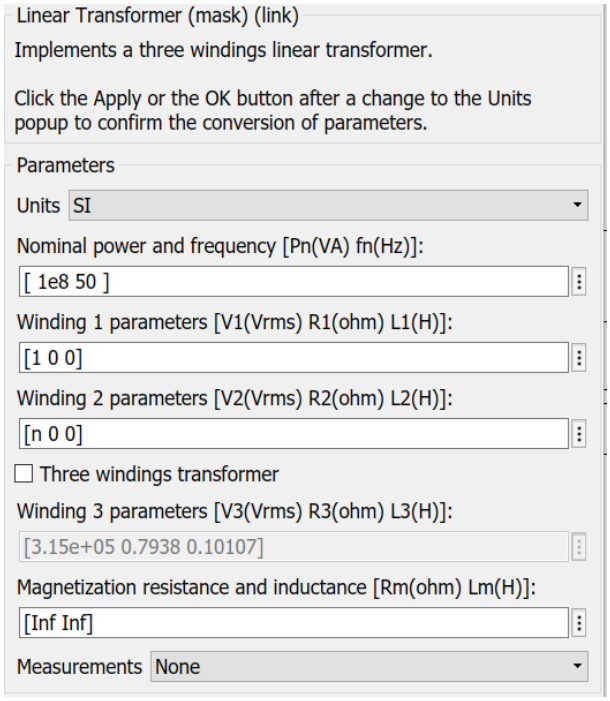

Figura 14. Parámetros Transformador.

Además de los componentes de Simulink para poder medir y mostrar gráficamente las señales, los cuales se van a ver en lo siguiente:

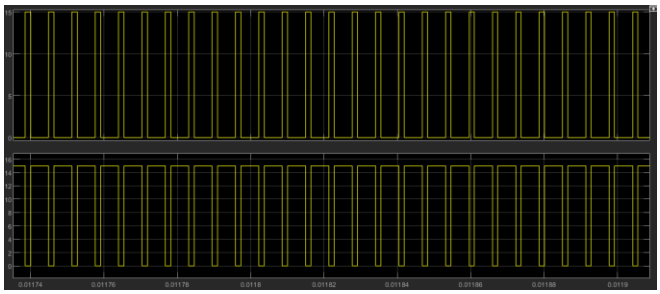

Figura 15. Señal de control y su complemento.

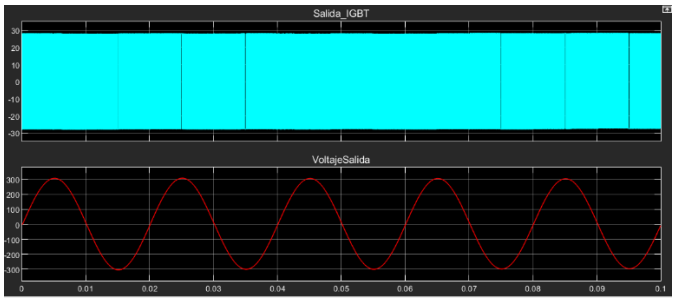

Figura 16. Voltaje de salida antes de pasar por transformador.

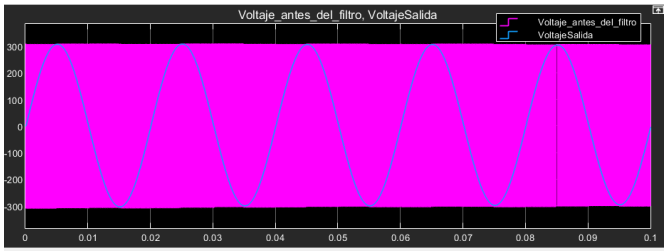

Figura 17. Voltaje de salida antes del filtro.

Aquí se puede ver las salidas de la señal de salida. En la primera gráfica, es la salida del voltaje antes de que pase por el transformador. En la segundo gráfico, es la señal después del transformador, la cual se puede ver que aumenta la magnitud de voltaje. Se nota que, en los dos gráficos, la señal antes del filtro no es una señal sinusoidal, sino una señal cuadrada, como se puede ver ampliado en el siguiente gráfico:

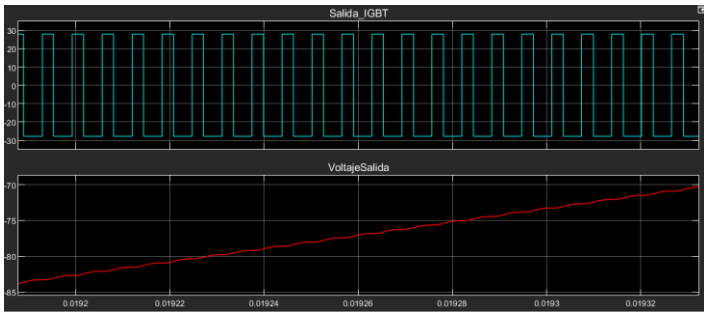

Figura 18. Grafico aumentado de la figura anterior.

La cual, después de que pase por el filtro, se convierte en una señal muy cercana a una senoidal.

En el siguiente grafico se muestra la señal de corriente después del filtro:

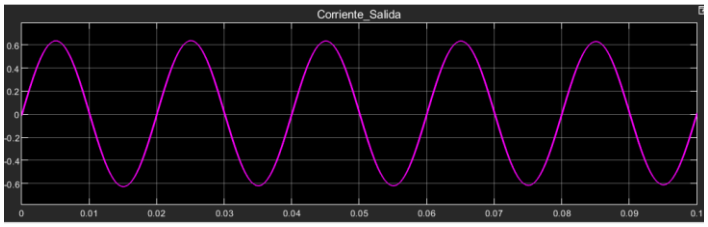

Figura 19. Voltaje de salida rectificada.

Ahora, se va a realizar un análisis de las señales para ver el THD.

THDv antes del transformador:

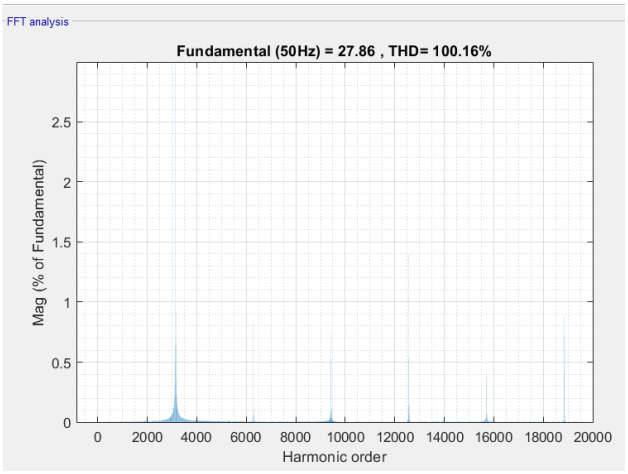

Figura 20. THDv antes del transformador.

Se puede ver las armónicas de la modulación, los cuales están alrededor de la frecuencia portadora, que, en este caso, están en los armónicos de orden de 2500, y sus múltiplos.

THDv después del transformador:

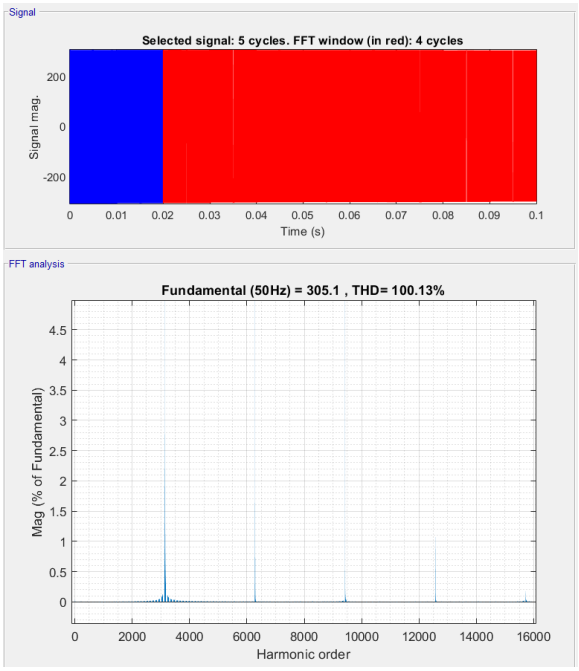

Figura 21. THDv después del transformador.

Se puede ver lo mismo que en el THDv anterior, no habiendo una clara diferencia de THDv antes del transformador y después del transformador.

THDv después del filtro:

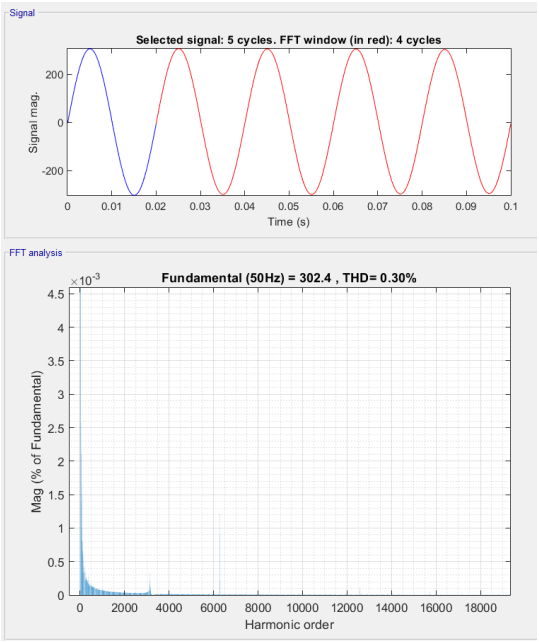

Figura 22. THDv de voltaje de salida filtrado.

Claramente se puede ver una diferencia con los THDv anteriores. Se puede ver, claramente, que el filtro realizo su trabajo de manera correcta, bajando el THDv de manera considerable, de 100% a menos de 1%. Además, aparte de unas pequeñas diferencias, se pudo ver que el THDv se aproxima a lo que se calculado de manera teórica.

THDi:

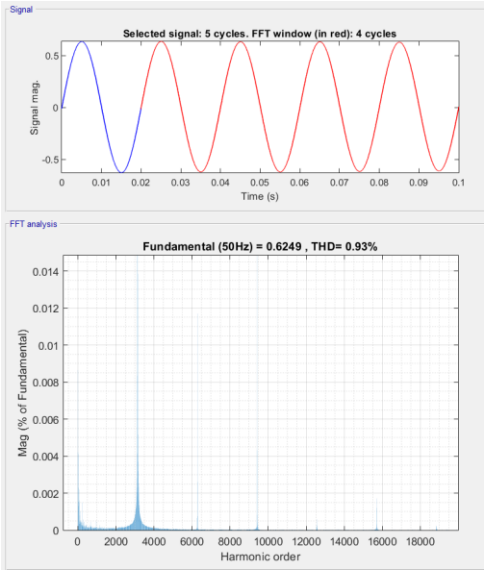

Figura 23. Corriente de salida filtrado.

De la misma manera que antes, el THDi mostrado aquí está muy cercano al calculado teóricamente, logrando así lo que se pidió en los requerimientos del proyecto.

Hay que recalcar que toda esta simulación se realizó con un Ma de 1 y un Mf de 500.

CÓDIGO MICROPIC

En lo referente al código de microPIC, se realiza unas modificaciones al código realizado en el proyecto anterior, como tener una frecuencia fija, que es la frecuencia portadora, 25 [kHz]. Una de ellas, es la consideración del tiempo muerto en los gates del IGBT.

Ya que se quiere realizar una modulación de frecuencia bipolar, los IGBT cambian de abierto y cerrado, en teoría, instantáneamente. En la realidad, hay un tiempo en el que, si uno no considera el tiempo muerto, los switches se encuentran ambos abiertos, siendo que uno debería estar cerrado y otro abierto, lo cual crea un cortocircuito, y esto es algo que se quiere evitar.

El PIC18F4550 contiene una opción para poder implementar este tiempo muerto, y así evitar el cortocircuito. Esta opción está relacionada en el siguiente registro [2]:

REGISTER 16-2: ECCP1DEL: PWM DEAD-BAND DELAY REGISTER

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PRSEN	PDC6(1)	PDC5(1)	PDC4(1)	PDC3(1)	PDC2(1)	PDC1(1)	PDC0(1)
bit 7							bit 0

Legend:			
R = Readable bit	W = Writable bit	U = Unimplemented bit, read as '0'	
-n = Value at POR	'1' = Bit is set	'0' = Bit is cleared	x = Bit is unknown

bit 7 **PRSEN:** PWM Restart Enable bit
1 = Upon auto-shutdown, the ECCPASE bit clears automatically once the shutdown event goes away; the PWM restarts automatically
0 = Upon auto-shutdown, ECCPASE must be cleared in software to restart the PWM

bit 6-0 **PDC6:PDC0:** PWM Delay Count bits(1)
Delay time, in number of FOSC/4 (4 * TOSC) cycles, between the scheduled and actual time for a PWM signal to transition to active.

Note 1: Reserved on 28-pin devices; maintain these bits clear.

Figura 24. Registro del PIC para el delay y banda muerta.

Nosotros queremos entregar un tiempo de delay de tiempo muerto lo suficientemente grande para que no pueda ocurrir este problema de cortocircuito. En nuestro caso, ya que el tiempo de delay se basa en múltiplos de $4 \cdot Tosc$, lo que es igual a 200 [ns], y el tiempo de apagado del IGBT es de 144 [ns], se tiene que, en número de tiempo de delay, toma el siguiente número de ciclos para apagarse el IGBT:

$$T_{deadband} = \frac{200\ [ns]}{144\ [ns]} = 1.3389$$

Entonces, tomando eso en cuenta, se va a configurar que el dead band, el tiempo muerto entre cada cambio, sea de 4 ciclos de delay.

Para poder aplicar que los duty cicle del PWM vayan cambiando constantemente, para emular la modulación SPWM, se guardan los vectores de duty cicle para cada modulación de amplitud determinada. Estos vectores, en el código, son vectores de una fila y muchas columnas, las cuales, el programa, cada cierto periodo, va recorriendo de en uno en uno cada valor del vector, lo cual, a la vez, ese valor de vector se lleva al registro correspondiente para cambiar el duty cicle.

Para poder conseguir los valores del vector se utilizó el siguiente programa: Smart_Sine.

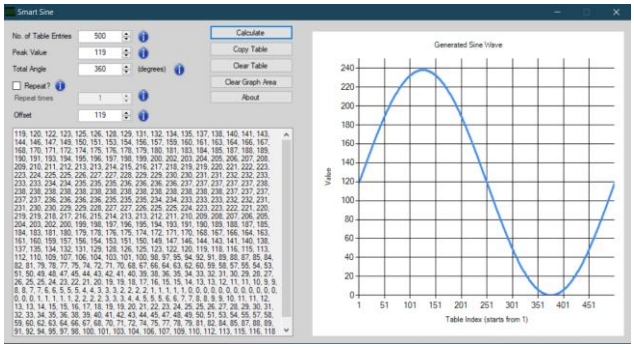

Figura 25. Imagen programa Smart_Sine.

Se tiene que, por los valores de que puede tomar el duty en nuestro código, por el valor de PR2 de 119, el duty tiene un rango de 0 a 199 para un duty de 0 a 100. Por esto, se puso un valor peak de 119, con un desfase de 119, y el ángulo total de 360°. Entonces, al tener los valores del vector, se hizo los cambios correspondientes para que esté entre 0 y 119.

Ya que el valor de duty está compuesto por 10 bits, 2 bits de DC1B, que provienen del registro CCP1CON, y 8 bits del registro CCPR1L, se tienen que separar el valor obtenido del duty en dos bytes, para poder cambiar los registros correspondientes. Para esto, se utilizó el siguiente código de Matlab:

```
mascara = uint16(3);

Tpwm = 1/25000;

Tmr2 = 4;

Tosc = 1/(48*10^(6));

PR2 = (Tpwm/(4*(Tosc)*(Tmr2)))-1;
```

```
%Ma = 0.1
Duty_1 = (1/(119*2*100)).*[11900, 11915, 11930, 11945, 11960, 11975, 11990, 12005, 12019, 12034,
%Ma = 0.15
Duty_2 = (1/(119*2*100)).*[11900, 11922, 11945, 11967, 11990, 12012, 12034, 12057, 12079, 12101,
%Ma = 0.2
Duty_3 = (1/(119*2*100)).*[11900, 11930, 11960, 11990, 12020, 12049, 12079, 12109, 12139, 12169,
%Ma = 0.25
Duty_4 = (1/(119*2*100)).*[11900, 11937, 11975, 12012, 12049, 12087, 12124, 12161, 12199, 12236,
%Ma = 0.3
Duty_5 = (1/(119*2*100)).*[11900, 11945, 11990, 12035, 12079, 12124, 12169, 12214, 12258, 12303,
%Ma = 0.35
Duty_6 = (1/(119*2*100)).*[11900, 11952, 12005, 12057, 12109, 12162, 12214, 12266, 12318, 12370,
%Ma = 0.4
Duty_7 = (1/(119*2*100)).*[11900, 11960, 12020, 12079, 12139, 12199, 12259, 12318, 12378, 12437,
```

Figura 26. Vectores de duty.

```
%Ma = 0.45
Duty_8 = (1/(119*2*100)).*[11900, 11967, 12034, 12101, 12168, 12235, 12302, 12369, 12435, 12502,
%Ma = 0.5
Duty_9 = (1/(119*2*100)).*[11900, 11975, 12050, 12124, 12199, 12274, 12348, 12423, 12497, 12571,
%Ma = 0.55
Duty_10 = (1/(119*2*100)).*[11900, 11982, 12064, 12147, 12229, 12311, 12393, 12475, 12557, 12639,
%Ma = 0.6
Duty_11 = (1/(119*2*100)).*[11900, 11990, 12079, 12169, 12259, 12348, 12438, 12527, 12617, 12706,
%Ma = 0.65
Duty_12 = (1/(119*2*100)).*[11900, 11997, 12094, 12192, 12289, 12386, 12483, 12580, 12676, 12773,
%Ma = 0.7
Duty_13 = (1/(119*2*100)).*[11900, 12005, 12109, 12214, 12319, 12423, 12527, 12632, 12736, 12840,
%Ma = 0.75
Duty_14 = (1/(119*2*100)).*[11900, 12012, 12124, 12236, 12348, 12460, 12572, 12684, 12796, 12907,
```

Figura 27. Vectores de duty.

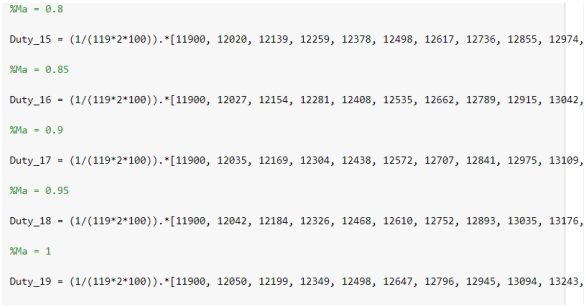

Figura 28. Vectores de duty.

```
%%Transformacion Duty

%Duty con ma = 0.1
Duty1_bits_totales =
uint16(4.*(PR2+1).*(round(Duty_1, 8)));

Duty1_2LSB = uint8(bitand(Duty1_bits_totales,
mascara));
Duty1_8MSB = uint8(bitsrl(Duty1_bits_totales, 2));

%Duty con ma = 0.15
Duty2_bits_totales =
uint16(4.*(PR2+1).*(round(Duty_2, 8)));

Duty2_2LSB = uint8(bitand(Duty2_bits_totales,
mascara));
Duty2_8MSB = uint8(bitsrl(Duty2_bits_totales, 2));

%Duty con ma = 0.2
Duty3_bits_totales =
uint16(4.*(PR2+1).*(round(Duty_3, 8)));

Duty3_2LSB = uint8(bitand(Duty3_bits_totales,
mascara));
Duty3_8MSB = uint8(bitsrl(Duty3_bits_totales, 2));

%Duty con ma = 0.25
Duty4_bits_totales =
uint16(4.*(PR2+1).*(round(Duty_4, 8)));

Duty4_2LSB = uint8(bitand(Duty4_bits_totales,
mascara));
Duty4_8MSB = uint8(bitsrl(Duty4_bits_totales, 2));

%Duty con ma = 0.3
Duty5_bits_totales =
uint16(4.*(PR2+1).*(round(Duty_5, 8)));

Duty5_2LSB = uint8(bitand(Duty5_bits_totales,
mascara));
Duty5_8MSB = uint8(bitsrl(Duty5_bits_totales, 2));

%Duty con ma = 0.35
Duty6_bits_totales =
uint16(4.*(PR2+1).*(round(Duty_6, 8)));

Duty6_2LSB = uint8(bitand(Duty6_bits_totales,
mascara));
Duty6_8MSB = uint8(bitsrl(Duty6_bits_totales, 2));

%Duty con ma = 0.4
Duty7_bits_totales =
uint16(4.*(PR2+1).*(round(Duty_7, 8)));

Duty7_2LSB = uint8(bitand(Duty7_bits_totales,
mascara));
Duty7_8MSB = uint8(bitsrl(Duty7_bits_totales, 2));

%Duty con ma = 0.45
Duty8_bits_totales =
uint16(4.*(PR2+1).*(round(Duty_8, 8)));

Duty8_2LSB = uint8(bitand(Duty8_bits_totales,
mascara));
Duty8_8MSB = uint8(bitsrl(Duty8_bits_totales, 2));

%Duty con ma = 0.5
Duty9_bits_totales =
uint16(4.*(PR2+1).*(round(Duty_9, 8)));

Duty9_2LSB = uint8(bitand(Duty9_bits_totales,
mascara));
Duty9_8MSB = uint8(bitsrl(Duty9_bits_totales, 2));

%Duty con ma = 0.55
Duty10_bits_totales =
uint16(4.*(PR2+1).*(round(Duty_10, 8)));

Duty10_2LSB = uint8(bitand(Duty10_bits_totales,
mascara));
Duty10_8MSB = uint8(bitsrl(Duty10_bits_totales, 2));

%Duty con ma = 0.6
Duty11_bits_totales =
uint16(4.*(PR2+1).*(round(Duty_11, 8)));

Duty11_2LSB = uint8(bitand(Duty11_bits_totales,
mascara));
Duty11_8MSB = uint8(bitsrl(Duty11_bits_totales, 2));
```

```
%Duty con ma = 0.65
Duty12_bits_totales =
uint16(4.*(PR2+1).*(round(Duty_12, 8)));

Duty12_2LSB = uint8(bitand(Duty12_bits_totales,
mascara));
Duty12_8MSB = uint8(bitsrl(Duty12_bits_totales, 2));

%Duty con ma = 0.7
Duty13_bits_totales =
uint16(4.*(PR2+1).*(round(Duty_13, 8)));

Duty13_2LSB = uint8(bitand(Duty13_bits_totales,
mascara));
Duty13_8MSB = uint8(bitsrl(Duty13_bits_totales, 2));

%Duty con ma = 0.75
Duty14_bits_totales =
uint16(4.*(PR2+1).*(round(Duty_14, 8)));

Duty14_2LSB = uint8(bitand(Duty14_bits_totales,
mascara));
Duty14_8MSB = uint8(bitsrl(Duty14_bits_totales, 2));

%Duty con ma = 0.8
Duty15_bits_totales =
uint16(4.*(PR2+1).*(round(Duty_15, 8)));

Duty15_2LSB = uint8(bitand(Duty15_bits_totales,
mascara));
Duty15_8MSB = uint8(bitsrl(Duty15_bits_totales, 2));

%Duty con ma = 0.85
Duty16_bits_totales =
uint16(4.*(PR2+1).*(round(Duty_16, 8)));

Duty16_2LSB = uint8(bitand(Duty16_bits_totales,
mascara));
Duty16_8MSB = uint8(bitsrl(Duty16_bits_totales, 2));

%Duty con ma = 0.9
Duty17_bits_totales =
uint16(4.*(PR2+1).*(round(Duty_17, 8)));

Duty17_2LSB = uint8(bitand(Duty17_bits_totales,
mascara));
Duty17_8MSB = uint8(bitsrl(Duty17_bits_totales, 2));

%Duty con ma = 0.95
Duty18_bits_totales =
uint16(4.*(PR2+1).*(round(Duty_18, 8)));

Duty18_2LSB = uint8(bitand(Duty18_bits_totales,
mascara));
Duty18_8MSB = uint8(bitsrl(Duty18_bits_totales, 2));

%Duty con ma = 1.0
Duty19_bits_totales =
uint16(4.*(PR2+1).*(round(Duty_19, 8)));

Duty19_2LSB = uint8(bitand(Duty19_bits_totales,
mascara));
Duty19_8MSB = uint8(bitsrl(Duty19_bits_totales, 2));
```

El código en MPLAB, y el cual fue integrado al PIC, es el siguiente:

```
#include <xc.h> //librerias
#include <stdint.h>
#include <stdio.h>
#include <stdlib.h>
#include "config.h"

// CONFIG1L
#pragma config PLLDIV = 5       // P
#pragma config CPUDIV = OSC1_PLL2//
#pragma config USBDIV = 2       // U

// CONFIG1H
#pragma config FOSC = HSPLL_HS  // O
#pragma config FCMEN = OFF      // F
#pragma config IESO = OFF       // I

// CONFIG2L
#pragma config PWRT = OFF       // P
#pragma config BOR = ON         // B
#pragma config BORV = 3         // B
#pragma config VREGEN = ON      // U
```

Figura 29. Bits de configuración.

```
// CONFIG2H
#pragma config WDT = ON
#pragma config WDTPS = 32768

// CONFIG3H
#pragma config CCP2MX = ON
#pragma config PBADEN = OFF
#pragma config LPT1OSC = ON
#pragma config MCLRE = ON

// CONFIG4L
#pragma config STVREN = ON
#pragma config LVP = ON
#pragma config ICPRT = OFF
#pragma config XINST = OFF
```

Figura 30. Bits de configuración.

```
// CONFIG5L
#pragma config CP0 = OFF
#pragma config CP1 = OFF
#pragma config CP2 = OFF
#pragma config CP3 = OFF

// CONFIG5H
#pragma config CPB = OFF
#pragma config CPD = OFF

// CONFIG6L
#pragma config WRT0 = OFF
#pragma config WRT1 = OFF
#pragma config WRT2 = OFF
#pragma config WRT3 = OFF
```

Figura 31. Bits de configuración.

```
// CONFIG7L
#pragma config EBTR0 = OFF
#pragma config EBTR1 = OFF
#pragma config EBTR2 = OFF
#pragma config EBTR3 = OFF

// CONFIG7H
#pragma config EBTRB = OFF
```

Figura 32. Bits de configuración.

En las imágenes anteriores del código, se muestran los bits de configuración.

```
void Init_PWM() //inicia el modulo PWM
{
    PR2 = 119;           // frecuencia
    CCPR1L = 43;         // duty cycle
    TRISCbits.RC2 = 0;          // pin RC2 como salida, señal principal, no modulada
    TRISDbits.RD5 = 0;          // pin RD5 como salida, señal secundaria, modulada
    CCP1CONbits.CCP1M = 0b1100; // modo PWM
    CCP1CONbits.P1M = 0b10;
    T2CONbits.TMR2ON = 1;       // enable timer 2
    T2CONbits.T2CKPS = 0b01;    //  prescale = 4
    ECCP1DEL = 0x04;
}
```

Figura 33. Función Init_PWM().

Aquí es la función Init_PWM(), la cual proviene del proyecto anterior, la cual inicia y configura los bits para que empiece el PWM. En este proyecto, se hizo una pequeña modificación para que se pueda trabajar como medio puente, y configurar los pines de salida.

```
void Init_USART() //Se inicia el modulo USART
{
    //Baud rate
    SPBRG = 25; // 77 -> 9600; 25 -> 115200
    BAUDCON= 0;
    // salida y entrada
    TRISCbits.RC6 = 0; // Tx
    TRISCbits.RC7 = 1; // Rx

    // setup PIC
    TXSTAbits.BRGH = 1; // 0 low speed , 1 high speed
    TXSTAbits.SYNC = 0; // 0 asincrono, 1 sincrono
    RCSTAbits.SPEN = 1; // habilitar Tx y Rx
    RCIE = 1;           // interrupcion serial

    // activar transmision y recepcion
    TXSTAbits.TX9 =  0;      // 8 bits
    TXSTAbits.TXEN = 1;     // activar transmision
    RCSTAbits.RX9 =  0;      // 8 bits
    RCSTAbits.CREN = 1;     // activar recepcion
```

Figura 34. Función Init_USART().

Esta función, Init_USART(), es para inicializar la comunicación serial, con su respectiva configuración de los puertos y el baud rate.

```
RCSTAbits.SREN = 0;
RCSTAbits.ADDEN =0;
RCSTAbits.FERR = 0;
RCSTAbits.OERR = 0;
RCSTAbits.RX9D = 0;
TXSTAbits.SENDB =1;
TXSTAbits.TRMT = 1;
TXSTAbits.TX9D = 0;
TXSTAbits.CSRC = 0;
TXSTAbits.TRMT = 1;
```

Figura 35. Bits de configuración de registro de la comunicación serial.

```
void main(void) //funcion general
{
    Init_PWM(); //llama a funcion de inicion PWM
    Init_USART(); //llama a funcion de USART

    while (1){ //while para recibir constantemente (loop)
        int s = 0;
        uint8_t ma;
        int i;
        if(RCIF){
            uint8_t dato = RCREG;
            RCIF = 0;
            int j = 0;
            while (j<20){
            for(i = 0;i<= 500;i++){
                switch(dato){
                    case 10:
                        CCPR1L = vector1_8MSB[i];
                        CCP1CONbits.DC1B = vector1_2LSB[i];
                        __delay_ms(4);
                        break;
                    case 20:
                        CCPR1L = vector2_8MSB[i];
                        CCP1CONbits.DC1B = vector2_2LSB[i];
                        __delay_ms(4);
                        break;
                    case 30:
                        CCPR1L = vector3_8MSB[i];
                        CCP1CONbits.DC1B = vector3_2LSB[i];
                        __delay_ms(4);
                        break;
```

Figura 36. Función principal.

Aquí, en la función principal, se inicia las funciones ya antes mencionadas y después un loop infinito, para que pueda siempre estar leyendo el pic si es que llega alguna comunicación serial o no.

Después de eso, cuando ya se leyó algún carácter, se verifica si es algún carácter que se encuentre dentro de los casos 10, 20, 30, …, 190. Si es así, se procede a cambiar el duty, el cual lee algunos de los vectores de duty, los cuales se diferencian la modulación de ancho. El duty, como fue explicado anteriormente, es cambiado constantemente, en un ciclo for, pasando por cada valor del vector, hasta que se reciba algún nuevo carácter.

```
case 40:
    CCPR1L = vector4_8MSB[i];
    CCP1CONbits.DC1B = vector4_2LSB[i];
    __delay_ms(4);
    break;
case 50:
    CCPR1L = vector5_8MSB[i];
    CCP1CONbits.DC1B = vector5_2LSB[i];
    __delay_ms(4);
    break;
case 60:
    CCPR1L = vector6_8MSB[i];
    CCP1CONbits.DC1B = vector6_2LSB[i];
    __delay_ms(4);
    break;
case 70:
    CCPR1L = vector7_8MSB[i];
    CCP1CONbits.DC1B = vector7_2LSB[i];
    __delay_ms(4);
    break;
case 80:
    CCPR1L = vector8_8MSB[i];
    CCP1CONbits.DC1B = vector8_2LSB[i];
    __delay_ms(4);
    break;
case 90:
    CCPR1L = vector9_8MSB[i];
    CCP1CONbits.DC1B = vector9_2LSB[i];
    __delay_ms(4);
    break;
```

Figura 37. Casos para cambio de duty.

```
case 100:
    CCPR1L = vector10_8MSB[i];
    CCP1CONbits.DC1B = vector10_2LSB[i];
    __delay_ms(4);
    break;
case 110:
    CCPR1L = vector11_8MSB[i];
    CCP1CONbits.DC1B = vector11_2LSB[i];
    __delay_ms(4);
    break;
case 120:
    CCPR1L = vector12_8MSB[i];
    CCP1CONbits.DC1B = vector12_2LSB[i];
    __delay_ms(4);
    break;
case 130:
    CCPR1L = vector13_8MSB[i];
    CCP1CONbits.DC1B = vector13_2LSB[i];
    __delay_ms(4);
    break;
case 140:
    CCPR1L = vector14_8MSB[i];
    CCP1CONbits.DC1B = vector14_2LSB[i];
    __delay_ms(4);
    break;
case 150:
    CCPR1L = vector15_8MSB[i];
    CCP1CONbits.DC1B = vector15_2LSB[i];
    __delay_ms(4);
    break;
```

Figura 38. Casos para cambio de duty.

```
                case 160:
                    CCPR1L = vector16_8MSB[i];
                    CCP1CONbits.DC1B = vector16_2LSB[i];
                    __delay_ms(4);
                    break;
                case 170:
                    CCPR1L = vector17_8MSB[i];
                    CCP1CONbits.DC1B = vector17_2LSB[i];
                    __delay_ms(4);
                    break;
                case 180:
                    CCPR1L = vector18_8MSB[i];
                    CCP1CONbits.DC1B = vector18_2LSB[i];
                    __delay_ms(40);
                    break;
                case 190:
                    CCPR1L = vector19_8MSB[i];
                    CCP1CONbits.DC1B = vector19_2LSB[i];
                    __delay_ms(4);
                    break;
            }

        }j++;
    }
    }
}
```

Figura 39. Casos para cambio de duty.

INTERFAZ GRAFICA

La interfaz gráfica fue realizada a través de la aplicación de Matlab AppDesigner, la cual es parecida en términos de código al que se realizó el proyecto anterior. El código es el siguiente:

```
classdef Interfaz_Proyecto_2 < matlab.apps.AppBase

    % Properties that correspond to app components
    properties (Access = public)
        UIFigure                    matlab.ui.Figure
        onoff                       matlab.ui.control.Button
        Image                       matlab.ui.control.Image
        Grupo4RafaelBurgosJocelynMatusLabel  matlab.ui.control.Label
        UIAxes                      matlab.ui.control.UIAxes
        Slider                      matlab.ui.control.Slider
        EditField                   matlab.ui.control.EditField
        ModulaciondeamplitudLabel   matlab.ui.control.Label
        EditField_2                 matlab.ui.control.NumericEditField
        VoltajeRMSLabel             matlab.ui.control.Label
        EscribirLabel               matlab.ui.control.Label
    end

    properties (Access = private)   %% Función comunicación PIC

        Conectado;
        Flag  = hex2dec('54');
        Ma % Description
        Matriz_valores = [0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5,0.55,0.6,0.65,0.7,0.75,0.8,0.85,0.9,0.9
    end

     methods (Access = private)     %% Inicializa funciones

        function FWrites(app)

            FlagF = app.Flag;
            Maf = app.Ma;
```

Figura 40. Inicio código de interfaz gráfica. Inicio de función Fwrites(app).

Aquí empieza el código, donde sea crea un vector de valores, el cual sirve para verificar si el valor que se introdujo en la interfaz es un valor correcto, verificar si el valor está dentro de los valores permitidos de modulación de amplitud.

Además, se encuentra el inicio de la función Fwrites(app), la cual se llama más adelante para poder enviar la información al pic.

```
FlagF = app.Flag;
Maf = app.Ma;
switch Maf
    case 0.1
        valor = 10;
    case 0.15
        valor = 20;
    case 0.2
        valor = 30;
    case 0.25
        valor = 40;
    case 0.3
        valor = 50;
    case 0.35
        valor = 60;
    case 0.4
        valor = 70;
    case 0.45
        valor = 80;
    case 0.5
        valor = 90;
    case 0.55
        valor = 100;
    case 0.6
        valor = 110;
    case 0.65
        valor = 120;
    case 0.7
        valor = 130;
```

Figura 41. Casos de modulación de amplitud.

```
    case 0.75
        valor = 140;
    case 0.8
        valor = 150;
    case 0.85
        valor = 160;
    case 0.9
        valor = 170;
    case 0.95
        valor = 180;
    case 1
        valor = 190;
  end
  s = serial('COM3');     % Nombre asignado al PIC
  set(s,'BaudRate',115200);
  fopen(s);
  fwrite(s, valor, 'uint8');   %Se envia Ma
  fclose(s);

 end

end  %% Termina de declarar funciones
```

Figura 42. Casos modulación de amplitud. Fin de función Fwrites(app).

Aquí se encuentran los valores de los casos correctos de modulación de amplitud, y, además, el envió de la información serial al pic.

```
% Callbacks that handle component events
methods (Access = private)

    % Button pushed function: onoff
    function onoffButtonPushed(app, event)
        info = strcmp(app.onoff.Text,'Conectado');%string compare para poder comparar las palabras y as
        if    info== 1
            app.onoff.Text= 'Desconectado';
            app.Conectado= 0;

        elseif info == 0
            app.onoff.Text= 'Conectado';
            app.Conectado=1;
        end
    end
```

Figura 43. Función interacción interfaz-usuario botón de conectado/desconectado.

```
% Value changed function: Slider
function SliderValueChanged(app, event)
    value = app.Slider.Value;
    [~, minIdx] = min(abs(value - event.Source.MinorTicks(:)));
    event.Source.Value = event.Source.MinorTicks(minIdx);
    MaE = event.Source.MinorTicks(minIdx);
    app.EditField.Value = num2str(MaE);

    if app.Conectado
        app.EscribirLabel.FontColor = 'k';
        if (app.Slider.Value >=0.1) && (app.Slider.Value <=1)
            app.Ma = MaE;
            app.EscribirLabel.Text = 'Escribir valores entre 0.1 y 1';
            t = linspace(0,1,10000);
            PlotSen = 220*MaE*sin(2*pi*t);
            plot(app.UIAxes,t,PlotSen,'r','linewidth',2);
            app.EditField_2.Value = (220*MaE);
            FWrites(app);
        else
            app.EscribirLabel.Text = 'Entregar valor entre e incluyendo 0.1 y 1';
        end
     else
        app.EscribirLabel.FontColor = 'r';
        app.EscribirLabel.Text = 'Por favor conectar el dispositivo';
    end
```

Figura 44. Función interacción interfaz-usuario slider.

```
% Value changed function: EditField
function EditFieldValueChanged(app, event)
    MaE = str2double(app.EditField.Value);

    if app.Conectado
        app.EscribirLabel.FontColor = 'k';
        if (ismember(MaE, app.Matriz_valores))
            app.Slider.Value = MaE;
            app.Ma = MaE;
            app.EscribirLabel.Text = 'Escribir valores entre 0.1 y 1';
            t = linspace(0,1,10000);
            PlotSen = 220*MaE*sin(2*pi*t);
            plot(app.UIAxes,t,PlotSen,'r','linewidth',2);
            app.EditField_2.Value = (220*MaE);
            FWrites(app);
        else
            app.EscribirLabel.Text = 'Valor entre 0.1 y 1, y valores discretos de 0.5';
        end
    else
        app.EscribirLabel.FontColor = 'r';
        app.EscribirLabel.Text = 'Por favor conectar el dispositivo';
    end
```

Figura 45. Función interacción interfaz-usuario campo numérico.

En las imágenes anteriores son funciones modificadas de AppDesigner sobre cómo debe ser el comportamiento del grafico a partir de la interacción con este, y obtener los datos de esta interacción para la entrega de la información serial al pic.

Y aquí estaría la interfaz gráfica:

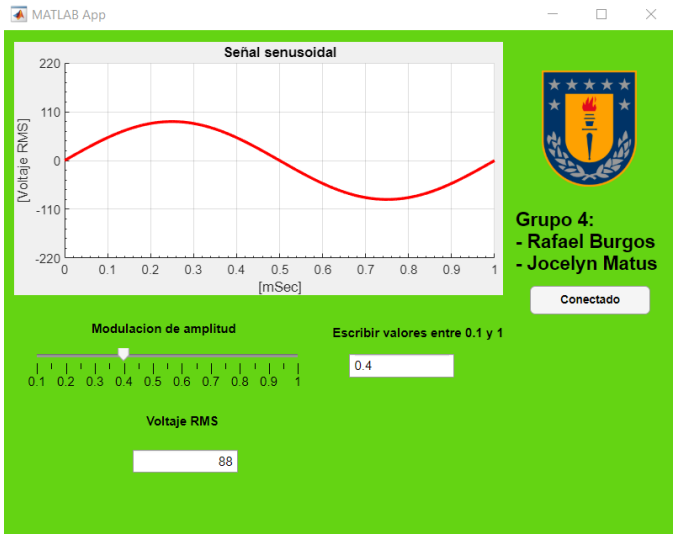

Figura 46. Interfaz gráfica en funcionamiento.

PCB

El diseño de la PCB fue el siguiente:

En primera instancia, el esquemático que se usó para la PCB es el siguiente:

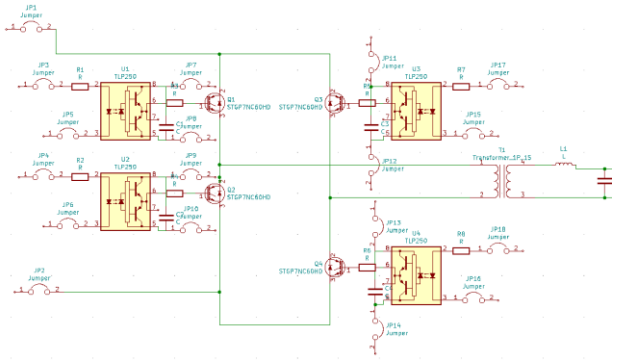

Figura 47. Esquemático PCB circuito de potencia.

Notando que se tienen puesto muchos jumpers para realizar las conexiones necesarias, como las conexiones a tierras de los optoacopladores o la entrada de voltaje.

En la siguiente imagen se ve las conexiones que se realizó en la PCB:

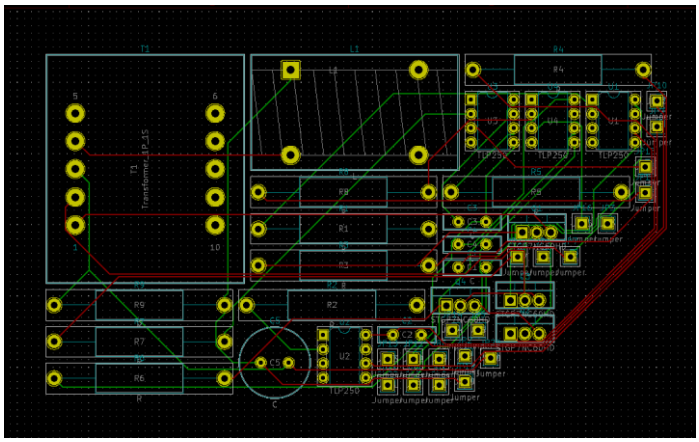

Figura 48. Conexiones placa PCB.

Y esta sería el modelo 3D de esta:

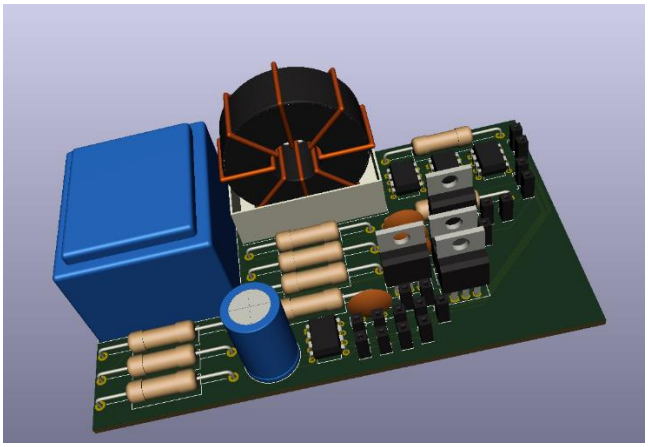

Figura 49. Modelo 3D PCB vista superior.

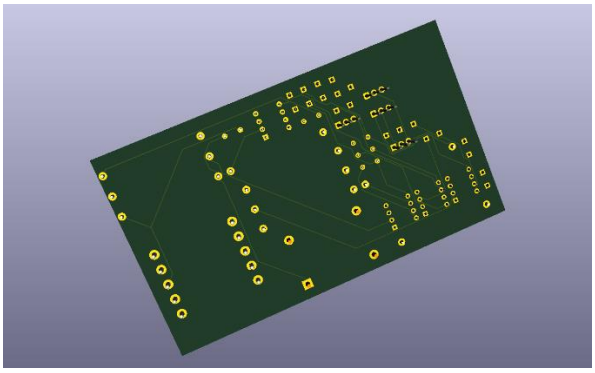

Figura 50. Modelo 3D PCB vista inferior.

CONCLUSIÓN

La realización de proyecto da a demostrar que los inversores pueden llegar a ser muy complejos de manejar, en su control y en su diseño. Para esto, hay que estar preparados de que exista la posibilidad de fracaso en los diseños iniciales. Aun así, el conocimiento entregado a través de este proyecto es valioso y puede llegar a ser un apoyo en el desarrollo como profesional más adelante.

REFERENCIAS

[1] Daniel W. Hart, P. D. (2010). *Power Electronics*. Vol 1°. New York: McGraw-Hill Education, 2011.

[2] *PIC18F2455/2550/4455/4550 Data Sheet*, Microchip, 2009. [Online]. Disponible en: https://ww1.microchip.com/downloads/en/devicedoc/39632e.pdf