Universidad de Alcalá
Escuela Politécnica Superior

GRADO DE INGENIERÍA ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Grado

Implementación en Hardware configurable de técnicas de modulación vectorial para inversores trifásicos de potencia

Autor: Alejandro Sáez Mora
Tutora: Mª Carmen Pérez Rubio

2018

UNIVERSIDAD DE ALCALÁ

Escuela Politécnica Superior

GRADO DE INGENIERÍA EN ELECTRÓNICA Y AUTOMÁTICA INDUSTRIAL

Trabajo Fin de Carrera

Implementación en Hardware configurable de técnicas de modulación vectorial para inversores trifásicos de potencia

Autor: Alejandro Sáez Mora

Director Mª Carmen Pérez Rubio

TRIBUNAL:

Presidente: Jesús Ureña Ureña

Vocal 1: Pedro Martín Sánchez

Vocal 2: Mª Carmen Pérez Rubio

FECHA: ……..

Agradecimientos

Gracias a mi familia y amigos por el apoyo recibido, Javi, Carlos, Omar, y especialmente a mi tutora Dª Mª Carmen Pérez Rubio por su implicación en el proyecto y por la paciencia durante estos casi 9 meses.

1 CONTENIDOS

2 ÍNDICE DE IMÁGENES Y TABLAS

3 ÍNDICE DE CÓDIGOS

4 RESUMEN

En términos generales, la principal tarea de la Electrónica de Potencia es la de procesar y controlar el flujo de la energía eléctrica de manera eficiente y de forma que ésta se adapte a las condiciones de una carga específica.

Atendiendo a las características de entrada y salida, se pueden establecer cuatro grandes grupos: conversión AC/DC, conversión DC/DC, conversión DC/AC y conversión AC/AC.

El siguiente Trabajo Fin de Grado (TFG), se centra en el estudio de la conversión DC/AC (inversor) mediante la técnica de modulación vectorial, (*SVM-Space Vector Modulation*), en su versión trifásica. Para ello, se desarrollará en VHDL (*Very High Speed Integrated Circuits Hardware Description Language)*, un código, que permitirá programar una FPGA (*Field Programmable Gate Array*) para controlar las señales PWM (Pulse Width Module), de la conmutación de los interruptores de potencia en una fuente Semiteach-IGBT. Finalmente, se realizarán simulaciones en Simulink-Matlab evaluando su correcto funcionamiento y una prueba real con descarga en placa y conexión de cargas de baja potencia.

Palabras claves: Inversor, FPGA, VHDL, modulación vectorial

4.1 ABSTRACT

This document is the final grade thesis of the degree "Electrónica y Automática Industrial" done by Alejandro Sáez Mora. It is a study about the implementation of a DC/AC converter on a FPGA through the PWM Space Vector Modulation. A VHDL code has been developed to manage the switches from a cube Semiteach-IGBT and a real test has been done with a low power carges. The main of the work is to serve as support for the practices classes in subjects about Power Electronics.

Keywords: Inversor, VHDL, FPGA, PWM Space Vector Modulation

5 RESUMEN EXTENDIDO

El trabajo realizado es el proyecto fin de grado del alumno Alejandro Sáez Mora, presentado en la universidad Politécnica Superior de Alcalá de Henares. Durante la experiencia en la carrera, las asignaturas tecnológicas contienen laboratorios donde se practican los conocimientos adquiridos en las clases teóricas. El campo de la ingeniería está en constante cambio y actualizarse a las nuevas tecnologías es un deber, incluyendo así a las nuevas estrategias de impartir las clases. Con este fin, nace este trabajo, que servirá como ayuda docente en las clases prácticas de asignaturas relacionadas con la electrónica de potencia. Además, forma parte de un conjunto de proyectos con el mismo objetivo donde se han implementado otro tipo de modulaciones en convertidores DC/AC con distintos códigos y diseños hardware.

El documento presente trata de la implementación en VHDL de un inversor de potencia DC/AC trifásico siguiendo la técnica de la modulación vectorial a través de la programación de una tarjeta FPGA. Ésta, es la encargada de realizar la modulación y enviar las señales moduladoras a un cubo de potencia, en el cual se han realizado los ensayos prácticos.

Los inversores tratan de convertir la corriente eléctrica de continua a alterna. Esto es logrado gracias a la conmutación de unos interruptores que actúan como puertas, dejando pasar o no la corriente, generando así señales cuadradas que pueden ser filtradas para obtener ondas senoidales.

Para llevar a cabo el control de los interruptores, se ha programado una FPGA donde son generadas las señales de control o PWM por medio de la técnica de modulación vectorial.

Además de la implementación en VHDL, el diseño fue desarrollado en Matlab con el fin de calcular los errores cometidos y de verificar el funcionamiento del inversor antes de realizar las pruebas prácticas en el laboratorio.

La información aquí presentada está contenida en cuatro grupos principales:

La memoria comienza explicando el contexto actual acerca de los convertidores, justificando el uso de las tarjetas FPGA como herramienta del diseño y referenciando trabajos consultados para la realización de este documento.

En la segunda parte se ha realizado una explicación de la modulación utilizada, así como su adaptación para la implementación hardware en el programa Matlab con el fin de poder cuantificar los errores que se han ido cometiendo al truncar los decimales.

El tercer apartado abarca las especificaciones de los dispositivos utilizados y el desarrollo completo de la implementación en VHDL realizada.

Por último, se muestran los resultados finales obtenidos en las pruebas prácticas reales en el laboratorio y los teóricos obtenidos en la herramienta Simulink de Matlab.

Para la elaboración del código se ha utilizado la aplicación ISE Project Navigator junto a ModelSim SE como simulador. Para facilitar la accesibilidad al diseño, se ha tratado de programar de forma estructural y modulada como se verá explicado la memoria.

Finalmente, el documento incluye las partes de código más relevantes generadas, un manual de usuario del hardware utilizado en las pruebas reales y las hojas de especificaciones de los dispositivos utilizados.

5.1 CONTEXTO

El TFG que aquí se presenta se encuentra dentro del proyecto de innovación docende “Proyectos para el fomento de la innovación en el proyecto de enseñanza-aprendizaje” del curso académico 2017-2018. En particular, este trabajo está contenido en el proyecto “Puesta en práctica de metodologías activas y su adaptación a planes bilingües en el área de ingeniería”. Referencia: UAH/EV914.

6 MEMORIA

6.1 INTRODUCCIÓN

6.1.1 Antecedentes

La electrónica de potencia es la rama encargada del estudio de dispositivos y circuitos dedicados al control y la conversión de la energía eléctrica. Antiguamente, estos procedimientos se realizaban con métodos electromagnéticos, sin embargo, con el avance de las tecnologías se ha buscado reducir costes y la optimización de los sistemas electrónicos.

Un convertidor DC/AC o también llamado inversor de voltaje, consiste en la transformación de tensión de corriente continua en una señal variable con frecuencia y magnitudes de salida controladas. El objetivo principal es obtener una señal senoidal para evitar introducir perturbaciones sobre las cargas y transformar energía que posteriormente no se usa. En el pasado, para generar la transformación, era necesario acoplar máquinas de corriente alterna junto a los generadores de corriente continua. En la actualidad, gracias al desarrollo de los dispositivos de potencia semiconductores y el aumento de la capacidad de computación de los procesadores digitales, se ha reducido en gran tamaño y coste, la conversión de energía.

El mecanismo empleado para hacer conmutar los dispositivos semiconductores es conocido como modulación. La técnica empleada determinará el aprovechamiento de los recursos, así como la calidad de la señal y las perdidas por conmutación.

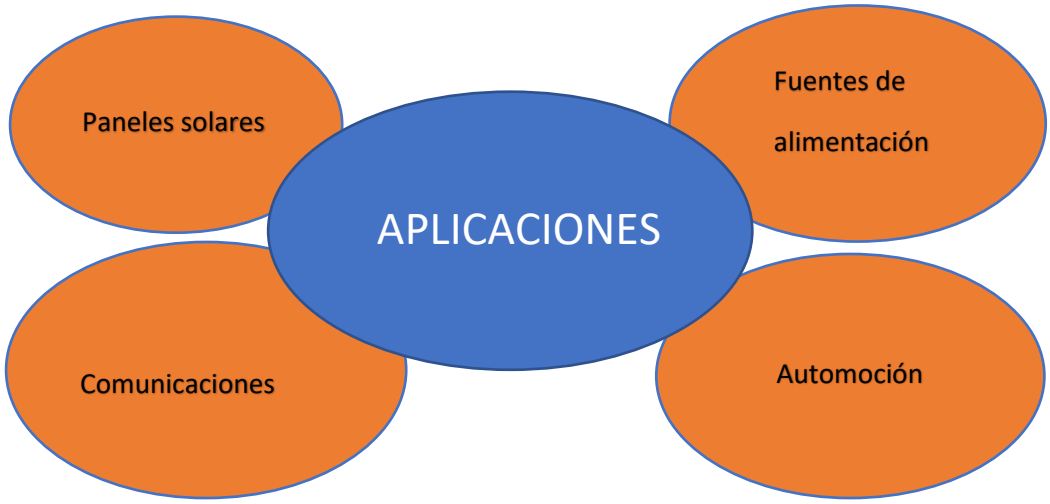


Imagen 6.1: Aplicaciones de los inversores

Las aplicaciones principales de los inversores mostradas en la imagen 6.1, van desde pequeñas fuentes de alimentación para ordenadores hasta aplicaciones industriales de alta potencia.

Todos los sistemas de almacenamiento de energía que son usados para dispositivos alternos tienen la necesidad del uso de convertidores.

En la actualidad, ya existen trabajos de carácter docente que utilizan FPGAs para el control de inversores trifásicos. especialmente, "Plataforma basada en MATLAB, FPGA Xilinx y Semiteach-IGBT para docencia en electrónica de potencia" G.Pérez, A.Garrigós, J.M.Blanes, R.Gutiérrez [1], donde mediante las herramientas System Generator o HDL Coder generan automáticamente el código en VHDL, y el Trabajo de Fin de Carrera "Modelado VHDL de estrategias de modulación para convertidores DC/AC para plataforma Semiteach-IGBT" E. Díaz [6], donde se implementa la modulación PWM trifásica con inyección de secuencias cero. El TFG que aquí se presenta, extiende el trabajo [6] a la implementación en *hardware* reconfigurable de la técnica SVM.

6.1.2 Justificación de uso de FPGA

Una FPGA (*Field Programmable Gate Array*) es un dispositivo que integran bloques lógicos y periféricos, cuyas interconexiones y funcionalidades son programables mediante software.

A diferencia de otros procesadores, las FPGAs tienen ejecución secuencial y concurrente, lo que permite llevar a cabo operaciones de manera paralela, siendo los procesos independientes unos a otros.

En los últimos años, el avance en los lenguajes de alto nivel junto con el desarrollo de nuevas aplicaciones ha hecho más accesible este tipo de tecnologías con un conocimiento básico de la placa y de la programación.

Para el diseño de un inversor de potencia, el uso de una FPGA tiene algunos inconvenientes, propios de trabajar con señales digitales en coma fija, si bien, el trabajo tiene desde el comienzo fines didácticos más que prácticos, por lo que el uso de esta tecnología es suficiente para llevarlo a cabo. Además, Xilinx, mayor distribuidor de FPGAs del mundo, ofrece una gama de bajo coste diseñada para el aprendizaje y la introducción a este tipo de programación.

6.1.3 Objetivos

Este proyecto está orientado a la docencia, por ello, su principal objetivo es mostrar a los alumnos la estrategia de modulación vectorial de convertidores DC/AC de forma real, sirviendo así de apoyo a los conocimientos impartidos en las clases teóricas de las asignaturas que incluyen esta parte del temario en sus resultados de aprendizaje.

Siguiendo la guía docente de la asignatura de Electrónica de Potencia del Grado en electrónica y Automática industrial de la Escuela Politécnica Superior de la Universidad de Alcalá [12], las competencias abarcadas en este proyecto son las siguientes:

- Competencias Genéricas:
    1. TR2-Conocimiento en materias básicas y tecnológicas, que les capacite para el aprendizaje de nuevos métodos y teorías, y les dote de versatilidad para adaptarse a nuevas situaciones.
    2. TR3, Capacidad de resolver problemas con iniciativa, toma de decisiones, creatividad, razonamiento crítico y de comunicar y transmitir conocimientos, habilidades y destrezas en el campo de la Ingeniería Industrial.
- Competencias de carácter personal:
    1. CEI4-Conocimiento aplicado de electrónica de potencia.
    2. CEI7-Conocimiento y capacidad para el modelado y simulación de sistemas.
- Resultados de aprendizaje:
    1. RAEI10. Explicar los conceptos generales de la electrónica de potencia.
    2. RAEI11. Describir los componentes básicos de los convertidores de potencia.
    3. RAEI12. Aplicar los conocimientos de electrónica a la resolución de problemas técnicos.
    4. RAEI13. Modelar, simular y diseñar sistemas de potencia.

6.1.4 ESTRUCTURA DEL DOCUMENTO

El documento ha sido redactado de forma escalonada, permitiendo así al lector ir adquiriendo el conocimiento para entender cada sección escrita.

La sección 6, Memoria, conforma la parte principal del trabajo. En ella se comienza con una base teórica sobre el tema tratado, y su adaptación a código VHDL. Los apartados posteriores, detallan los resultados obtenidos tanto en simulación como en la práctica, junto a los errores cometidos en el proceso.

Finalmente, en las secciones posteriores, se adjuntan las hojas de características de los dispositivos, las partes del código más relevantes, un manual de usuario y el presupuesto general del proyecto.

6.2 REVISIÓN DE TÉCNICAS DE MODULACIÓN VECTORIAL Y SIMULACIONES EN MATLAB

6.2.1 Inversores

La corriente alterna es el sistema idóneo de generación de electricidad y de su distribución, sin embargo, el almacenamiento y los numerosos dispositivos que trabajan con señales continuas, generan la necesidad de buscar un método de conversión de la energía, naciendo así los convertidores. Hoy en día su uso se ha generalizado en muchas aplicaciones favoreciendo su desarrollo en la industria.

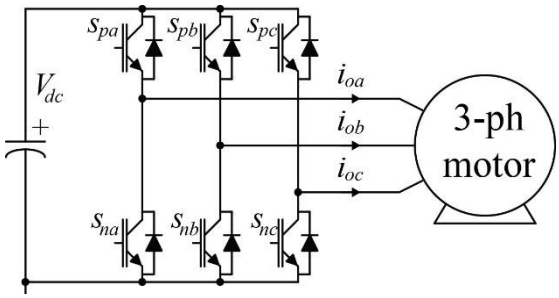


Imagen 6.2: Esquema de un inversor. "Departamento de electrónica UTFSM"

Los convertidores que transforman la energía de DC/AC son denominados inversores, y están constituidos, en el caso trifásico, por un circuito de seis interruptores de potencia y seis diodos conectados en antiparalelo tal y como se muestra en la imagen 6.2.

Mediante la apertura y cierre de los interruptores, generan tensiones trifásicas variables en amplitud y frecuencia, a partir de la fuente de tensión continua. El flujo de potencia es reversible, es decir, en modo inversor, la mayor parte del tiempo la potencia fluye del lado DC al lado AC, debido a que la tensión y la corriente son alternas y su funcionamiento pasa por los cuatro cuadrantes vistos en la imagen 6.3.b, mientras que en modo rectificador la energía eléctrica es convertida del lado AC al DC.

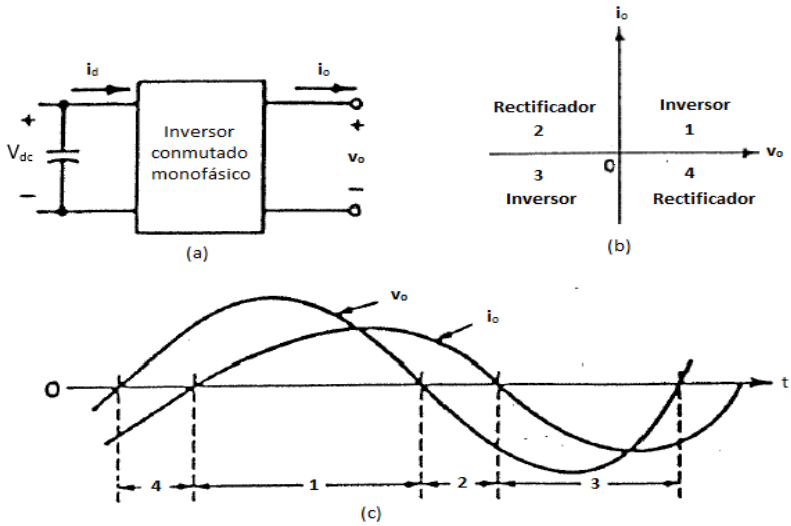


Imagen 6.3: a) Esquema de inversor monofásico conmutado, b) Cuadrantes de funcionamiento, c) Tensión y corriente de salida [7]

En la imagen 6.3.a y 6.3.c se puede ver el diagrama más básico de un inversor conmutado, donde a su entrada las señales son continuas y a su salida se obtienen ondas alternas.

La estrategia seguida para hacer conmutar los interruptores es denominada modulación. Esta afecta a la calidad de la salida obtenida y la eficiencia final del inversor. El método más extendido y del cual nace la técnica vectorial utilizada en el presente proyecto es la modulación basada en una onda portadora. Esta consiste en el uso de una señal senoidal de referencia más una portadora triangular, de forma que en la intersección de las dos determinan la conmutación de los interruptores, según puede verse en la Fig. 6.4

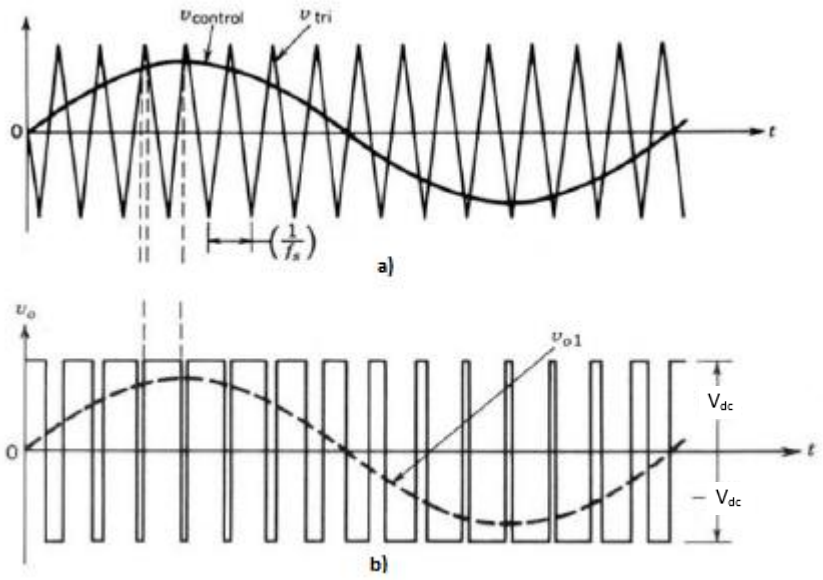


Imagen 6.4: a) Señal de control y señal triangular. b) Tensión de salida y primer armónico del inversor en puente completo [7]

De esta forma, el armónico fundamental de la señal de salida, v_{o1} de la imagen 6.4.b, reproduce la onda senoidal de referencia. Sin embargo, el aprovechamiento de la energía continua es inferior al 80%, por lo que es necesario la inyección de señales homopolares, que no son reflejadas en la carga, para mejorar la eficiencia del convertidor. Además, está basada en un sistema monofásico, siendo necesario el uso de tres ondas portadoras desfasadas para trasladarlo al caso trifásico, FIg.6.5.

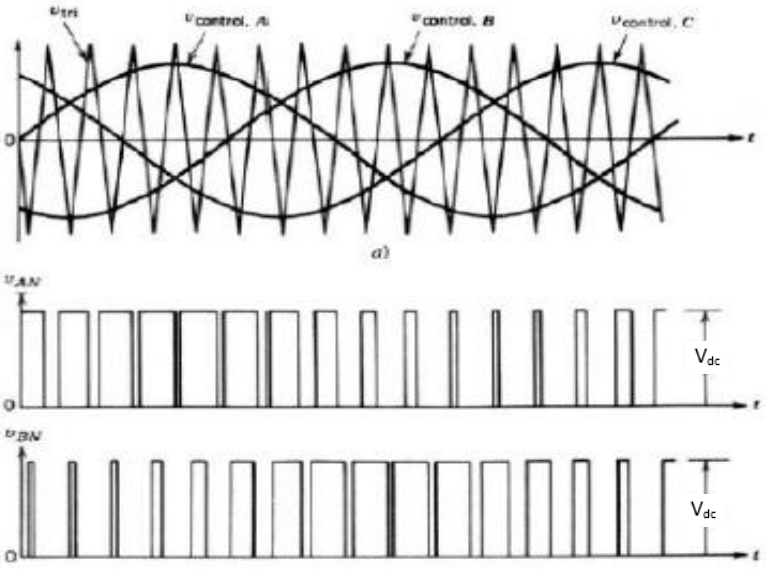


Imagen 6.5: a) Señal de control trifásica y señal triangular. b) Tensiones de linea-neutro a la salida del inversor [7]

6.2.2 Modulación vectorial

6.2.2.1 Introducción

En 1986, Van der Broek planteó una nueva técnica capaz de obtener los mismos resultados que la modulación PWM con la inyección de armónicos, [21].

Mediante la representación espacial de los vectores, consiguió sustituir el sistema trifásico por un solo vector, cuya velocidad de giro marca la frecuencia de la tensión alterna a la salida del inversor, como se muestra en la Fig.6.6.

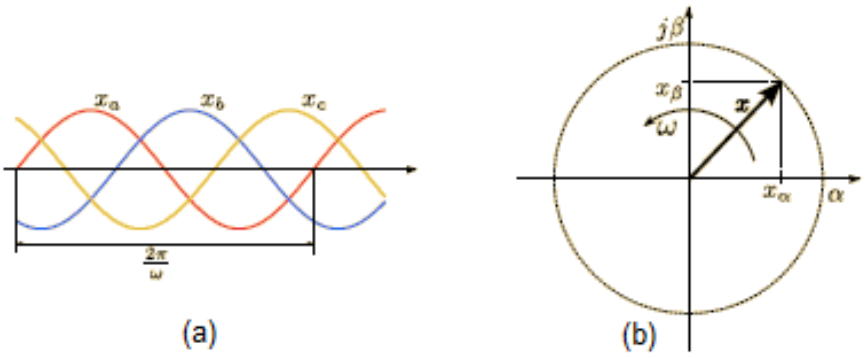


Imagen 6.6: a) Sistema trifásico. b) Representación espacial de los vectores [7]

Las ventajas de este método dependen de su aplicación final a la que este destinada, si bien, su ventaja fundamental hoy en día es su mayor sencillez de implementación digital frente a la modulación PWM senoidal vista anteriormente.

A continuación, se va a explicar en qué consiste la técnica vectorial para su posterior implementación en la tarjeta FPGA.

6.2.2.2 Teoría

Representación vectorial

La representación vectorial con vectores espaciales trata de representar un sistema compuesto de tres magnitudes dependientes en el tiempo, en un sistema bidimensional. En este caso, la magnitud trifásica dependiente en el tiempo sitentizada será el voltaje en la carga del inversor.

La implementación hardware empieza con las especificaciones de cada dispositivo utilizado, y la explicación mediante diagrama de bloques del código generado.

Por último, se calculan los errores cometidos y se muestran los resultados obtenidos tanto simulados en Matlab como en la descarga en placa.

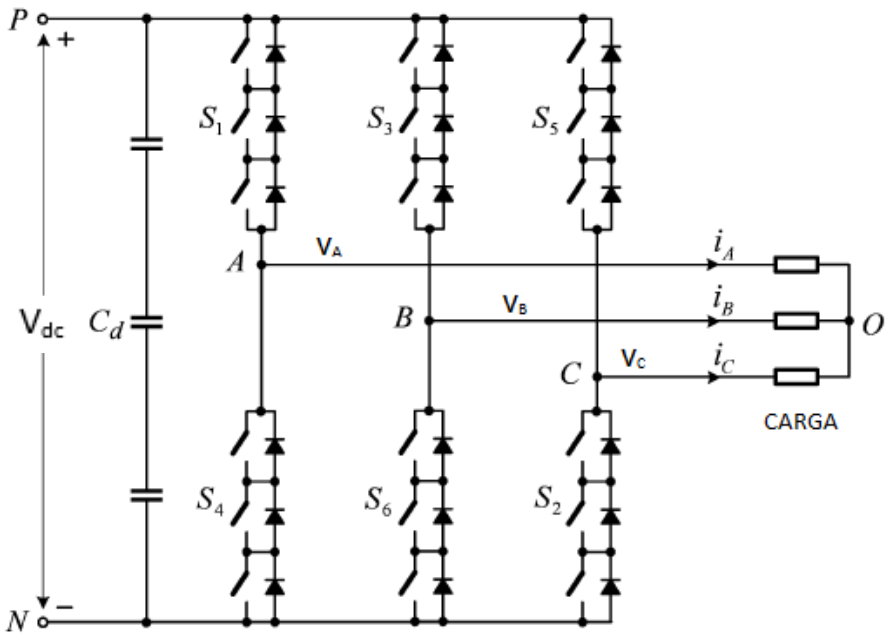


Imagen 6.7: Esquema de un inversor trifásico [4]

En sistemas trifásicos equilibrados como el mostrado en la imagen 6.7, sin hilo neutro, sus tensiones o corrientes siguen la siguiente ecuación:

$$va(t) + vb(t) + vc(t) = 0 \qquad \text{<6.1>}$$

Donde la suma de las corrientes o tensiones instantáneas en cualquier instante de tiempo debe ser igual a 0, por lo que una de las componentes puede ser despejada en función de las otras dos:

$$va(t) = -vb(t) - vc(t) \qquad \text{<6.2>}$$

Para reducir la ecuación 6.2 a dos componentes se realiza una transformación trifásica en vectores espaciales. En el documento "Representación vectorial de variables trifásicas" Emilio José Bueno Peña [8], detalla los pasos seguidos para llevar a cabo la transformación, dando como resultado la siguiente matriz:

Invariante en potencia	Invariante en amplitud
$abc \rightarrow \alpha\beta\gamma$	
$T_{abc\rightarrow\alpha\beta\gamma} = \sqrt{\frac{2}{3}}\begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$	$T_{abc\rightarrow\alpha\beta\gamma} = \frac{2}{3}\begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \\ \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} & \frac{1}{\sqrt{2}} \end{bmatrix}$

Imagen 6.8: Matrices transformación abc-αβγ [8]

La transformación elegida para el desarrollo de la modulación es la invariante en amplitud.

De esta manera, la señal trifásica *abc* es expresada en nuevo sistema *αβγ*, Fig.6.9, siendo *α* la proyección de a en el plano *X, β* la perpendicular a *α* según la regla de la mano derecha, y γ es el resultado del producto vectorial de las anteriores *α x β*, siendo la componente sobre su eje la suma instantánea de *abc* que es 0.

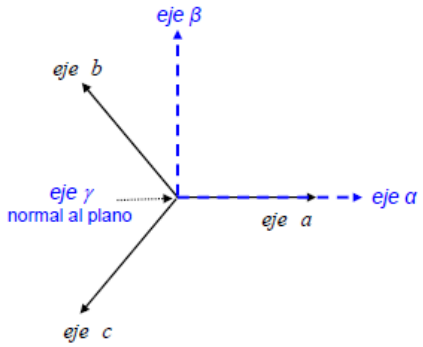


Imagen 6.9: Localización de los ejes abc y αβγ en el espacio [8]

Estados de conmutación

El estado de los interruptores del inversor puede ser representado por estados de conmutación, considerando únicamente dos posibilidades para cada rama:

- P: El interruptor superior de la rama está encendido (*On*) y el inferior apagado (*Off*)
- O: El interruptor inferior de la rama está encendido (*On*) y el inferior apagado (*Off*)

	Rama A			*Rama B*			*Rama C*		
Estado	S_1	S_4	V_{an}	S_3	S_6	V_{bn}	S_5	S_2	V_{cn}
P	On	Off	Vdc	On	Off	Vdc	On	Off	Vdc
O	Off	On	0	Off	On	0	Off	On	0

Tabla 6.1: Estados de conmutación

La nomenclatura utilizada para los interruptores en la tabla 6.1, sigue la aparecida en el esquema de la imagen 6.7. Combinando los estados de conmutación para las tres ramas, aparecen ocho posibles casos:

Espacio vectorial, Estados de conmutación e interruptores activos				
Vector	**Estados de conmutación**	**Interruptores activos**	**V_{abc}**	**$V_{\alpha\beta}$**
$\vec{V}_0$	[PPP] [OOO]	S_1, S_3, S_5 S_4, S_6, S_2	$[V_{dc}\ V_{dc}\ V_{dc}]$ [0 0 0]	0 0
$\vec{V}_1$	[POO]	S_1, S_6, S_2	$[V_{dc}\ 0\ 0]$	$\frac{2}{3} V_{dc}\, e^{j0}$
$\vec{V}_2$	[PPO]	S_1, S_3, S_2	$[V_{dc}\ V_{dc}\ 0]$	$\frac{2}{3} V_{dc}\, e^{j\pi/3}$
$\vec{V}_3$	[OPO]	S_4, S_3, S_2	$[0\ V_{dc}\ 0]$	$\frac{2}{3} V_{dc}\, e^{j2\pi/3}$
$\vec{V}_4$	[OPP]	S_4, S_3, S_5	$[0\ V_{dc}\ V_{dc}]$	$\frac{2}{3} V_{dc}\, e^{j\pi}$
$\vec{V}_5$	[OOP]	S_4, S_6, S_5	$[0\ 0\ V_{dc}]$	$\frac{2}{3} V_{dc}\, e^{j4\pi/3}$
$\vec{V}_6$	[POP]	S_1, S_6, S_5	$[V_{dc}\ 0\ V_{dc}]$	$\frac{2}{3} V_{dc}\, e^{j5\pi/3}$

Tabla 6.2: Vector referencia, estados de conmutación e interruptores

Los vectores del espacio vectorial *αβ* son obtenidos a partir de la matriz de transformación invariante en amplitud de la imagen 6.8, multiplicando por la terna V_{abc} correspondiente a cada estado.

Los siete vectores formados son los llamados estáticos, y pueden ser representados en los ejes *αβ* formando el siguiente espacio vectorial hexagonal:

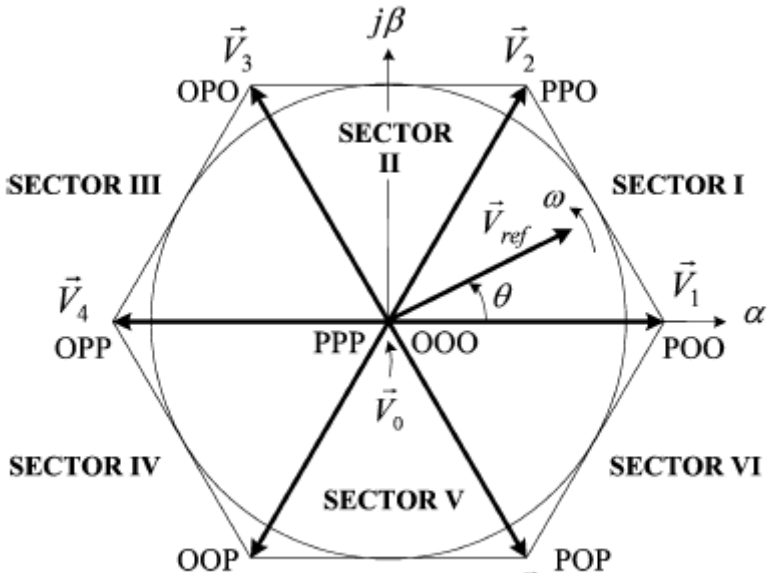


Imagen 6.10: Espacio vectorial αβ [4]

En la imagen 6.10, se pueden apreciar los ocho estados de conmutación comentados anteriormente, siendo [PPP] y [000] los denominados vectores nulos que son utilizados para reducir el número de transiciones en la conmutación de los interruptores.

El espacio vectorial queda dividido en seis sectores que serán de gran utilidad para la implementación de la modulación de forma digital.

Funcionamiento del modulador

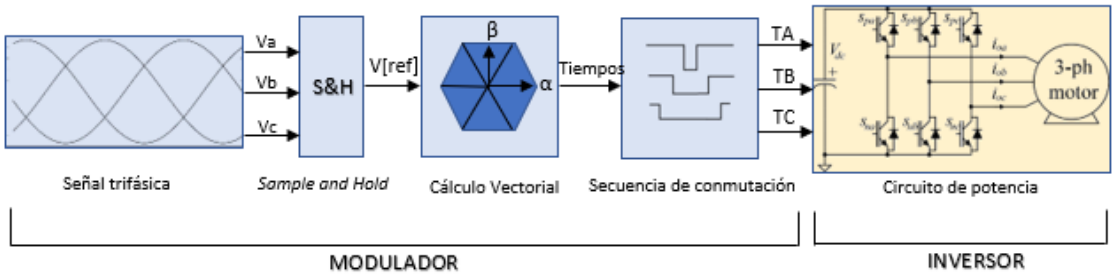


Imagen 6.11: Diagrama de bloques de la modulación vectorial de un inversor trifásico.

La imagen 6.11 muestra la secuencia de funcionamiento de la modulación vectorial. Durante cada periodo de muestreo, marcado por el S&H (*Sample and Hold*), una muestra de la señal trifásica es retenida, $V_{ref,}$ y transformada al espacio vectorial αβ. Los vectores representados en la imagen 6.10, conforman una base de vectores que sintetizan la tensión muestreada en tiempos de conmutación de cada interruptor.

Estos tiempos son transformados en las señales moduladoras aplicando un sistema secuenciador. De esta manera, el vector referencia gira a una velocidad angular proporcional a la frecuencia del primer armónico de la tensión de línea de la señal trifásica *F1*, siendo evaluado cada muestra de la frecuencia de muestreo *fs*.

El cociente de ambas frecuencias es denominado índice de modulación de frecuencia y es el que ajusta el contenido armónico a la salida del inversor, ecuación 6.3.

$$Mf = \frac{Fs}{F1} \qquad <6.3>$$

El módulo del vector referencia marca la zona de modulación en la que se estará trabajando. Puede ser diferenciada de manera sencilla gráficamente, véase la imagen 6.13.

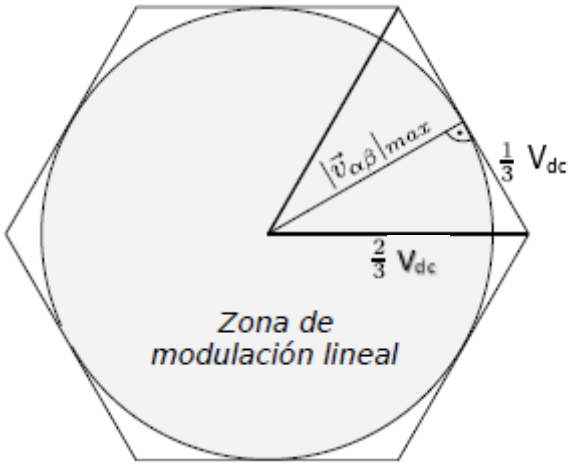


Imagen 6.12: Zona de modulación [7]

Si el vector es más grande que el círculo inscrito en el hexágono, el modulador trabaja en zona de sobremodulación, en caso contrario, la zona de trabajo es lineal. La ecuación 6.4 muestra el valor máximo del vector V_{ref} para trabajar en zona lineal.

$$|Vref| = \frac{\sqrt{3}}{3} Vdc \qquad <6.4>$$

En caso de que el circulo trazado por el vector no corte en ningún momento el hexágono, se trabaja en modulación de onda cuadrada.

El cociente entre el módulo de la referencia y el valor de la componente continua del inversor es llamado índice de modulación de amplitud, *Ma*, y es el parámetro que permite controlar la amplitud del primer armónico a la salida:

$$Ma = \frac{Vref}{Vdc} \qquad <6.5>$$

Tiempos de conmutación

Tal y como se ha explicado, la referencia V_{ref} puede ser sintetizado por tres vectores estacionarios adyacentes, que dependen del sector en el que se encuentre. Los tiempos de conmutación calculados representan el ciclo de trabajo de los interruptores durante cada periodo de muestreo.

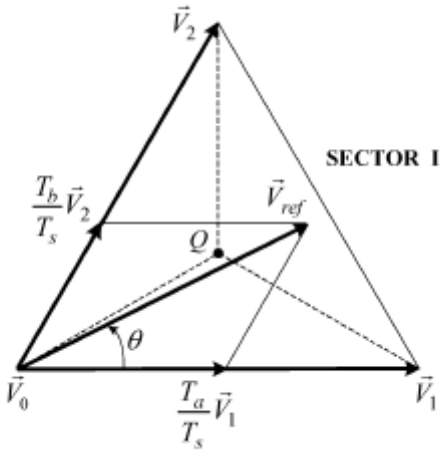


Imagen 6.13: V[ref] sintetizado por V1 y V2 [4]

En la imagen 6.13, correspondiente al sector I del hexágono, se puede ver como la tensión es descompuesta por los vectores adyacentes a ella, según las ecuaciones 6.6 y 6.7.

$$\overrightarrow{\mathrm{Vref}} * \mathrm{Ts} = \overrightarrow{\mathrm{V1}} * \mathrm{Ta} + \overrightarrow{\mathrm{V2}} * \mathrm{Tb} + \overrightarrow{\mathrm{V0}} * \mathrm{T0} \qquad \text{<6.6>}$$

$$\mathrm{Ts} = \mathrm{Ta} + \mathrm{Tb} + \mathrm{T0} \qquad \text{<6.7>}$$

Donde T_a,T_b y T_0 son los tiempos de conmutación de V_1, V_2 y V_0 respectivamente. Las ecuaciones 6.6 y 6.7 pueden ser expresadas en función del ángulo que forma la referencia con los vectores estáticos, como demuestran las ecuaciones siguientes:

$$\mathrm{Ta} = \frac{\sqrt{3} * \mathrm{Ts} * \mathrm{Vref}}{\mathrm{Vdc}} \sin\left(\frac{\pi}{3} - \theta\right) \qquad \text{<6.8>}$$

$$\mathrm{Tb} = \frac{\sqrt{3} * \mathrm{Ts} * \mathrm{Vref}}{\mathrm{Vdc}} \sin(\theta) \qquad \text{para } 0 \leq \theta \leq \frac{\pi}{3} \qquad \text{<6.9>}$$

$$\mathrm{Ts} = \mathrm{Ta} + \mathrm{Tb} + \mathrm{T0} \qquad \text{<6.10>}$$

Secuencia de conmutación

El último paso de la modulación vectorial, siguiendo el diagrama de bloques 6-11, es la elección de la secuencia de conmutación de los interruptores. Para ello, no hay un

modelo único a seguir, aunque sí deben cumplirse dos pautas en todos ellos para reducir el número de conmutaciones y, por tanto, la energía disipada en el proceso:

1. La transición entre dos estados solo debe involucrar dos interruptores de la misma rama.
2. La transición entre sectores no requiere un número mínimo de conmutaciones.

La secuencia más comúnmente utilizada es la denominada siete-segmentos, donde el periodo de muestreo es dividido en siete partes y las secuencias comienzan con el estado nulo [OOO], véase tabla 6.3.

Secuencia de conmutación Siete -segmentos							
Sector	1	2	3	4	5	6	7
I	[OOO]	[POO]	[PPO]	[PPP]	[PPO]	[POO]	[OOO]
II	[OOO]	[OPO]	[PPO]	[PPP]	[PPO]	[OPO]	[OOO]
III	[OOO]	[OPO]	[OPP]	[PPP]	[OPP]	[OPO]	[OOO]
IV	[OOO]	[OOP]	[OPP]	[PPP]	[OPP]	[OOP]	[OOO]
V	[OOO]	[OOP]	[POP]	[PPP]	[POP]	[OOP]	[OOO]
VI	[OOO]	[POO]	[POP]	[PPP]	[POP]	[POO]	[OOO]

Tabla 6.3: Secuencia de conmutación Siete- segmentos

El tiempo establecido para cada una de las siete partes depende del sector en el que se esté localizado el vector de referencia. La secuencia de tiempos según el sector viene mostrada en la siguiente tabla:

Secuencia de conmutación Siete -segmentos							
Sectores	1	2	3	4	5	6	7
I,III,V	$\frac{T0}{4}$	$\frac{Tb}{2}$	$\frac{Ta}{2}$	$\frac{T0}{2}$	$\frac{Ta}{2}$	$\frac{Tb}{2}$	$\frac{T0}{4}$
II,IV,VI	$\frac{T0}{4}$	$\frac{Ta}{2}$	$\frac{Tb}{2}$	$\frac{T0}{2}$	$\frac{Tb}{2}$	$\frac{Ta}{2}$	$\frac{T0}{4}$

Tabla 6.4: Secuencia de tiempos

De esta forma, las señales trifásicas generadas en la zona alterna del inversor por la conmutación de los interruptores son las siguientes:

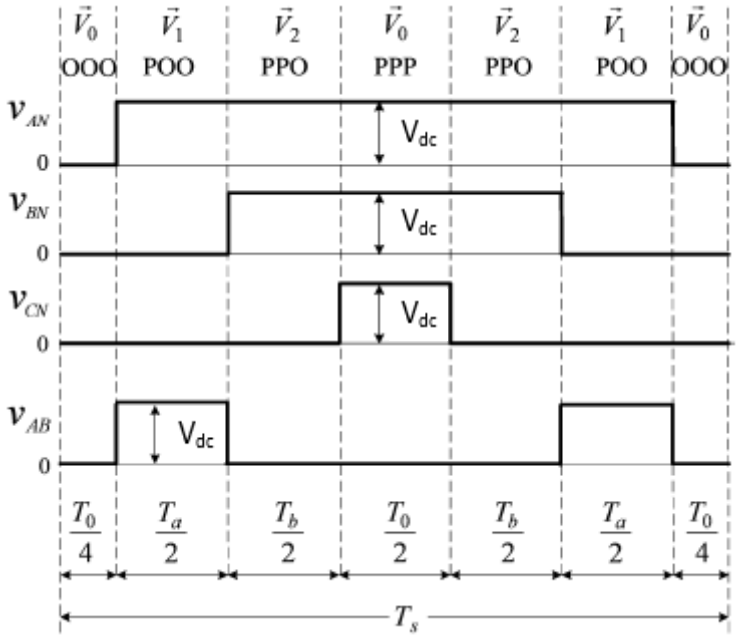
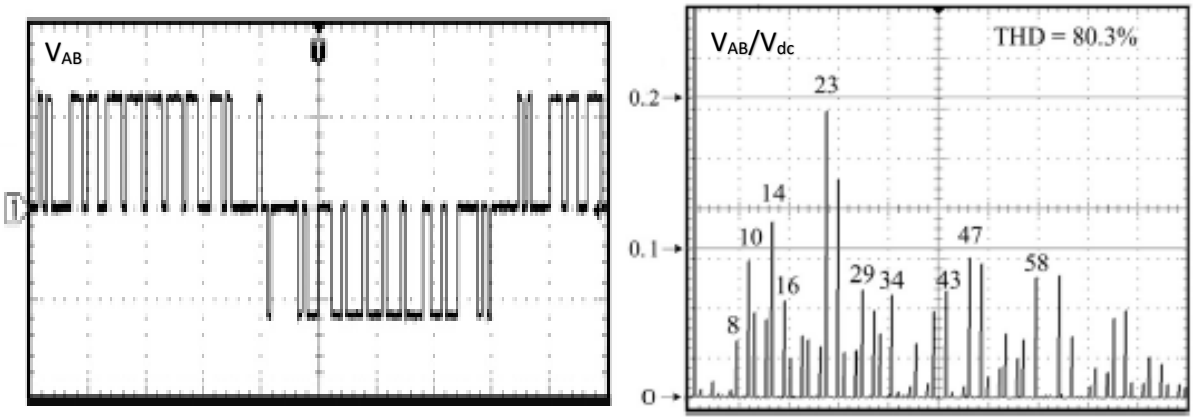


Imagen 6.14: Secuencia siete segmentos para el vector Vref en el sector I

La imagen 6.14 corresponde a una referencia del primer sector por la secuencia de conmutación realizada.

Análisis espectral

La simulación de un inversor trifásico operando bajo modulación vectorial con una frecuencia fundamental F1= 60Hz, una frecuencia de muestreo Fs=720Hz y un índice de modulación de amplitud Ma=0.8, da como resultado la siguiente distribución de armónicos:



(a) (b)

Imagen 6.15: a) Tensión de línea Vab. B) Espectro armónico [4]

En la gráfica 6.15.a, puede observarse como la señal de línea no presentan una simetría de media onda, $\vartheta ab(wt) \neq -\vartheta ab(wt + \pi)$. Además, como se puede ver a continuación con más detalle en la imagen 6.16, aparecen armónicos pares e impares de baja frecuencia, que, a pesar de tener una amplitud baja, no llegan a ser nulos.

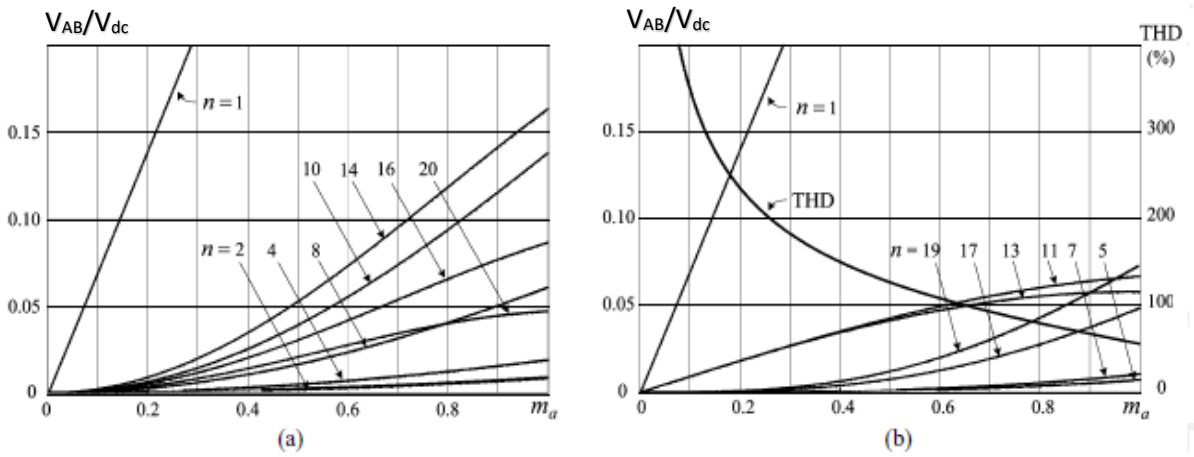


Imagen 6.16: a) Armónicos pares. b) Armónicos impares [4]

Eliminación de armónicos pares

Los armónicos pares no tienen gran impacto en motores controlados por variadores de media tensión, sin embargo, en aplicaciones de red deben cumplir con estándares IEEE. Estas normativas son más restrictivas con la amplitud de los armónicos pares, por lo que se va a tratar de eliminarlos del diseño.

Para ello, basta con cambiar la secuencia de conmutación de los interruptores, de tal forma, que la señal de tensión línea-línea, 6.15.a, tenga simetría par. Esto se consigue dividiendo cada sector del espacio vectorial hexagonal en dos partes.

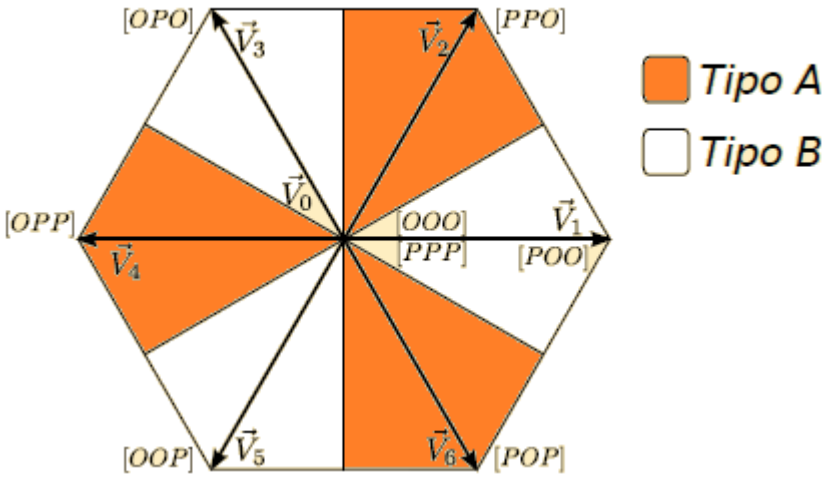


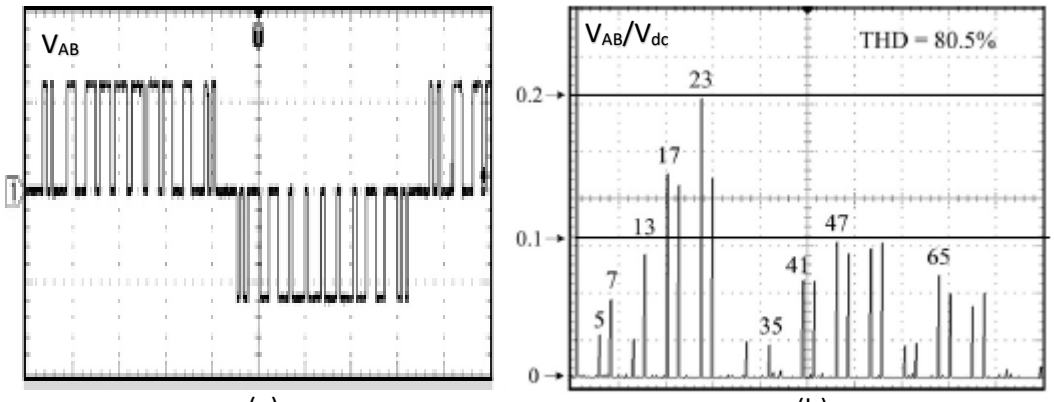
Imagen 6.17: Espacio vectorial para la eliminación de armónicos pares [7]

Las zonas tipo A comienzan y terminan con el vector estático nulo [OOO], mientras que la zona tipo B empieza y termina con el vector [PPP]. La tabla, 6.5, muestra la secuencia de conmutación para todos los sectores y zonas.

Secuencia de conmutación Eliminación armónicos pares							
Sector	1	2	3	4	5	6	7
I-a	[OOO]	[POO]	[PPO]	[PPP]	[PPO]	[POO]	[OOO]
I-b	[PPP]	[PPO]	[POO]	[OOO]	[POO]	[PPO]	[PPP]
II-a	[PPP]	[PPO]	[OPO]	[OOO]	[OPO]	[PPO]	[PPP]
II-b	[OOO]	[OPO]	[PPO]	[PPP]	[PPO]	[OPO]	[OOO]
III-a	[OOO]	[OPO]	[OPP]	[PPP]	[OPP]	[OPO]	[OOO]
III-b	[PPP]	[OPP]	[OPO]	[OOO]	[OPO]	[OPP]	[PPP]
IV-a	[PPP]	[OPP]	[OOP]	[OOO]	[OOP]	[OPP]	[PPP]
IV-b	[OOO]	[OOP]	[OPP]	[PPP]	[OPP]	[OOP]	[OOO]
V-a	[OOO]	[OOP]	[POP]	[PPP]	[POP]	[OOP]	[OOO]
V-b	[PPP]	[POP]	[OOP]	[OOO]	[OOP]	[POP]	[PPP]
VI-a	[PPP]	[POP]	[POO]	[OOO]	[POO]	[POP]	[PPP]
VI-b	[OOO]	[POO]	[POP]	[PPP]	[POP]	[POO]	[OOO]

Tabla 6.5: Secuencia de conmutación para la eliminación de armónicos pares

El análisis espectral obtenido de esta forma, Fig.6.18.b, no contiene los armónicos pares, manteniendo la distorsión armónico total del mismo valor, pero a costa de aumentar el número de conmutaciones de los interruptores.



(a) (b)

Imagen 6.18: a) Tensión de línea Vab. b) Análisis espectral sin armónicos pares [6]

En la imagen 6.18.a, se observa claramente como la señal de línea tiene simetría par:

$$\vartheta ab(wt) = -\vartheta ab(wt + \pi). \qquad <6.11>$$

6.2.3 Adaptación de la teoría para la implementación hardware en código VHDL

6.2.3.1 Introducción

Una vez explicada la teoría de la modulación, es necesaria adaptarla para su codificación en VHDL y su posterior implementación en FPGA. Para reducir el tiempo de conmutación y los recursos consumidos, es necesario simplificar las operaciones para poder implementarlas de manera sencilla a tecnología digital.

Con el fin de simplificar aún más el código se va a trabajar con coma fija, asumiendo los errores cometidos por truncamiento en cada operación realizada.

Para llevar a cabo este apartado, se va a seguir la nomenclatura y el guion del trabajo "*FPGA Implementation of Simplified SVPWM Algorithm for Three Phase Voltage Source Inverter*" [1].

6.2.3.2 Nomenclatura

Antes de comenzar es importante destacar los cambios de la nomenclatura que se han tomado en el documento consultado.

La representación vectorial de las variables trifásicas serán en el plano *d-q*, siendo *q* la componente vertical que antes era *β*, y *d* el eje horizontal del plano sustituyendo a *α*.

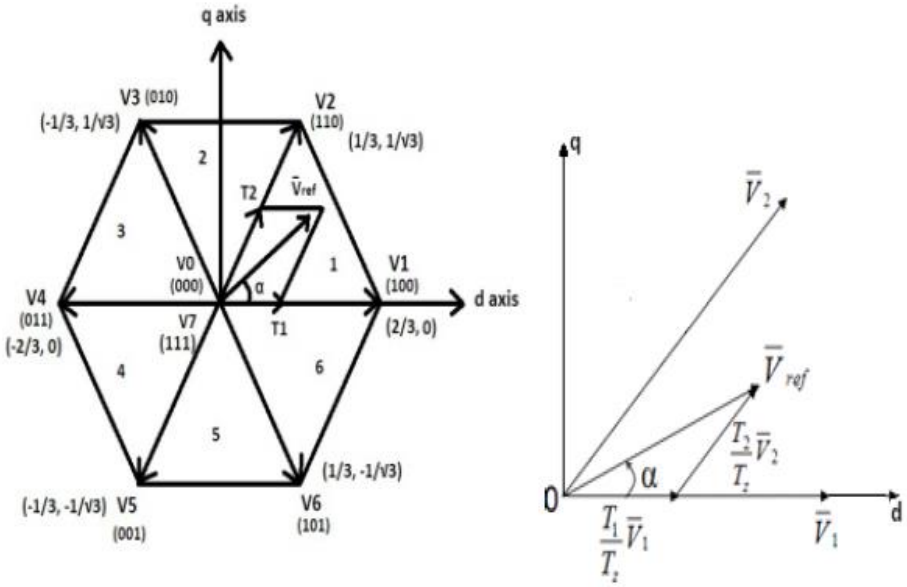


Imagen 6.19: Espacio vectorial y ejes de coordenadas d-q [1]

6.2.3.3 Desarrollo

La simplificación del algoritmo se basa en el cálculo del sector del hexágono en el que se está trabajando para la obtención de los tiempos. Evitando así el uso de raíces cuadradas, multiplicaciones y divisiones que no pueden ser realizadas a nivel de bit de manera sencilla.

La transformación de la señal trifásica de entrada *abc* al espacio vectorial *d-q* por medio de la matriz 6-8 vista en teoría, incluye el uso de raíces cuadradas como muestran las siguientes ecuaciones:

$$Vd = \frac{2}{3}Va - \frac{1}{3}Vb - \frac{1}{3}Vc \quad \text{<6.12>}$$

$$Vq = \frac{1}{\sqrt{3}}Vb - \frac{1}{\sqrt{3}}Vc \quad \text{<6.13>}$$

Para prescindir de las raíces y las fracciones de 3, se definen dos variables intermedias proporcionales a las actuales, X_d y X_q:

$$Xd = 2 * Va - Vb - Vc \quad \text{<6.14>}$$

$$Xq = Vb - Vc \quad \text{<6.15>}$$

Siendo la transformación producida:

$$Vd = \frac{1}{3}[Xd] \quad \text{<6.16>}$$

$$Vq = \frac{1}{\sqrt{3}}[Xq] \quad \text{<6.17>}$$

El sector de trabajo puede ser calculado por medio de tres sencillas condiciones:

1. Signo de X_d
2. Signo de X_q
3. $|X_d| > |X_q|$

La tercera condición informa del cambio de sector producido en un mismo cuadrante. Va a ser demostrada, ya que, difiere del documento referencia que se está siguiendo:

El cambio de sector se produce en el radio del hexágono formado por el espacio vectorial, de forma que, en ese punto, el ángulo formado con V_d es de 60º.

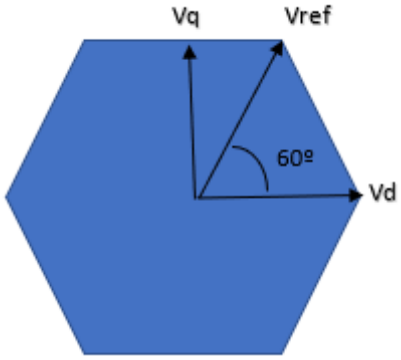


Imagen 6.20: Variación del sector I a II

Si descomponemos el vector referencia en componentes longitudinales a los ejes:

$$Vq = Vref * sen(60º) \quad \text{<6.18>}$$

$$\downarrow \quad \frac{1}{\sqrt{3}}[Xq]$$

$$Xq = \frac{3}{2}Vref \quad \text{<6.19>}$$

De la misma forma, obtenemos la otra variable intermedia, X_d:

$$Vd = Vref * cos(60º) \quad \text{<6.20>}$$

$$\downarrow \quad \frac{1}{3}[Xd]$$

$$Xd = \frac{3}{2}Vref \quad \text{<6.21>}$$

Ambas variables intermedias tienen el mismo módulo en este punto del hexágono, por lo tanto, se comparan en valor absoluto para determinar el cambio de sector.

En el caso de la cancelación de armónicos pares, en la imagen 6.18 se observaba como cada sector era dividido en dos partes iguales. Para la implementación de la secuencia de conmutación posterior, es necesario conocer en que parte de cada sector se encuentra la referencia. Por tanto, es necesario una cuarta condición para su identificación.

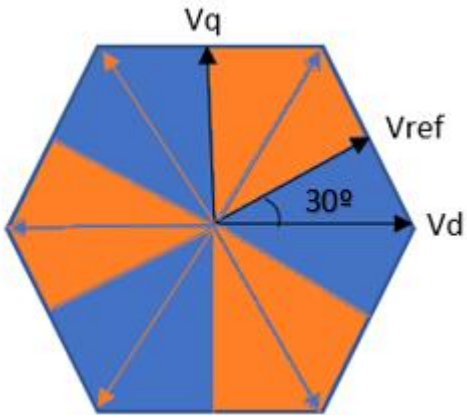


Imagen 6.21: Cambio de zona en el sector I

El procedimiento seguido es el mismo que en las ecuaciones 6.19 y 6.21 pero con el ángulo que corresponde:

$$Vq = Vref * sen(30º) \quad \text{<6.22>}$$

$$\downarrow \frac{1}{\sqrt{3}}[Xq]$$

$$Xq = \frac{\sqrt{3}}{2} Vref \quad \text{<6.23>}$$

$$Vd = Vref * cos(30º) \quad \text{<6.24>}$$

$$\downarrow \frac{1}{3}[Xd]$$

$$Xd = \frac{3 * \sqrt{3}}{2} Vref \quad \text{<6.25>}$$

Por lo tanto, la igualdad entre las ecuaciones 6.23 y 6.25 permitirá identificar el cambio de zona en los sectores I, III, IV y VI.

$$|Xd| > |3*Xq| \quad \text{<6.26>}$$

Continuando el desarrollo, las condiciones que determinan el sector donde se encuentra el vector referencia son mostradas en la siguiente tabla, 6.6.

SECTOR	CONDICIÓN
I	Xd>0 & Xq>0 &|Xd|>|Xq|
II	Xd>0 &|Xq|>|Xd|
III	Xd>0 & Xq<0 &|Xd|>|Xq|
IV	Xd<0 & Xq<0 &|Xd|>|Xq|
V	Xd<0 &|Xq|>|Xd|
VI	Xd<0 & Xq>0 &|Xd|>|Xq|

Tabla 6.6: Selección del sector

Las ecuaciones del cálculo de los tiempos vistas en el apartado teórico, 6.8, 6.9 y 6.10, pueden ser reescritas en función del sector de la siguiente manera:

$$T_n = \frac{\sqrt{3}\,Tz}{Vdc}\left[\sin\left(\frac{\pi}{3} * n\right) * Vd - \cos\left(\frac{\pi}{3} * n\right) Vq\right] \quad <6.27>$$

$$T_{n+1} = \frac{\sqrt{3}\,Tz}{Vdc}\left[-\sin\left(\frac{\pi}{3}(n-1)\right) * Vd - \cos\left(\frac{\pi}{3} * (n-1)\right) Vq\right] \quad <6.28>$$

$$[T0] = Tz - Tn - Tn1 \quad <6.29>$$

Donde, $T_n = T_1$, $T_{n+1} = T_2$, $T_z = T_s$, $V_d = V_1$, $V_q = V_2$, y n el número del sector.

Al tratarse de senos y cosenos con seis posibles valores únicamente, uno por cada sector, su resultado puede ser introducido en memoria a través de una tabla de verdad, formándose así una expresión en forma matricial:

$$\begin{bmatrix} Tn \\ Tn1 \end{bmatrix} = \frac{\sqrt{3}\,Tz}{Vdc} * M0 * \begin{bmatrix} Xd \\ Xq \end{bmatrix} \quad <6.30>$$

Donde M_0 es una matriz de dimensiones 4x4 con los siguientes coeficientes:

SECTOR	I	II	III	IV	V	VI
M_{00}	1/2	1/2	0	-1/2	-1/2	0
M_{01}	-1/2	1/2	1	1/2	-1/2	-1
M_{10}	0	-1/2	-1/2	0	1/2	1/2
M_{11}	1	1/2	-1/2	-1	-1/2	1/2

Tabla 6.7: Matriz M0

$$[M0] = \begin{bmatrix} M00 & M01 \\ M10 & M11 \end{bmatrix} \quad <6.31>$$

Las expresiones finales de los tiempos quedan en función de raíz de tres, a pesar de existir bloques en VHDL que lo calculan, al no trabajar con decimales, el error cometido es grande, por lo que se evitará normalizando las señales de entrada Va, Vb y Vc a raíz de tres.

$$[Tn] = \frac{\sqrt{3}\,Tz}{Vdc} * [M00 \quad M01] * \begin{bmatrix} Xd \\ Xq \end{bmatrix} \qquad <6.32>$$

$$[Tn1] = \frac{\sqrt{3}\,Tz}{Vdc} * [M10 \quad M11] * \begin{bmatrix} Xd \\ Xq \end{bmatrix} \qquad <6.33>$$

$$[T0] = Tz - Tn - Tn1 \qquad <6.34>$$

De esta manera, se ha conseguido un algoritmo sencillo de codificar en VHDL, reduciendo las operaciones que conllevan mayores recursos y minimizando los errores cometidos por el truncamiento de los decimales.

6.2.4 Simulación Matlab

Antes de implementar la modulación vectorial en la FPGA, se ha realizado una simulación en Matlab con el fin de verificar la adaptación del algoritmo teórico visto en el apartado anterio..

El siguiente guión ha sido elaborado por la tutora de este proyecto, MC.Perez junto a las correcciones del alumno que lo presenta, A.Sáez.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Algoritmo simplificado para modulación SVPWM, para vector de esTn(i)dos
%simétrico. Según artículos:
%1) "FPGA ImplemenTn(i)cion of Simplified SVPWM Algorithm for Three Phase
%VolTn(i)ge Source Inverter", D. N. Sonawave, M. S. SuTn(i)one, B. N- Choudhari
%and A. Badukar, International Journal of Computer and Electrical
%Engineering, vol. 2, no. 6, December 2010, pp. 1793-8163.
%2) "FPGA based space vector pulse width modulation technique
%implemenTn(i)tion for three phase inverter", R. Karmokar, S. Mungekar, T.
%Vaity, International Journal of Computer and Electrical
%Engineering, Vol. 8, Issue 2, April 2017, pp. 36-46.
%MCPR 25/05/2017
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
 clear all;
 close all;
 format long
 VAB1ma=[];
 modu=[];
 %Vdc 12 bits, y amplitud de la carga normalizada a raiz de tres
Vdc=(2^12);
%Datos del circuito
RI=0.0001;
CI=0.001;
R=10;
L=0.01;
Ron=1e-6;
%Pequeños incrementos para la simulación matlab
td=0;
e=1e-10;
eps=1e-10;
%Parámetros de la simulación
f1=100;
T1=1/f1;
mf=12;
fs=mf*f1;
Ts=1/fs;
ma=0.99;
%Formación de las tensiones de la carga que queremos obtener
samples=mf;%Numero de muestras para almacenar las tensiones van, vbn y vcn
t=0:(1/f1)/samples:1/f1-1/f1/samples;
va=(ma*Vdc/sqrt(3))*sin(2*pi*f1*t);
vb=(ma*Vdc/sqrt(3))*sin(2*pi*f1*t-120*pi/180);
vc=(ma*Vdc/sqrt(3))*sin(2*pi*f1*t+120*pi/180);
%Representacion de los senos generados
figure; plot(t,va,t,vb,t,vc);title('Tensión en la carga');legend('van', 'vbn', 'vcn');
% figure; plot(va,'b');hold on;plot(vb,'r');hold on;plot(vc,'k');title('Tensión en la carg
a');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Modulación vectorial
Xd=2*va-vb-vc;
Xq=vb-vc;
figure; plot(Xd);hold on; plot(Xq);title('Xd y Xq');legend('Xd', 'Xq');
Vd=1/3*(Xd);
Vq=(1/sqrt(3))*(Xq);
figure; plot(Vd);hold on; plot(Vq);title('Vd y Vq');legend('Vd', 'Vq');
%Cálculo del ángulo theta para la representación de los resultados
theta=atan2(Vq,Vd)*180/pi;
theta=theta-theta(1);
for i=1:1:length(theta)
```

```
    if theta(i)<=0
        theta(i)=theta(i)+360;
    end;
end
theta2=0:360/mf:(360/mf)*length(va)-360/mf;%por si quiero pintar varios períodos
%theta
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Determinación del sector
%Se añade 1e-10 para discernir entre sectores siguiendo un mismo criterio
%cuando el vector cae justo en el límite entre uno y otro
 sector_prueba=[];
for i=1:1:length(Xd)
    if Xd(i)>0 && (Xq(i)+1e-10)>0 && abs(Xd(i))>abs(Xq(i))
        sector_prueba(i)=1;
    elseif (Xd(i)>0 && Xq(i)>0 && abs(Xd(i))<(abs(Xq(i))+1e-10))  || (Xd(i)<0 && Xq(i)>0 &
& (abs(Xd(i))+1e-10)<abs(Xq(i)))
        sector_prueba(i)=2;
    elseif Xd(i)<0 && (Xq(i)-1e-10)>0 && (abs(Xd(i))+1e-10)>abs(Xq(i))
        sector_prueba(i)=3;
    elseif Xd(i)<0 && (Xq(i)-1e-10)<0 && abs(Xd(i))>abs(Xq(i))
        sector_prueba(i)=4;
    elseif (Xd(i)>0 && Xq(i)<0 && (abs(Xd(i))+1e-10)<abs(Xq(i))) || (Xd(i)<0 && Xq(i)<0 &&
 abs(Xd(i))<abs(Xq(i)))
        sector_prueba(i)=5;
    elseif Xd(i)>0 && (Xq(i)+1e-10)<0 && (abs(Xd(i))+1e-10)>abs(Xq(i))
        sector_prueba(i)=6;
     else
         sector_prueba(i)=5;%Para el primer dato
    end;
end;
sector=sector_prueba;
%Calculo de tiempos, según mi versión
Tn_aux=[];
Tn1_aux=[];
T0_aux=[];
for i=1:1:length(sector)
    switch sector(i)
    case 1
        Tn(i)=Xd(i)/2-Xq(i)/2;
        Tn1(i)=Xq(i);
    case 2
        Tn(i)=Xd(i)/2+Xq(i)/2;
        Tn1(i)=Xq(i)/2-Xd(i)/2;
    case 3
        Tn(i)=Xq(i);
        Tn1(i)=-Xq(i)/2-Xd(i)/2;
    case 4
        Tn(i)=-Xd(i)/2+Xq(i)/2;
        Tn1(i)=-Xq(i);

    case  5
        Tn(i)=-Xd(i)/2-Xq(i)/2;
        Tn1(i)=Xd(i)/2-Xq(i)/2;
    case 6
        Tn(i)=-Xq(i);
        Tn1(i)=Xd(i)/2+Xq(i)/2;
    otherwise
        Tn(i)=0;
        Tn1(i)=0;
        T0(i)=0;
end;
```

```
 if Tn(i)<0
     Tn(i)=0;
 end;
   Tn(i)=Tn(i)*(Ts/Vdc);
   Tn1(i)=Tn1(i).*(Ts/Vdc);
   T0(i)=Ts-(Tn(i)+Tn1(i));
end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Generación PWM según artículo
timetotal=[];
signAtotal=[];
signBtotal=[];
signCtotal=[];
control=[];
tiempocontrol=[];

%Generación PWM según artículo
timetotal=[];
signAtotal=[];
signBtotal=[];
signCtotal=[];
control=[];
tiempocontrol=[];

S1=[0 0 0 0 0 0;0 0 0 0 0 0; 0 0 0 0 0 0];%PWMA sector 1 a 6; PWMB sector 1 a 6; PWMC sect
or 1 a 6;
S2=[1 0 0 0 0 1; 0 1 1 0 0 0;0 0 0 1 1 0];
S3=[1 1 0 0 1 1;1 1 1 1 0 0;0 0 1 1 1 1];
S4=[1 1 1 1 1 1; 1 1 1 1 1 1; 1 1 1 1 1 1];
S5=S3;
S6=S2;
S7=S1;

for i=1:1:length(sector)

    if sector(i)==1 || sector(i)==3 || sector(i)==5
      t0=e; t1= T0(i)/4; t2=T0(i)/4+Tn(i)/2; t3=T0(i)/4+Tn(i)/2+Tn1(i)/2; t4=T0(i)/4+Tn(i)
/2+Tn1(i)/2+T0(i)/2; t5=T0(i)/4+Tn(i)/2+Tn1(i)/2+T0(i)/2+Tn1(i)/2; t6=T0(i)/4+Tn(i)/2+Tn1(
i)/2+T0(i)/2+Tn1(i)/2+Tn(i)/2; t7=T0(i)/4+Tn(i)/2+Tn1(i)/2+T0(i)/2+Tn1(i)/2+Tn(i)/2+T0(i)/
4;

      %Para depurar
      ta(i)=S1(1,sector(i))*T0(i)/4+S2(1,sector(i))*Tn(i)/2+S3(1,sector(i))*Tn1(i)/2+S4(1,
sector(i))*T0(i)/2+S5(1,sector(i))*Tn1(i)/2+S6(1,sector(i))*Tn(i)/2+S7(1,sector(i))*T0(i)/
4;
      tb(i)=S1(2,sector(i))*T0(i)/4+S2(2,sector(i))*Tn(i)/2+S3(2,sector(i))*Tn1(i)/2+S4(2,
sector(i))*T0(i)/2+S5(2,sector(i))*Tn1(i)/2+S6(2,sector(i))*Tn(i)/2+S7(2,sector(i))*T0(i)/
4;
      tc(i)=S1(3,sector(i))*T0(i)/4+S2(3,sector(i))*Tn(i)/2+S3(3,sector(i))*Tn1(i)/2+S4(3,
sector(i))*T0(i)/2+S5(3,sector(i))*Tn1(i)/2+S6(3,sector(i))*Tn(i)/2+S7(3,sector(i))*T0(i)/
4;

    else
     t0=e; t1= T0(i)/4; t2=T0(i)/4+Tn1(i)/2; t3=T0(i)/4+Tn1(i)/2+Tn(i)/2; t4=T0(i)/4+Tn1(i
)/2+Tn(i)/2+T0(i)/2; t5=T0(i)/4+Tn1(i)/2+Tn(i)/2+T0(i)/2+Tn(i)/2; t6=T0(i)/4+Tn1(i)/2+Tn(i
)/2+T0(i)/2+Tn(i)/2+Tn1(i)/2; t7=T0(i)/4+Tn1(i)/2+Tn(i)/2+T0(i)/2+Tn(i)/2+Tn1(i)/2+T0(i)/4
;
      %Para depurar
      ta(i)=S1(1,sector(i))*T0(i)/4+S2(1,sector(i))*Tn1(i)/2+S3(1,sector(i))*Tn(i)/2+S4(1,
sector(i))*T0(i)/2+S5(1,sector(i))*Tn(i)/2+S6(1,sector(i))*Tn1(i)/2+S7(1,sector(i))*T0(i)/
4;
```

```
      tb(i)=S1(2,sector(i))*T0(i)/4+S2(2,sector(i))*Tn1(i)/2+S3(2,sector(i))*Tn(i)/2+S4(2,
sector(i))*T0(i)/2+S5(2,sector(i))*Tn(i)/2+S6(2,sector(i))*Tn1(i)/2+S7(2,sector(i))*T0(i)/
4;
      tc(i)=S1(3,sector(i))*T0(i)/4+S2(3,sector(i))*Tn1(i)/2+S3(3,sector(i))*Tn(i)/2+S4(3,
sector(i))*T0(i)/2+S5(3,sector(i))*Tn(i)/2+S6(3,sector(i))*Tn1(i)/2+S7(3,sector(i))*T0(i)/
4;

    end;
        time=[t0 t1 t1+e t2 t2+e t3 t3+e t4 t4+e t5 t5+e t6 t6+e t7];


        signA = [S1(1,sector(i)) S1(1,sector(i)) S2(1,sector(i)) S2(1,sector(i)) S3(1,sect
or(i)) S3(1,sector(i)) S4(1,sector(i)) S4(1,sector(i)) S5(1,sector(i)) S5(1,sector(i)) S6(
1,sector(i)) S6(1,sector(i)) S7(1,sector(i)) S7(1,sector(i))];
        signB = [S1(2,sector(i)) S1(2,sector(i)) S2(2,sector(i)) S2(2,sector(i)) S3(2,sect
or(i)) S3(2,sector(i)) S4(2,sector(i)) S4(2,sector(i)) S5(2,sector(i)) S5(2,sector(i)) S6(
2,sector(i)) S6(2,sector(i)) S7(2,sector(i)) S7(2,sector(i))];
        signC = [S1(3,sector(i)) S1(3,sector(i)) S2(3,sector(i)) S2(3,sector(i)) S3(3,sect
or(i)) S3(3,sector(i)) S4(3,sector(i)) S4(3,sector(i)) S5(3,sector(i)) S5(3,sector(i)) S6(
3,sector(i)) S6(3,sector(i)) S7(3,sector(i)) S7(3,sector(i))];


        timetotal=[timetotal, time+(i-1)*Ts.*ones(1,length(time))];


       if i>1
            for m=length(time):-1:1
                if(timetotal(length(timetotal)-m)>=timetotal(length(timetotal)-m+1))
                    timetotal(length(timetotal)-m+1)=timetotal(length(timetotal)-m)+e;
                end
            end;
       end;


        signAtotal=[signAtotal, signA];
        signBtotal=[signBtotal, signB];
        signCtotal=[signCtotal, signC];

end;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%Para depuración
figure;plot(theta2,ta,theta2, tb, theta2, tc);%ta_on; tb_on y tc_on respecto al angulo the
ta

isequal(round(1e8*ta(1:end-mf/3)),round(1e8*tb(mf/3+1:end)))%ta y tb deberían ser iguales
pero con desfase 120°
isequal(round(1e8*ta(1:end-2*mf/3)),round(1e8*tc(2*mf/3+1:end)))%ta y tc deberían ser igua
les pero con desfase 120°

isequal(round(1e8*tb(1:end-mf/3)),round(1e8*tc(mf/3+1:end)))%tb y tc iguales pero con desf
ase 120°
%figure;plot(theta,ta);
sim('ModulacionVectorialMCP')
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
Representacion espacio vectorial
    Vdc=sqrt(3);

    Vmax=2*Vdc/3;
    figure(13)
    axis([-Vdc +Vdc -Vdc +Vdc])
    title('Representacion en alfa-beta de un periodo del primer armonico de salida para ca
```

```
da ma')
    grid ON;
    xlabel('V_alfa')
    ylabel('V_beta')
    plot([Vmax Vmax*cos(pi/3)], [0 Vmax*sin(pi/3)], 'm-*')
    hold on
    plot([Vmax*cos(pi/3) Vmax*cos(2*pi/3)], [Vmax*sin(pi/3) Vmax*sin(2*pi/3)], 'm-*')
    plot([Vmax*cos(2*pi/3) Vmax*cos(3*pi/3)], [Vmax*sin(2*pi/3) Vmax*sin(3*pi/3)], 'm-*')
    plot([Vmax*cos(3*pi/3) Vmax*cos(4*pi/3)], [Vmax*sin(3*pi/3) Vmax*sin(4*pi/3)], 'm-*')
    plot([Vmax*cos(4*pi/3) Vmax*cos(5*pi/3)], [Vmax*sin(4*pi/3) Vmax*sin(5*pi/3)], 'm-*')
    plot([Vmax*cos(5*pi/3) Vmax*cos(6*pi/3)], [Vmax*sin(5*pi/3) Vmax*sin(6*pi/3)], 'm-*')

     for k=length(ual)-mf+1:1:length(ual)
        plot([0 ual(k)], [0 ubl(k)], 'b-o')
        hold on
    end
    plot (ual, ubl);
```

Código 6-1: Adaptación de la modulación vectorial en Matlab

A continuación, se van a observar los resultados de la simulación con valores estándar solo para comprobar el correcto funcionamiento del código generado. Los parámetros de la modulación utilizados son:

Ma=0.99 Mf=36 F1=100Hz

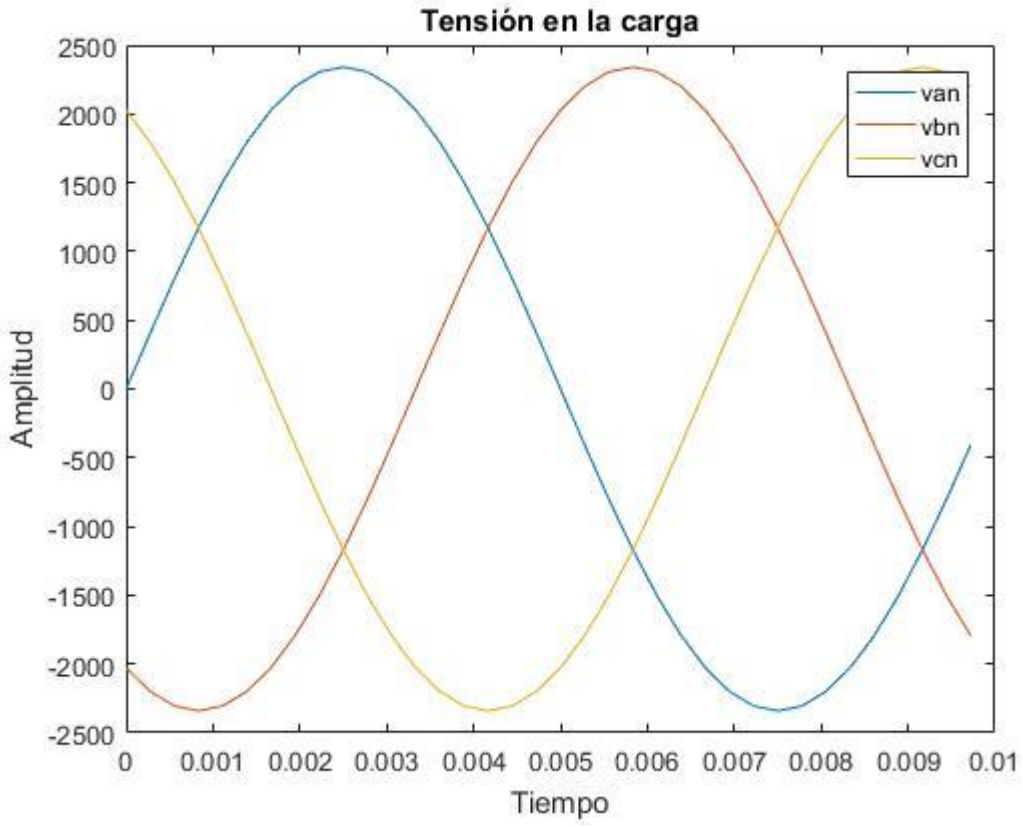


Imagen 6.22: Tensión de Carga

Esta es la tensión que se desea obtener a la salida del inversor, y, por lo tanto, la que va a ser discretizada para su modulación. La amplitud, como se comentó en la teoría, ha sido normalizada a raíz de tres obteniéndose así el siguiente valor:

$$Amplitud = \frac{2^{12}}{\sqrt{3}} = 2365 \qquad <6.35>$$

Se han escogido 12 bits de resolución para minimizar el error por cuantificación cometido al trabajar con coma fija y poder exportar las señales por el DAC de 12 bits que se verá posteriormente.

Las variables intermedias X_d y X_q generadas son:

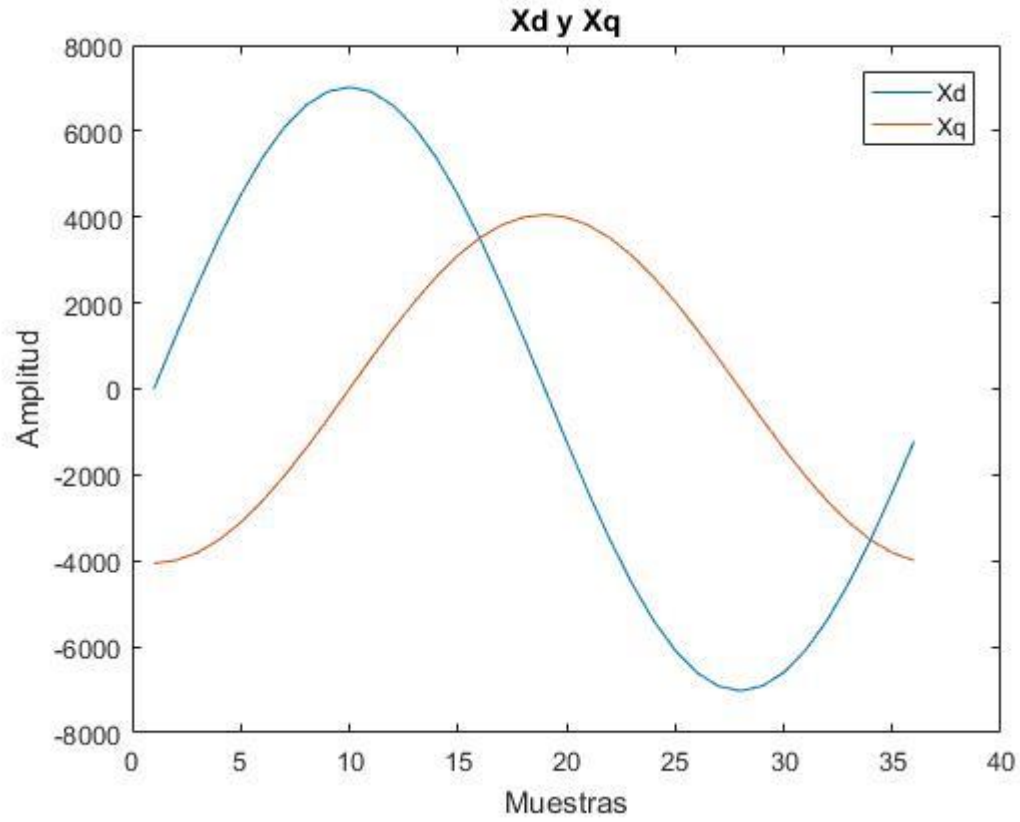


Imagen 6.23: Variables intermedias Xd y Xq

La visualización de los resultados permitirá deducir el tamaño mínimo que necesitan las señales para calcular estas variables:

$$Valor\ máximo\ Xd = 7023.5 = 14\ bits$$

$$Valor\ máximo\ Xq = 4055 = 13\ bits$$

Se ha sumado un bit más a cada una para la parte negativa de la señal.

La representación de los tiempos calculados para cada muestra tomada de la señal de carga es una forma clara de conocer si la modulación realizada está bien hecha.

La forma de onda es semejante a la que se puede encontrar realizando una simulación PWM con la adición de una secuencia cero, donde el aprovechamiento del bus DC es del 100%

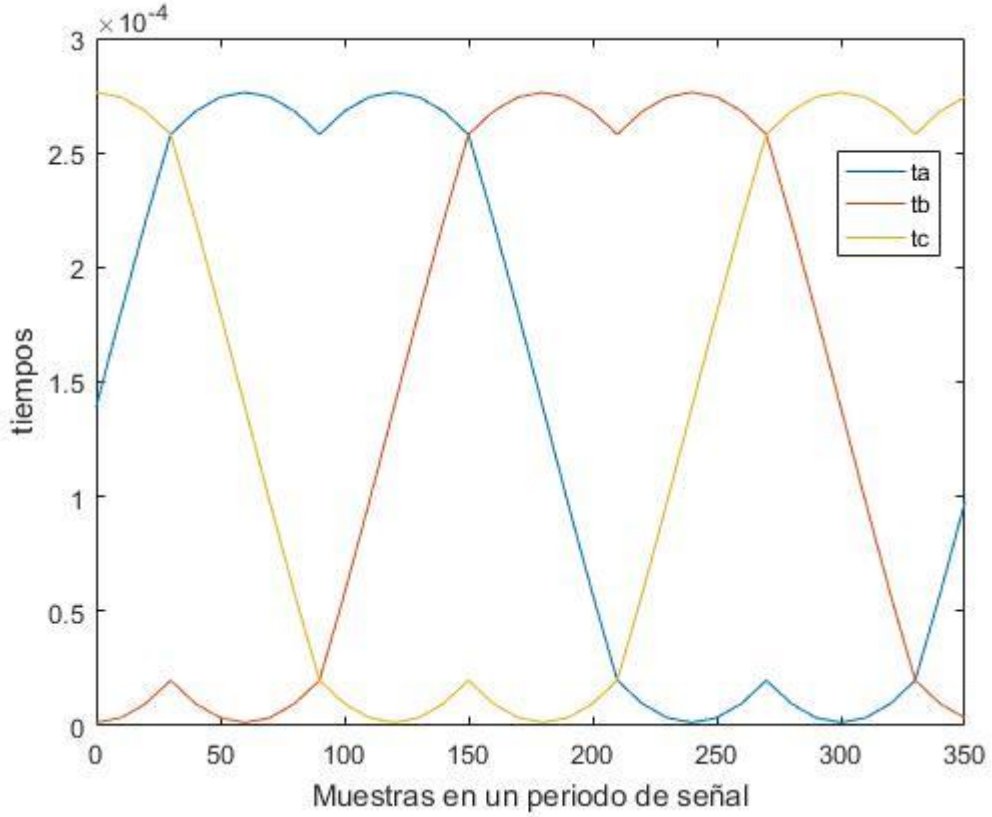


Imagen 6.24: Tiempos por muestra en un periodo completo

Al aumentar *Mf*, el número de muestras cogidas por periodo es mayor, y, por lo tanto, visualmente se apreciará menos los escalones de cuantificación.

La simulación del código se ha realizado mediante la herramienta de simulación interactiva simulink-Matlab, diseñada en un TFG previo [16]. En los anexos del manual de usuario, sección 9.4.3, se ha incluido una captura de pantalla del circuito elaborado en dicho TFG. Las señales de control, encargadas de gobernar el comportamiento de los interruptores del inversor son las siguientes:

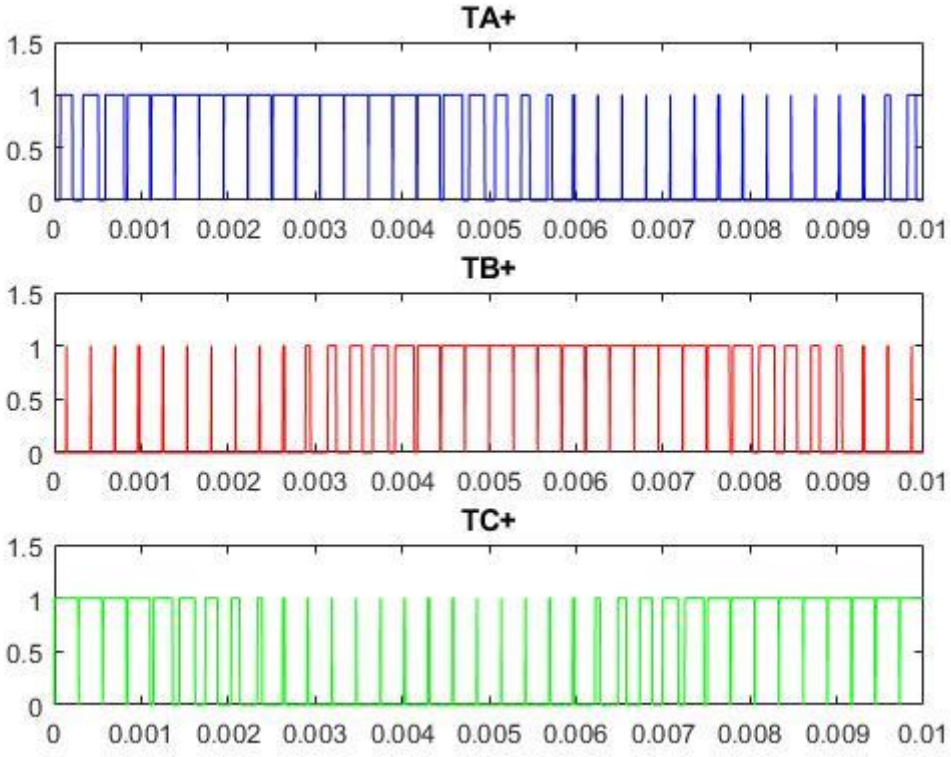


Imagen 6.25: Señales moduladoras TA+, TB+, TC+

Solo han sido representadas las señales de la rama de arriba del inversor para una mayor claridad en los resultados, siendo las otras tres las inversas de cada una de ellas.

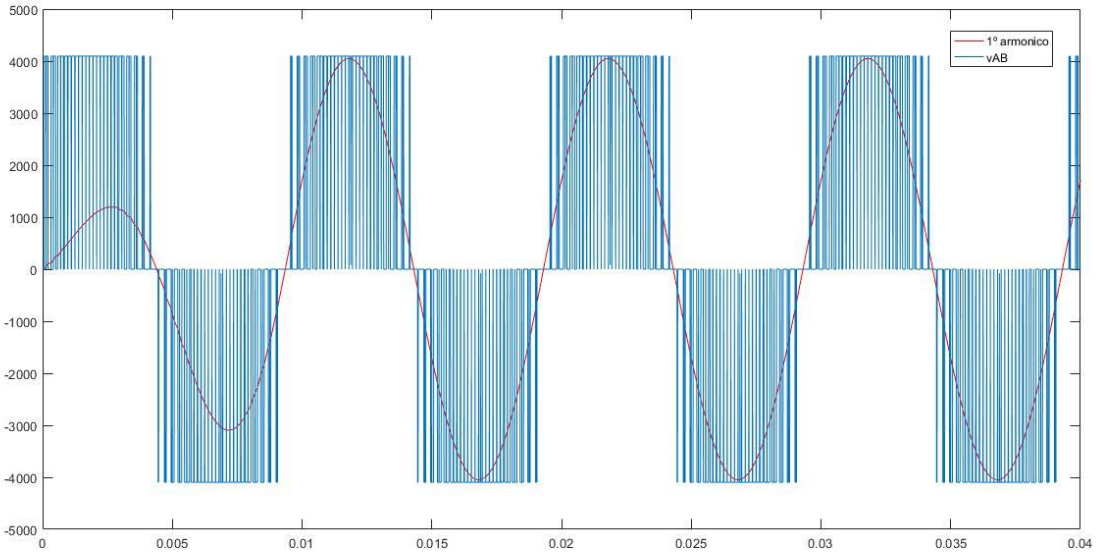


Imagen 6.26: Tensión de línea de salida VAB y primer armónico

La señal de línea-línea obtenida tiene una forma semejante a la imagen 6-15-a mostrada en la teoría. A pesar de ser difícil apreciarlo, la señal vAB no sigue una simetría par, como se puede comprobar viendo su contenido armónico:

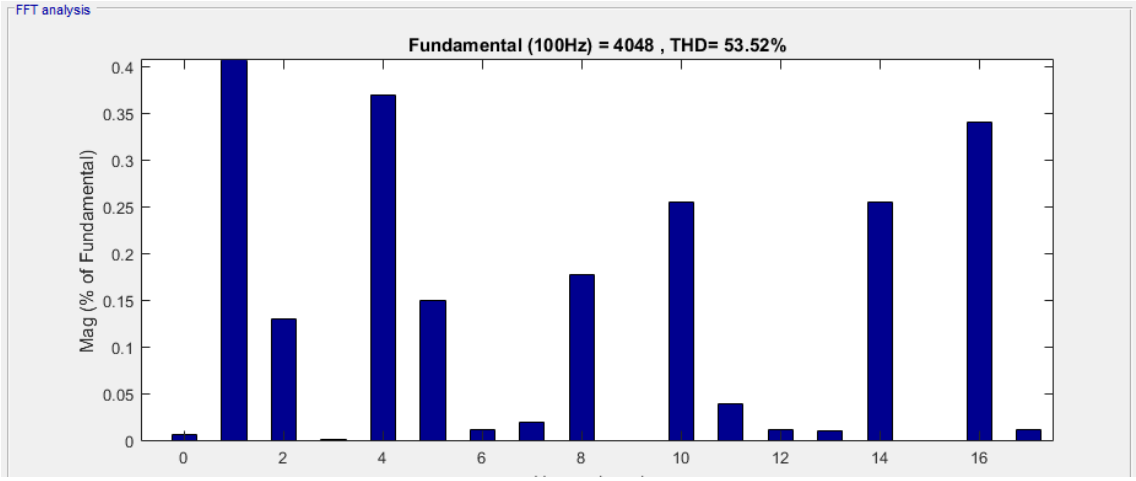


Imagen 6.27: Espectro Armónico

Como era de esperar, aparecen armónicos pares de baja frecuencia que no serían aceptados para ciertas aplicaciones.

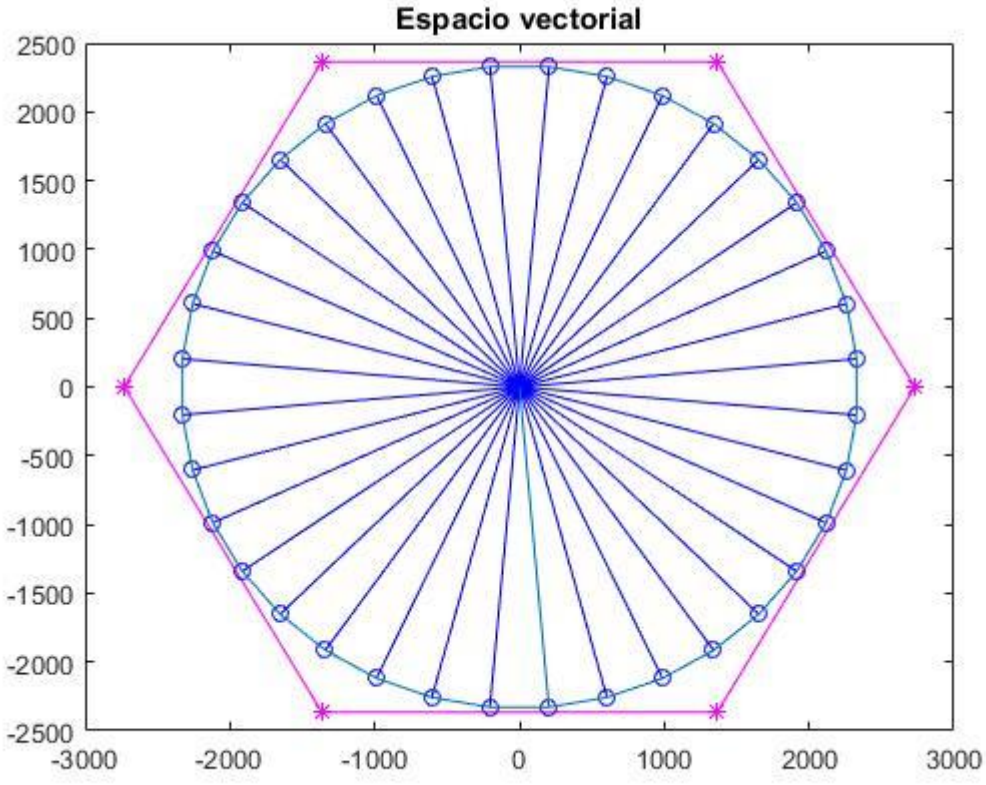


Imagen 6.28: Representación del espacio vectorial obtenido αβ

Por último, la representación vectorial del vector referencia que ha sido convertido en cada muestra para un periodo completo es similar al visto en teoría.

6.3 IMPLEMENTACIÓN HARDWARE DE ESTRATEGIAS DE MODULACIÓN VECTORIAL

6.3.1 Introducción

Una vez finalizada la explicación teórica de la modulación vectorial y su posterior adaptación a hardware, se va a comentar las especificaciones de los dispositivos utilizados para su implementación para las posteriores pruebas prácticas, así como, el diseño elaborado en VHDL.

6.3.1.1 FPGA

La FPGA utilizada para la implementación del inversor es la tarjeta Nexys 2 de Digilent basada en una FPGA Spartan3E-XC3S1200E FG320-4 de Xilinx Spartan3E [20].

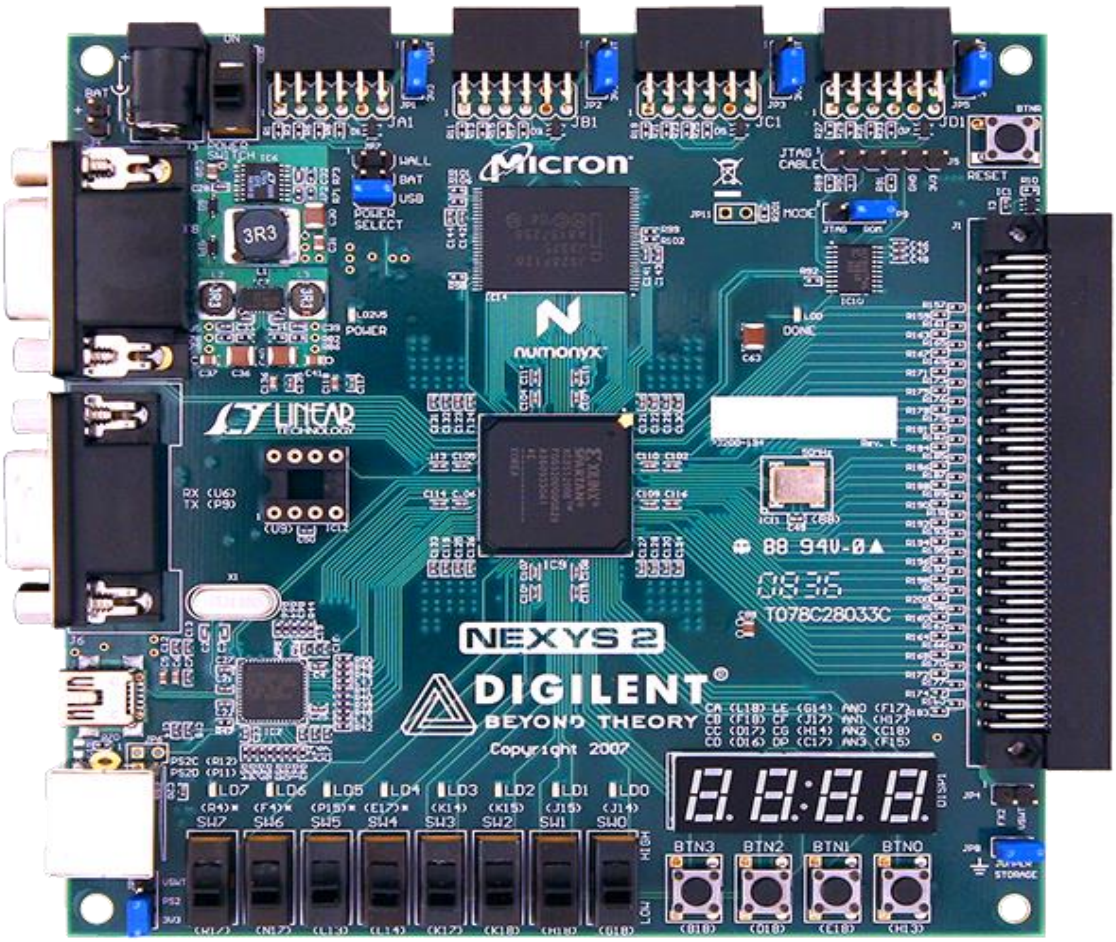


Imagen 6.29:FPGA Spartan3E-XC3S1200E FG320-4 de Xilinx Spartan3E [20]

Xilinx es una compañía de tecnología americana, de gran prestigio en la fabricación de tarjetas reprogramables tipo FPGA. Dentro de su familia de productos, se encuentra la línea de bajo coste Spartan, que permite a los estudiantes iniciarse en la programación y desarrollar sus conocimientos. Además, la Universidad de Alcalá de Henares mantiene un convenio con la compañía mediante el cual, Xilinx facilita un software básico, la

aplicación ISE Project Navigator, con el que realizar los primeros diseños en un entorno educativo y de bajo presupuesto.

Las características principales de la placa mostrada en la imagen anterior son:

1. Está conformada por 500K puertas lógicas.
2. Alimentación por vía USB siendo posible el uso de baterías.
3. Cuenta con 16MB de memoria RAM&ROM.
4. Interruptores de conmutación eficientes.
5. Oscilador interno de 50MHz más una conexión para un segundo oscilador.
6. 60 conexiones de entrada/salida
7. 8 leds, 4 displays de 7 segmentos, 4 pulsadores y 8 interruptores.

Las entradas y salidas utilizadas para la simulación en placa del diseño VHDL han sido los mostrados en las imágenes 6.30 y 6.31:

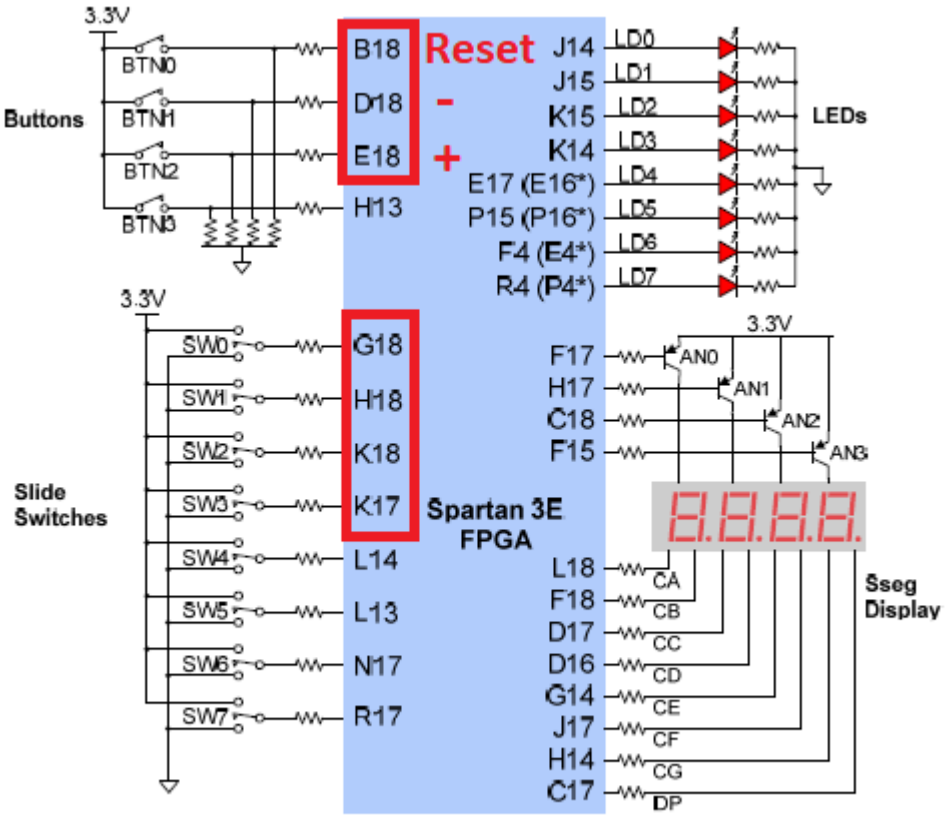
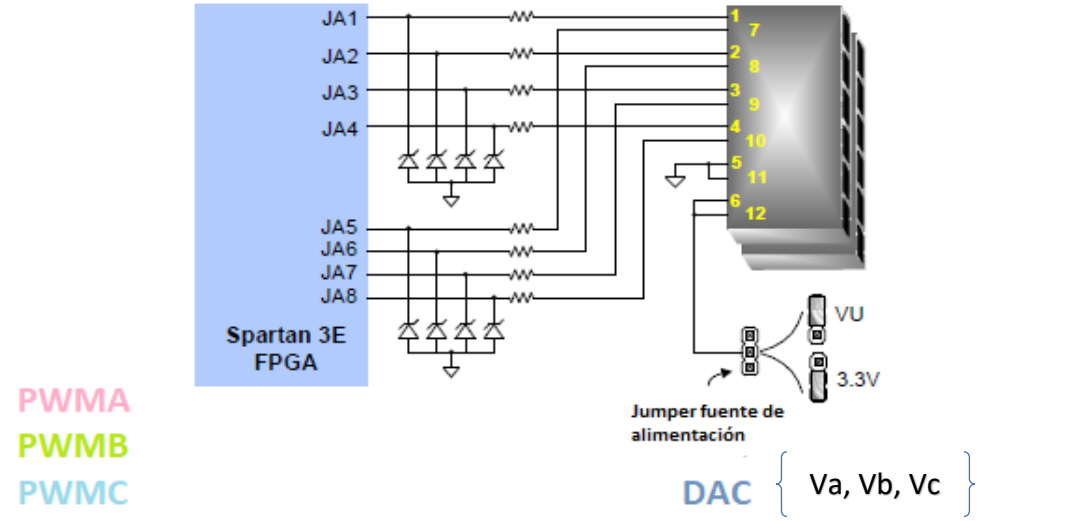


Imagen 6.30: Nexys 2, Entradas/Salidas y conexiones [20]

- Los displays van a representar los valores de los tres parámetros principales de la modulación: el índice de modulación amplitud *Ma*, frecuencia de la señal de carga *F1* y el índice de modulación de frecuencia *Mf*.

- Los interruptores o switches usados se utilizan para modificar dichos parámetros en plena simulación. Las combinaciones y su modo de funcionamiento quedan detallados en el manual de usuario, sección 9.
- El uso de los Leds no ha sido necesario en el diseño final, a pesar de su gran utilidad en simulaciones intermedias para visualizar el funcionamiento de señales.

Los conectadores periféricos encargados de sacar las señales moduladoras al inversor y la señal trifásica de la carga al DAC son los siguientes:



Conectores Pines PMOD							
Pmod JA		Pmod JB		Pmod JC		Pmod JD	
JA1: L15	JA7: K13	JB1: M13	JB7: P17	JC1: G15	JC7: H15	JD1: J13	JD7: K14[1]
JA2: K12	JA8: L16	JB2: R18	JB8: R16	JC2: J16	JC8: F14	JD2: M18	JD8: K15[2]
JA3: L17	JA9: M14	JB3: R15	JB9: T18	JC3: G13	JC9: G16	JD3: N16	JD9: J15[3]
JA4: M15	JA10: M16	JB4: T17	JB10: U18	JC4: H16	JC10: J12	JD4:: P18	JD10: J14[4]

Imagen 6.31: Nexys 2, Conectores Pmod [20]

La asignación de cada pin queda detallada en el apartado siguiente. Se han asignado 6 pines para las señales moduladoras, tal y como se indica en la figura, y 8 pines para conectar los dos puertos DAC.

6.3.1.2 Cubo de potencia Semiteach-IGBT

El cubo de Potencia Semiteach-IGBT [7.3] es un dispositivo constituido por componentes electrónicos de potencia, con conectores de seguridad tipo banana y BNC y diferentes sistemas de seguridad que hacen su funcionalidad puramente educativa. Ha sido diseñado para un uso docente donde los errores son habituales, por ello, tiene una alta protección a fallos.

Imagen 6.32: Cubo de potencia Semiteach-IGBT [7.3]

El uso de un banco de transistores IGBT como interruptores de potencia, mostrado en la siguiente imagen, es adecuado para velocidades de conmutación de alrededor de 100 KHz. Este será un dato de gran importancia para entender los resultados finales.

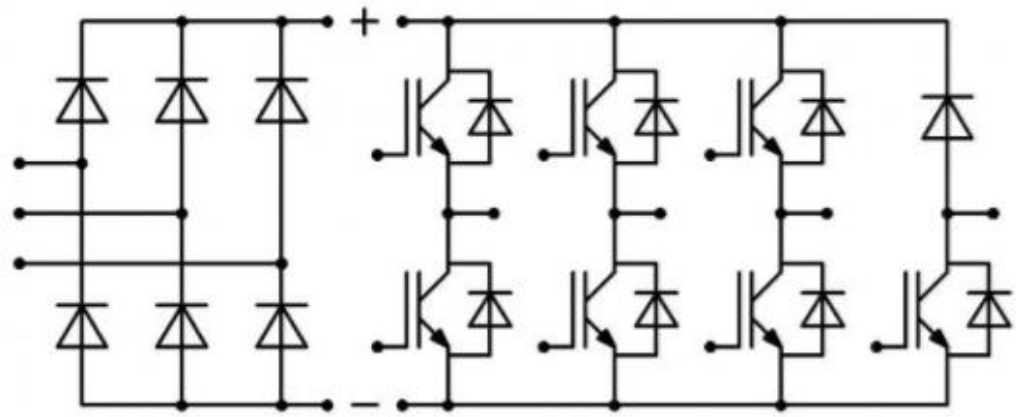


Imagen 6.33: Esquema del cubo de potencia [7.3]

Las conexiones quedan definidas en la tabla 6.8, extraída del TFG de E.Díaz debido a la similitud del material hardware utilizado en ambos trabajos:

Nº	Tipo	Función	Nivel de Voltaje	Máximo nivel de corriente
0	Tierra del panel	Conexión tierra	0	30A
1	Conector Banana 4mm	Fuente de alimentación del ventilador	230V/50Hz	1A
2	Conector Banana 4mm	Termal trip	15V	5A
3	Conector Banana 4mm	Entrada rectificador	230/ 400V	30A
4	Conector Banana 4mm	Salida DC rectificador	600VDC (rojo positivo, azul negativo)	30A
5	Conector Banana 4mm	Entradas DC del inversor IGBT	600VDC (rojo positivo, azul negativo)	30A
6	Conector Banana 4mm	Inversor IGBT AC + salidas choppers	400 VAC / 600 VDC	30A
7	Coaxial aislado BNC	Entrada PWM inversor	C-MOS lógico 0/15V, 0V= IGBT abierto, 15V=IGBT cerrado	1A
8	Coaxial aislado BNC	Entrada PWM chopper	C-MOS lógico 0/15V, 0V= IGBT abierto, 15V=IGBT cerrado	1A
9	Coaxial aislado BNC	Error de salida	C-MOS lógico 0/15V	1A
10	Conector Banana 4mm	15V Alimentación del driver	15V	5A
11	Conector Banana 4mm	0V Alimentación del driver	15V	5A
12	Conector Banana 4mm	Sensor de temperatura	0-5V	1A

Tabla 6.8: Conexiones del cubo de potencia Semiteach-IGBT [6]

Las conexiones necesarias para el diseño han sido la 1,5,6,7,10 y 11.

A pesar de sus medidas de seguridad, es recomendable utilizar el cubo bajo la supervisión de un tutor, por ello, se añadirá en el manual de usuario las especificaciones clave a seguir para evitar su deterioro.

6.3.1.3 Tarjeta de adaptación

La tarjeta de adaptación consiste en un circuito integrado en una placa board diseñada en el Trabajo de Fin de Grado de "Configuración y prueba de convertidores DC/DC para prácticas de laboratorio" desarrollado por C. de Evan [15]. Este diseño tiene la funcionalidad de ampliar las señales PWM emitidas por la FPGA de 3,3V a nivel alto a

15V para poder excitar los transistores IGBTs contenidos en el cubo de potencia. Además, aísla la tarjeta FPGA de la parte de potencia. Las características más destacables para su uso son las siguientes:

- Tensión máxima de alimentación 20VDC.
- Integrado por 2 chips HCPL-2231.
- Bajo consumo de corriente (1.6mA a 1.8mA)

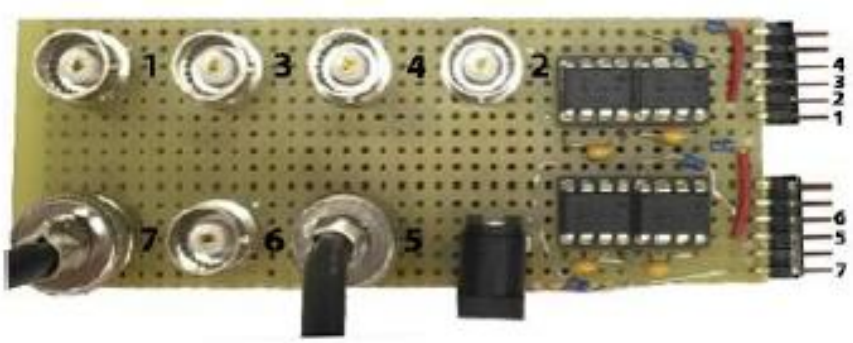


Imagen 6.34: Circuito de Adaptación [15]

La asignación numérica de puertos mostrada en la imagen 6.36 y en la tabla 6.9, relaciona las señales moduladoras de entrada con sus homólogas amplificadas. La enumeración seguida, es la misma que en el TFG de E.Diaz por especificaciones del diseño, para mantener una linealidad de entrada y salida en los proyectos.

Salida	Señal PWM
1	TA+
2	TA-
3	TB+
4	TB-
5	TC+
6	TC-

Tabla 6.9: Asignación de pines del circuito de adaptación

6.3.1.4 DAC

El siguiente componente utilizado en el diseño es el convertidor digital-analógico PmodDA2 de Digilent, siendo su referencia en Texas Instruments DAC121S101 [18].

Imagen 6.35: Imagen del DAC PmodDA" de Digilent [18]

Sus características principales son las siguientes:

- Convertidor digital-analógico de 12 bits.
- Dos canales de conversión simultánea.
- Bajo consumo
- Diseño compacto (2.5 cm x 2.0cm) para diseños flexibles.
- Cuenta con 6 pines Pmod con interfaz GPIO.
- Resolución de aproximadamente 1mV.

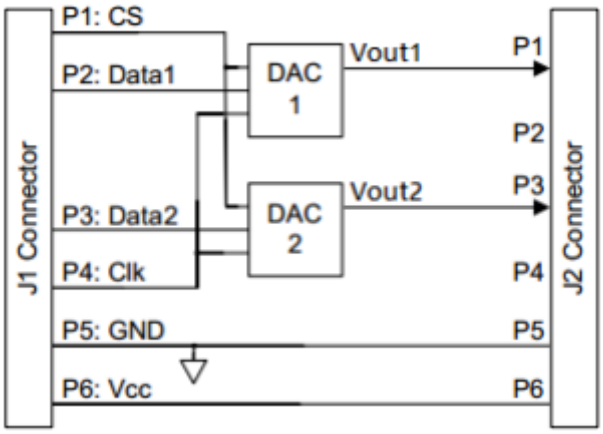


Imagen 6.36: Esquema de conexiones PmodDA2 de digilent [18]

El DAC es utilizado para la visualización en el osciloscopio de la señal trifásica moduladora generada internamente en la FPGA. Sus conexiones de entrada, *J1 Connector*, van unidas a los conectores Pmod de la imagen 6-37, y su salida, *J2 Connector* es conectada directamente al osciloscopio.

Al tener únicamente dos canales de conversión, fueron utilizados dos dispositivos simultáneamente para la visualización de las tres señales.

PIN	SEÑALES DE SALIDA FPGA	SEÑALES DE SALIDA DAC
P1	SYNC	Vout1
P2	DINA	N/C
P3	DINB	Vout2
P4	SCLK	N/C
P5	GND	GND
P6	VCC	VCC

Tabla 6.10: Asignación de pines DAC

6.3.1.5 Carga

La única carga probada a la salida del inversor ha sido una bombilla de baja potencia por fase. Su bajo presupuesto y su facilidad visual para observar el funcionamiento del diseño la hacen el elemento útil para evaluar de forma rápida y visual las pruebas prácticas realizadas.

Imagen 6.37: Carga de baja potencia, Bombilla [6]

6.3.1.6 Diseño final

La conexión final del diseño es la mostrada en la siguiente imagen:

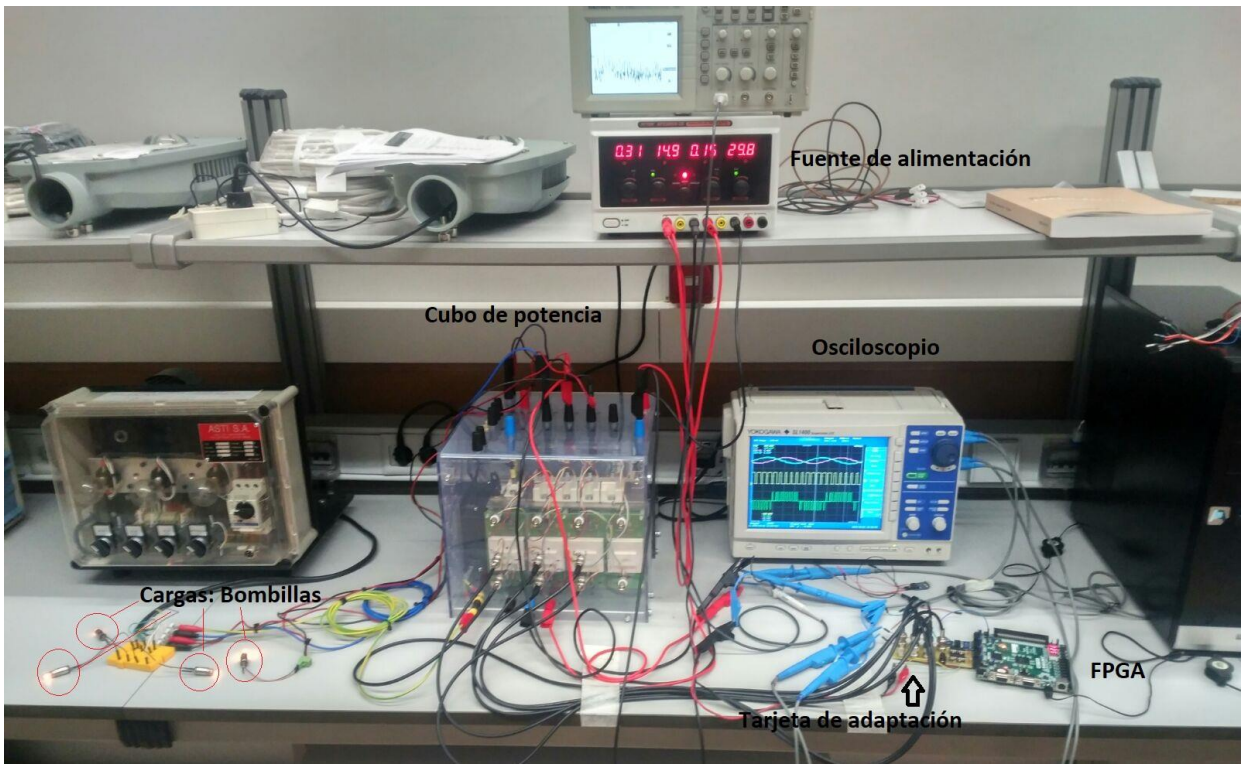


Imagen 6.38: Diseño final de la simulación práctica

Todo el material hardware utilizado fue cedido por el departamento de Electrónica de la Escuela Politécnica Superior de la Universidad de Alcalá, lugar donde se llevó a cabo el trabajo.

6.3.2 Implementación VHDL

Una vez explicados los componentes físicos del diseño, se continuará con la estrategia seguida para la programación del código en VHDL, mediante la representación de los diagramas de bloques que se han utilizado. Para ello, se van a describir los módulos principales y la parte de código con mayor relevancia.

6.3.2.1 Módulo de mayor jerarquía

El código construido ha sido dividido en múltiples bloques para facilitar su acceso y entendimiento. La entidad de mayor jerarquía generada mostrada en la siguiente imagen está compuesta de cuatro módulos: Displays, Entradas, VaVbVc y Modulación.

A continuación, se van describir brevemente la función de cada uno de ellos, así como la asignación de las señales de entrada y salida realizada.

Todos los diagramas de bloques que van a ser mostrados en este apartado siguen la nomenclatura del código. Las señales y módulos dibujados son los principales para la explicación del programa, de forma que no son las únicas que aparecen en el diseño completo.

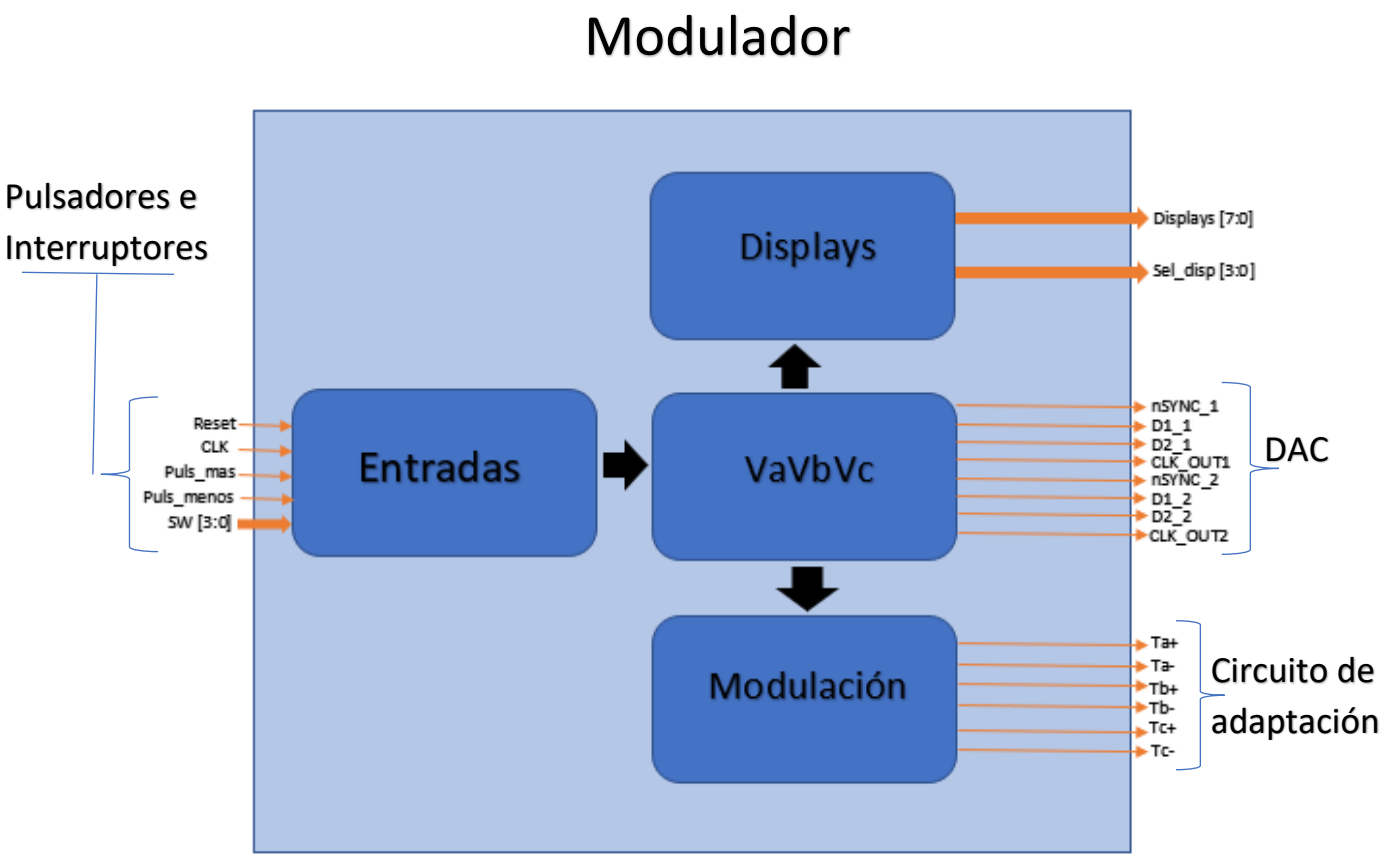


Imagen 6.39: Diagrama de bloques de mayor jerarquía

El módulo Entradas, es el encargado de recibir todas las señales de entrada de la tarjeta y procesarlas para su posterior uso en el resto del programa. La señal CLK es el oscilador interno que proporciona la FPGA de 50MHz. Los interruptores utilizados han sido los cuatro primeros empezando por la derecha, según la imagen 6.29.

SEÑALES DE ENTRADA		
Funcionalidad	**Señales del diseño**	**Pines FPGA**
SEÑAL RESET	**Reset**	BTN0:B18
OSCILADOR 50 MHZ	**CLK**	GCLK: B8
AUMENTA PARÁMETRO	**Pulsador_mas**	BTN2:E18
DISMINUYE PARÁMETRO	**Pulsador_menos**	BTN1:D18
CONFIG MF	**SW(0)**	SW0:G18

CONFIG MA	**SW(1)**	SW1:H18
CONFIG F1	**SW(2)**	SW2:K18

Tabla 6.11: Asignación de pulsadores e interruptores

El siguiente bloque que tiene el diseño es VaVbVc. En él, se generan las 3 señales senoidales y se controlan los tres parámetros fundamentales de la modulación. Además, contiene el módulo DA2refComp que procesa los senos para enviarlos a las salidas donde estará conectado el DAC.

SEÑALES DE SALIDA: VAVBVC		
Funcionalidad	**Señales del diseño**	**Pines FPGA**
SEÑAL DE SINCRONIZACIÓN DEL DAC1	**nSYNC_1**	JC1:G15
DATOS DE ENTRADA AL DAC1	**D1_1**	JC2:J16
DATOS DE ENTRADA AL DAC2	**D2_1**	JC3:G13
SEÑAL DE RELOJ DEL DAC1	**CLK_OUT1**	JC4:H16
SEÑAL DE SINCRONICZACIÓN DEL DAC2	**nSYNC_2**	JD1:J13
DATOS DE ENTRADA AL DAC2	**D2_1**	JD2:M18
DATOS DE ENTRADA AL DAC2	**D2_2**	JD3:N18
SEÑAL DE RELOJ DEL DAC2	**CLK_OUT2**	JD4:P18

Tabla 6.12: Asignación de pines DAC-FPGA

Displays, es el bloque encargado de la gestión tanto de displays como de leds, en este caso no se ha utilizado ningún led, pero sería el bloque asignado para implementar su control.

SEÑALES DE SALIDA: DISPLAYS	
SEÑALES DEL DISEÑO	**Pines FPGA**
DISPLAY(0)	CG:H14
DISPLAY(1)	CF:J17

DISPLAY(2)	CE:G14
DISPLAY(3)	CD:D16
DISPLAY(4)	CC:D17
DISPLAY(5)	CB:F18
DISPLAY(6)	CA:L18
DISPLAY(7)	DP:C17
SEL_DISP(0)	AN0:F17
SEL_DISP(1)	AN1:H17
SEL_DISP(2)	AN2:C18
SEL_DISP(3)	AN3:F15

Tabla 6.13: Asignación de pines de los displays

Por último, en el bloque Modulación están localizadas todas las cuentas necesarias relacionadas con la modulación vectorial del inversor y, por consiguiente, donde se generan las señales moduladoras PWM que controlarán las conmutaciones de los interruptores en el bloque de potencia.

SEÑALES DE SALIDA: MODULACIÓN	
SEÑALES DEL DISEÑO	Pines FPGA
TA+	JB7:P17
TA-	JB8:R16
TB+	JB9:T18
TB-	JB10:U18
TC+	JA9:M14
TC-	JA10:M16

Tabla 6.14: Asignación pines señales moduladoras

6.3.2.2 Módulo Entradas

Las señales producidas por los puertos de la placa no siempre funcionan de la manera correcta, por ello, es necesario su procesado antes de su uso en el programa principal. Este bloque tiene cuatro objetivos:

1) Filtrar la señal de reloj interna que proporciona la placa para reducir el *jitter*, el *skew* y mejorar el rendimiento del proceso.
2) Eliminar el efecto antirrebotes que se produce en el accionamiento de los pulsadores.

3) Codificar los interruptores en un array para su implementación en la modulación
4) Generar dos temporizadores para los displays

El diagrama de bloques simplificado que muestra el funcionamiento es el siguiente:

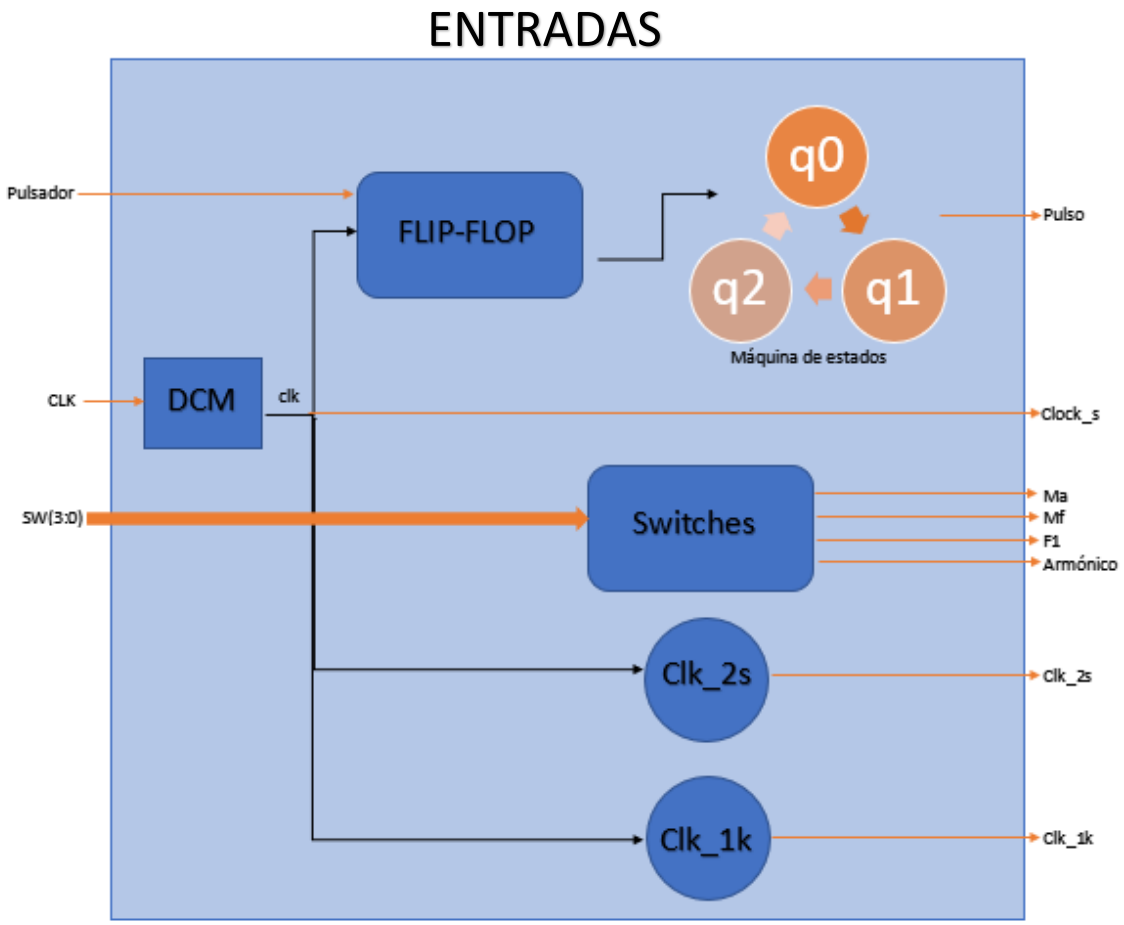
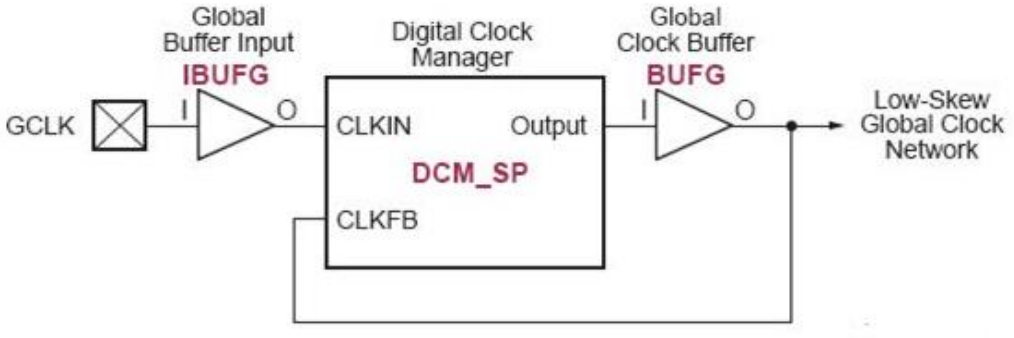


Imagen 6.40: Diagram de bloques ENTRADAS

La señal de reloj es filtrada por el DCM (*Digital Clock Manager*). La FPGA nexys 2 cuenta con ocho bloques internos DCM cuyo diagrama de bloque se expone a continuación. La señal de reloj a su salida será la utilizada en el resto de módulos del diseño



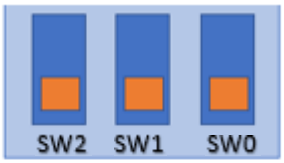
Para los pulsadores de entrada, se ha utilizado un sistema antirrebotes que detecta la señal por flanco de subida y genera un único pulso por activación:

- Flip Flop es un biestable con un retardo de activación cuya función es cambiar de estado transcurrido un tiempo para evitar el periodo de transición y los posibles rebotes que se producen en él.
- Después, se encuentra una pequeña máquina de estados que se activa a nivel alto y se desactiva a nivel bajo, de forma, que solo genera un único pulso de periodo CLK, por cada accionamiento de los botones de la FPGA.

Los dos temporizadores construidos, Clk_2s y Clk_1k, son de dos segundos y un milisegundo respectivamente:

- Clk_1k conmutará la activación de los displays cada un milisegundo, de forma, que estarán parpadeando a 1KHz, inapreciable para el ojo humano.
- Clk_2s permuta la visualización del parámetro de simulación con su valor correspondiente.

Por último, el módulo Switches recibe la señal de cada interruptor y la envía al diseño en un bus de datos. Por la disponibilidad de recursos suficientes en la placa, se ha podido utilizar un interruptor por cada parámetro, dejando un estado de reposo donde los pulsadores no tienen ningún efecto en el diseño.



Configuración interruptores	Parámetros
	Mf
	Ma
	F1

Tabla 6.15: Configuración interruptores

6.3.2.3 Módulo Displays

El módulo display recibe en un bus de datos, las direcciones de cada carácter que se va a representar en la salida.

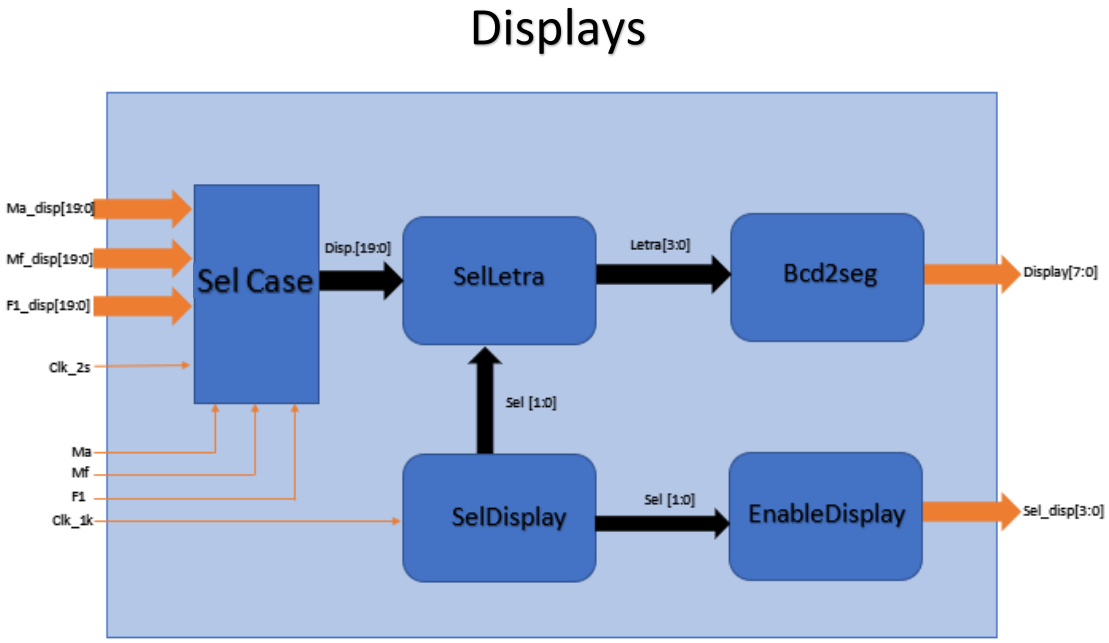


Imagen 6.41: Diagrama de bloques DISPLAYS

Tal y como está representado, SelCase es un módulo que recibe en buses de datos de 20 bits, las cuatro direcciones de las cuatro cifras que serán representadas, además, de recoger las señales de los tres primeros interruptores que configuran el parámetro que se está modificando. Por último, también tiene como entrada la señal de reloj con un pulso cada dos segundos, que conmutará el valor numérico con el parámetro configurado en los displays como se puede ver a continuación:

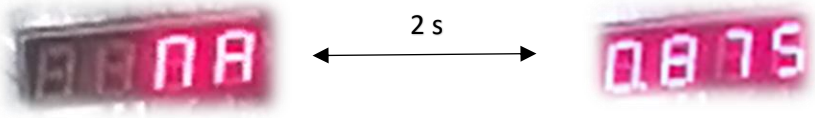


Imagen 6.42: Display en la simulación real

El siguiente módulo, SelDisplay, es el encargado de gobernar la activación de los displays, de forma que, cada un milisegundo un display es activado a nivel bajo por medio del bloque EnableDisplay y su carácter que debe representar es enviado por SelLetra que actúa como un multiplexor, mandando el bus de 4 bits a Bcd2seg donde las direcciones acceden a una tabla de verdad.

6.3.2.4 Módulo VaVbVc

A continuación, se explica cómo se han generado la señal trifásica que conforma la modulación vectorial de los inversores.

El siguiente diagrama de bloques explica los pasos seguidos para la obtención de los senos.

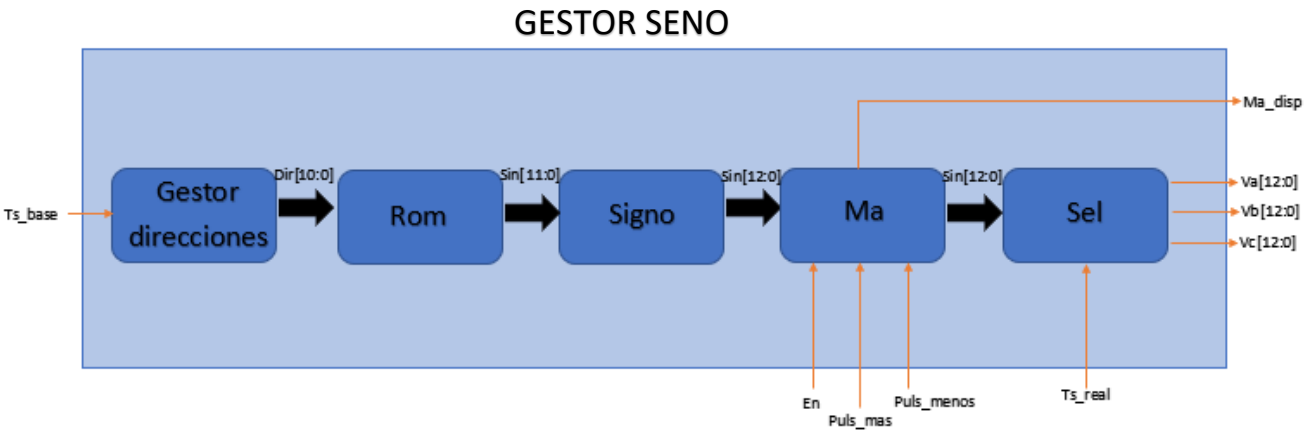


Imagen 6.43: Diagrama de bloques GESTOR SENO

Para simplificar el diseño se decidió trabajar con una única onda senoidal con un periodo de 5760 muestras, almacenadas en memoria Rom. Sin embargo, para ahorrar espacio y gracias a la simetría par que presentan los senos, solo se ha almacenado un cuarto de la onda, la zona roja en la imagen 6-46 , en 1440 muestras.

El bloque Gestor direcciones, se encarga de generar el vector que irá recorriendo la memoria y activando una señal de signo según el cuadrante en el que se encuentre cada muestra, tal y como se muestra en el siguiente diagrama de flujo:

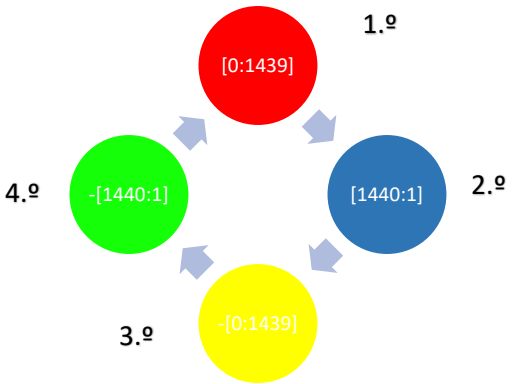


Imagen 6.44: Flujo de datos de la máquina de estados

De esta forma, se accede a la memoria Rom, donde están almacenadas las muestras del primer cuadrante. El desfase de 120º entre las ondas trifásicas se obtiene desfasando los contadores que acceden a la memoria.

La amplitud del seno está normalizada a raíz de 3 para evitar el uso de raíces en los cálculos posteriores de la modulación:

$$Amplitud = \frac{2^{12}}{\sqrt{3}} = 2365 \qquad <6.36>$$

El seno obtenido es el siguiente, donde los colores diferencian los cuatro sectores, similares a la máquina de estados anterior.

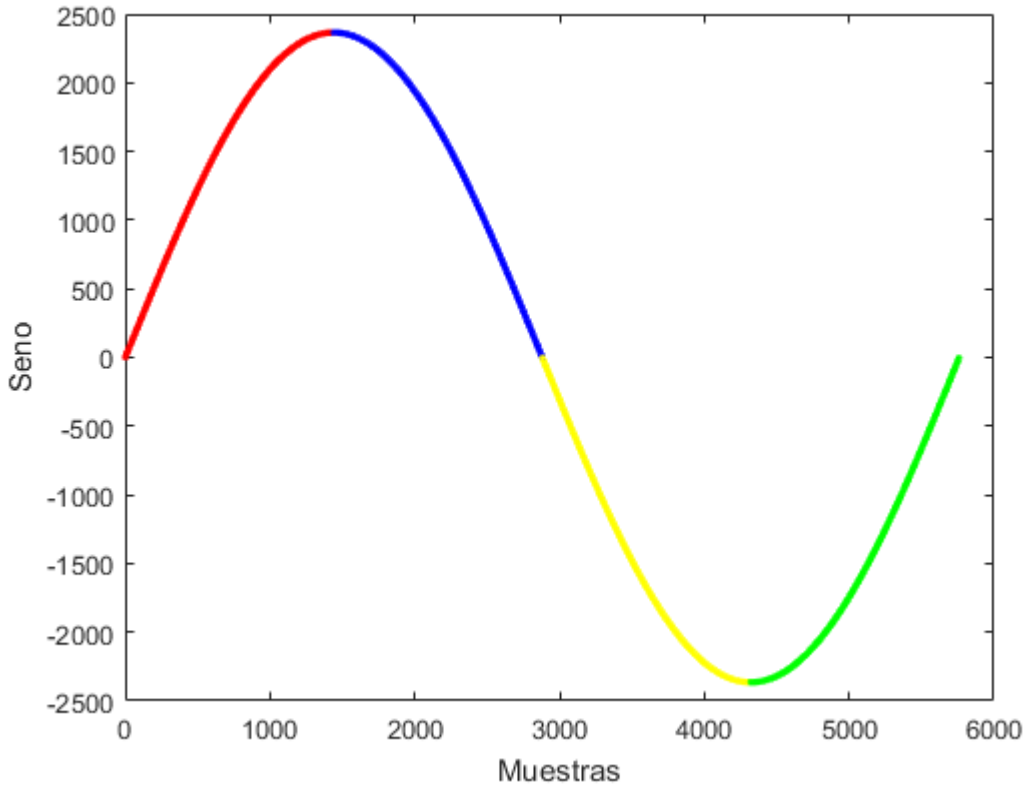


Imagen 6.45: 4 sectores de una onda senoidal

Siguiendo con el diagrama 6.44, el módulo signo, como su nombre indica, añade el signo a cada muestra duplicando el último bit en el caso de ser positivo, o convirtiéndolo a complemento 2 en caso de ser negativo.

El bloque Ma es el encargado de gestionar el índice de modulación de amplitud y de aplicarlo a las muestras. A él, llegan las señales de los pulsadores generadas por el módulo Entradas y una señal *enable* que se activa dependiendo de la secuencia de los interruptores de la FPGA. Los índices con los que se podrá trabajar son los siguientes:

Índices de modulación de amplitud: Ma								
1	0.875	0.75	0.625	0.5	0.25	0.125	0.03125	0.015625

Tabla 6.16: Índices de modulación de amplitud: Ma

Para evitar el uso de un multiplicador y trabajar con decimales, se han escogido modulaciones fracciones de 2, de tal forma que se pueda realizar mediante un desplazamiento de bits.

Los índices de modulación que no son fracciones de 2 se han formado igualmente por desplazamiento de bits, sumando los índices de menor grado, por ejemplo:

$$0.875 = 0.5 + 0.25 + 0.125 \qquad <6.37>$$

La señal Ts_Base marca el tiempo de muestreo mínimo que al que trabajará el diseño, siendo la inversa de ésta, la frecuencia a la que se accederá a la memoria Rom. Por otro lado, Ts_Real establece el tiempo de muestreo real que tendrá la señal trifásica a la salida, actualizando los valores de los senos en el bloque Sel en cada pulso.

El siguiente diagrama muestra la secuencia de bloques utilizados para generar ambas señales:

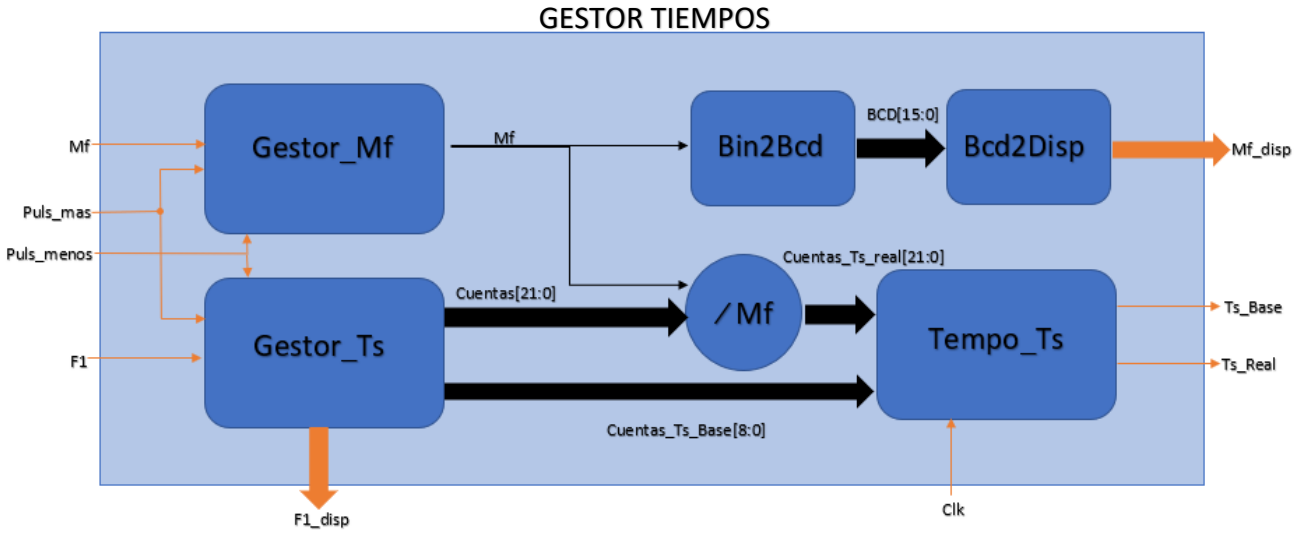


Imagen 6.46: Diagrama de bloques GESTOR TIEMPOS

Gestor_Ts selecciona la frecuencia de la señal *F1* que se desea obtener en la señal trifásica. El número de ciclos de clk de la FPGA de 20MHz que hay que contar para obtener un tiempo Ts Base son:

$$Cuentas\ Ts\ Base = \frac{50MHz}{5760 * F1} \qquad <6.38>$$

Las cuentas del tiempo de muestreo real que se necesita para tener el periodo deseado son:

$$Cuentas\ Ts\ Real = \frac{50MHz}{Mf * F1}$$

Al trabajar con un rango de frecuencia bajo, se ha almacenado en una tabla de verdad las cuentas de Ts_Base y las de Ts Real, sin dividir entre Mf, para cada frecuencia disponible.

Las frecuencias del diseño elegidas que marcarán la frecuencia del armónico fundamental de la señal AC del inversor DC/AC son las siguientes:

FRECUENCIA DE LAS ONDAS TRIFÁSICAS: F1	
FRECUENCIA	**Nº de cuentas Ts_Base**
100	87
90	96
80	109
70	124
60	145
50	174
40	217
30	289
20	434

El bloque Tempo_Ts es el encargado de contar el número ciclos de reloj hasta alcanzar el número de cuentas calculadas anteriormente.

En esta imagen puede observarse como la onda azul es el cuarto de seno base con un periodo de muestreo de $\frac{T1}{5760}$, mientras que los círculos rojos equivalen al cuarto de seno real querido con un índice de modulación de 40 y un tiempo de muestreo de $\frac{T1}{40}$, siendo $T1 = \frac{1}{F1}$.

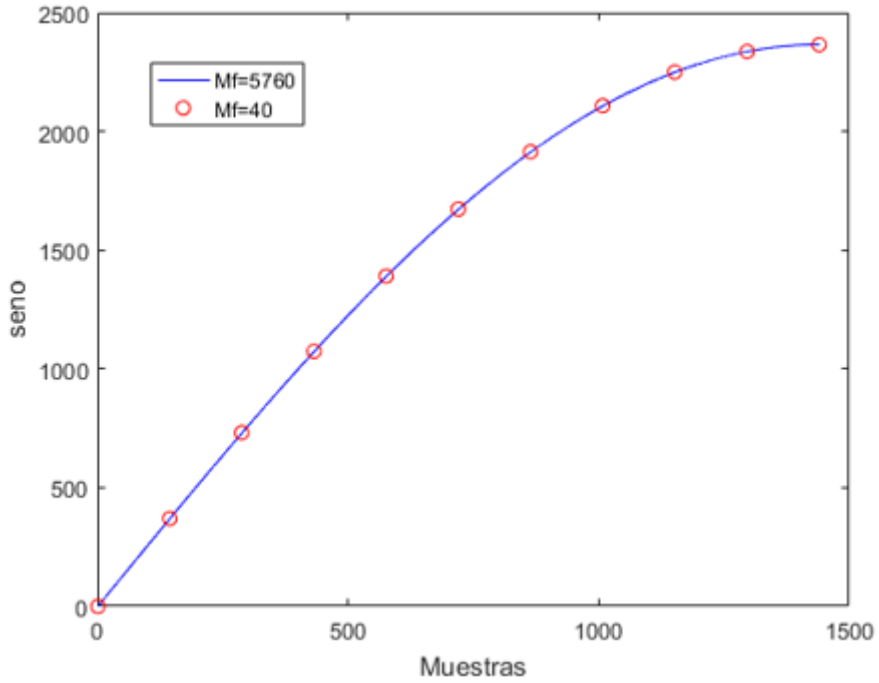


Imagen 6.47: Señal seno base (línea azul, y señal seno real muestreada (puntos)

Una vez explicados las partes principales de este módulo, el bloque general queda de la siguiente forma:

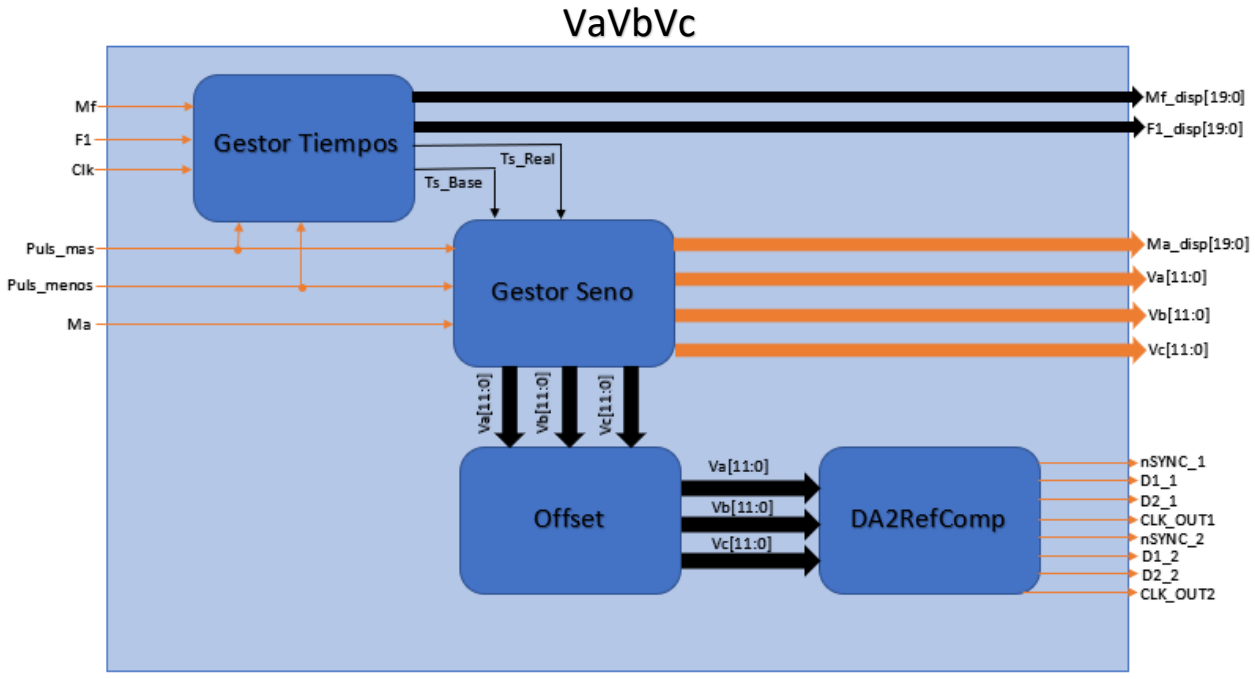


Imagen 6.48: Diagrama de Bloques VaVcVc

Los senos generados y que serán utilizados para los cálculos de la modulación son los siguientes:

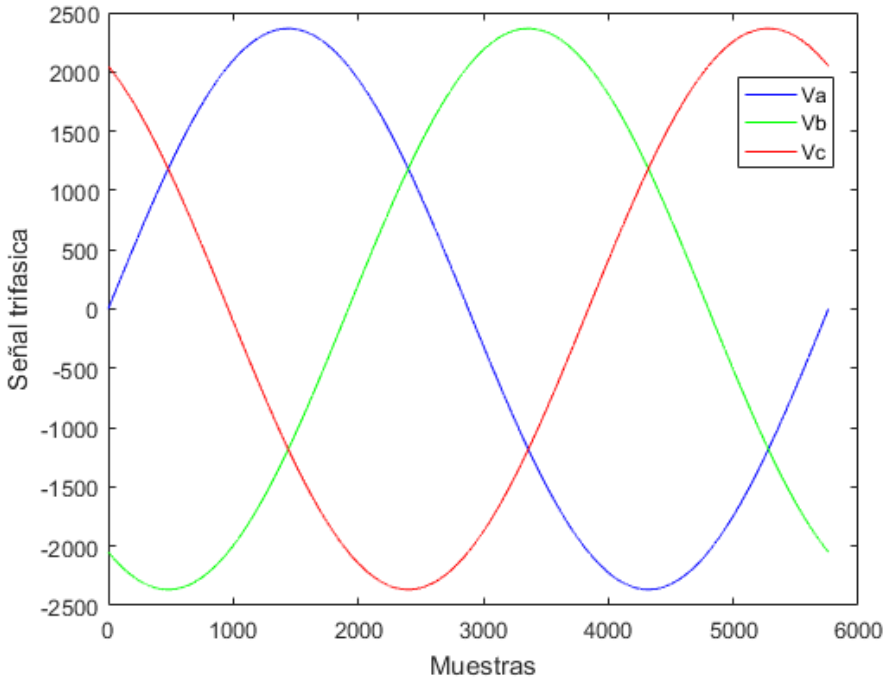


Imagen 6.49: Senos moduladores generados

Para ser representados por el DAC, es necesario ajustarlo a su SPAN y aplicar un offset a las señales para eliminar la parte negativa. Para ello, se ha sumado a cada seno el valor de la amplitud, sin embargo, el DAC utilizado, PmodDA2 de Digilent, es de 12 bits como ya se comentó y el doble de la amplitud necesita 13 bits para ser codificada, por lo que, se ha optado por acotar las salidas a un valor máximo de 4095. De esta forma, los senos que serán enviados a través del DAC con un Ma=1 son:

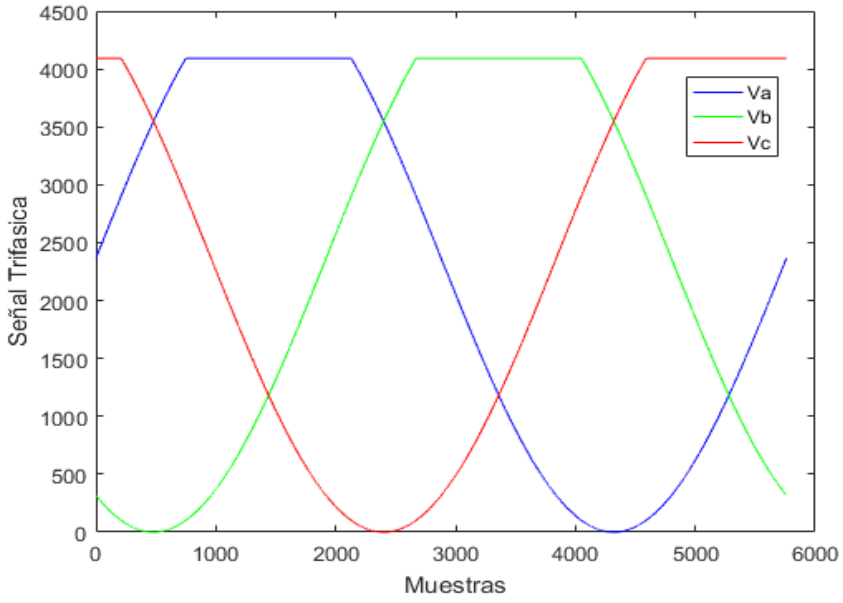


Imagen 6.50: Senos representados por el DAC

Este inconveniente no tiene ningún efecto sobre la modulación general, tan solo es un error visual que puede ser corregido disminuyendo el índice de modulación de amplitud.

6.3.2.5 Bloque Modulación

Por último, el bloque Modulación es el encargado de hacer todas las operaciones y de obtener finalmente las señales moduladoras encargadas de hacer conmutar los interruptores.

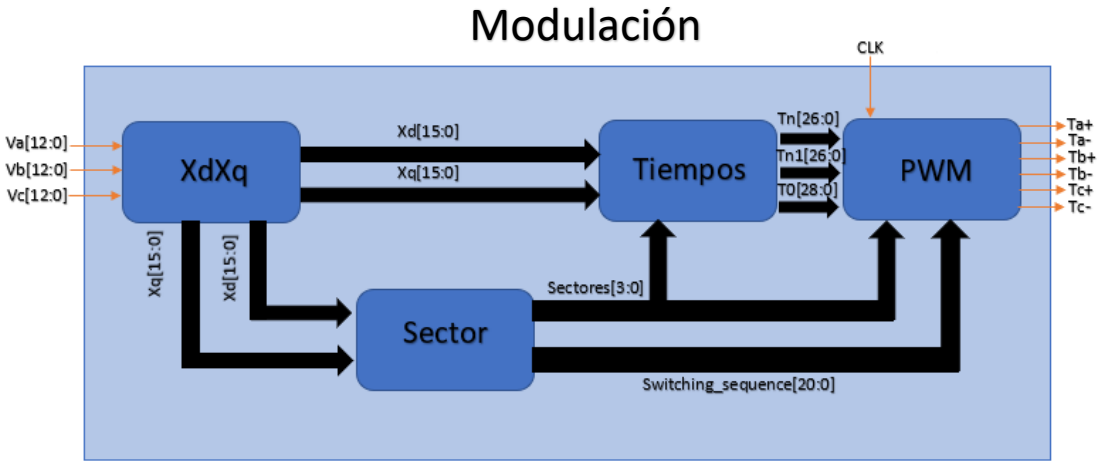


Imagen 6.51: Diagrama de bloques MODULACIÓN

El proceso seguido es el explicado en el apartado 6.2.3, siguiendo el documento citado. Las señales de salida del bloque Tiempos se corresponden ya con las cuentas de ciclos de reloj para obtener dichos tiempos.

Las secuencias de conmutación que siguen las señales moduladoras, es enviado por el bus Switching sequence de 21 bits.

No ha sido necesario incluir tiempos muertos en las señales de salida porque el propio cubo de potencia lo contiene internamente.

6.3.3 Errores cometidos

Para calcular las cuentas utilizadas para el cálculo de los periodos y el tiempo de muestreo, se cometió un error por redondeo al número entero más cercano, además, de la cuantificación realizada en los datos del cuarto de seno introducido para trabajar con coma fija.

Para evaluar los errores se desarrolló el siguiente código en matlab con cuantificadores que simulan los datos utilizados en el diseño.

Errores

```
clear all;
close all;
%Errores Cometidos
%Errores debidos a la frecuencia
q=quantizer('fixed','ufix','wrap',[18 0]);%Quantificación por truncamiento
q1=quantizer('fixed','round','wrap',[12 0]);%Quantificación por truncamiento
f1=20:10:100;
aux1=15:5:100;
aux2=200:100:5700;
aux3=5705:5:5760;
mf=[aux1 aux2 aux3];
ma=0.2:0.1:1;
%Error para baja frecuencia F1(20Hz)
contador_sen1=1./(20e-9*mf*f1(1));
contador_sen2=quantize(q,contador_sen1);
frecuencia_1_error_incial=1./(contador_sen2.*mf*20e-9);
error_en_frecuencia_f1_baja_frecuencia=frecuencia_1_error_incial-f1(1);
figure;
subplot(2,2,1);
plot(mf,error_en_frecuencia_f1_baja_frecuencia,'m');
title('F1 20Hz')
xlabel('mf');
ylabel('error de f1 en Hz');
hold on;
%Error  en media frecuencia F1(50Hz)
contador_sen1=1./(20e-9*mf*f1(4));
contador_sen2=quantize(q,contador_sen1);
frecuencia_1_error_incial=1./(contador_sen2.*mf*20e-9);
error_en_frecuencia_f1_media_frecuencia=frecuencia_1_error_incial-f1(4);
subplot(2,2,2);
plot(mf,error_en_frecuencia_f1_media_frecuencia,'m');
title('F1 50Hz')
xlabel('mf');
ylabel('error de f1 en Hz');
hold on;
%Error para media-alta frecuencia F1(70Hz)
contador_sen1=1./(20e-9*mf*f1(6));
contador_sen2=quantize(q,contador_sen1);
frecuencia_1_error_incial=1./(contador_sen2.*mf*20e-9);
error_en_frecuencia_f1_media_alta_frecuencia=frecuencia_1_error_incial-f1(6);
subplot(2,2,3);
plot(mf,error_en_frecuencia_f1_media_alta_frecuencia,'m');
title('F1 70Hz')
xlabel('mf');
ylabel('error de f1 en Hz');
hold on;
%Error  en media frecuencia F1(100Hz)
contador_sen1=1./(20e-9*mf*f1(9));
contador_sen2=quantize(q,contador_sen1);
frecuencia_1_error_incial=1./(contador_sen2.*mf*20e-9);
error_en_frecuencia_f1_alta_frecuencia=frecuencia_1_error_incial-f1(9);
subplot(2,2,4);
plot(mf,error_en_frecuencia_f1_alta_frecuencia,'m');
title('F1 100Hz')
xlabel('mf');
ylabel('error de f1 en Hz');
hold on;
%Error de la frecuencia en porcentaje
figure;
```

```
subplot(2,2,1);
plot(mf,100*(error_en_frecuencia_f1_baja_frecuencia/f1(1)),'m');
title('F1 20Hz')
xlabel('mf');
ylabel('error de f1 en %');
hold on;
subplot(2,2,2);
plot(mf,100*(error_en_frecuencia_f1_media_frecuencia/f1(4)),'m');
title('F1 50Hz')
xlabel('mf');
ylabel('error de f1 en %');
hold on;
subplot(2,2,3);
plot(mf,100*(error_en_frecuencia_f1_media_alta_frecuencia/f1(6)),'m');
title('F1 70Hz')
xlabel('mf');
ylabel('error de f1 en %');
hold on;
subplot(2,2,4);
plot(mf,100*(error_en_frecuencia_f1_alta_frecuencia/f1(9)),'m');
title('F1 100Hz')
xlabel('mf');
ylabel('error de f1 en %');
hold on;
%Errores debidos a la cuantificacion de los senos
 Vdc=(2^12);
 f1=100;
 T1=1/f1;
 contador_T1=T1/(20e-9);
 mf=100;
 fs=mf*f1;
 Ts=f1/5760;
 t=0:(1/f1)/mf:1/f1-1/f1/mf;
 va=(1*Vdc/sqrt(3))*sin(2*pi*f1*t);%Como se comentó, introducimos aquí el Raiz de tres par
a evitar trabajar con el resto del programa
 q=quantizer('fixed','ceiling','saturate',[13 0]);
 va=quantize(q,va);
 vaVhdl=[va(1:(end/4)+1) fliplr(va(2:end/4)) -va(1:(end/4)+1) -fliplr(va(2:end/4))];
 Ma=[1 0.5 0.25 0.125 0.03125 0.015625];
 VaVhdl=[];
 q1=quantizer('fixed','floor','saturate',[13 0]);
 for i=1:1:6
     VaVhdl(i,:)=quantize(q1,vaVhdl*Ma(i));
 end;

 VaVhdl(7,:)=VaVhdl(4,:);
 VaVhdl(8,:)=VaVhdl(5,:);
 VaVhdl(9,:)=VaVhdl(6,:);
 VaVhdl(6,:)=VaVhdl(3,:);
 VaVhdl(5,:)=VaVhdl(2,:);
 VaVhdl(4,:)=VaVhdl(5,:)+VaVhdl(7,:);
 VaVhdl(3,:)=VaVhdl(5,:)+VaVhdl(6,:);
 VaVhdl(2,:)=VaVhdl(5,:)+VaVhdl(6,:)+VaVhdl(7,:);

 Ma=[1 0.875 0.75 0.625 0.5 0.25 0.125 0.03125 0.015625];
 for i=1:1:9
 VAVHDL(i,:)=(Ma(i)*vaVhdl);
 Va(i,:)=(Ma(i)*Vdc/sqrt(3))*sin(2*pi*f1*t);
 VAA(i,:)=quantize(q1,[Va(i,1:(end/4)+1) fliplr(Va(i,2:end/4)) -Va(i,1:(end/4)+1) -fliplr(
Va(i,2:end/4))]);
 end;
```

```
Error_Va=[];
Error_Va_porcen=[];
Error_Va_total=[];
Error_Va_total_porcen=[];
Error_Vc=[];
contA=0;
contA_porcentaje=0;
cont_cuanti=0;
%Calculo del error cometido
for i=1:1:9
    contA=0;
    contA_porcentaje=0;
    cont_cuanti=0;
    for j=1:1:mf
    Error_cuanti(i,j)=abs(abs(Va(i,j))-abs(VAA(i,j)));
    Error_cuanti_porcentaje(i,j)=abs(abs(Va(i,j))-abs(VAA(i,j)))*100/((ma(i)*2365));
    contA=contA+abs(abs(VAVHDL(i,j))-abs(VaVhdl(i,j)));
    contA_porcentaje=contA_porcentaje+abs(abs(VAVHDL(i,j))-abs(VaVhdl(i,j)))*100/(ma(i)*2
365);
    end;
    Error_Va(i)=contA/mf;
    Error_Va_porcen(i)=contA_porcentaje/mf;
end
hold on;
muestras=1:1:mf;
figure;
subplot(2,1,1);
plot(Ma,Error_Va,'r')
subplot(2,1,2);
plot(Ma,Error_Va_porcen,'r')
figure;
subplot(2,2,1)
plot(muestras,Error_cuanti(1,:))
subplot(2,2,2)
plot(muestras,Error_cuanti(3,:))
subplot(2,2,3)
plot(muestras,Error_cuanti(5,:))
subplot(2,2,4)
plot(muestras,Error_cuanti(7,:))
figure;
subplot(2,2,1)
plot(muestras,Error_cuanti_porcentaje(1,:))
subplot(2,2,2)
plot(muestras,Error_cuanti_porcentaje(3,:))
subplot(2,2,3)
plot(muestras,Error_cuanti_porcentaje(5,:))
subplot(2,2,4)
plot(muestras,Error_cuanti_porcentaje(7,:))
```

Published with MATLAB® R2016b

Código 6-2: Cálculo de errores de cuantificación en Matlab

Error de la frecuencia fundamental

El error evaluado abarca el redondeo de las cuentas de ciclos de reloj del Ts_base junto al truncamiento producido al dividir por *Mf*.

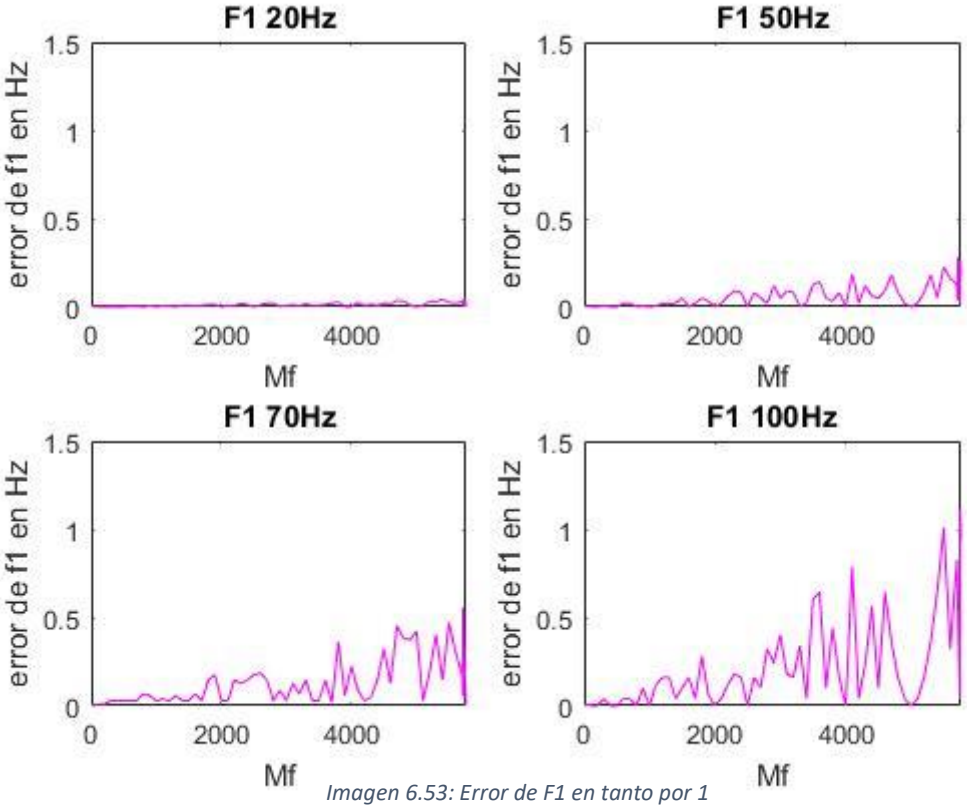


Imagen 6.53: Error de F1 en tanto por 1

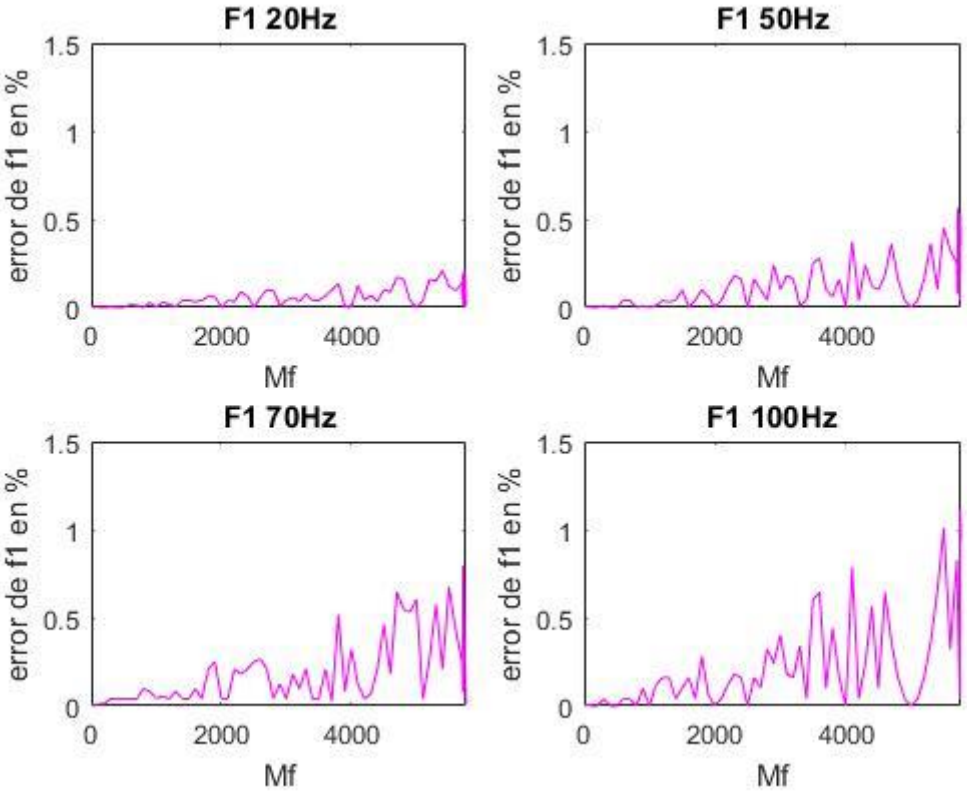


Imagen 6.52: Error de F1 en tanto por 100

Como puede observarse en las gráficas anteriores, el error aumenta progresivamente con Mf. Una forma de corregirlo es ajustar las cuentas de Ts_Base antes de ser divididas por Mf, de forma que, para un índice de modulación de 5760, el error sea nulo.

El error de la frecuencia fundamental no tiene gran relevancia en el diseño final, además, de ser de pequeña magnitud como muestran las gráficas, no afecta en los cálculos de la modulación.

Error de amplitud

A continuación, se ha calculado el error medio de todas las muestras para cada índice de modulación implementado, cometido por el truncamiento de los decimales al generar las muestras y al realizar el desplazamiento de bits en la multiplicación por Ma.

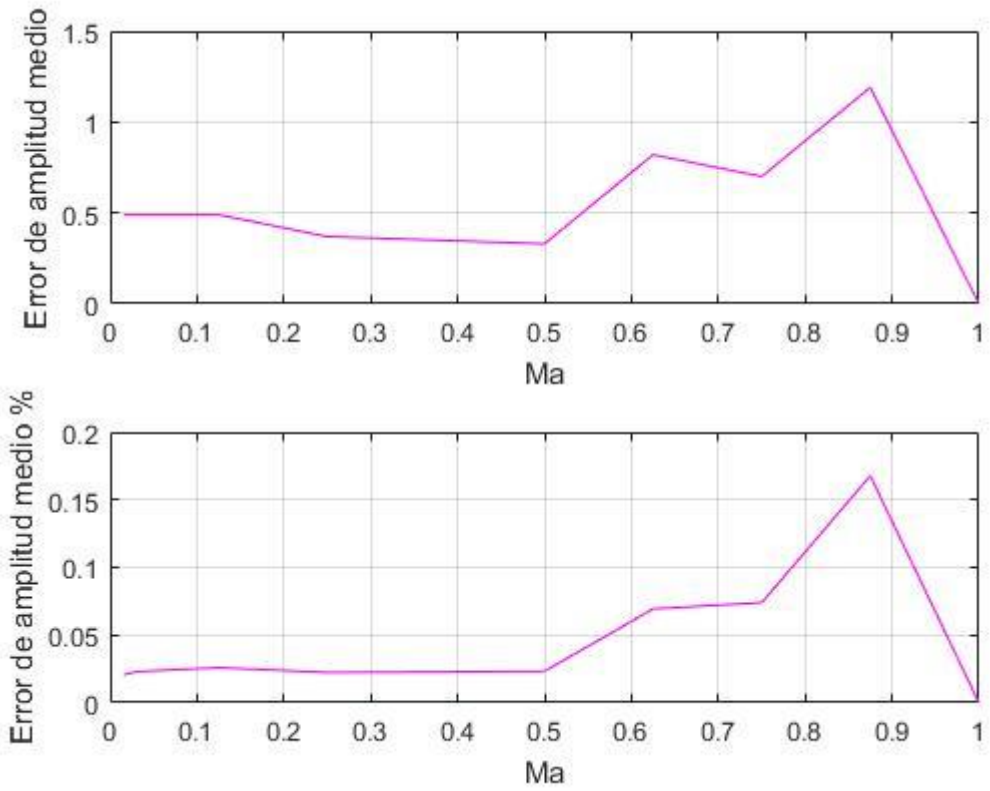


Imagen 6.54: Error de amplitud medio de cada muestra en un ciclo

Como era de esperar, para un índice de modulación igual a 1, el error es 0, ya que, las muestras fueron generadas con la amplitud máxima.

Las gráficas 6.56 y 6.57 presentan el error distribuido en cada muestra para diferentes modulaciones tanto de amplitud como de frecuencia.

El índice con mayor error corresponde a Ma= 0.875 debido a que arrastra la diferencia de las modulaciones inferiores.

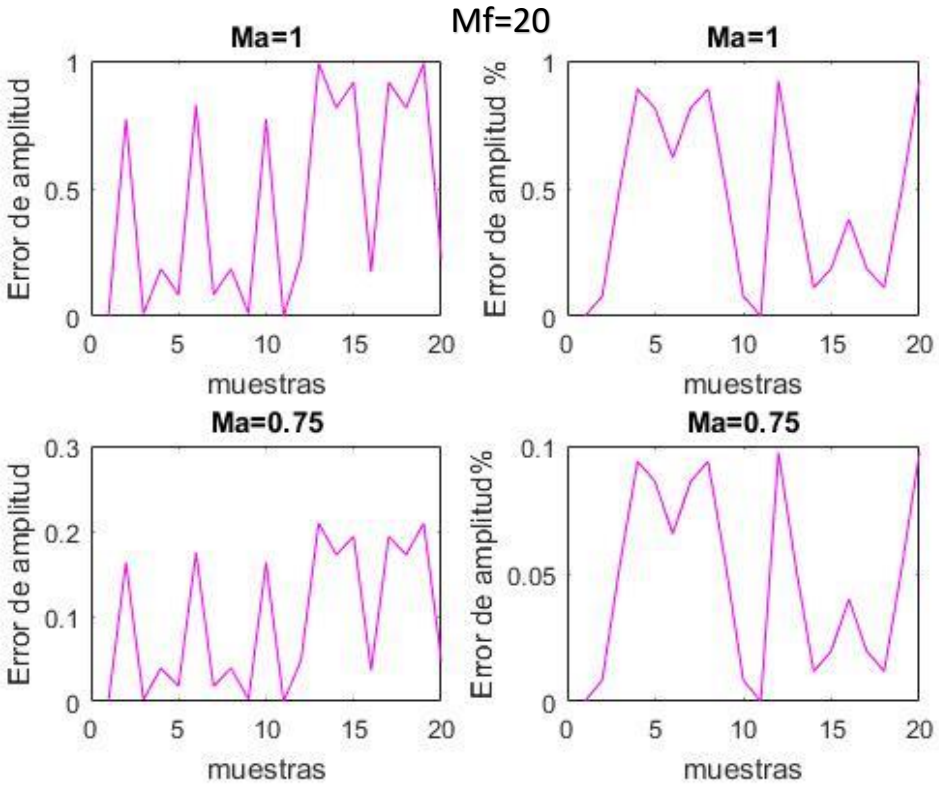


Imagen 6.56: Error de amplitud por muestra en tanto por 1

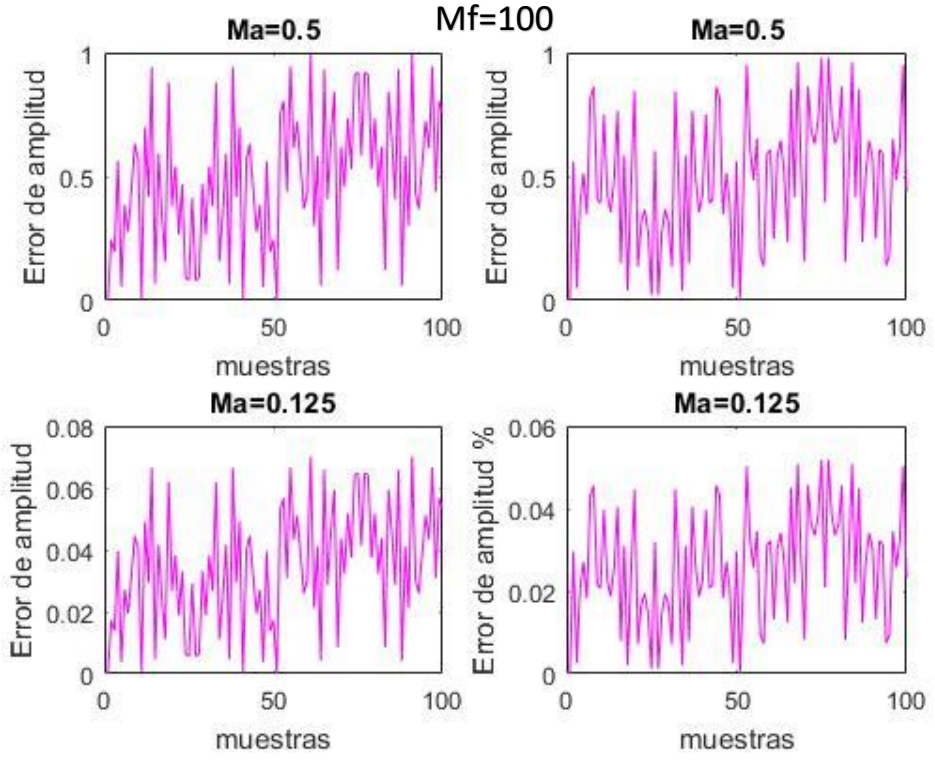


Imagen 6.55: Error de amplitud por muestra en tanto por 100

6.4 RESULTADOS

En este apartado se van a mostrar las gráficas obtenidas tanto en la simulación funcional del código VHDL en ModelSim como la descarga en placa realizada en el laboratorio, comparándolas así con las gráficas teóricas obtenidas en los apartados anteriores.

6.4.1 Resultados ModelSim

La versión utilizada del simulador no contiene todas las opciones de edición, por lo que los datos obtenidos fueron exportados a Matlab.

Se va a comenzar mostrando la señal trifásica va, vb y vc, utilizada en la entrada del modulador para ser muestreada y comenzar el funcionamiento del proceso.

Las gráficas 6.57, 6.58 y 6.59 muestran el cambio producido en las ondas al variar sus parámetros de funcionamiento.

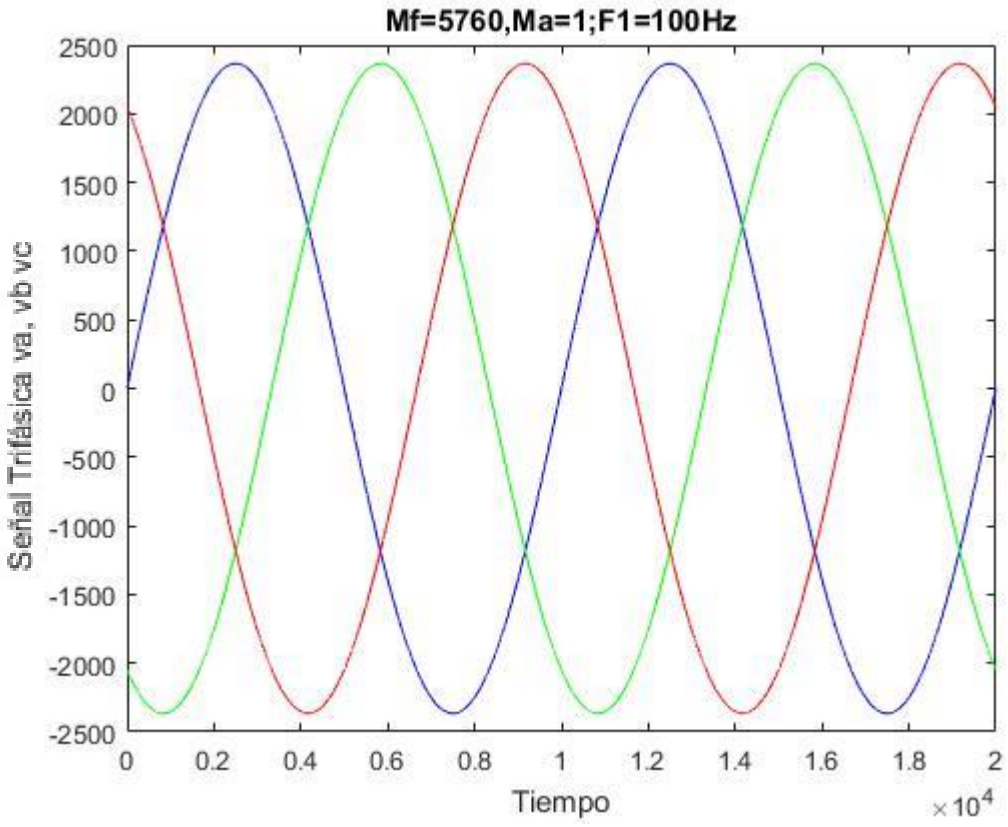


Imagen 6.57: Señal trifásicas va (roja), vb (verde), vc (azul) obtenida en VHDL

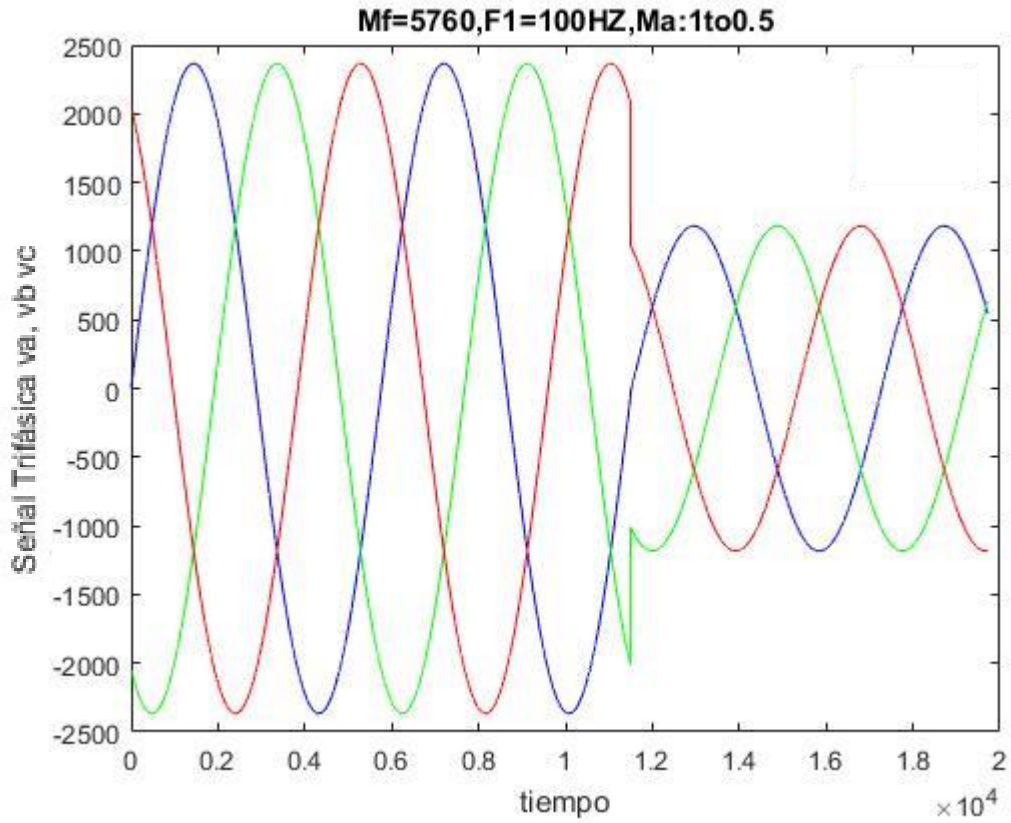


Imagen 6.58: Señal trifásica va (roja), vb (verde), vc (azul) con variación de Ma

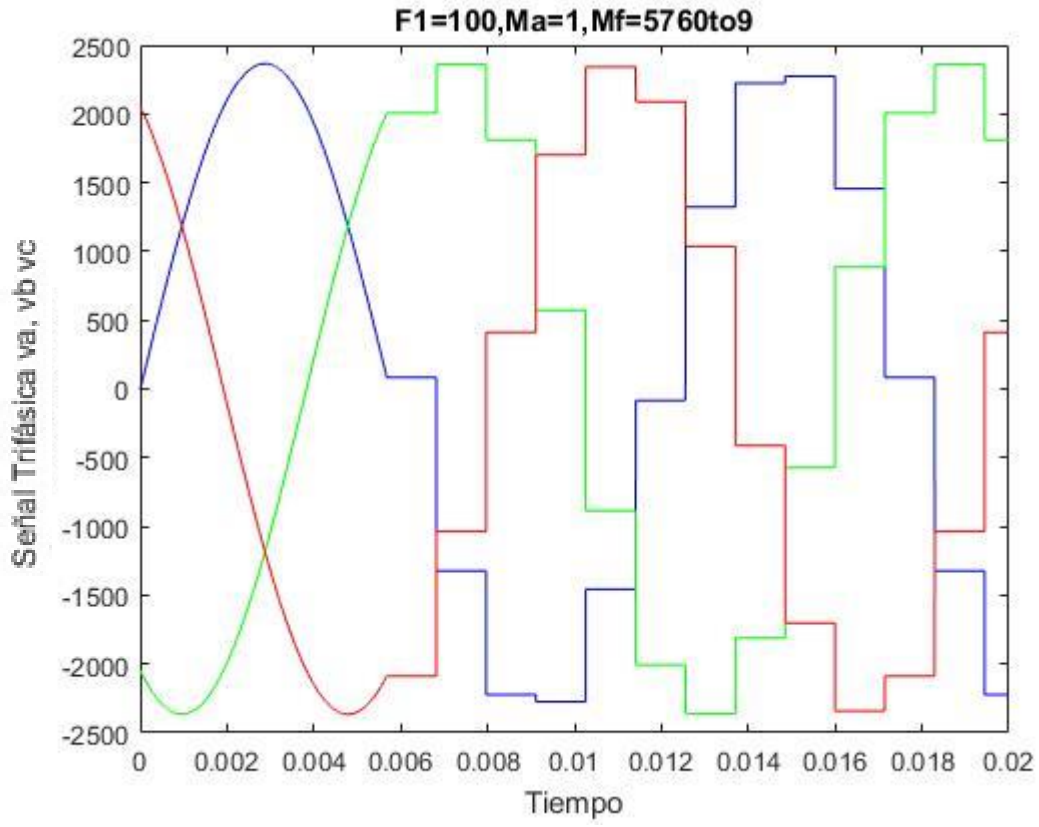


Imagen 6.59: Señal trifásica va (roja), vb (verde), vc (azul) con variación de Mf obtenida en VHDL

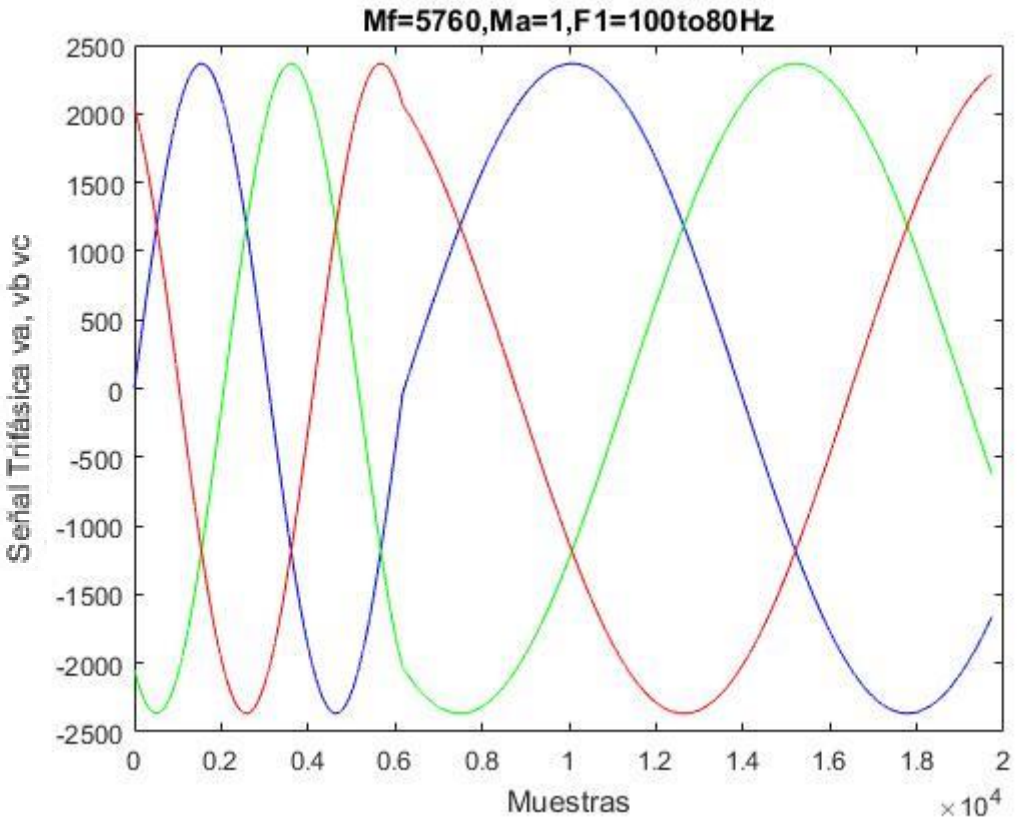


Imagen 6.60: Señal trifásica va (roja), vb (verde), vc (azul) con variación de F1 obtenida en VHDL

La variación de los parámetros representados anteriormente fue realizada mediante código, siendo los incrementos de gran tamaño. En la práctica real, las variaciones únicamente pueden ser ejecutadas a través de los pulsadores e interruptores, siendo los incrementos de la unidad mínima correspondiente a cada caso.

Para comprobar el correcto funcionamiento del diseño realizado, se ha simulado en Matlab un código que simula las muestra obtenidas por VHDL con un Mf=30, Ma=0.99 y F1=100Hz. Para ello, se han utilizado cuantificadores en cada operación del código, simulando así perfectamente los datos generados en Xilinx. Las gráficas resultantes son las siguientes:

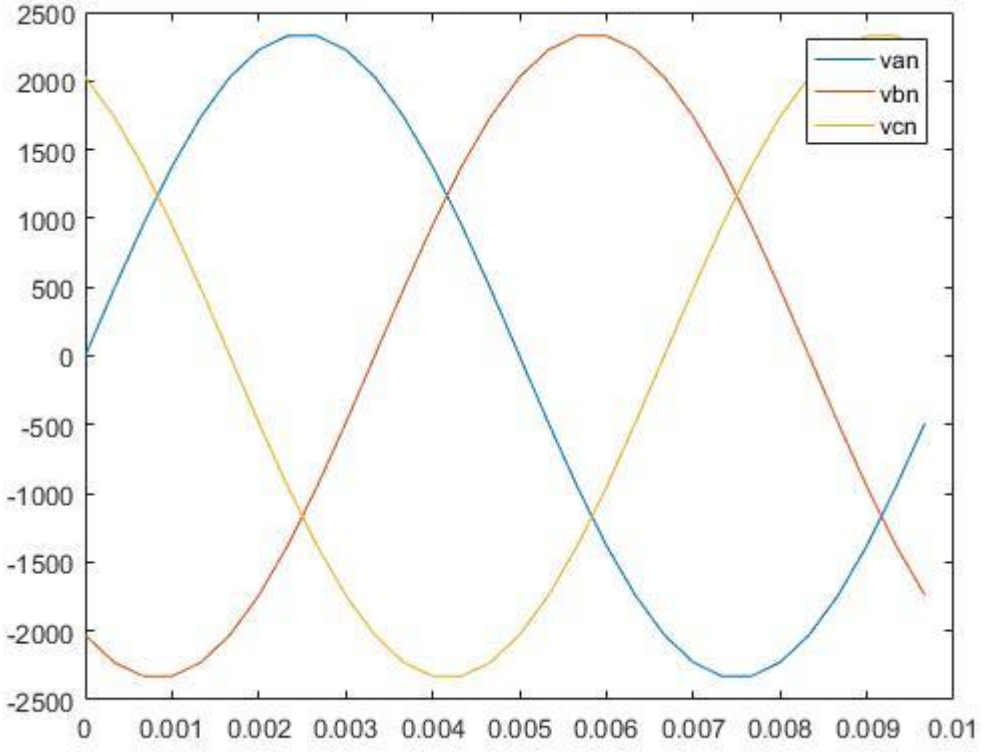


Imagen 6.61: Señal trifásica van (roja), vbn (verde), vcn (azul)

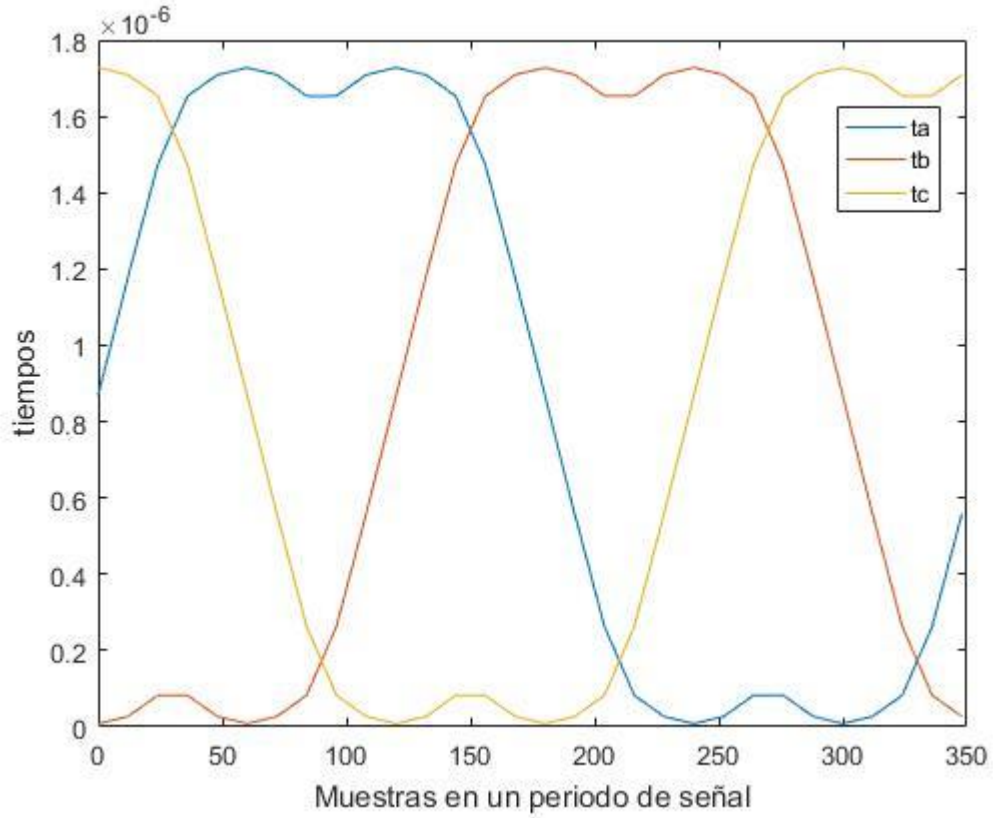


Imagen 6.62: Tiempos de conmutación en un periodo

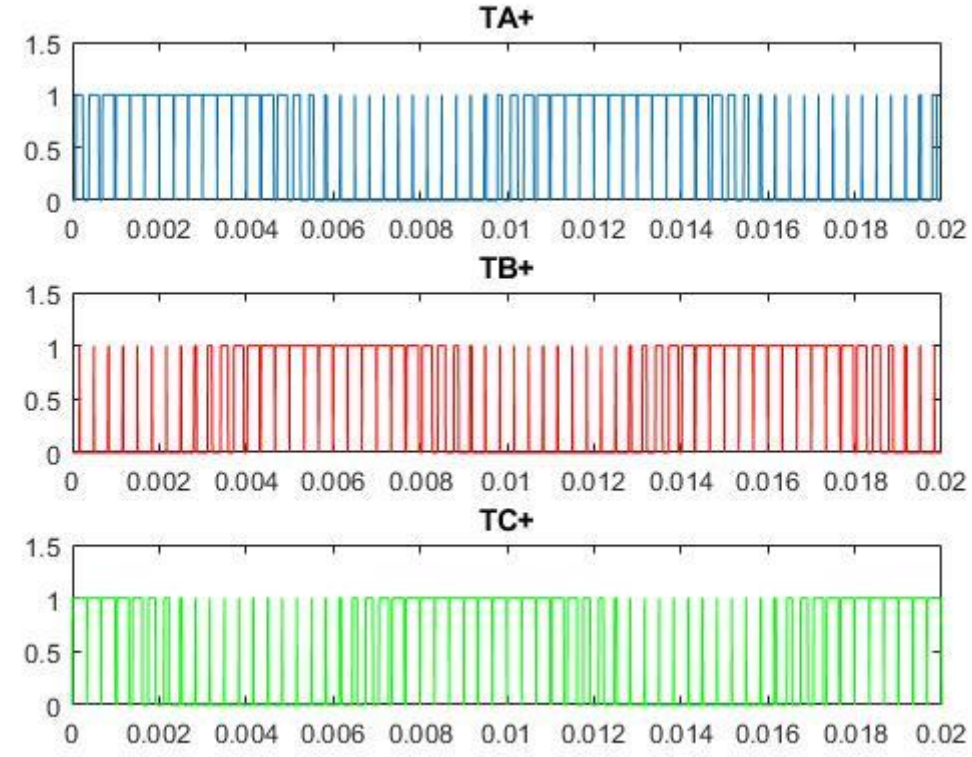


Imagen 6.64: Señales moduladoras de la primera rama del inversor

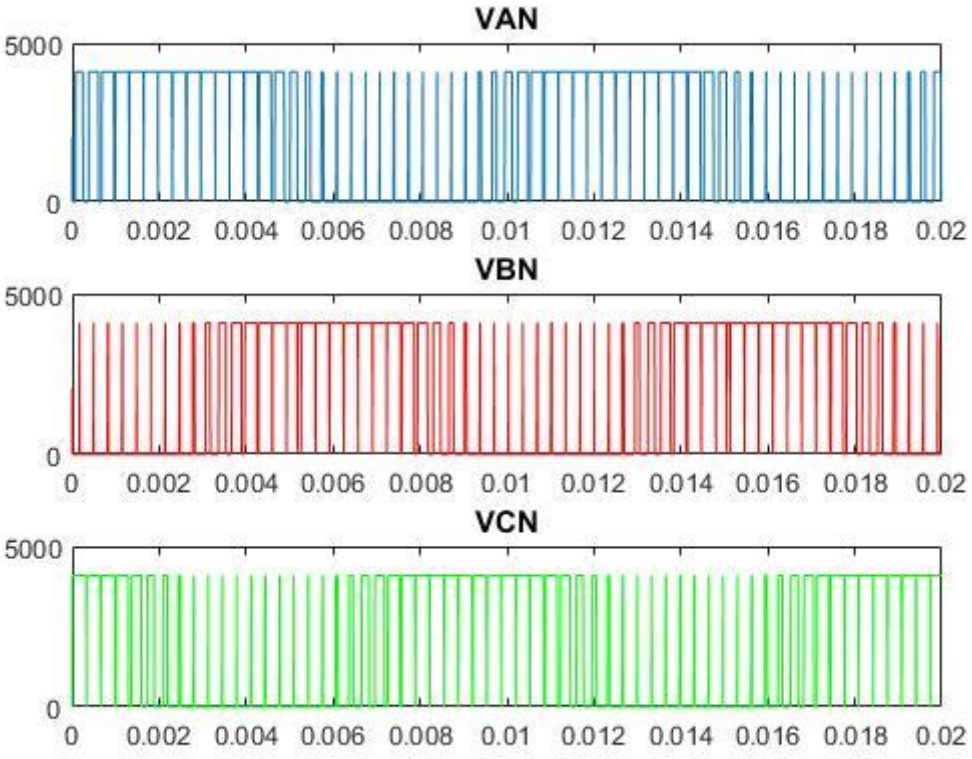


Imagen 6.63: Tensiones de fase obtenidas a la salida

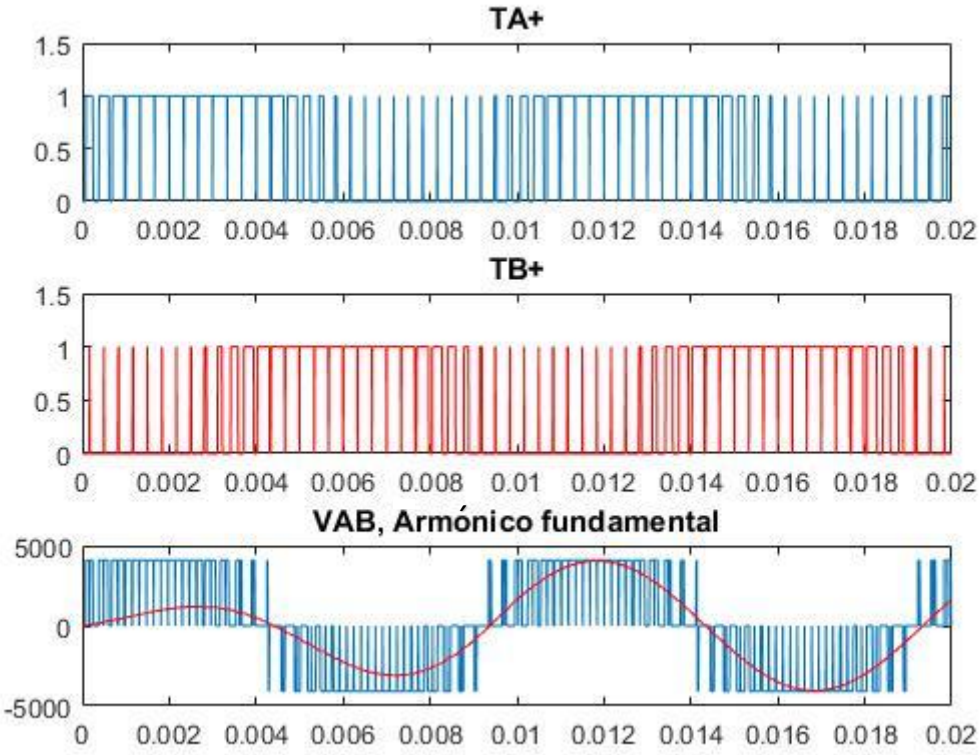


Imagen 6.65: Señales moduladoras TA+ y TB+ junto a la tensión de línea VAB y su primer armónico

La salida obtenida sigue una forma de onda similar a la vista en teoría. A pesar de haber utilizado un índice de modulación de frecuencia relativamente bajo, Mf=30, junto a una frecuencia fundamental elevada, F1=100Hz, aparecen pulsos en las señales moduladoras con un ciclo de trabajo muy pequeño. Esto tendrá una gran repercusión en los resultados prácticos ya que los interruptores utilizados en el cubo de potencia no son capaces de conmutar tan rápido.

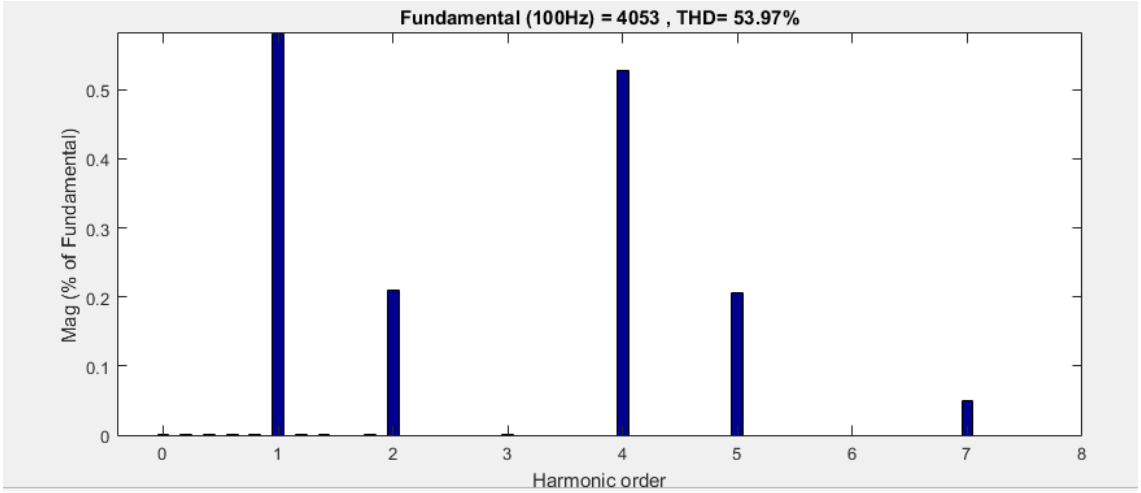


Imagen 6.66: Espectro armónico

Los armónicos resultantes, son pares en frecuencias bajas como era de esperar al utilizar la secuencia de conmutación de siete segmentos. La magnitud del armónico fundamental depende de la tensión continua utilizada (2^{12} V) y el índice de modulación de amplitud Ma=0.99.

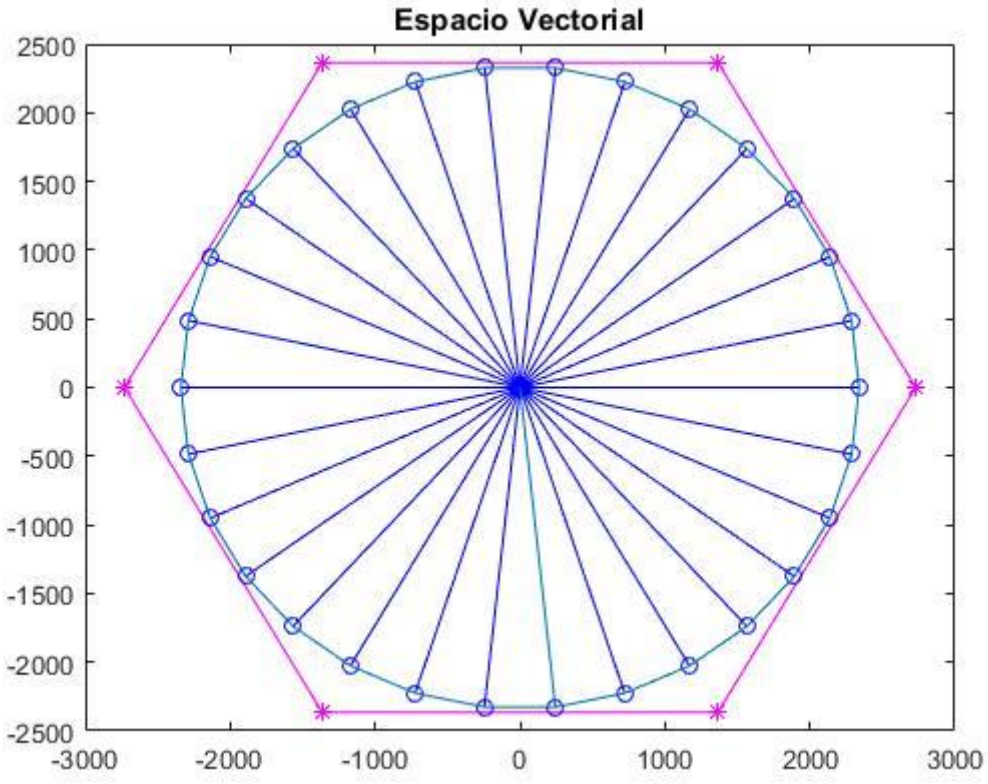


Imagen 6.67: Espacio Vectorial final

El espacio vectorial final muestra los 30 vectores de referencia utilizados en el ciclo completo en el sistema de referencia de *αβ.*

6.4.2 Resultados FPGA

La descarga en placa fue realizada en días consecutivos. Primero, fueron comprobadas nuevamente las señales generadas por el código en la FPGA sin conectar con el módulo de potencia. Para ello se utilizó el osciloscopio YOKOGOWA de 4 canales. Sin embargo, solo se pudieron representar dos de las tres ondas que componen la señal trifásica va, vb y vc, al disponer de un único DAC.

Las siguientes imágenes representan las señales reales obtenidas para distintas configuraciones de los parámetros de configuración, con el fin de comprobar el correcto funcionamiento del diseño antes de la conexión del cubo de potencia y la carga.

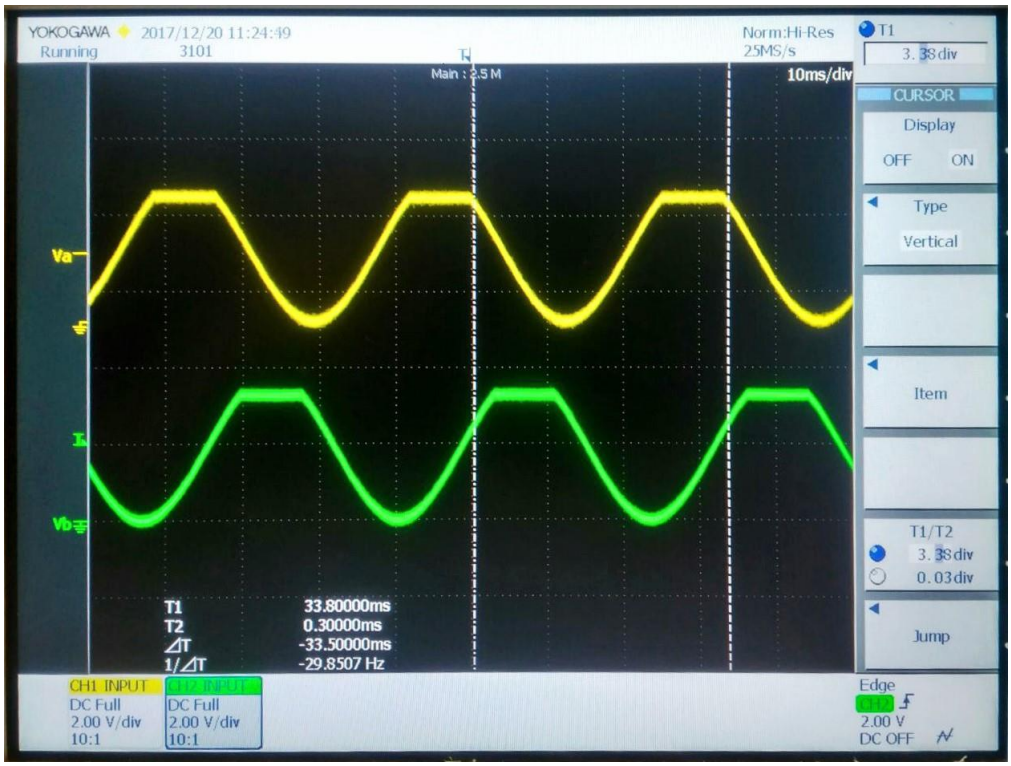


Imagen 6.68: Señales Va y Vb. Parámetros Ma=1, Mf=5760, F1=30

Tal y como se comentó anteriormente, el DAC utilizado es de 12 bits y la señal trifásica utilizada para la modulación es de 13 bits al estar desplazada verticalmente, es por ello, que, en la gráfica, las ondas aparecen recortadas en sus valores máximos de amplitud. A partir de una modulación de amplitud de Ma=0.625, el error es corregido.

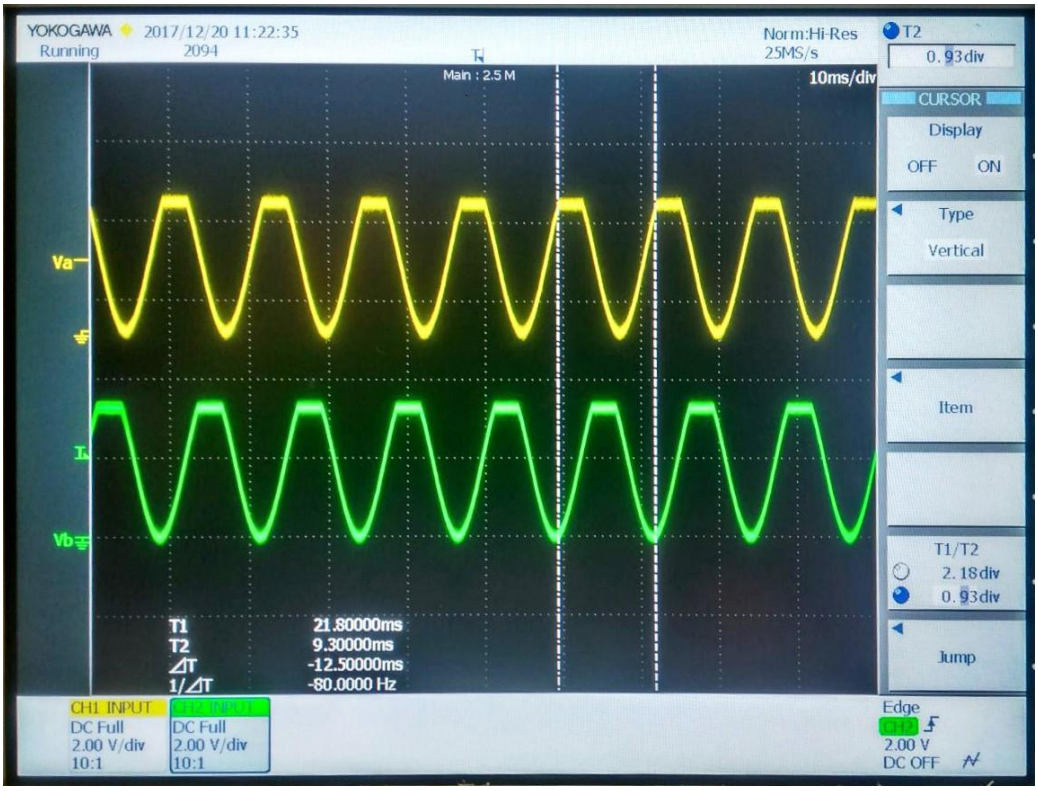


Imagen 6.70: Señales Va y Vb. Parámetros: Ma=1, Mf=5760, F1=80

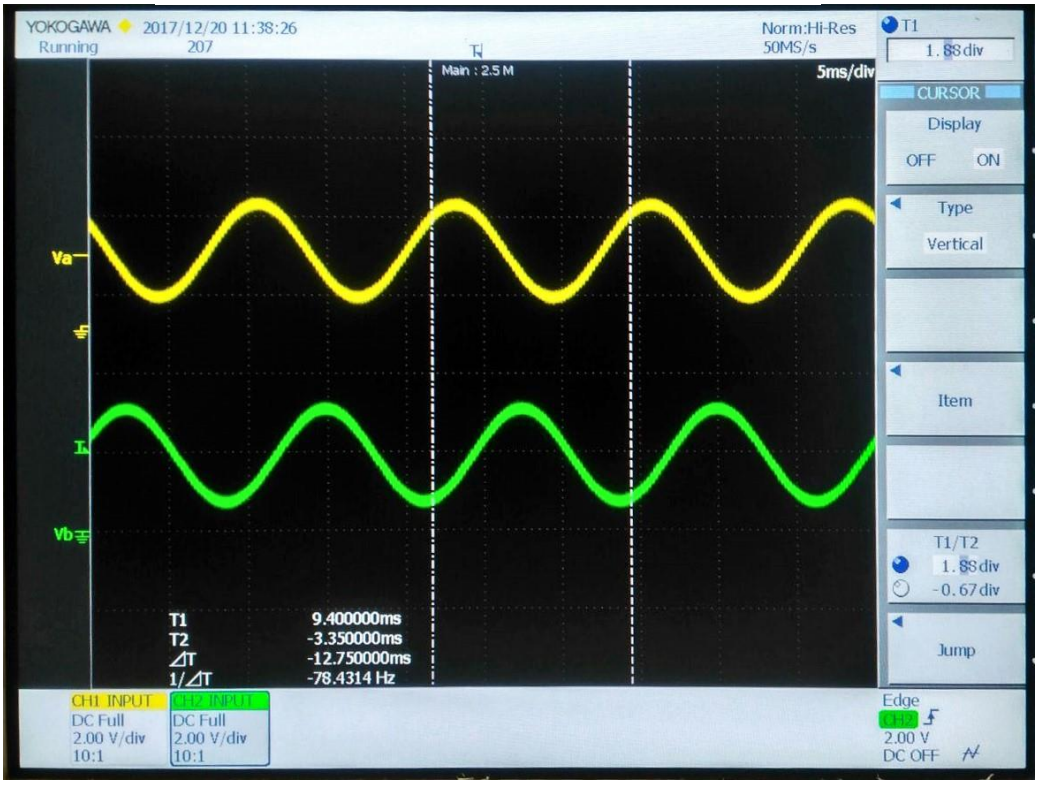


Imagen 6.69: Señales Va y Vb. Parámetros Ma=0.625, Mf=20, F1=80

Comparando las imágenes anteriores 6.68 y 6.69, se puede observar como la frecuencia de la señal es dependiente del Mf utilizado.

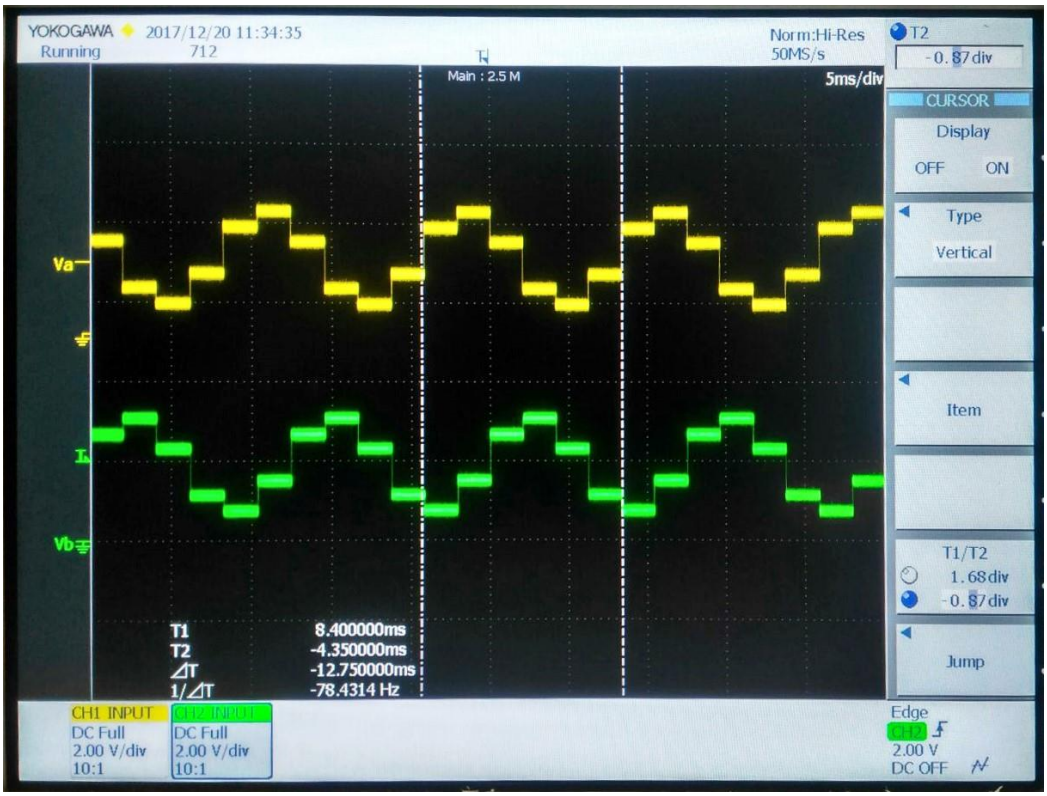


Imagen 6.71: Señales Va y Vb. Parámetros: Ma=0.625, Mf=6, F1=80

Imagen 6.72: Señales de control TA+, TB+. Parámetros: Ma=1, F1=100, Mf=5760

Imagen 6.71: Señal de control TC+. Parámetros: Ma=1, Mf=5760, F1=100

Las señales de control TA y TB, no se corresponden con el mismo periodo de simulación que la señal TC, debido a la disponibilidad de cuatro sondas de osciloscopio únicamente. Utilizando los parámetros por defecto tras reset, Ma=1, Mf=5760, F1=100Hz, las señales tienen escalones de gran velocidad como se puede observar en TA+ de la imagen 6.72.

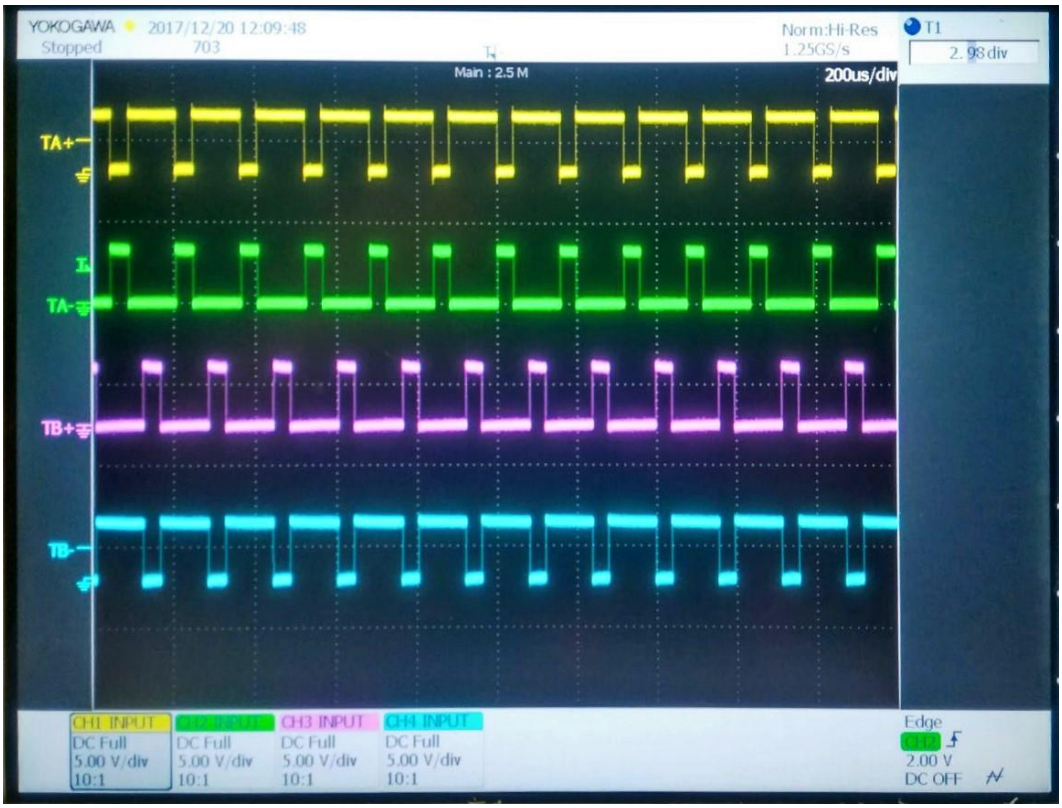


Imagen 6.72: Señales de control TA+, TA-, TB+, TB-. Parámetros: Ma=0.5, Mf=90, F1=70

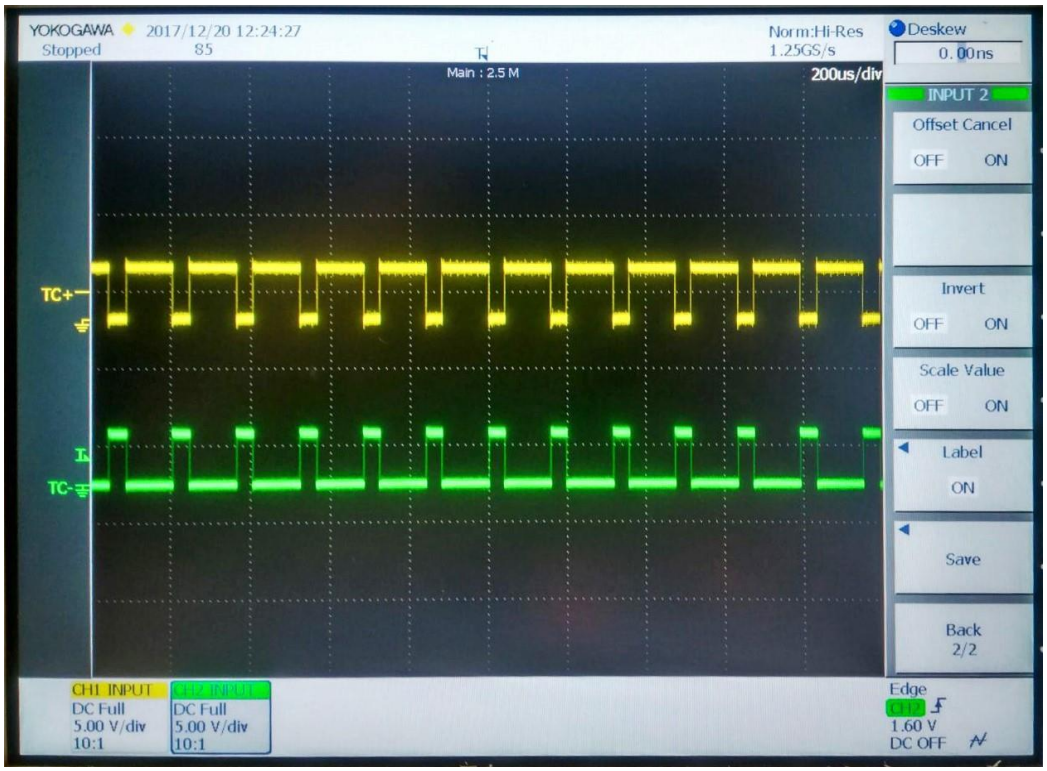


Imagen 6.73: Señales de control TC+, TC-. Parámetros: Ma=0.5, Mf=90, F1=70

Al ir disminuyendo *Mf* y la frecuencia de la señal, las ondas moduladoras se hacen más lentas, con pulsos de mayor ciclo de trabajo.

A continuación, se va a mostrar una comparación de las señales de control junto a la simulación Matlab para observar gráficamente la similitud de las ondas. Para ello se ha escogido el índice de modulación de frecuencia más bajo, Mf=6, junto a la frecuencia de la señal más elevada, para poder comparar un periodo completo de simulación.

Imagen 6.74: Señales de control TA+, TA-, TB+, TB-. Parámtros: Ma=0.5, Mf=6, F1=100

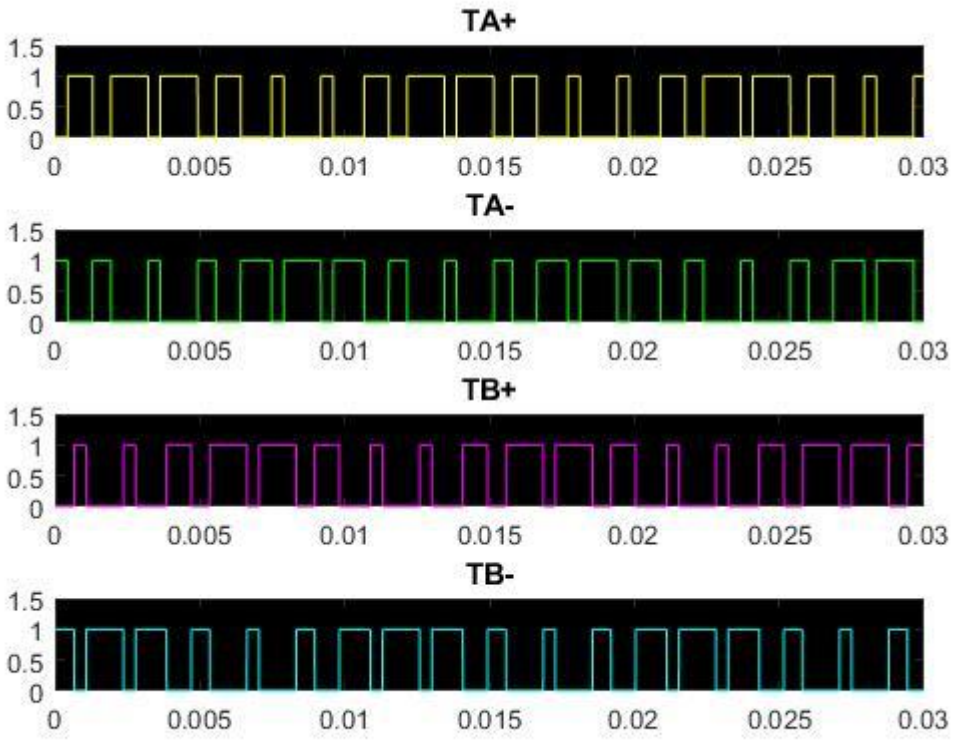


Imagen 6.75: Señales de control TA+, TA-, TB+, TB-. Parámetros Ma=0.5, Mf=6, F1=100. MATLAB

Comparando las imágenes anteriores se puede observar la similitud de las ondas, a pesar del ligero desfase que hay entre ellas.

6.4.3 Pruebas reales

Una vez validadas las señales de control que regulan la conmutación de los interruptores de potencia, se muestran las gráficas obtenidas en la práctica real llevada a cabo en el laboratorio.

En todas las imágenes siguientes, se van a mostrar las mismas ondas de salidas y en el mismo orden, que corresponden a:

1. Señales Va (Rosa) y Vb (Azul) obtenidas a través del DAC.
2. En la parte intermedia, la tensión de fase V_{AN} (amarillo).
3. Por último, la tensión de línea-línea generada en la salida del inversor V_{AB}, (verde).

Imagen 6.76: Práctica real. Parámetros: Ma=0.5, Mf=20, F1=100Hz

Se puede observar claramente los 20 pulsos que tiene V_{AB}, señal verde, en un periodo completo. Únicamente se muestran dos de las tres ondas de la señal trifásica al no disponer de más sondas en el osciloscopio usado.

A continuación, serán vistas más gráficas con diferentes configuraciones de los parámetros de modulación con el fin de poder ver las formas de ondas que van tomando las salidas.

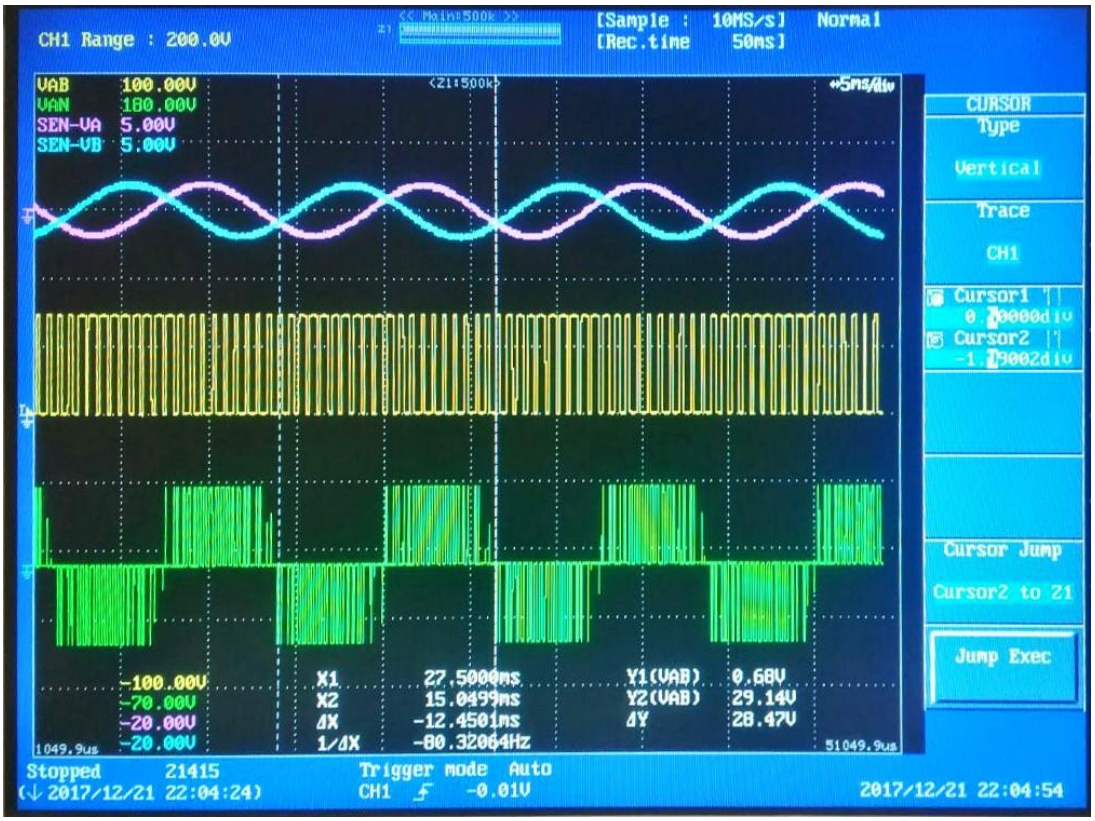


Imagen 6.77: Práctica Real. Parámetros: Ma=0.5Mf=20F1=80

En la tensión de salida $V_{AB,}$ señal verde, se observan pulsos cuya amplitud no llega alcanzar el valor de $V_{dc.}$ Para ciertas configuraciones de los parámetros de modulación, las conmutaciones producidas en los interruptores son demasiado rápidas, produciendo el efecto visto en la imagen. Por ello, la práctica real con los dispositivos aquí utilizados, solo será válida para aquellas configuraciones donde las señales PWM sean lo suficientemente lentas para completar las conmutaciones de los interruptores en todo el periodo.

Realizando pruebas se pudo comprobar que el índice de modulación de frecuencia máximo que era capaz de soportar el cubo estaba en torno a 30.

En las siguientes imágenes se observará las señales de salida para distintas configuraciones acotando los parámetros de la simulación para la cual el inversor trabaja adecuadamente.

Imagen 6.78: Práctica Real. Parámetros: Ma=0.5 Mf=20 F1=40

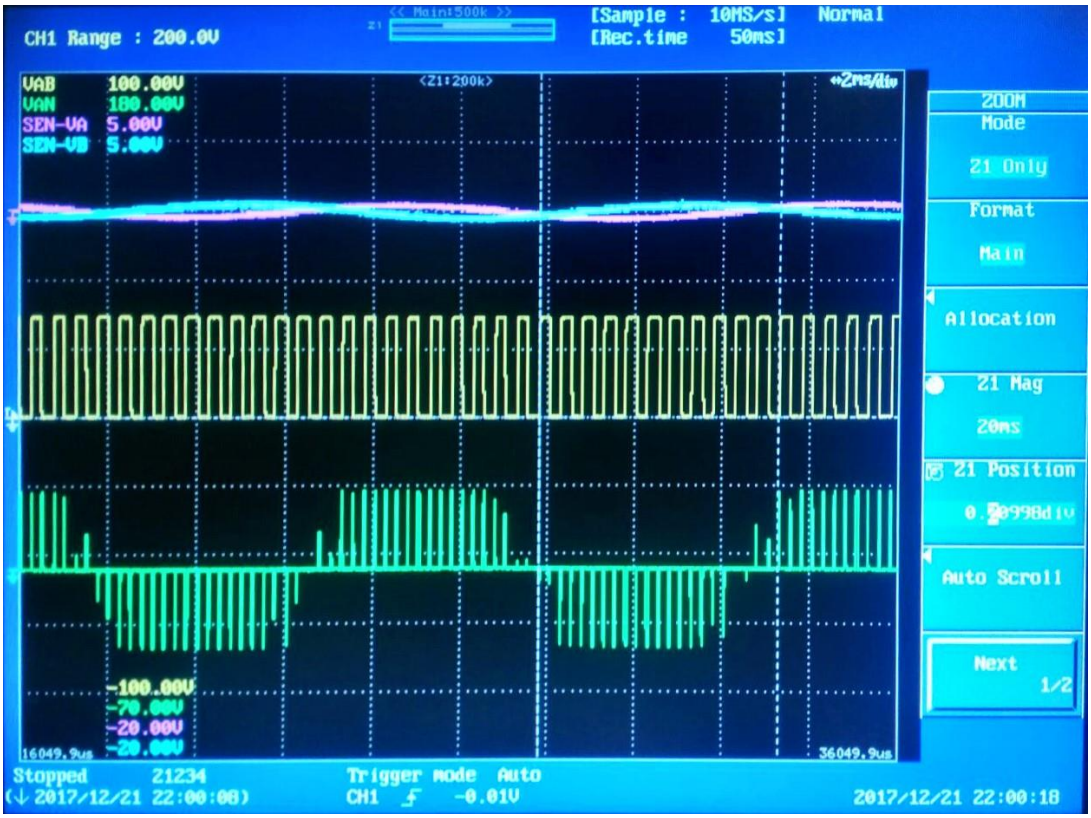


Imagen 6.79: Práctica Real. Parámetros: Ma=0.125 Mf=20 F1=100

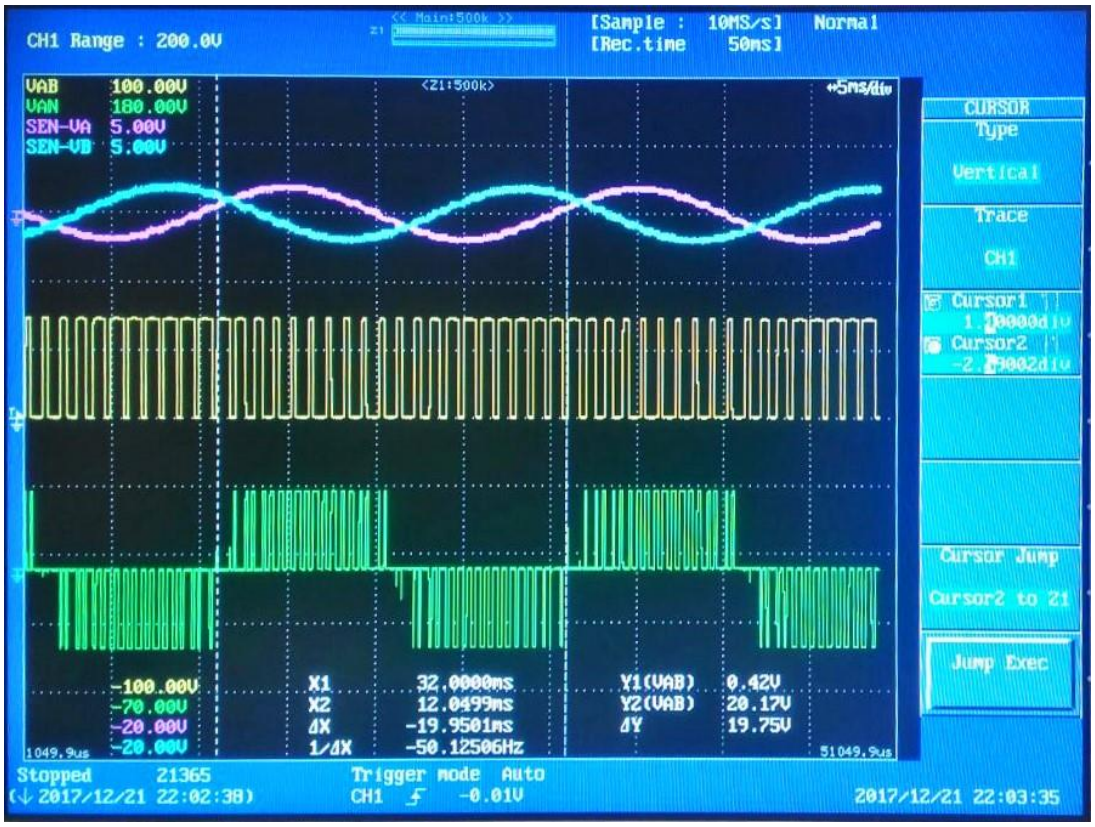


Imagen 6.80:Práctica Real. Parámetros: Ma=0.5 Mf=20 F1=50

Imagen 6.81: Práctica Real. Parámetros: Ma=0.125 Mf=10 F1=100

6.4.4 Concordancia de resultados

Imagen 6.83: Práctica Real. Parámetros: Ma=0.5 Mf=20 F1=20

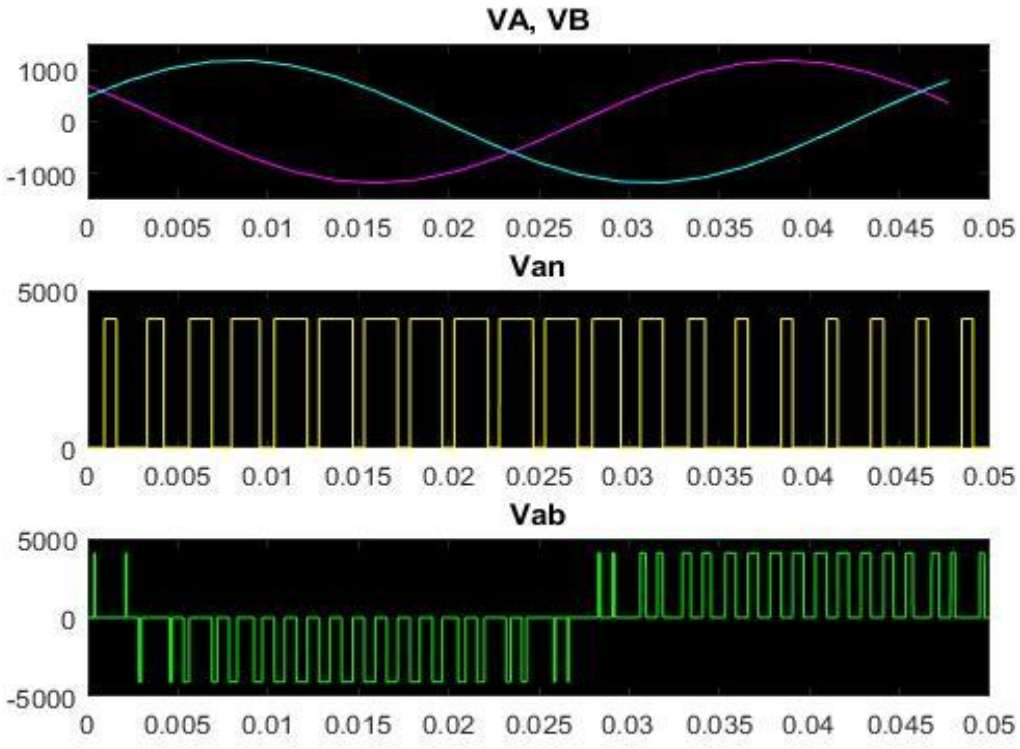


Imagen 6.82: Simulación de Matlab. Parámetros: Ma=0.5, Mf=20, F1=20

Para ondas lentas, los resultados obtenidos son similares a los generados mediante Matlab.

El diseño es capaz de generar las señales PWM que modulan un inversor trifásico para todo el rango de parámetros configurados. Las limitaciones visualizadas en los resultados prácticos, provocadas por el cubo de potencia Semiteach-IGBT utilizado, pueden solucionarse empleando otros interruptores que puedan conmutar a mayores frecuencias. Aún así, es posible mostrar el funcionamiento de la modulación vectorial en frecuencias bajas.

6.4.5 Recursos utilizados

Los recursos utilizados en la implementación del diseño fueron obtenidos a través del simulador ISE Project Navigator. Los resultados reflejan que la frecuencia máxima es de 50.133MHz, siendo el clk interno de la FPGA empleado de 50Mhz.

La tabla muestra un porcentaje máximo de utilización del 37% correspondientes a los Slices ocupados. El total de multiplicadores usados es de 4, siendo necesarios dos de ellos, en la obtención de los tiempos en cuentas de la FPGA.

```
Timing summary:
---------------

Timing errors: 0  Score: 0  (Setup/Max: 0, Hold: 0)

Constraints cover 862332654 paths, 0 nets, and 16209 connections

Design statistics:
   Minimum period:  19.947ns{1}   (Maximum frequency:  50.133MHz)
```

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Total Number Slice Registers	2,505	17,344	14%
Number used as Flip Flops	2,493		
Number used as Latches	12		
Number of 4 input LUTs	4,141	17,344	23%
Number of occupied Slices	3,271	8,672	37%
Number of Slices containing only related logic	3,271	3,271	100%
Number of Slices containing unrelated logic	0	3,271	0%
Total Number of 4 input LUTs	4,798	17,344	27%
Number used as logic	4,111		
Number used as a route-thru	657		
Number used as Shift registers	30		
Number of bonded IOBs	35	250	14%
Number of RAMB16s	2	28	7%
Number of BUFGMUXs	3	24	12%
Number of DCMs	1	8	12%
Number of MULT18X18SIOs	4	28	14%
Average Fanout of Non-Clock Nets	2.46		

Imagen 6.84: Resultados de la implementación del diseño en la FPGA Spartan 3E 1200

6.5 CONCLUSIONES Y FUTUROS TRABAJOS

En este trabajo se ha realizado la implementación VHDL de un esquema de modulación vectorial SVM para control de inversores trifásicos. Se han realizado todas las fases de diseño:

1. En primer lugar, se ha creado un modelo teórico en Matlab-Simulink del algoritmo de modulación SVM a implementar en VHDL. El modelo ha permitido evaluar los errores de cuantificación por los bits utilizados en el diseño digital.
2. Se ha realizado el diseño VHDL del algoritmo de modulación vectorial SVM, incluyendo simulación funcional, síntesis, implementación y simulación temporal.
3. Se ha comprobado experimentalmente mediante una FPGA de Xilinx, un cubo de potencia SemiTeach IGBT y una carga resistiva, el adecuado funcionamiento del circuito y sus restricciones.
4. Por último, se han contrastado los resultados simulados con los prácticos obtenidos.

Con la realización de este trabajo, y junto al realizado por otros compañeros en sus TFG correspondientes, se puede dar por concluida la implementación de las distintas modulaciones de inversores impartidas en la asignatura de Electrónica de potencia en la Universidad de Alcalá en tarjetas FPGA.

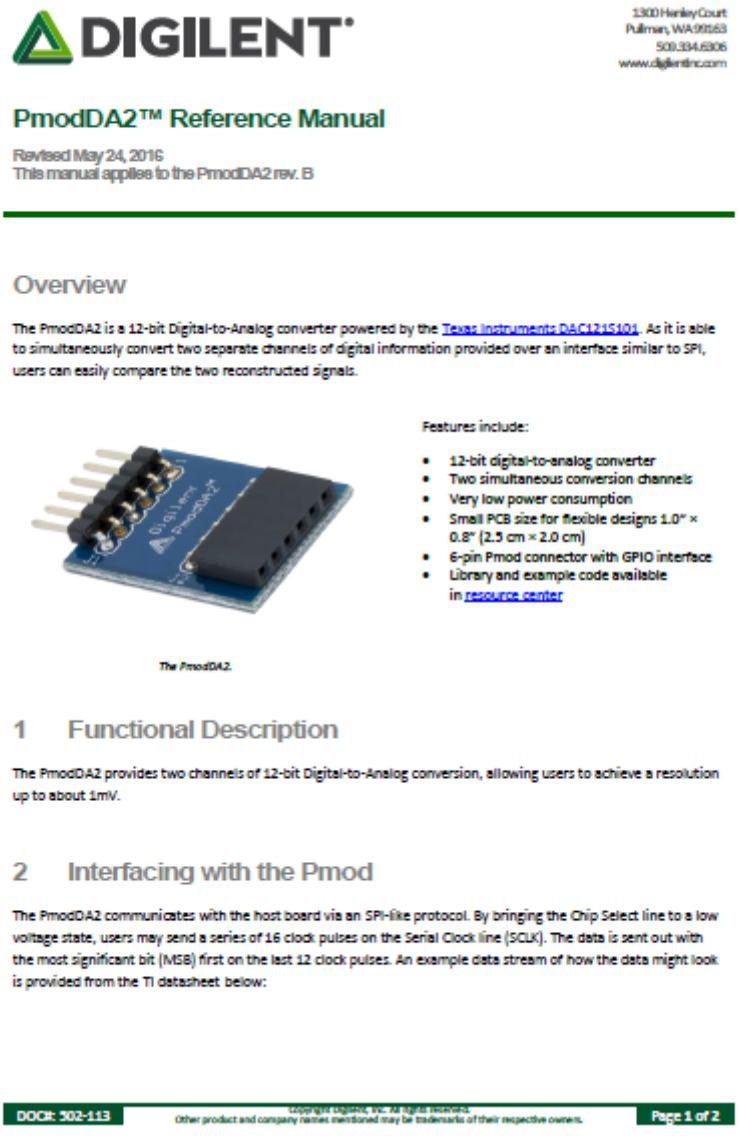
Referente al diseño aquí realizado, cabe la posibilidad de introducir la eliminación de los armónicos pares de la tensión V_{AB} a la salida del inversor, vista en el apartado teórico. Observando los recursos utilizados, la implementación del seno completo o ampliar el número de muestras guardadas es viable y reduciría los errores cometidos respecto a la simulación teórica.

Por último, se podría plantear la unificación del trabajo realizado por Edel Díaz [6] acerca de la modulación SPWM y la inyección de secuencias cero, junto el trabajo aquí presente con el fin de mejorar la comodidad de poder ser explicado en una clase práctica sin tener que estar descargando uno a uno en la FPGA.

7 PLIEGO DE CONDICIONES

En esta sección, se va a adjuntar las hojas de características de los dispositivos utilizados en el diseño. Si se desea obtener una información más detallada, cada uno de ellos han sido referenciados en la bibliografía.

7.1 DAC



DIGILENT

1300 Henley Court
Pullman, WA 99163
509.334.6306
www.digilentinc.com

PmodDA2™ Reference Manual

Revised May 24, 2016
This manual applies to the PmodDA2 rev. B

Overview

The PmodDA2 is a 12-bit Digital-to-Analog converter powered by the Texas Instruments DAC121S101. As it is able to simultaneously convert two separate channels of digital information provided over an interface similar to SPI, users can easily compare the two reconstructed signals.

Features include:

- 12-bit digital-to-analog converter
- Two simultaneous conversion channels
- Very low power consumption
- Small PCB size for flexible designs 1.0" × 0.8" (2.5 cm × 2.0 cm)
- 6-pin Pmod connector with GPIO interface
- Library and example code available in resource center

The PmodDA2.

1 Functional Description

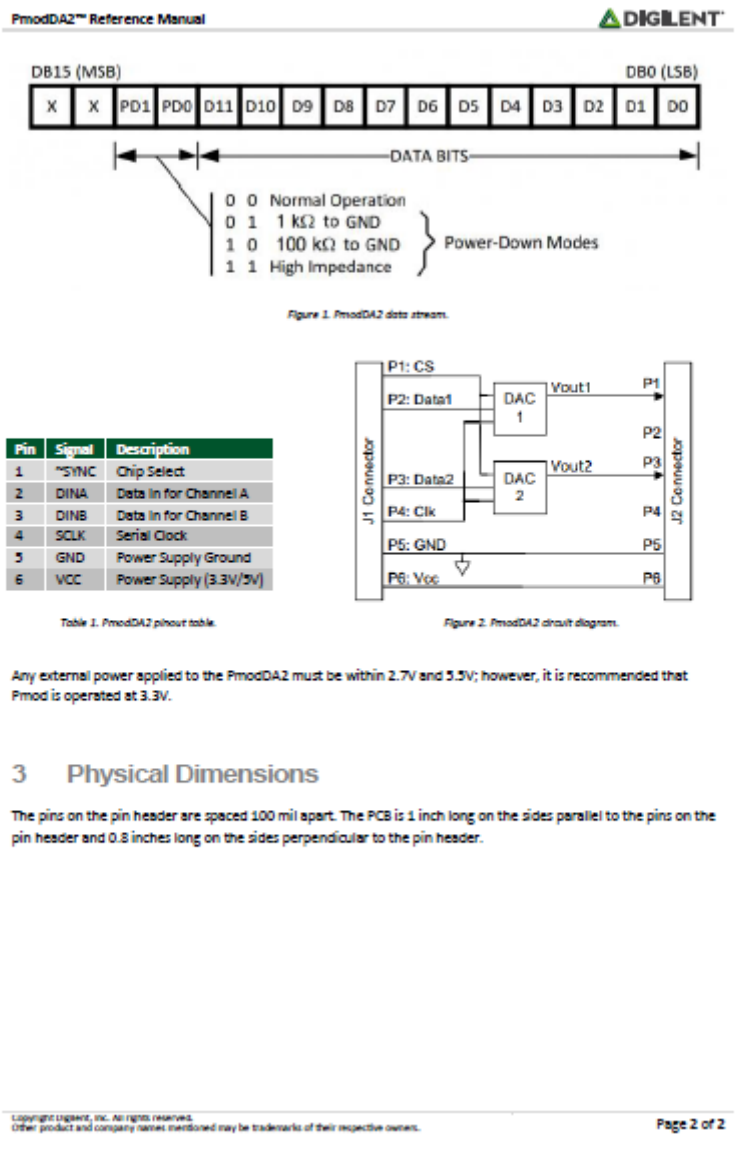
The PmodDA2 provides two channels of 12-bit Digital-to-Analog conversion, allowing users to achieve a resolution up to about 1mV.

2 Interfacing with the Pmod

The PmodDA2 communicates with the host board via an SPI-like protocol. By bringing the Chip Select line to a low voltage state, users may send a series of 16 clock pulses on the Serial Clock line (SCLK). The data is sent out with the most significant bit (MSB) first on the last 12 clock pulses. An example data stream of how the data might look is provided from the TI datasheet below:

DOC#: 502-113 Page 1 of 2

Imagen 7.1: Hoja de características 1 DAC [18]



PmodDA2™ Reference Manual — DIGILENT

Figure 1. PmodDA2 data stream.

Pin	Signal	Description
1	~SYNC	Chip Select
2	DINA	Data In for Channel A
3	DINB	Data In for Channel B
4	SCLK	Serial Clock
5	GND	Power Supply Ground
6	VCC	Power Supply (3.3V/5V)

Table 1. PmodDA2 pinout table.

Figure 2. PmodDA2 circuit diagram.

Any external power applied to the PmodDA2 must be within 2.7V and 5.5V; however, it is recommended that Pmod is operated at 3.3V.

3 Physical Dimensions

The pins on the pin header are spaced 100 mil apart. The PCB is 1 inch long on the sides parallel to the pins on the pin header and 0.8 inches long on the sides perpendicular to the pin header.

Page 2 of 2

Imagen 7.2: Hoja de características 2 DAC [18]

7.2 TARJETA FPGA

Debido a la extensión del documento completo, solo se ha adjuntado las partes más importantes utilizadas en el trabajo.

Peripheral Connectors

The Nexys2 board provides four two-row 6-pin Pmod connectors that together can accommodate up to 8 Pmods. The four 12-pin connectors each have 8 data signals, two GND pins, and two Vdd pins. All data signals include short circuit protection resistors and ESD protection Diodes. A jumper block adjacent to each Pmod connector can connect the Pmod's Vdd signal to the Nexys2 board's 3.3V supply or to the input power bus (VU). If the jumper is set to VU and USB power is driving the main power bus, care should be taken to ensure no more than 200mA is consumed by the Pmod. Further, if the jumper is set to VU, a voltage source connected to the Pmod can drive the main power bus of the Nexys2 board, so care should be taken to avoid connecting conflicting power supplies.

The Pmod connectors are labeled JA (nearest the power jack), JB, JC, and JD (nearest the expansion connector). Pinouts for the Pmod connectors are provided in the table below.

More than 30 low-cost are available for attachment to these connectors. Pmods can either be attached directly, or by using a small cable. Available Pmods include A/D and D/A converters, motor drivers, speaker amplifiers, distance measuring devices, etc. Please see www.digilentinc.com for more information.

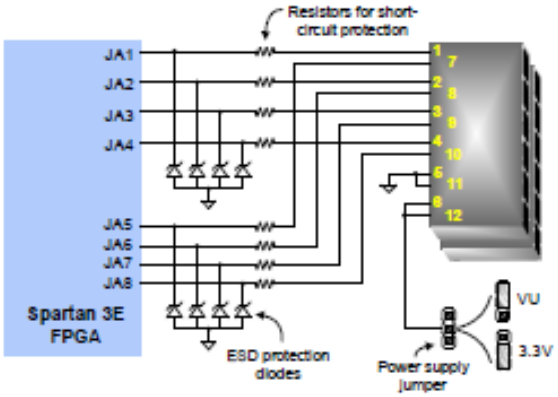


Figure 23: Nexys2 Pmod connector circuits

Table 3: Nexys2 Pmod Connector Pin Assignments

Pmod JA		Pmod JB		Pmod JC		Pmod JD	
JA1: L15	JA7: K13	JB1: M13	JB7: P17	JC1: G15	JC7: H15	JD1: J13	JD7: K14[1]
JA2: K12	JA8: L16	JB2: R18	JB8: R16	JC2: J16	JC8: F14	JD2: M18	JD8: K15[2]
JA3: L17	JA9: M14	JB3: R15	JB9: T18	JC3: G13	JC9: G16	JD3: N18	JD9: J15[3]
JA4: M15	JA10: M16	JB4: T17	JB10: U18	JC4: H16	JC10: J12	JD4:: P18	JD10: J14[4]

Notes: [1] shared with LD3 [2] shared with LD3 [3] shared with LD3 [4] shared with LD3

Imagen 7.3: Hoja de características 1 FPGA [17]

series resistor for protection against short circuits (a short circuit would occur if an FPGA pin assigned to a pushbutton or slide switch was inadvertently defined as an output).

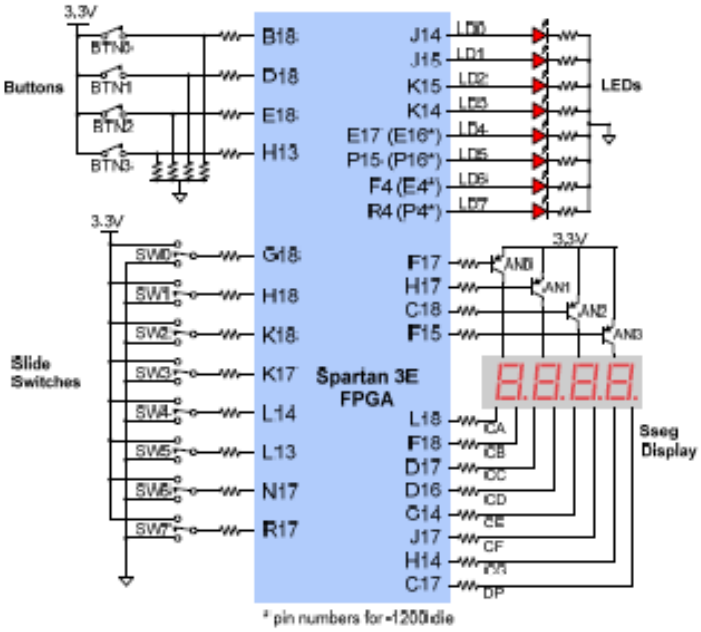


Figure 8: Nexys2 I/O devices and circuits

Outputs: LEDs

Eight LEDs are provided for circuit outputs. LED anodes are driven from the FPGA via 390-ohm resistors, so a logic '1' output will illuminate them with 3-4ma of drive current. A ninth LED is provided as a power-on LED, and a tenth LED indicates FPGA programming status. Note that LEDs 4-7 have different pin assignments due to pinout differences between the -500 and the -1200 die.

Outputs: Seven-Segment Display

The Nexys2 board contains a four-digit common anode seven-segment LED display. Each of the four digits is composed of seven segments arranged in a "figure 8" pattern, with an LED embedded in each segment. Segment LEDs can be individually illuminated, so any one of 128 patterns can be displayed on a digit by illuminating certain LED segments and leaving the others dark. Of these 128 possible patterns, the ten corresponding to the decimal digits are the most useful.

Imagen 7.4: Hoja de características 2 FPGA [17]

7.3 CUBO DE POTENCIA SEMITEACH-IGBT

SEMISTACK - IGBT

SEMITRANS Stack[1)]

Three-phase rectifier + inverter with brake chopper

SEMITEACH - IGBT
SKM 50 GB 123D
SKD 51
P3/250F

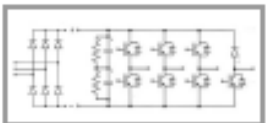
Features

- Multi-function IGBT converter
- Transparent enclosure to allow visualization of every part
- IP2x protection to minimize safety hazards
- External banana/BNC type connectors for all devices
- Integrated drive unit offering short-circuit detection/cut-off, power supply failure detection, interlock of IGBTs + galvanic isolation of the user
- Forced-air cooled heatsink

Typical Applications

- Education: One stack can simulate almost all existing industrial applications:
- 3-phase inverter+brake chopper
- Buck or boost converter
- Single phase inverter
- Single or 3-phase rectifier

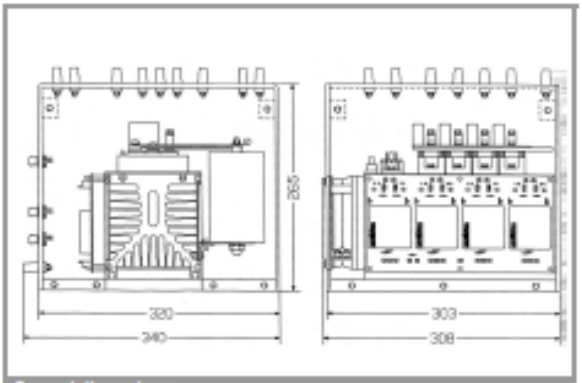
1) Photo non-contractual



B6U + B6CI + E1CIKF

Circuit	I_{rms} (A)	V_{ac} / V_{dcmax}	Types
B6CI	30	440 / 750	SEMITEACH - IGBT

Symbol	Conditions	Values	Units
I_{rms}	no overload	30	A
	IGBT - 4x SKM 50 GB 123D		
V_{CES}		1200	V
$V_{CE(sat)}$	I_C= 50A, V_{GE}= 15V, chip level; T_j= 25(125)°C	2,7 (3,5)	V
V_{GES}		±20	V
I_C	T_{case}= 25 (80)°C	50 (40)	A
I_{CM}	T_{case}= 25 (80)°C; t_p= 1ms	100 (80)	A
$V_{R(max)}$	Rectifier - 1x SKD 51/14 without filter with filter	 3 x 480 3 x 380	 V V
C_{equ}	DC Capacitor bank - Electrolytic 2x 2200µF/400V total equivalent capacitance	1100 / 800	µF / V
V_{DCmax}	max. DC voltage applied to the capacitor bank	750	V
	Driver - 4x SKHI 22		
Power supply		0 / 15	V
Current consumption	max; per driver	16	mA
Thermal trip	Normally Open type (NO)	71	°C



General dimensions

This technical information specifies semiconductor devices but promises no characteristics. No warranty or guarantee expressed or implied is made regarding delivery, performance or suitability.

2 Power Electronic Systems – SEMISTACK 08-06-2005 © by SEMIKRON

Imagen 7.5: Hoja de características cubo de potencia semiteach-IGBT

8 PRESUPUESTO DEL PROYECTO

El presupuesto del proyecto refleja el importe económico que hubiera conllevado el trabajo en un diseño real. Sin embargo, este proyecto es el Trabajo fin de Grado del alumno que lo presenta, y todo el material utilizado y los recursos destinados a él han sido cedidos por el Departamento de Electrónica de la Universidad de Alcalá.

Más allá de los materiales empleados, la mayor parte del valor de este proyecto recae en la programación del hardware, donde la destreza y el ingenio del ingeniero son claves para el desarrollo de la aplicación en una duración de tiempo concreta.

Antes de elaborar el presupuesto, se va a mostrar el Diagrama de Gant y las horas empleadas aproximadas en cada una de las fases.

8.1 DIAGRAMA DE GANT

En la siguiente tabla se muestra la distribución temporal del trabajo de manera aproximada, dividiendo cada mes en cuatro semanas.

Etapas	Mayo				Junio				Septiembre				Octubre				Noviembre				Diciembre			
Revisión teórica de los convertidores DC/AC																								
Estudio detallado de la modulación SPM																								
Modelado en Matlab y cálculo de errores.																								
Modelado en VHDL y simulación en ModelSim																								

Comprobación de simulaciones en Matlab																						
Puesta en marcha y realización de pruebas																						
Redacción del proyecto																						

Tabla 8.1: Diagrama de Grant

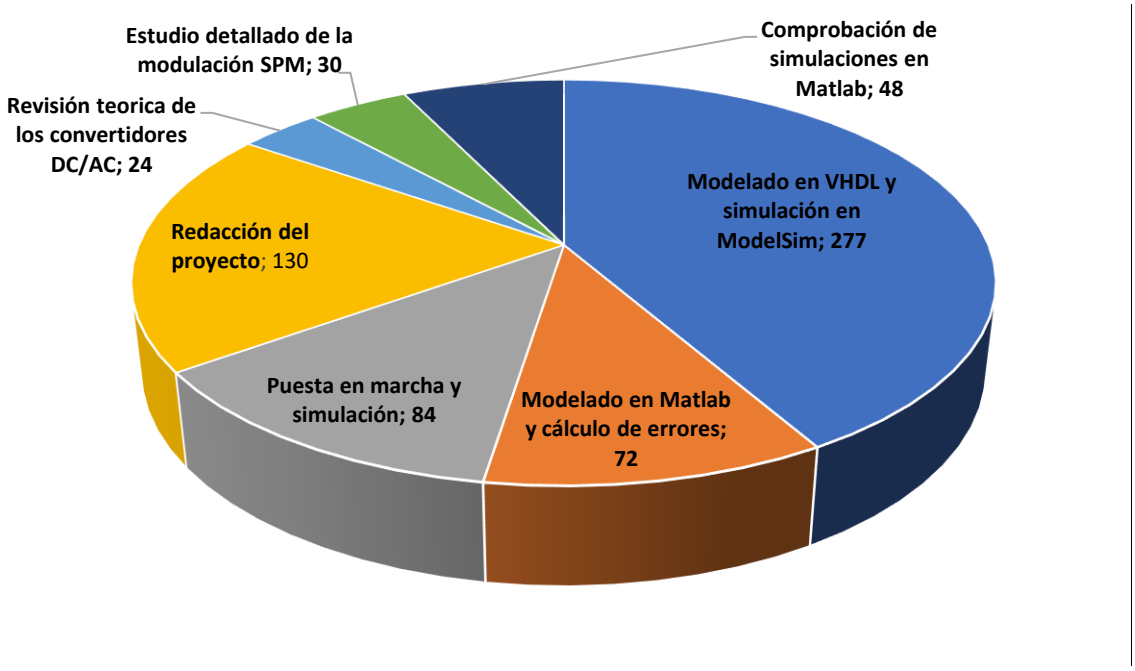


Imagen 8.1: Distribución del trabajo

8.2 PRESUPUESTO

El valor del material empleado ha sido estimado en función de su amortización. Para ello, se ha supuesto que los equipos tengan una vida útil de 5 años, y el tiempo de uso ha sido de un total de 6 meses.

Según el tipo de coste, se ha dividido el presupuesto en tres partes:

8.2.1 Coste del Hardware

Material utilizado	Unidades	Valor
Cubo de potencia Seimkron Semiteach-IGBT	1	1.800€
Tarjeta Digilent nexys-2 Spartane··-XC3S1200E FG320-4	1	150€
Fuente de alimentación Delta Elektronika SM 35-45	1	2.700€
DAC Digilent Pmod2	1	50€
Osciloscopio Digital Yokogawa DL9140L	1	5.000€
Circuito de adaptación [X]	1	100€
PC Acer Aspire V, i7	1	900€
	Coste Total= 10.700€	
	Coste Real (6 meses/60 meses útiles) = 1.700€	

Tabla 8.2 Coste Hardware

8.2.2 Trabajo del ingeniero

Competencias	Horas	Coste/Hora
Desarrollo del software	277	65€
Pruebas en Matlab	120	65€
Pruebas prácticas	84	65€
Redacción del proyecto	160	40€
Total	611	37.665€
	Coste total	37.665€

Tabla 8.3: Coste Ingeniero

En el trabajo llevado a cabo por el ingeniero no han sido incluidas las horas de formación del alumno, ni las reuniones de seguimiento realizadas con la tutora del proyecto, alrededor de una a la semana de los meses trabajados.

8.2.3 Otros costes

Por último, se adjuntan los costes indirectos del material software utilizado como son sus licencias.

Licencia:	Coste:
Licencia de Xilinx ISE 14.2	1.200€
Licencia Matlab R2016b	2.000€
Licencia Adept 2.3	0€
Licencia Microsoft Office 2016	439€
Total	3.639€
Coste Real (6 meses/60meses útiles)	363€

Tabla 8.4: Otros costes

8.2.4 Presupuesto total estimado

Concepto	Coste
Coste del hardware	1.700€
Trabajo del ingeniero	37.665€
Otros costes	363€
	Total: 39.730€

Tabla 8.5: Presupuesto total

9 MANUAL DE USUARIO

Este apartado va a consistir en una pequeña guía para llevar a cabo la práctica tanto en simulación como en la descarga a placa. En el Anexo, queda detallada la distribución de los archivos dentro del disco y los códigos utilizados tanto de VHDL como de MATLAB.

9.1 SIMULACIÓN EN MATLAB

Si se desea realizar la simulación en Matlab, es necesario abrir los siguientes documentos:


ModuaciónVectorial_SimplificadoMCP.m


RST_to_ab.m


ModuaciónVectorialMCP.slx

- ModulaciónVectorial_SimplificadoMCP es el *Script* donde está contenido el código de la modulación vectorial. En él, pueden modificarse los datos del circuito y los parámetros de simulación.
- RST_to_ab se utiliza para representar el espacio vectorial alpha-beta como en la imagen 6-28.
- ModulaciónVectorialMCP es el archivo de Simulink donde está contruído el circuito de un inversor trifásico [16].

El circuito ya viene con los parámetros de simulación ajustados, por lo que no es necesario variarlos. El tiempo de ejecución es opcional. Se recomienda utilizar tiempos bajos si el Mf es muy elevado o F1 es muy bajo para evitar simulaciones muy largas.

Si se desea trabajar con las muestras cuantificadas tal y como está programado en VHDL, se debe abrir el siguiente archivo junto a los dos últimos anteriores:


ModuaciónVectorial_Cuantificada.m

Su estructura y uso es semejante a ModulaciónVectorial_SimplificadoMCP, donde los datos y parámetros de simulación aparecen al principio de código para ser su modificación.

Para poder visualizar las señales generadas a partir de VHDL en Matlab, se ha adjuntado junto a un código adaptado, Modulacion_Vectorial_AS_ Simulacion, el *TestBench* que genera los diferentes *Scripts* con los datos. Para poder representar las gráficas, únicamente es necesario importar los datos desde la ventana principal de Matlab "*Import Data*", darle un nombre, y ejecutar las muestras mediante "*plot"* u otro comando de representación.

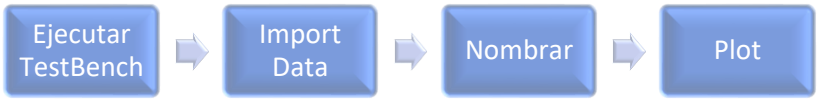


Imagen 9.1: Diagrama datos Modelsim-Matlab

A menudo, las muestras que han sido generadas no son las que uno desea representar, para ello, conviene recorrer la parte del vector que se quiere visualizar.

9.2 SIMULACIÓN EN MODELSIM

La simulación del código se ha realizado mediante la aplicación ModelSim por su facilidad para visualizar las formas de onda. Para su ejecución, basta con seleccionar ModelSim como Simulador dentro de la aplicación ISE Project Navigator:

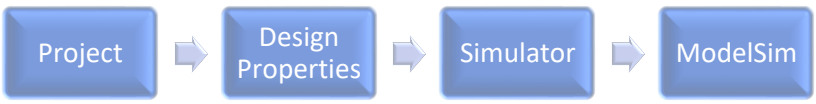


Imagen 9.2: Diagrama funcionamiento ModelSim

De esta forma, debería aparecer en la ventana de diseño el icono de ModelSim como muestra la siguiente imagen:

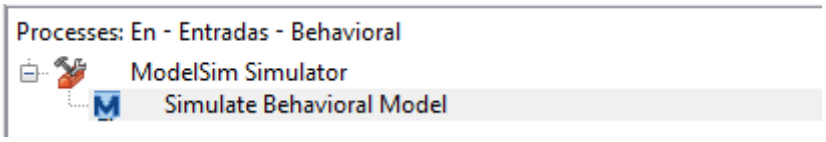


Imagen 9.3: Captura Xilinx ISE project

El proyecto recomendado para realizar pruebas en simulación es el contenido en la carpeta Modulacion_Vectorial_AS_ Simulacion. El testbench aquí adjuntado tiene señales que permiten fácilmente configurar los parámetros en los instantes de tiempo que se deseen. Además, contiene el proceso para exportar datos en los archivos de escritura. Para ello es conveniente seguir ciertas pautas:

1. Actualizar el directorio en las líneas de código correspondientes, al ordenador donde se va a llevar a cabo la simulación.
2. Dependiendo de las variables que se quieren exportar y de los parámetros de la simulación, es necesario pensar previamente cuantas muestras se necesitan obtener por número de ciclos. En el archivo adjunto, está configurado para obtener muestras de las señales senoidales y las ondas moduladoras.
3. Evitar exportar los datos antes de completar todas las inicializaciones, poniendo un tiempo espera.
4. Poner un límite de tiempo en la simulación para evitar generar archivos demasiado pesado en caso de despiste.

Los documentos de escritura serán generados una vez se comience la simulación en ModelSim. Si se quieren visualizar en la misma aplicación las señales, basta con arrastrarlas a la ventana wave:

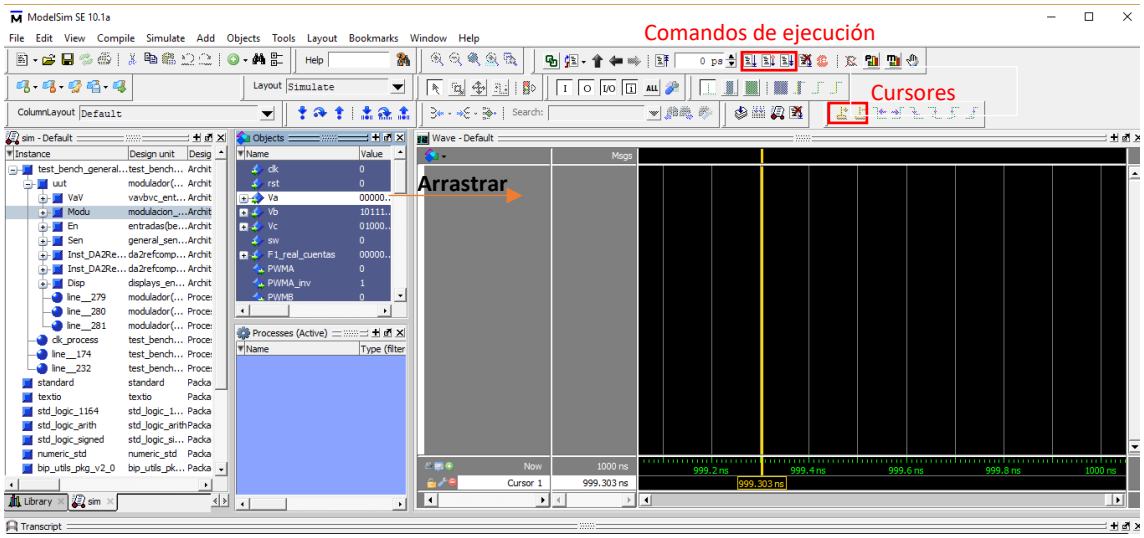


Imagen 9.4: Captura ModelSim

Los cursores (línea vertical amarilla), indican el instante de tiempo, y son necesarios para comprobar los periodos de las señales.

Para observar las ondas senoidales en formato analógico solo hay hacer click derecho en la señal dentro de la ventana Wave-default, “Format” y seleccionar Automatic analog.

Nota: No es necesario customizar el formato analógico en un primer principio, una vez realizada la simulación, si situamos el ratón tal y como muestra la siguiente imagen, la propia aplicación ajustará la señal a sus valores máximos y mínimos:

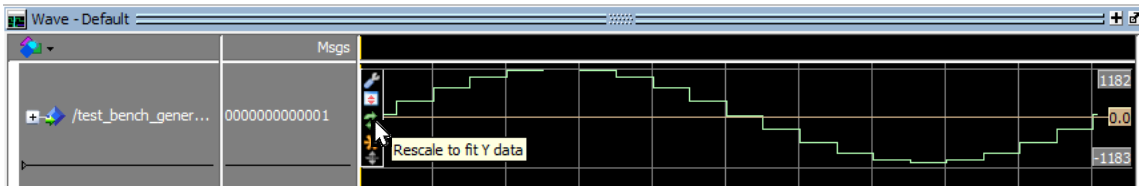


Imagen 9.5: Gráfica en ModelSim

9.3 DESCARGA EN PLACA

Antes de realizar las pruebas en el laboratorio de trabajo es necesario cumplir unas normas de seguridad con el fin de evitar accidentes. Este apartado ha sido transcrito del TFG de Edel Díaz [6], ya que ha sido utilizado el mismo Hardware para ambos proyectos:

9.3.1 Medidas de seguridad

Normas Generales:

1. Antes de trabajar con cualquier aparato electrónico, es recomendado leerse sus manuales de usuario, con el fin de establecer las conexiones correctas y evitar cortocircuitos y la sobrealimentación de sus fuentes.
2. Mantener fuera de la zona de trabajo cualquier tipo de líquido o sustancia inflamable.
3. Cerciorarse de que las conexiones son las correctas antes de encender las fuentes de alimentación.
4. Siempre que se quiera quitar un cable o hacer un cambio en las conexiones desconectar primero el sistema.
5. Utilizar material en buen estado y pedir ayuda al tutor/a si ocurre algún problema.
6. No exceder los límites de trabajo de cada dispositivo.
7. En general, los aparatos utilizados tienen un coste considerable y deben ser tratados con cuidado.

Normas de seguridad del cubo de potencia Semiteach-IGBT

1. No alimentar el dispositivo a más de 750 VDC, el cubo no contiene ningún sistema de protección ante sobretensiones.
2. Una vez apagado el sistema, los condensadores pueden permanecer cargados durante unos minutos.
3. Está prohibido desplazar el aparato mientras esté conectado a la red eléctrica.

4. En caso de fallo o de señales de salida indeseadas, tener en cuenta la señal de error que proporciona la unidad.
5. No exceder una corriente de trabajo de 30 A.
6. El cubo dispone un sensor térmico en el ventilador que informa de posibles sobrecalentamientos.
7. Se recomienda conectar el sistema de refrigeración correctamente y no cubrirlo en ningún momento

Normas de seguridad del circuito de adaptación

1. No alimentar a más de 20 V.
2. No desconectar los cables mientras se encuentre conectado a la corriente.

Normas de seguridad de la tarjeta Digilent Nexys2 Spartan3E-XC3S1200E

1. No superar una alimentación de 5V por vía USB. Comprobar que la alimentación externa proporcione la corriente necesaria a la placa.
2. Un mal uso de la placa puede provocar la rotura de los pines, por ello, se ruega cuidado al conectar los cables.

9.3.2 Conexionado del sistema

Antes de comenzar este apartado se deben leerte atentamente las normas de seguridad anteriores para prevenir accidentes.

Las conexiones que se deben realizar en el diseño general son las siguientes:

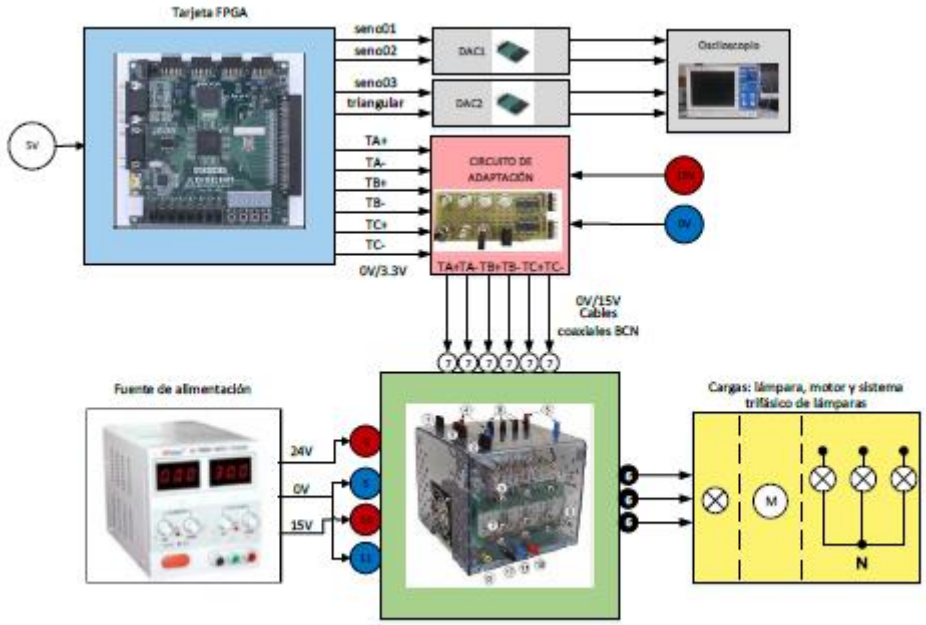


Imagen 9.6: Esquema general del sistema de potencia. (TFG E. Díaz [6])

Las conexiones del cubo de potencia Semiteach-IGBT quedan detalladas en el anexo XX. Si es la primera vez que se realiza el montaje es conveniente consultar al tutor antes de la puesta en marcha.

La descarga del programa en la placa se lleva a cabo conectando la placa al ordenador mediante conector USB-miniUSB. La aplicación Adept de Digilent permite establecer la conexión entre los dispositivos y enviar el archivo. bit con el programa.

Las conexiones entre la tarjeta FPGA y los módulos DAC y el circuito de adaptación son las siguientes:

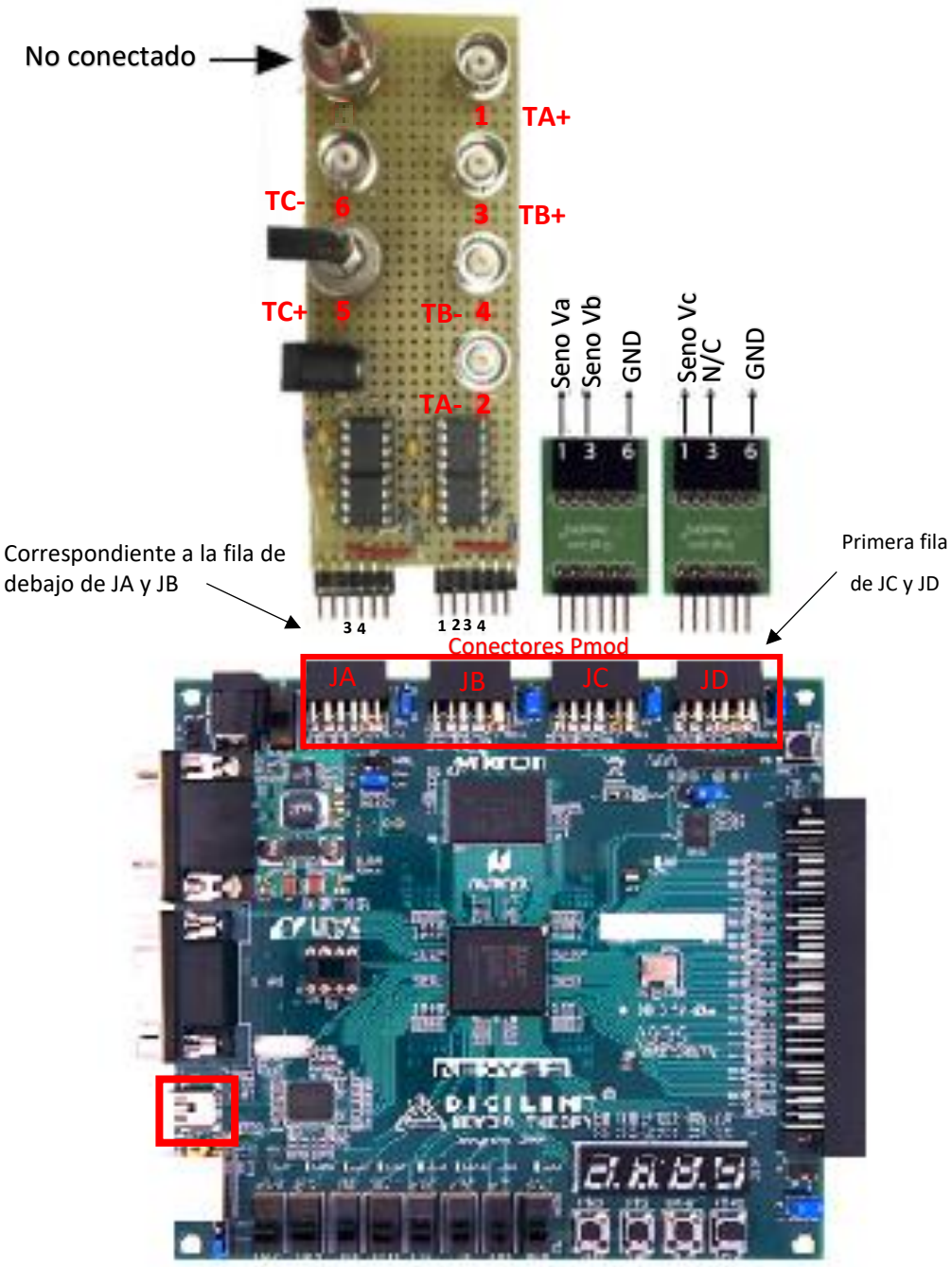


Imagen 9.7: Conexión FPGA-Circuito de adaptación [6]

Conexiones Pmod-DAC

PINES JC Y JD	SEÑALES DE SALIDA FPGA	SEÑALES DE SALIDA DAC
P1	SYNC	Vout1
P2	DINA	N/C
P3	DINB	Vout2
P4	SCLK	N/C
P5	GND	GND
P6	VCC	VCC

Tabla 9.1: Conexiones Pmod-DAC

Conexiones Pmod-Circuito de adaptación

SEÑALES DE SALIDA: MODULACIÓN	
SEÑALES DEL DISEÑO	Pines FPGA
TA+	JB7:P17
TA-	JB8:R16
TB+	JB9:T18
TB-	JB10:U18
TC+	JA9:M14
TC-	JA10:M16

Tabla 9.2: Conexiones Pmod-Circuito de adaptación

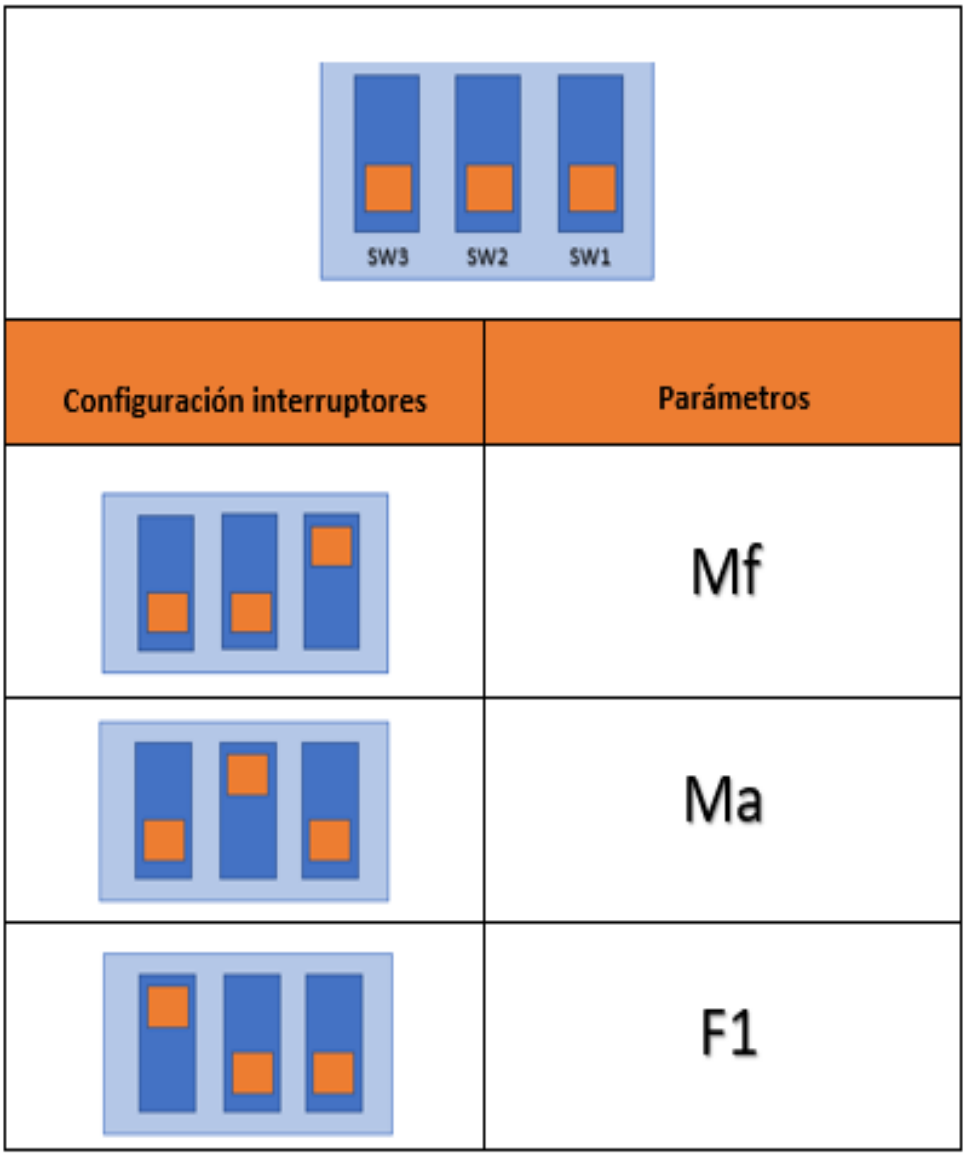
9.3.3 Modo de funcionamiento

El programa generado tiene una configuración por defecto tras reset de Mf=5760, Ma=1 y F1=100Hz.

Una vez iniciada la práctica, los interruptores y pulsadores que permiten modificar los parámetros del inversor se detallan en la siguiente tabla X.

El diseño lleva incorporado un sistema antirrebotes que permite incrementar en una unidad del parámetro correspondiente por cada accionamiento de los pulsadores.

En caso de saturación del programa, cuando las señales de salida no respondan a las configuraciones, pulsar el reset general de la placa y volver a descargar el archivo. bit en la placa.



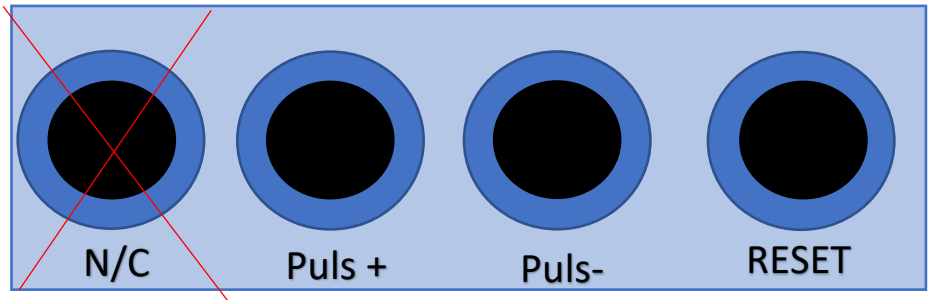


Imagen 9.8: Configuración de interruptores y pulsadores

	Posibles configuraciones de trabajo		
Pulsadores	Ma	MF	F1
Pulsador + / Pulsador -	1	(5760:5700) -> 10	100
	0.875	(5700:5500) -> 100	90
	0.75	(5500:500) -> 500	80
	0.625	(500:100) -> 100	70
	0.5	(100: 10) -> 10	60
	0.25	(10:6) -> 1	50
	0.125		40
	0.03125		30
	0.015		20

Tabla 9.3: Valores parámetros de modulación

IMPORTANTE: Si las pruebas van a ser llevadas a cabo con el cubo de potencia descrito en este documento, utilizar únicamente configuraciones de Mf y F1 bajas, <30 y <70 correspondientemente. Tal y como se ha desarrollado en el trabajo relacionado con este manual, los interruptores de potencia no son capaces de conmutar tan rápido como en simulación.

9.4 ANEXOS

9.4.1 Conexiones del cubo de potencia Semiteach-IGBT

El cubo de Potencia Semiteach-IGBT es un dispositivo constituido por componentes electrónicos de potencia, con conectores de seguridad tipo banana y BNC y diferentes sistemas de seguridad que hacen su funcionalidad puramente educativa. Ha sido diseñado para un uso docente donde los errores son habituales, por ello, tiene una alta protección a fallos.

magen 9.9: Conexiones cubo de potencia Semiteach_IGBT [7.3]

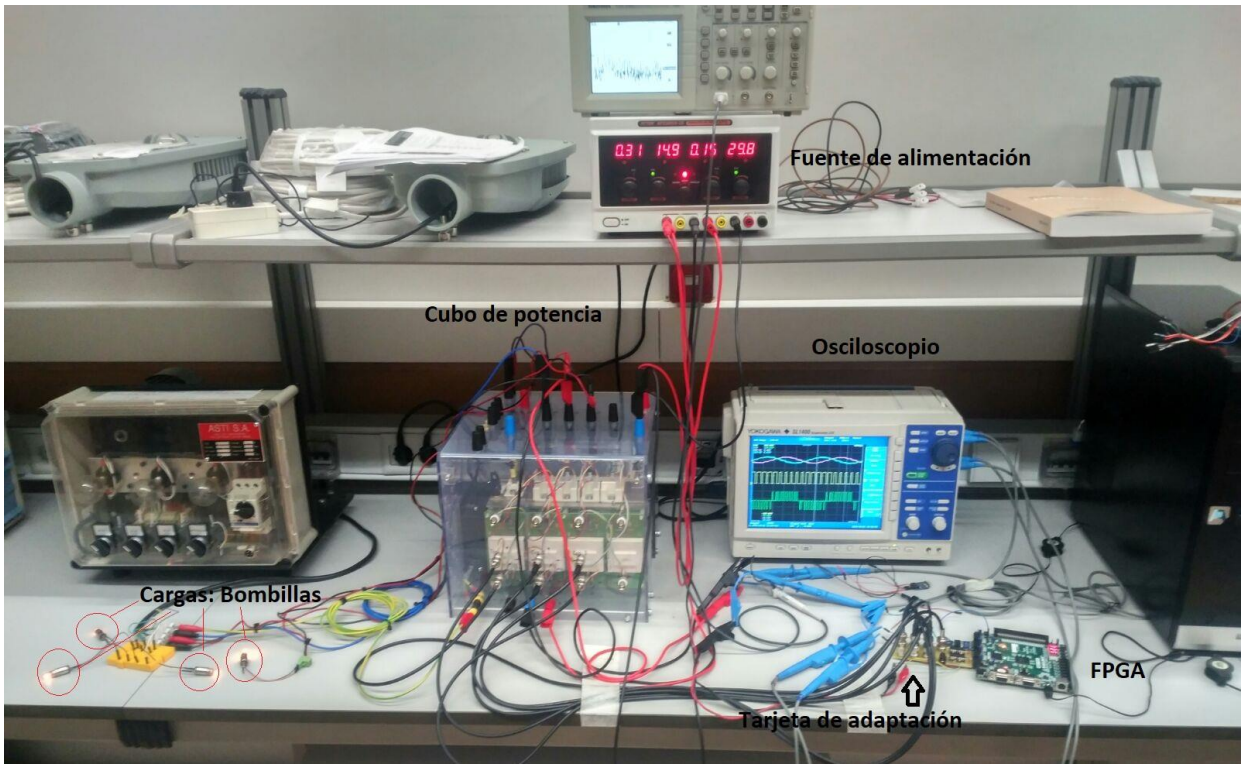
La siguiente tabla muestra todos los conectores presentes en el cubo. Fue elaborada por E.Díaz [6] en el desarrollo de su TFG.

Nº	Tipo	Función	Nivel de Voltaje	Máximo nivel de corriente
0	Tierra del panel	Conexión tierra	0	30A
1	Conector Banana 4mm	Fuente de alimentación del ventilador	230V/50Hz	1A
2	Conector Banana 4mm	Termal trip	15V	5A
3	Conector Banana 4mm	Entrada rectificador	230/ 400V	30A
4	Conector Banana 4mm	Salida DC rectificador	600VDC (rojo positivo, azul negativo)	30A
5	Conector Banana 4mm	Entradas DC del inversor IGBT	600VDC (rojo positivo, azul negativo)	30A
6	Conector Banana 4mm	Inversor IGBT AC + salidas choppers	400 VAC / 600 VDC	30A
7	Coaxial aislado BNC	Entrada PWM inversor	C-MOS lógico 0/15V, 0V= IGBT abierto, 15V=IGBT cerrado	1A
8	Coaxial aislado BNC	Entrada PWM chopper	C-MOS lógico 0/15V, 0V= IGBT abierto, 15V=IGBT cerrado	1A
9	Coaxial aislado BNC	Error de salida	C-MOS lógico 0/15V	1A
10	Conector Banana 4mm	15V Alimentación del driver	15V	5A
11	Conector Banana 4mm	0V Alimentación del driver	15V	5A
12	Conector Banana 4mm	Sensor de temperatura	0-5V	1A

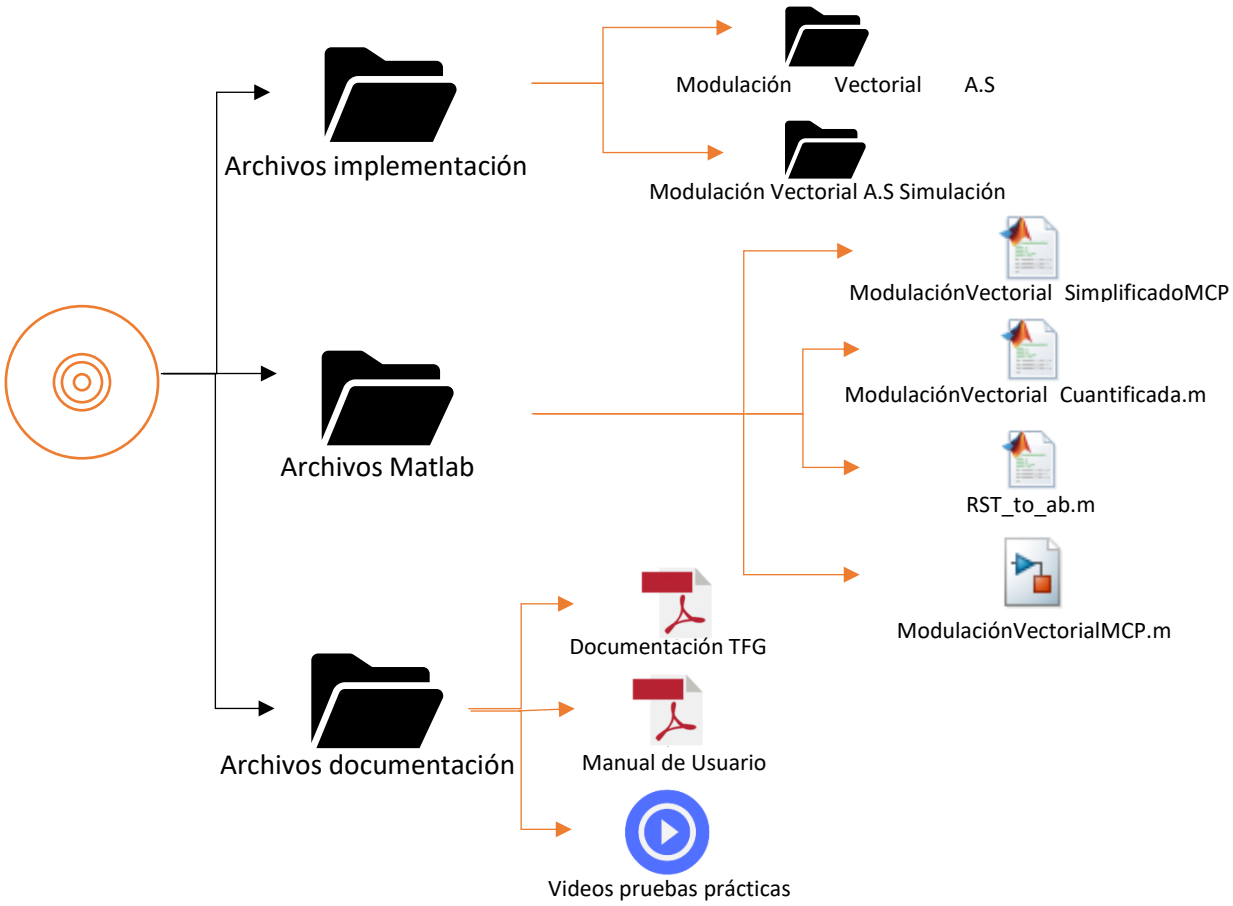
Tabla 9.4: Conexiones cube Semiteach-IGBT [6]

Las conexiones necesarias para este diseño son: 1,5,6,7,10 y 11.

9.4.2 Conexión general del sistema

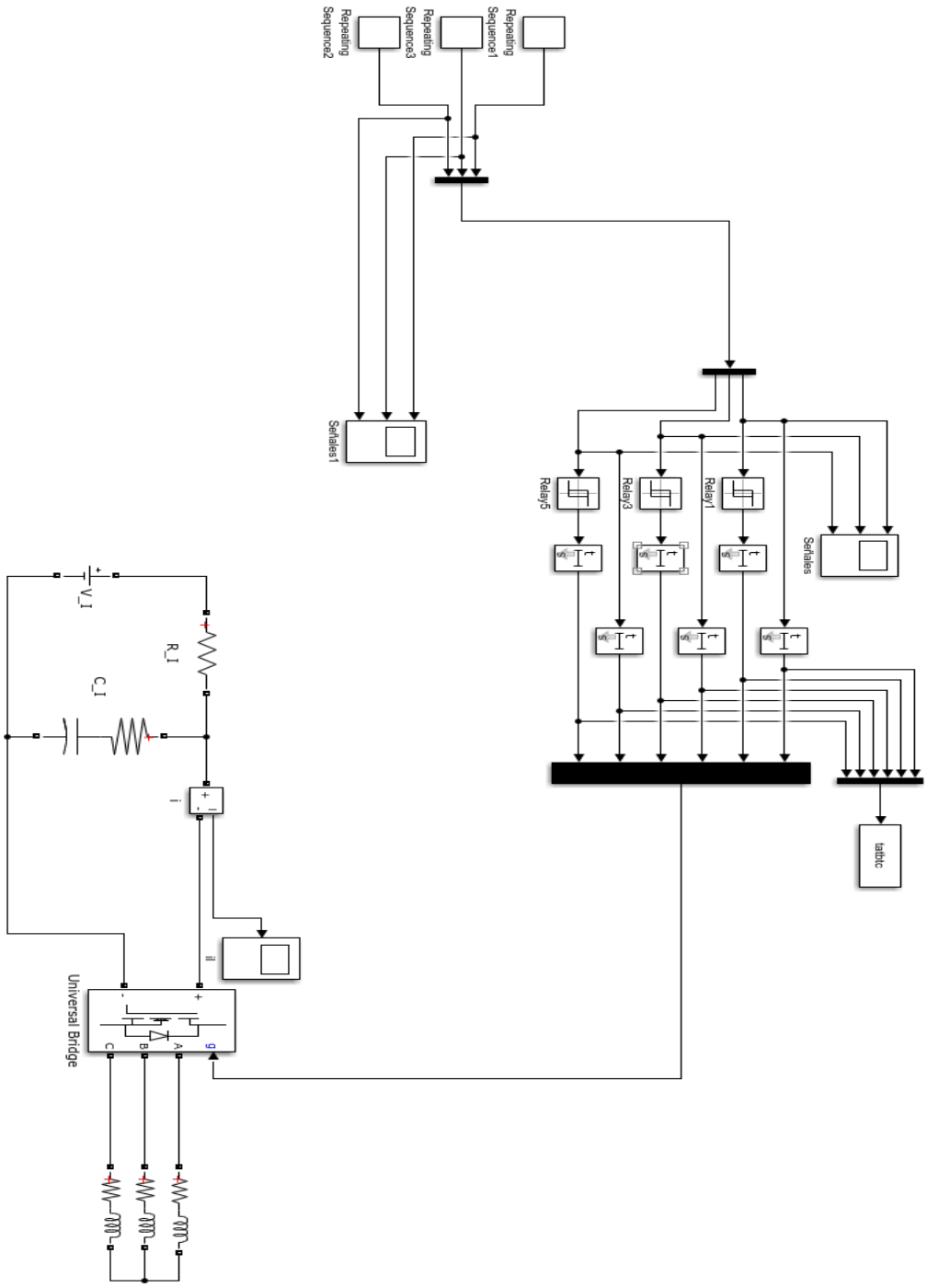


9.4.3 Organización de los archivos en el disco



9.4.4 Circuito de inversor trifásico en Simulink [16]

El circuito original incluye bloques para el seguimiento de los resultados, que no fueron incluidos para poder visualizar mejor el diseño principal.



10 CÓDIGOS VHDL RELEVANTES

10.1 DISEÑO ANTIRREBOTES

10.1.1 FLIP-FLOP

```
entity Flip_flop is
    Port ( clk : in  STD_LOGIC;
           btn_in : in  STD_LOGIC;
           btn_out : out  STD_LOGIC);
end Flip_flop;

architecture Behavioral of Flip_flop is

constant CNT_SIZE : integer := 19;
    signal btn_prev   : std_logic ;
    signal counter    : std_logic_vector(CNT_SIZE downto 0);
begin
    process(clk)
    begin
     if (clk'event and clk='1') then
          if (btn_prev xor btn_in) = '1' then
               counter <= (others => '0');
               btn_prev <= btn_in;
          elsif (counter(CNT_SIZE) = '0') then
               counter <= counter + 1;
          else
               btn_out <= btn_prev;
          end if;
     end if;
    end process;
end Behavioral;
```

Código 10-1: FLIP-FLOP

10.1.2 Antirrebotes

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity antirrebotes_ent is
     port(
          clk: in std_logic;--50Mhz
          entrada: in std_logic;--Es usado tanto para el
pulsador 1 como el pulsador 2, para evitar posibles anomalias
      salida: out std_logic
          );
end entity antirrebotes_ent;

architecture antirrebotes_arch of antirrebotes_ent is
type FSMstate is (q0, q1, q2);
signal est_actual : FSMstate;
begin
process(clk)
     begin
```

```
          if clk='1' and clk'event then
               case est_actual is
                    when q0=>
                         if (entrada='1') then est_actual<=q1;
                         else est_actual<=q0;
                         end if;
                    when q1=> --Entra cuando se acciona el
pulsador y genera el mismo un unico pulso
                         est_actual<=q2;
                    when q2=> --No se podra activar otra vez
hasta que no se haya soltado el pulsador
                         if (entrada='0') then est_actual<=q0;
                         else est_actual<=q2;
                         end if;
               end case;
          end if;
end process;
process(est_actual)
begin
     case est_actual is
          when q0=>
               salida<='0';
          when q1=>
               salida<='1';
          when q2=>
               salida<='0';
     end case;
end process;
end antirrebotes_arch;
```

Código 10-2: Antirrebotes

10.2 CONTADOR SENO

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
entity contador_seno_va_ent is
     port(
          rst: in std_logic;
          clk: in std_logic;--50Mhz
          Ts_Base: in std_logic;--Tiempo de muestreo base
          change: out std_logic;--Enable para acceder a la ROM
          signo: out std_logic;--Bit de signo
          salida_rom:out std_logic_vector(10 downto 0)-- Accede
a la memoria rom
          );
end entity contador_seno_va_ent;

architecture contador_seno_va_arch of contador_seno_va_ent is
signal entrada: std_logic_vector(3 downto 0);
begin
process(clk,rst)
variable aux,cnt: integer;
     begin
```

```
          if(rst='1') then
               change<='1';
               cnt:=0;--Contador de dirección
               aux:=0;--Auxiliar para el cambio entre
cuadrantes
               entrada<="0001";
               salida_rom<=CONV_STD_LOGIC_VECTOR(cnt, 11);
              signo<='0';
          elsif(clk 'event and clk='1') then
               if Ts_Base='1' then entrada is
                    change<='1';
                    case entrada is
                         when "0001" =>
                              cnt:=cnt+1-aux;
                              aux:=0;
                              if cnt=1439 then
                                   entrada<="0010";
                                   aux:=2;
                                   end if;
                                   signo<='0';
                         when "0010"=>
                              cnt:=cnt-1+aux;
                              aux:=0;
                              if cnt=1 then
                                   entrada<="0100";
                                   aux:=2;
                                   end if;
                                   signo<='0';
                         when "0100"=>
                              cnt:=cnt+1-aux;
                              aux:=0;
                              if cnt=1439 then
                                   entrada<="1000";
                                   aux:=2;
                                   end if;
                                   signo<='1';
                         when "1000"=>
                              cnt:=cnt-1+aux;
                              aux:=0;
                              if cnt=1 then
                                   entrada<="0001";
                                   aux:=2;
                                   end if;
                                   signo<='1';
                         when others=>
                              cnt:=0;
                    end case;
                    salida_rom<=CONV_STD_LOGIC_VECTOR(cnt,
11);
               else change<='1';
               end if;
          end if;
end process;
end contador_seno_va_arch;
```

Código 10-3: Contador Seno

10.3 GESTOR TS_BASE

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_unsigned.all;
use IEEE.std_logic_arith.all;

entity Gestor_Ts_Base_ent is
    Port ( rst : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           Pulsador1 : in  STD_LOGIC;
           Pulsador2: in std_logic;
           Ts_Real_cuentas : out  STD_LOGIC_VECTOR (21 downto
0);--Cuentas para el tiempo de muestreo real
           F1_disp : out STD_LOGIC_VECTOR(19 downto 0);--
Direcciones para el display
           Ts_Base_cuentas : out  STD_LOGIC_VECTOR (8 downto
0));--Cuentas del tiempo de muestreo base
end Gestor_Ts_Base_ent;
architecture Gestor_Ts_Base_arch of Gestor_Ts_Base_ent is
begin
    process(clk,rst)
    variable cont: integer;
    begin
     if(rst='1') then
          cont:=8;--Iniciamos con 100 Hz
          Ts_Real_cuentas<="00011110100001001000000"; --500000
          Ts_Base_cuentas<="001010111";--87,100Hz
          F1_disp<="01011010100101000101";--100H
     elsif(clk 'event and clk='1') then
         if (pulsador1 = '1')AND cont<8 then
          cont:=cont+1;
          elsif (Pulsador2='1')AND cont>0 then
          cont:=cont-1;
          end if;
           case cont is
           when 0 =>
           Ts_Base_cuentas<="110110010";--434,20Hz
           Ts_Real_cuentas<="10011000100101101000000";--2500000
           F1_disp<="01100010100010101001";--20Hc
         when 1=>
           Ts_Base_cuentas<="100100001";--289,30Hz
           Ts_Real_cuentas<="01100101101110011101011";--1666667
           F1_disp<="01101010100010101001";--30Hc
           when 2 =>
           Ts_Base_cuentas<="011011001";--217,40Hz
           Ts_Real_cuentas<="01001100010010110100000";--1250000
           F1_disp<="01110010100010101001";--40Hc
           when 3 =>
           Ts_Real_cuentas<="00111101000010010000000";--1000000
           Ts_Base_cuentas<="010101110";--174,50Hz
           F1_disp<="01110010100010101001";--50Hc
           when 4 =>
           Ts_Real_cuentas<="00110010110111001101010";--833333
           Ts_Base_cuentas<="010010001";--145,60Hz
           F1_disp<="10000010100010101001";--60Hc
```

```
            when 5 =>
            Ts_Real_cuentas<="00101011100110001011l0";--714286
            Ts_Base_cuentas<="001111100";--124,70Hz
            F1_disp<="10001010100010101001";--70Hc
            when 6 =>
            Ts_Real_cuentas<="00100110001001011010l0";--625000
            Ts_Base_cuentas<="001101101";--109,80Hz
            F1_disp<="10010010100010101001";--80Hc
            when 7 =>
            Ts_Real_cuentas<="00100001110000000000";--552960
            Ts_Base_cuentas<="001100000";--96,90Hz
            F1_disp<="10011010100010101001";--90Hc
            when others =>
            Ts_Real_cuentas<="00011110100001001000l0";--500000
            Ts_Base_cuentas<="001010111";--87,100Hz
            F1_disp<="01011010100101000101";--100H
            end case;
     end if;
    end process;
end Gestor_Ts_Base_arch;
```

Código 10-4:Gestor Ts Base

10.4 MF_ENT

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity Mf_ent is
    Port ( rst : in  STD_LOGIC;
           clk : in  STD_LOGIC;
           Pulsador1 : in  STD_LOGIC;--Pulsador +
           Pulsador2 : in  STD_LOGIC;--Pulsador -
          en: out STD_LOGIC;--Enable para activar el divisor
           mf : out  STD_LOGIC_VECTOR (12 downto 0));
end Mf_ent;

architecture Mf_arch of Mf_ent is

begin
    process(clk,rst)
      variable cont: Integer;
      variable Mf_cont:Integer;
    begin
     if(rst='1') then
           Mf_cont:=5760;
           Mf<=CONV_STD_LOGIC_VECTOR(Mf_cont, 13);
           en<='1';
     elsif(clk 'event and clk='1') then
         if (Pulsador1 = '1') AND Mf_cont<5760 then
                en<='1';
           if Mf_cont<10 then
                cont:=1;
               Mf_cont:=Mf_cont+cont;
```

```
                Mf<= CONV_STD_LOGIC_VECTOR(Mf_cont, 13);
            elsif (Mf_cont>10)AND(Mf_cont<100) then
                cont:=10;
                Mf_cont:=Mf_cont+cont;
                 Mf<= CONV_STD_LOGIC_VECTOR(Mf_cont, 13);
            elsif Mf_cont>=100 AND Mf_cont<500 then
                cont:=100;
                Mf_cont:=Mf_cont+cont;
                 Mf<= CONV_STD_LOGIC_VECTOR(Mf_cont, 13);
            elsif Mf_cont>=500 AND Mf_cont<5500 then
                cont:=500;
                Mf_cont:=Mf_cont+cont;
                 Mf<= CONV_STD_LOGIC_VECTOR(Mf_cont, 13);
            elsif Mf_cont>=5500 AND Mf_cont<5700 then
                 cont:=100;
                 Mf_cont:=Mf_cont+cont;
                 Mf<= CONV_STD_LOGIC_VECTOR(Mf_cont, 13);
            else
                 cont:=10;
                 Mf_cont:=Mf_cont+cont;
                 Mf<= CONV_STD_LOGIC_VECTOR(Mf_cont, 13);
           end if;
           elsif (Pulsador2='1') AND Mf_cont>6 then
                en<='1';
           if Mf_cont<=10 then
                cont:=1;
                Mf_cont:=Mf_cont-cont;
                 Mf<= CONV_STD_LOGIC_VECTOR(Mf_cont, 13);
            elsif (Mf_cont>10)AND(Mf_cont<=100) then
                 cont:=10;
                Mf_cont:=Mf_cont-cont;
                 Mf<= CONV_STD_LOGIC_VECTOR(Mf_cont, 13);
            elsif Mf_cont>100 AND Mf_cont<=500 then
                 cont:=100;
                 Mf_cont:=Mf_cont-cont;
                 Mf<= CONV_STD_LOGIC_VECTOR(Mf_cont, 13);
            elsif Mf_cont>500 AND Mf_cont<=5500 then
                cont:=500;
                Mf_cont:=Mf_cont-cont;
                 Mf<= CONV_STD_LOGIC_VECTOR(Mf_cont, 13);
            elsif Mf_cont>5500 AND Mf_cont<=5700 then
                cont:=100;
                 Mf_cont:=Mf_cont-cont;
                Mf<= CONV_STD_LOGIC_VECTOR(Mf_cont, 13);
            else
                 cont:=10;
                 Mf_cont:=Mf_cont-cont;
                 Mf<= CONV_STD_LOGIC_VECTOR(Mf_cont, 13);
           end if;
           else en<='1';
           end if;
            Mf<= CONV_STD_LOGIC_VECTOR(Mf_cont, 13);
           end if;
    end process;
end Mf_arch;
```

Código 10-5: Mf_ent

10.5 MA_ENT

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;
use IEEE.std_logic_arith.all;

entity Ma_ent is
Generic(TAM_SEÑAL12:INTEGER);
    Port ( Sen_in: in std_logic_vector(TAM_SEÑAL12 downto 0);
           clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
            ma_sel: in std_logic_vector(3 downto 0);
           ma_disp: out std_logic_vector(19 downto 0);
           Sen_out : out  STD_LOGIC_VECTOR (TAM_SEÑAL12 downto
0));
end Ma_ent;

architecture Ma_arch of Ma_ent is
begin
process(clk,rst)
variable aux1,aux2,aux3,aux4:integer;
      begin
      if rst='1' then
           Sen_out<=(others =>'0');
          ma_disp<=(OTHERS => '0');
          aux1:=0;
          aux2:=0;
          aux3:=0;
          aux4:=0;
      elsif(clk 'event and clk='1') then
          case ma_sel is
               when "0000" =>
                    Sen_out<=Sen_in;--Ma=1
                    Ma_disp<="10100101001010001011";--1
               when "0001" =>
                    aux1:=CONV_INTEGER(Sen_in(TAM_SEÑAL12
downto 1));--Ma=0.5
                    aux2:=CONV_INTEGER(Sen_in(TAM_SEÑAL12
downto 2));--Ma=0.25
                    aux3:=CONV_INTEGER(Sen_in(TAM_SEÑAL12
downto 3));--Ma=0.125
                    aux4:=aux1+aux2+aux3;

     Sen_out<=CONV_STD_LOGIC_VECTOR(aux4,TAM_SEÑAL12+1);--
Ma=0.875 (0001); Ma=0.75 (0010); Ma=0.625 (0011); Ma=0.5 (0100);
Ma=0.25 (0101); Ma=0.125 (0110); Ma=0.03125 (0111); Ma=0.015625
(1000)
               Ma_disp<="10110100101000101111";--0.875
               when "0010" =>
                    aux1:=CONV_INTEGER(Sen_in(TAM_SEÑAL12
downto 1));--Ma=0.5
                    aux2:=CONV_INTEGER(Sen_in(TAM_SEÑAL12
downto 2));--Ma=0.25
                    aux3:=aux1+aux2;--
                    Ma_disp<="10100101101000101111";--0.75
```

```
     Sen_out<=CONV_STD_LOGIC_VECTOR(aux3,TAM_SEÑAL12+1);--
Ma=0.75
                when "0011" =>
                     aux1:=CONV_INTEGER(Sen_in(TAM_SEÑAL12
downto 1));--Ma=0.5
                     aux2:=CONV_INTEGER(Sen_in(TAM_SEÑAL12
downto 3));--Ma=0.125
                     Ma_disp<="10110100000110001111";--0.625
                     aux3:=aux1+aux2;

     Sen_out<=CONV_STD_LOGIC_VECTOR(aux3,TAM_SEÑAL12+1);--
Ma=0.625
                when "0100" =>
                Sen_out<=Sen_in(TAM_SEÑAL12)&Sen_in(TAM_SEÑAL12
downto 1);--Ma=0.5
                     Ma_disp<="10100101001011001111";--0.5
                when "0101"=>

     Sen_out<=Sen_in(TAM_SEÑAL12)&Sen_in(TAM_SEÑAL12)&Sen_in(TAM
_SEÑAL12 downto 2);--Ma=0.25
                     Ma_disp<="10100101100110001111";--0.25
                when "0110"=>

     Sen_out<=Sen_in(TAM_SEÑAL12)&Sen_in(TAM_SEÑAL12)&Sen_in(TAM
_SEÑAL12)&Sen_in(TAM_SEÑAL12 downto 3);--Ma=0.125
                     Ma_disp<="10110010110110001111";--0.125
                when "0111"=>

     Sen_out<=Sen_in(TAM_SEÑAL12)&Sen_in(TAM_SEÑAL12)&Sen_in(TAM
_SEÑAL12)&Sen_in(TAM_SEÑAL12)&Sen_in(TAM_SEÑAL12)&Sen_in(TAM_SEÑ
AL12 downto 5);--Ma=0.03125
                     Ma_disp<="10110010100110101011";--0.031
                when "1000"=>
     Sen_out<=Sen_in(11)&Sen_in(11)&Sen_in(TAM_SEÑAL12)&Sen_in(T
AM_SEÑAL12)&Sen_in(TAM_SEÑAL12)&Sen_in(TAM_SEÑAL12)&Sen_in(TAM_S
EÑAL12 downto 6);--Ma=0.015625
                     Ma_disp<="10110010100101101111";--0.015
                when OTHERS=>
                     Sen_out<=Sen_in;
                end case;
           end if;
     end process;
end Ma_arch;
```

Código 10-6: Ma_ent

10.6 XdXq

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;

entity XdXq is
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           Va : in  STD_LOGIC_VECTOR (12 downto 0);
           Vb : in  STD_LOGIC_VECTOR (12 downto 0);
           Vc : in  STD_LOGIC_VECTOR (12 downto 0);
           Xd_out : out  STD_LOGIC_VECTOR (15 downto 0);
                  Xq_out: out  STD_LOGIC_VECTOR (15 downto 0));
end XdXq;
architecture Behavioral of XdXq is
begin
     Process(clk,rst)
     variable auxXd,auxXq,auxVa,auxVb,auxVc:INTEGER;
     begin
          if rst='1' then
               Xd_out<=(OTHERS=>'0');
               Xq_out<=(OTHERS=>'0');
          elsif clk'event and clk='1' then
               auxVa:=CONV_INTEGER(Va&'0');
               auxVb:=CONV_INTEGER(Vb);
               auxVc:=CONV_INTEGER(Vc);
               auxXq:=auxVb-auxVc;
               auxXd:=auxVa-auxVb-auxVc;
               Xd_out<= CONV_STD_LOGIC_VECTOR(auxXd,16);
               Xq_out<= CONV_STD_LOGIC_VECTOR(auxXq,16);
          end if;
     end process;
end Behavioral;
```

Código 10-7: Xd y Xq

10.7 TIEMPOS TN TN1

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;

entity Tns_ent is
Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           Xq : in  STD_LOGIC_VECTOR (15 downto 0);
           Xd : in  STD_LOGIC_VECTOR (15 downto 0);
           Sector : in  STD_LOGIC_VECTOR (2 downto 0);
           ce : out STD_LOGIC;--enable para el multiplicador
           Tn : out  STD_LOGIC_VECTOR (16 downto 0);
           Tn1 : out  STD_LOGIC_VECTOR (16 downto 0));
end Tns_ent;
architecture Behavioral of Tns_ent is
begin
     process(rst,clk)
     variable aux1,Tn_s,Tn1_s,auxXd,auxXq: INTEGER;
     begin
          if rst='1' then
               Tn<="00000000000000000";
               Tn1<="00000000000000000";
          elsif clk='1' AND clk'event then
               ce<='0';
               if Conv_integer(Xq)/=auxXq OR
Conv_integer(Xd)/=auxXd then
                    ce<='1';
                    end if;
                    auxXq:=Conv_INTEGER(Xq);
                    auxXd:=Conv_INTEGER(Xd);
               case Sector is
                    when "001" =>
                    Tn_s:=Conv_integer(Xd(15 downto 1)) -
Conv_integer(Xq(15 downto 1));
                   Tn<=CONV_STD_LOGIC_VECTOR(Tn_s,17);
                   Tn1<=Xq(15)&Xq(15 downto 0);
                    when "010" =>
                    Tn_s:=Conv_integer(Xq(15 downto 1)) +
Conv_integer(Xd(15 downto 1));
                    Tn1_s:=Conv_integer(Xq(15 downto 1)) -
Conv_integer(Xd(15 downto 1));
                    Tn1<=CONV_STD_LOGIC_VECTOR(Tn1_s,17);
                    Tn<=CONV_STD_LOGIC_VECTOR(Tn_s,17);
                    when "011" =>
                    Tn<=Xq(15)&Xq(15 downto 0);
                    Tn1_s:=0- Conv_integer(Xq(15 downto 1)) -
Conv_integer(Xd(15 downto 1));
                    Tn1<=CONV_STD_LOGIC_VECTOR(Tn1_s,17);
                    when "100" =>
                    Tn_s:=Conv_integer(Xq(15 downto 1)) -
Conv_integer(Xd(15 downto 1));
                    Tn1_s:=0-Conv_integer(Xq(15 downto 0));
                    Tn<=CONV_STD_LOGIC_VECTOR(Tn_s,17);
```

```
                    Tn1<=CONV_STD_LOGIC_VECTOR(Tn1_s,17);
                    when "101" =>
                    Tn_s:=0- Conv_integer(Xd(15 downto 1)) -
Conv_integer(Xq(15 downto 1));
                    Tn1_s:= Conv_integer(Xd(15 downto 1)) -
Conv_integer(Xq(15 downto 1));
                    Tn<=CONV_STD_LOGIC_VECTOR(Tn_s,17);
                    Tn1<=CONV_STD_LOGIC_VECTOR(Tn1_s,17);
                    when "110" =>
                    Tn_s:=0- Conv_integer(Xq(15 downto 0));
                    Tn1_s:=Conv_integer(Xq(15 downto 1)) +
Conv_integer(Xd(15 downto 1));
                    Tn<=CONV_STD_LOGIC_VECTOR(Tn_s,17);
                    Tn1<=CONV_STD_LOGIC_VECTOR(Tn1_s,17);
                    when OTHERS =>
                    Tn<="00000000000000000";
                    Tn1<="00000000000000000";
                    end case;
                end if;
     end process;
end Behavioral;
```

Código 10-8: Tiempos Tn y Tn1

10.8 SECTOR

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.STD_LOGIC_signed.all;

entity Sector_ent is
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           XdXq : in  STD_LOGIC_VECTOR (2 downto 0);--|Xd|>
           Xd: in STD_LOGIC_VECTOR(15 downto 0);
           Xq : in  STD_LOGIC_VECTOR (15 downto 0);
           Xq3 : in STD_LOGIC_VECTOR(17 downto 0);--Xq*3
           switching_sequence: out STD_LOGIC_VECTOR(20 downto
0);--Secuencia de conmutacion siete segmentos
           switching_sequence_Ha: out STD_LOGIC_VECTOR(20 downto
0);--Secuencia de conmutación sin armonicos pares Zona A
           switching_sequence_Hb: out STD_LOGIC_VECTOR(20 downto
0);--Secuencia de conmutacion sin armonicos pares Zona B
           armonic: out STD_LOGIC;--0 =>A,--1=>B
           Sector : out  STD_LOGIC_VECTOR (2 downto 0));
end Sector_ent;

architecture Behavioral of Sector_ent is
begin
     Process(clk,rst)
     variable auxXd,auxXq,auxXq3,auxnotXd,auxnotXq3:INTEGER;
     begin
     if rst='1' then
          Sector<="001";
          Armonic<='0';
```

```
        Switching_sequence<="000100110111110100000";--Sector
        1
        Switching_sequence_Ha<="000100110111110100000";--1-a
        Switching_sequence_Hb<="111110100000100110111";--1-b
    elsif clk'event and clk='1' then
        auxXd:=Conv_INTEGER(Xd(15 downto 0));
        auxnotXd:=0-auxXd;
        auxXq:=Conv_INTEGER(Xq(15 downto 0));
        auxXq3:=Conv_INTEGER(Xq3(17 downto 0));
        auxnotXq3:=0-auxXq3;
         case XdXq is
             when "111" =>
             Sector<="001";--Sector 1
             Switching_sequence<="000100110111110100000";
             if auxXq3>auxXd then
                  Armonic<='1';
             else
                  Armonic<='0';
             end if;
        Switching_sequence_Ha<="000100110111110100000";--1-a
        Switching_sequence_Hb<="111110100000100110111";--1-b
             when "110" =>
             Sector<="010";--Sector 2
             Switching_sequence<="000010110111110010000";
             if auxXd<0 then
                  Armonic<='1';
             else
                  Armonic<='0';
             end if;
        Switching_sequence_Ha<="111110010000010110111";--2-a
        Switching_sequence_Hb<="000010110111110010000";--2-b
             when "010" => Sector<="010";--Sector 2
             Switching_sequence<="000010110111110010000";
             if auxXd<0 then
                  Armonic<='1';
             else
                  Armonic<='0';
             end if;
        Switching_sequence_Ha<="111110010000010110111";--2-a
        Switching_sequence_Hb<="000010110111110010000";--2-b
             when "011" => Sector<="011";--Sector 3
             Switching_sequence<="000010011111011010000";
             if auxXq3<auxnotXd then
                  Armonic<='1';
             else
                  Armonic<='0';
             end if;
        Switching_sequence_Ha<="000010011111011010000";--3-a
        Switching_sequence_Hb<="111011010000010011111";--3-b
             when "001" =>
             Sector<="100";--Sector 4
             Switching_sequence<="000001011111011001000";
             if auxnotXq3>auxnotXd then
                  Armonic<='1';
             else
                  Armonic<='0';
```

```
                end if;
            Switching_sequence_Ha<="11101100100000101111";--4-a
            Switching_sequence_Hb<="00000101111011001000";--4-b
                when "100" =>
                Sector<="101";--Sector 5
                Switching_sequence<="000001101111101001000";
                if auxXd>0 then
                    Armonic<='1';
                else
                    Armonic<='0';
                end if;
            Switching_sequence_Ha<="000001101111101001000";--5-a
            Switching_sequence_Hb<="11101001000001101111";--5-b
                when "000" =>
                Sector<="101";--Sector 5
                if auxXd>0 then
                    Armonic<='1';
                else
                    Armonic<='0';
                end if;
                Switching_sequence<="000001101111101001000";
            Switching_sequence_Ha<="000001101111101001000";--5-a
            Switching_sequence_Hb<="11101001000001101111";--5-b
                when OTHERS =>
                Sector<="110";--Sector 6
                if auxnotXq3<auxXd then
                    Armonic<='1';
                else
                    Armonic<='0';
                end if;
                Switching_sequence<="000100101111101100000";
            Switching_sequence_Ha<="11110110000100101111";--6-a
            Switching_sequence_Hb<="00010010111101100000";--6-b
                end case;
            end if;
     end process;
end Behavioral;
```

Código 10-9: Sector

10.9 PWM

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;

entity PWM is
    Port ( clk : in  STD_LOGIC;
           rst : in  STD_LOGIC;
           T0 : in  STD_LOGIC_VECTOR (28 downto 0);
           Tn : in  STD_LOGIC_VECTOR (26 downto 0);
           Tn1 : in  STD_LOGIC_VECTOR (26 downto 0);
           Xd : in  STD_LOGIC_VECTOR (15 downto 0);
           Xq : in  STD_LOGIC_VECTOR (15 downto 0);
           sector : in STD_LOGIC_VECTOR(2 downto 0);
```

```
           armonic: in STD_LOGIC;--Zona A B
           Eliminacion_pares: in STD_LOGIC;--SW
           switching_sequence: in STD_LOGIC_VECTOR(20 downto 0);
           switching_sequence_Ha: in STD_LOGIC_VECTOR(20 downto
0);
           switching_sequence_Hb: in STD_LOGIC_VECTOR(20 downto
0);
            PWMA : out  STD_LOGIC;--TA+
           PWMA_inv: out STD_LOGIC;--TA-
            PWMB : out  STD_LOGIC;--TB+
           PWMB_inv : out STD_LOGIC;--TB-
            PWMC : out  STD_LOGIC;--TC+
           PWMC_inv : out STD_LOGIC);--TC-
end PWM;

architecture Behavioral of PWM is
signal aux,aux2: std_logic_vector(15 downto 0);
begin
     Process(clk,rst)
     variable cont,auxT04,auxTn2,auxTn12,auxT02:INTEGER;
     begin
     if rst='1' then
          PWMA<='0';
          PWMB<='0';
          PWMC<='0';
          PWMA_inv<='1';
          PWMB_inv<='1';
          PWMC_inv<='1';
          cont:=0;
          aux<=Xd;
          aux2<=Xq;
     elsif clk'event and clk='1' then
          auxT02:=conv_integer(T0(20 downto 1));--T0/2
          auxT04:=conv_integer(T0(20 downto 2));--T0/4
          auxTn2:=conv_integer(Tn(20 downto 1));--Tn/2
          auxTn12:=conv_integer(Tn1(20 downto 1));--Tn1/2
     if (aux /= Xd) OR (aux2 /= Xq) then
          cont:=0;
     end if;
     if Eliminacion_Pares='0' then--Desactivada
     if conv_integer(T0)< 0 then
          PWMA<=switching_sequence(17);
          PWMB<=switching_sequence(16);
          PWMC<=switching_sequence(15);
          PWMA_inv<=not(switching_sequence(17));
          PWMB_inv<=not(switching_sequence(16));
          PWMC_inv<=not(switching_sequence(15));
     elsif sector="001" OR sector="011" OR sector="101" then
     if cont< auxT04 then
          PWMA<=switching_sequence(20);
          PWMB<=switching_sequence(19);
          PWMC<=switching_sequence(18);
          PWMA_inv<=not(switching_sequence(20));
          PWMB_inv<=not(switching_sequence(19));
          PWMC_inv<=not(switching_sequence(18));
     elsif cont< (auxT04+auxTn2) then
```

```
            PWMA<=switching_sequence(17);
            PWMB<=switching_sequence(16);
            PWMC<=switching_sequence(15);
            PWMA_inv<=not(switching_sequence(17));
            PWMB_inv<=not(switching_sequence(16));
            PWMC_inv<=not(switching_sequence(15));
      elsif cont< (auxT04+auxTn2+auxTn12)then
            PWMA<=switching_sequence(14);
            PWMB<=switching_sequence(13);
            PWMC<=switching_sequence(12);
            PWMA_inv<=not(switching_sequence(14));
            PWMB_inv<=not(switching_sequence(13));
            PWMC_inv<=not(switching_sequence(12));
      elsif cont< (auxT04+auxT02+auxTn2+auxTn12)then
            PWMA<=switching_sequence(11);
            PWMB<=switching_sequence(10);
            PWMC<=switching_sequence(9);
            PWMA_inv<=not(switching_sequence(11));
            PWMB_inv<=not(switching_sequence(10));
            PWMC_inv<=not(switching_sequence(9));
      elsif cont< (auxT04+auxT02+auxTn2+auxTn12+auxTn12)then
            PWMA<=switching_sequence(8);
            PWMB<=switching_sequence(7);
            PWMC<=switching_sequence(6);
            PWMA_inv<=not(switching_sequence(8));
            PWMB_inv<=not(switching_sequence(7));
            PWMC_inv<=not(switching_sequence(6));
      elsif
cont<(auxT04+auxT02+auxTn2+auxTn12+auxTn12+auxTn2)then
            PWMA<=switching_sequence(5);
            PWMB<=switching_sequence(4);
            PWMC<=switching_sequence(3);
            PWMA_inv<=not(switching_sequence(5));
            PWMB_inv<=not(switching_sequence(4));
            PWMC_inv<=not(switching_sequence(3));
      elsif cont<
(conv_integer(T0)+auxTn2+auxTn12+auxTn12+auxTn2)then
            PWMA<=switching_sequence(2);
            PWMB<=switching_sequence(1);
            PWMC<=switching_sequence(0);
            PWMA_inv<=not(switching_sequence(2));
            PWMB_inv<=not(switching_sequence(1));
            PWMC_inv<=not(switching_sequence(0));
      end if;
      else
      if cont< (auxT04)then
            PWMA<=switching_sequence(20);
            PWMB<=switching_sequence(19);
            PWMC<=switching_sequence(18);
            PWMA_inv<=not(switching_sequence(20));
            PWMB_inv<=not(switching_sequence(19));
            PWMC_inv<=not(switching_sequence(18));
      elsif cont< (auxT04+auxTn12)then
            PWMA<=switching_sequence(17);
            PWMB<=switching_sequence(16);
            PWMC<=switching_sequence(15);
```

```
            PWMA_inv<=not(switching_sequence(17));
            PWMB_inv<=not(switching_sequence(16));
            PWMC_inv<=not(switching_sequence(15));
      elsif cont< (auxT04+auxTn2+auxTn12)then
            PWMA<=switching_sequence(14);
            PWMB<=switching_sequence(13);
            PWMC<=switching_sequence(12);
            PWMA_inv<=not(switching_sequence(14));
            PWMB_inv<=not(switching_sequence(13));
            PWMC_inv<=not(switching_sequence(12));
      elsif cont< (auxT02+auxT04+auxTn2+auxTn12)then
            PWMA<=switching_sequence(11);
            PWMB<=switching_sequence(10);
            PWMC<=switching_sequence(9);
            PWMA_inv<=not(switching_sequence(11));
            PWMB_inv<=not(switching_sequence(10));
            PWMC_inv<=not(switching_sequence(9));
      elsif cont< (auxT02+auxT04+auxTn2+auxTn12+auxTn2)then
            PWMA<=switching_sequence(8);
            PWMB<=switching_sequence(7);
            PWMC<=switching_sequence(6);
            PWMA_inv<=not(switching_sequence(8));
            PWMB_inv<=not(switching_sequence(7));
            PWMC_inv<=not(switching_sequence(6));
      elsif cont<
(auxT02+auxT04+auxTn2+auxTn12+auxTn12+auxTn2)then
            PWMA<=switching_sequence(5);
            PWMB<=switching_sequence(4);
            PWMC<=switching_sequence(3);
            PWMA_inv<=not(switching_sequence(5));
            PWMB_inv<=not(switching_sequence(4));
            PWMC_inv<=not(switching_sequence(3));
      elsif
cont<(conv_integer(T0)+auxTn2+auxTn12+auxTn12+auxTn2)then
            PWMA<=switching_sequence(2);
            PWMB<=switching_sequence(1);
            PWMC<=switching_sequence(0);
            PWMA_inv<=not(switching_sequence(2));
            PWMB_inv<=not(switching_sequence(1));
            PWMC_inv<=not(switching_sequence(0));
end if;
else
      if armonic='0' then --A
      if conv_integer(T0)< 0 then
            PWMA<=switching_sequence_Ha(17);
            PWMB<=switching_sequence_Ha(16);
            PWMC<=switching_sequence_Ha(15);
            PWMA_inv<=not(switching_sequence_Ha(17));
            PWMB_inv<=not(switching_sequence_Ha(16));
            PWMC_inv<=not(switching_sequence_Ha(15));
      else
      if cont< auxT04 then
            PWMA<=switching_sequence_Ha(20);
            PWMB<=switching_sequence_Ha(19);
            PWMC<=switching_sequence_Ha(18);
            PWMA_inv<=not(switching_sequence_Ha(20));
```

```
            PWMB_inv<=not(switching_sequence_Ha(19));
            PWMC_inv<=not(switching_sequence_Ha(18));
      elsif cont< (auxT04+auxTn2) then
            PWMA<=switching_sequence_Ha(17);
            PWMB<=switching_sequence_Ha(16);
            PWMC<=switching_sequence_Ha(15);
            PWMA_inv<=not(switching_sequence_Ha(17));
            PWMB_inv<=not(switching_sequence_Ha(16));
            PWMC_inv<=not(switching_sequence_Ha(15));
      elsif cont< (auxT04+auxTn2+auxTn12)then
            PWMA<=switching_sequence_Ha(14);
            PWMB<=switching_sequence_Ha(13);
            PWMC<=switching_sequence_Ha(12);
            PWMA_inv<=not(switching_sequence_Ha(14));
            PWMB_inv<=not(switching_sequence_Ha(13));
            PWMC_inv<=not(switching_sequence_Ha(12));
      elsif cont< (auxT04+auxT02+auxTn2+auxTn12)then
            PWMA<=switching_sequence_Ha(11);
            PWMB<=switching_sequence_Ha(10);
            PWMC<=switching_sequence_Ha(9);
            PWMA_inv<=not(switching_sequence_Ha(11));
            PWMB_inv<=not(switching_sequence_Ha(10));
            PWMC_inv<=not(switching_sequence_Ha(9));
      elsif cont< (auxT04+auxT02+auxTn2+auxTn12+auxTn12)then
            PWMA<=switching_sequence_Ha(8);
            PWMB<=switching_sequence_Ha(7);
            PWMC<=switching_sequence_Ha(6);
            PWMA_inv<=not(switching_sequence_Ha(8));
            PWMB_inv<=not(switching_sequence_Ha(7));
            PWMC_inv<=not(switching_sequence_Ha(6));
      elsif
cont<(auxT04+auxT02+auxTn2+auxTn12+auxTn12+auxTn2)then
            PWMA<=switching_sequence_Ha(5);
            PWMB<=switching_sequence_Ha(4);
            PWMC<=switching_sequence_Ha(3);
            PWMA_inv<=not(switching_sequence_Ha(5));
            PWMB_inv<=not(switching_sequence_Ha(4));
            PWMC_inv<=not(switching_sequence_Ha(3));
      elsif cont<
(conv_integer(T0)+auxTn2+auxTn12+auxTn12+auxTn2)then
            PWMA<=switching_sequence_Ha(2);
            PWMB<=switching_sequence_Ha(1);
            PWMC<=switching_sequence_Ha(0);
            PWMA_inv<=not(switching_sequence_Ha(2));
            PWMB_inv<=not(switching_sequence_Ha(1));
            PWMC_inv<=not(switching_sequence_Ha(0));
      end if;
      end if;
else
      if conv_integer(T0)< 0 then
            PWMA<=switching_sequence_Hb(17);
            PWMB<=switching_sequence_Hb(16);
            PWMC<=switching_sequence_Hb(15);
            PWMA_inv<=not(switching_sequence_Hb(17));
            PWMB_inv<=not(switching_sequence_Hb(16));
            PWMC_inv<=not(switching_sequence_Hb(15));
```

```
     elsif sector="001" OR sector="011" OR sector="101" then
     if cont< auxT04 then
          PWMA<=switching_sequence_Hb(20);
          PWMB<=switching_sequence_Hb(19);
          PWMC<=switching_sequence_Hb(18);
          PWMA_inv<=not(switching_sequence_Hb(20));
          PWMB_inv<=not(switching_sequence_Hb(19));
          PWMC_inv<=not(switching_sequence_Hb(18));
     elsif cont< (auxT04+auxTn2) then
          PWMA<=switching_sequence_Hb(17);
          PWMB<=switching_sequence_Hb(16);
          PWMC<=switching_sequence_Hb(15);
          PWMA_inv<=not(switching_sequence_Hb(17));
          PWMB_inv<=not(switching_sequence_Hb(16));
          PWMC_inv<=not(switching_sequence_Hb(15));
     elsif cont< (auxT04+auxTn2+auxTn12)then
          PWMA<=switching_sequence_Hb(14);
          PWMB<=switching_sequence_Hb(13);
          PWMC<=switching_sequence_Hb(12);
          PWMA_inv<=not(switching_sequence_Hb(14));
          PWMB_inv<=not(switching_sequence_Hb(13));
          PWMC_inv<=not(switching_sequence_Hb(12));
     elsif cont< (auxT04+auxT02+auxTn2+auxTn12)then
          PWMA<=switching_sequence_Hb(11);
          PWMB<=switching_sequence_Hb(10);
          PWMC<=switching_sequence_Hb(9);
          PWMA_inv<=not(switching_sequence_Hb(11));
          PWMB_inv<=not(switching_sequence_Hb(10));
          PWMC_inv<=not(switching_sequence_Hb(9));
     elsif cont< (auxT04+auxT02+auxTn2+auxTn12+auxTn12)then
          PWMA<=switching_sequence_Hb(8);
          PWMB<=switching_sequence_Hb(7);
          PWMC<=switching_sequence_Hb(6);
          PWMA_inv<=not(switching_sequence_Hb(8));
          PWMB_inv<=not(switching_sequence_Hb(7));
          PWMC_inv<=not(switching_sequence_Hb(6));
     elsif
cont<(auxT04+auxT02+auxTn2+auxTn12+auxTn12+auxTn2)then
          PWMA<=switching_sequence_Hb(5);
          PWMB<=switching_sequence_Hb(4);
          PWMC<=switching_sequence_Hb(3);
          PWMA_inv<=not(switching_sequence_Hb(5));
          PWMB_inv<=not(switching_sequence_Hb(4));
          PWMC_inv<=not(switching_sequence_Hb(3));
     elsif cont<
(conv_integer(T0)+auxTn2+auxTn12+auxTn12+auxTn2)then
          PWMA<=switching_sequence_Hb(2);
          PWMB<=switching_sequence_Hb(1);
          PWMC<=switching_sequence_Hb(0);
          PWMA_inv<=not(switching_sequence_Hb(2));
          PWMB_inv<=not(switching_sequence_Hb(1));
          PWMC_inv<=not(switching_sequence_Hb(0));
     end if;
     end if;
end if;
end if;
```

```
aux<=Xd;
aux2<=Xq;
cont:=cont+1;
end if;
end process;
end Behavioral;
```

Código 10-10: PWM

11 BIBLIOGRAFÍA

[1] D. N. Sonawane, M. S. Sutaone, B.N. Choudhari and Abhijeet Badurkar "FPGA Implementation of Simplified SVPWM Algorithm for Three Phase Voltage Source Inverter", International Journal of Computer and Electrical Engineering, Vol2, No 6, December,2010.

[2] Reema KArmokar, Shubham Mungekar, Trupti Vaity, "FPGA Based space vector pulse width modulation technique implementation for three phase inverter", Department of Electronics, Fr. Conceicao Rodrigues College of Engineering, Bandra, Mumbai, India.

[3] P. Rodríguez Cortés "Modulación vectorial tridimensional de inversores en puente completo", https://upcommons.upc.edu/bitstream/handle/2117/93607/07Prc07de09.pdf.

[4] M.N. Qureshi, "Two-level Voltage Source Inverter", 1, December,2016.

[5] Ned Mohan, Tore M. Undeland and William P. Robbins, "Power Electronics. Third Edition" Willer, 2003.

[6] Edel Díaz Llerena, "Trabajo fin de Carrera: Modelado VHDL de estrategias de modulación para convertidores DC/AC para plataforma Semiteach-IGBT", Dept. Electrónica, Escuela Politécnica Superior, Universidad de Alcalá.

[7] Jesus Ureña Ureña, Miguela Ángel Sotelo Vázquez, Fco. Javier Rodríguez Sánchez, Rafael Bravo Navarro, Mariano Dominguez Herranz, Emilio J. Bueno Peña and Pedro A. Revenga de Toro "Electronica de Potencia", Servicio de Publicaciones UAH, 1999.

[8] Emilio José Bueno Peña, "Representación vectorial de variables trifásicas", Departamento de Electrónica, Universidad de Alcalá.

[9] Salvador Seguí Chilet, Francisco J.Gimeno Sales, Rafael Masot Peris, Salvador Orts Grau, "Control Vectorial de inversores trifásicos/1" Dept. de ingeniería Electrónica Universidad Politécnica de Valencia.

[10] Manuel J. Bellido, "Apuntes de Prácticas sobre Diseño y Simulación con VHDL empleando el entorno de XILINX: ISE", Diseño de Computadores, Octubre de 2012.

[11] Mauricio A.Tonelli, "Modulación Vectorial de Inversores de Potencia", Facultad de ingeniería, dept Electrotecnia, Universidad Nacional de la Plata.

[12] "Guía Docente Electrónica de Potencia", Grado en ingeniería Electrónica y Automática Industrial (G60), Universidad de Alcalá, Curso Académico 2015/2016.

[13] "Methodologies and HDL for the design of digital systems", Bachelor's Degree in Electronics and Industrial Automation Engineering.

[14] "Laboratorio. ISE&ModelSim Tutorial", Diseñi Electrónico, Master Universitario en Sistema Electrónicos Avanzados, Sistemas Inteligentes.

[15] Cristina de Evan de Marcos, "Configuración y prueba de convertidores DC/DC para prácticas de laboratorio", Trabajo de fin de Grado, Universidad de Alcalá, septiembre 2015.

[16] Josua Cabeza Picazo "Interfaz gráfica para simulación de convertidores DC/AC para entorno docente" Dept. Electrónica, Escuela Politécnica Superior, Universidad de Alcalá, 2017.

[17] Digilent, Inc "Digilent Nexys2 Board Reference Manual", Especificaciones del Producto,2011. Disponible en: http://www.digilent.com.

[18] Digilent, Inc "Digilent PmodDA™ Digital to Analog module Converter Board Reference Manual", 2006 Disponible en: http//www.digilent.com.

[19] Xilinx, Inc. "Using Digital Clock Managers(DCMs) in spartan-3-FPGAs", 2006. Disponible en http//www.xilinx.com.

[20] G.Pérez, A.Garrigós, J.M.Blanes, R.Gutiérrez. "Plataforma basada en MATLAB, FPGA Xilinx y Semiteach-IGBT para docencia en electrónica de potencia" , Julio 2013.

[21] H. Van der Broek, H. Skudelny, and G. Stanke, "Analysis and realization of a pulse width modulator bassed on voltage space vectors," in Proc. IEEE Ind. Applicant. Conf, 1986, pp.c244-251.

Universidad de Alcalá
Escuela Politécnica Superior

ESCUELA POLITECNICA
SUPERIOR