Proyecto Fin de Carrera
Ingeniería Industrial

SIMULACIONES TEORICAS EN MODELOS REALES DE INVERSORES TRINIVEL TRIFASICOS

Autor: Daniel Armenta Camacho
Tutor: M. Ángeles Martín Prats

Departamento de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Proyecto Fin de Carrera
Ingeniería Industrial

SIMULACIONES TEORICAS EN MODELOS REALES DE INVERSORES TRINIVEL TRIFASICOS

Autor:
Daniel Armenta Camacho

Tutor:
M. Ángeles Martín Prats
Profesor titular

Departamento de Ingeniería Electrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Proyecto Fin de Carrera: SIMULACIONES TEORICAS EN MODELOS REALES DE INVERSORES TRINIVEL TRIFASICOS

Autor: Daniel Armenta Camacho

Tutor: M. Ángeles Martín Prats

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

A mis padres

A mi mujer

Y a mis hijos

Por ser incondicionales

Agradecimientos

Este proyecto no hubiera sido posible sin un equipo con el que trabajar codo con codo para llegar a una meta tan ambiciosa. Tengo que agradecer al equipo formado por Ramón Portillo, Fernando Cortés y Francisco José Benítez el interés y la entrega en la consecución de un proyecto conjunto. Y por encima de todo, agradezco a nuestra estrella polar por guiarnos, dirigirnos y animarnos en el dia a dia de nuestra investigación: gracias M. Ángeles Prats por contar con nosotros, por no dejar que nos desanimásemos en cada problema y adudarnos a superarlo y seguir andando. Y gracias por la paciencia de varios años en la espera.

Y gracias a todos mis profesores y amigos del departamento que me abrieron las puertas del apasionante mundo de la electrónica, que finalmente me ha llevado a poder realizar este Proyecto.

Especial agradecimiento a mi familia que durante mucho tiempo me ha animado a retomar el proyecto, que ha aguantado mi ausencia incluso en momentos complicados y que ha estado a mi lado independientemente de mi estado de animo. En particular, gracia a mi mujer (mi vida) por enseñarme y demostrarme en persona que nunca es tarde para retomar la universidad.

Y gracias a mis padres por no perder la esperanza y seguir convencidos de que este día llegaría.

Resumen

Convertir un proyecto de investigación en un dispositivo real sobre el que aplicar unas hipótesis de trabajo conlleva generalmente bastante inversión de tiempo y dinero en pruebas, ajustes de dimensionamiento de dispositivos y adaptación a las condiciones reales. Contar con un buen sistema de simulaciones que aplique los algoritmos matemáticos teóricos sobre modelos más reales de los dispositivos y poder así predecir la respuesta del sistema basándose en datos más cercanos a la realidad, se convierte en una pieza clave para reducir al máximo el dinero y tiempo necesarios para pruebas en laboratorio antes de pasar al prototipo definitivo.

Introducción

Este proyecto pretende complementar el trabajo realizado en otros 3 proyectos fin de carrera por Ramón Portillo Guisado "*Modelado y Simulación de Inversores Multinivel"* [1], Fernando Cortés Ponce "*Desarrollo de Técnicas de Control y su Implementación en DSP para Inversores Multinivel*" [2] y Francisco José Benítez Camacho "*Diseño de un prototipo de inversor trinivel e implementación del hardware de control"* [3]; 4 proyectos que colaboraron en la tesis doctoral de MªÁngeles Martín Prats: *Nuevas técnicas de modulación vectorial para conversores electrónicos de potencia multinivel* [4].

La finalidad de dicha tesis doctoral fue el estudio e implementación de algoritmos de modulación y técnicas de control de convertidores multinivel en aplicaciones de media y gran potencia. Fue acompañada, además, del montaje y puesta en marcha del prototipo de un convertidor DC-AC trinivel de potencia para la aplicación experimental de los algoritmos de control de dicha tesis y posteriores.

El problema principal en el diseño de la bancada fñisica es el dimensionamiento de los distintos elementos de medida, control y operación. Si los valores nominales de tensión, intensidad, (...), de alguno de los elementos de medida queda por debajo de los valores reales (reales, no los calculados previamente) se generaría un error de medida, mal funcionamiento o incluso rotura del elemento. Lo contrario, la elección del elemento con unos valores nominales muy por encima de los valores reales del sistema, llevaría a un desembolso económico mayor que el necesario (debe recordar el lector que hablamos de electrónica de potencia), así como una posible pérdida de precisión. En el ejemplo más básico, podemos entender los problemas que tendría un medidor de intensidad mal dimensionado.

En cuanto a la aplicación del algoritmo de control teórico sobre el montaje real, existen distintas incertidumbres sobre la seguridad e integridad de los componentes electrónicos de potencia. Uno de los principales peligros de aplicar directamente el control teórico sobre la bancada radica en los tiempos de respuesta (encendido y apagado) de los interruptores de potencia.

Aunque los valores nominales de tiempos característicos de los componentes electrónicos vienen facilitados por los catálogos, el hecho de montar varios elementos eléctricamente conectados hace que el cálculo de cómo responderán ante el funcionamiento del algoritmo no sea trivial, ya que deja de ser un montaje básico como el utilizado para dichos catálogos. Debido a que estos algoritmos suelen crearse para altas frecuencias de conmutación, es importante fijar estos límites físicos y ajustar el algoritmo en consecuencia, para evitar sobretensiones y cortocircuitos en la bancada.

Estas dos razones (dimensionamiento y aplicación del algoritmo ideal al sistema real) son las que desembocan en la necesidad de simular el comportamiento real de los dispositivos de potencia trabajando según el esquema del montaje para conocer todas estas limitaciones físicas. Además, como consecuencia del mejor conocimiento de la respuesta real de los componentes, podrá concluirse qué acciones sobre el algoritmo tendrán mayor o menor efecto en la bancada una vez montada. Así, se genera una herramienta de estudio previo muy precisa que reduce los riesgos que conllevaría experimentar directamente sobre los dispositivos de potencia, los cuales no suelen avisar con una ventana animada ante un error, sino que directamente se deterioran, ya sea visiblemente (corto plazo) o interiormente (largo plazo).

Índice

Agradecimientos **i**

Resumen **iii**

Introducción **iv**

Índice **v**

Índice de Figuras **vi**

1 Preliminares **9**

1.1 Breve introducción a los inversores (conversores DC-AC) *9*

1.1.1 Inversor dos niveles: 9

1.1.2 Inversor trifásico de 2 niveles 10

1.1.3 Inversores PWM 10

1.1.4 Inversor Trinivel y multinivel 11

1.1.5 Inversor Trinivel con carga R-L 13

1.2 Punto de partida: simulaciones ideales *14*

1.3 Diferencias entre el sistema real y el ideal *17*

1.3.1 Tiempos muertos 18

1.3.2 Rampa de subida 18

2 Conversión de Matlab a PSPICE **20**

2.1 Pasos previos a la aplicación del Conversor M-P. *21*

2.2 Conversor M-P: REDUCE *21*

2.3 Conversor M-P: AJUSTA *24*

2.4 Conversor M-P: DESPLAZA *25*

2.5 Conversor M-P: ESCALA *27*

2.6 Conversor M-P: GRABA *27*

3 Simulaciones en PSPICE **29**

3.1 Estabilidad de PSPICE en electrónica de Potencia *29*

3.2 Parámetros de simulación. *29*

3.3 Pruebas de validez del modelo del IGBT *30*

3.3.1 Simulación de un Inversor monofásico PWM 30

3.3.2 Simulación de un Inversor trifásico PWM 31

3.3.3 Simulación de un Inversor trifásico trinivel ideal sin carga 32

3.3.4 Simulación de un Inversor trifásico trinivel ideal con carga 34

3.3.5 Simulación TEST de un IGBT 36

3.3.6 Simulación Inversor Trinivel Trifásico con IGBT SKM300 37

4 SLPS: Integracion completa **39**

Conclusiones. **44**

ANEXO A: Código MATLAB completo. **45**

Referencias **59**

Índice de Conceptos **60**

Glosario **61**

Bibliografía **62**

Índice de Figuras

Figura 1-1: Inversores monofásicos de dos niveles en medio puente y en puente completo 9
Figura 1-2: Inversor trifásico de dos niveles de puente completo 10
Figura 1-3: Tensiones de salida con PWM em Circuito monofásico 10
Figura 1-4: Tensiones de salida con PWM en Circuito trifásico 11
Figura 1-5: Inversor trinivel NPC 11
Figura 1-6: Tensiones de salida con PWM en Inversor trinivel NPC 12
Figura 1-7: Inversor trinivel ideal con carga R-L 13
Figura 1-8: Algoritmo aplicado sobre inversor trinivel ideal con carga R-L 14
Figura 1-9: Control de la simulación real a partir de datos sistema ideal 15
Figura 1-10: Cortocircuito producido por el cierre de interruptores de una rama 17
Figura 1-11: Tarjeta de control alimentando a dispositivo disparador de IGBT 19
Figura 2-1: Eliminación de puntos redundantes en la señal de conmutación 22
Figura 2-2: Correspondencia de las tablas originales de Matlab con los IGBT 23
Figura 2-3: Conversión de escalones en rampas 24
Figura 2-4: Nomenclatura de los distintos IGBT 25
Figura 2-5: Aplicación de tiempos muertos 26
Figura 3-1: Esquema del Inversor monofásico PWM 29
Figura 3-2: Esquema del Inversor monofásico PWM 30
Figura 3-3: Intensidad de Rama del Inversor monofásico PWM 30
Figura 3-4: Esquema del Inversor trifásico PWM 31
Figura 3-5: Intensidad del Inversor trifásico PWM 31
Figura 3-6: Esquema del Inversor trifásico trinivel sin carga 32
Figura 3-7: Tensión de Fase del Inversor trifásico trinivel sin carga 32
Figura 3-8: Detalle de la tensión de Fase del Inversor trifásico trinivel sin carga 33
Figura 3-9: Tensión de Línea del Inversor trifásico trinivel sin carga 33
Figura 3-10: Detalle de la tensión de Línea del Inversor trifásico trinivel sin carga 33
Figura 3-11: Esquema del Inversor trifásico trinivel con carga 34
Figura 3-12: Tensión de Fase del Inversor trifásico trinivel con carga 34
Figura 3-13: Tensión de Línea del Inversor trifásico trinivel con carga 35
Figura 3-14: Intensidades de Rama del Inversor trifásico trinivel con carga 35
Figura 3-15: Esquema del Inversor de 2 niveles con IGBT SKM300 36
Figura 3-16: Tensión – Intensidad del Inversor de 2 niveles con IGBT SKM300 36
Figura 3-17: Esquema del Inversor Trinivel Trifásico con IGBT SKM300 37
Figura 3-18: Intensidad en régimen estacionario del Inversor Trinivel Trifásico con IGBT SKM300 37

Figura 3-19: Evolución de Intensidad de Rama del Inversor Trinivel Trifásico con IGBT SKM300 37
Figura 3-20: Intensidades de Rama del Inversor Trinivel Trifásico con IGBT SKM300 38
Figura 3-21: Transitorio de estabilización de curva de intensidad 38
Figura 3-22: Transitorio de estabilización de la tensión 38
Figura 4-1: Bucle cerrado en simulación ideal, con salida de datos para PSpice 39
Figura 4-2: Bucle cerrado en simulación ideal, con salida de datos para PSpice 40
Figura 4-3: Configuración de datos del bloque SLPS. 41
Figura 4-4: Parámetros de simulación de PSpice dentro del bloque SLPS. 42
Figura 4-5: Coordinación en el intercambio de datos entre Simulink y PSpice. 42

1 PRELIMINARES

1.1 Breve introducción a los inversores (conversores DC-AC)

Un inversor es un sistema (circuito electrónico) utilizado para convertir corriente continua en corriente alterna. La función de un inversor es cambiar un voltaje de entrada de corriente directa a un voltaje simétrico de salida de corriente alterna, con la magnitud y frecuencia deseada por el usuario o el diseñador. Los inversores son utilizados en una gran variedad de aplicaciones, desde pequeñas fuentes de alimentación de dispositivos electrónicos, hasta aplicaciones industriales que manejan grandes potencias. Los inversores también son utilizados para convertir la corriente continua generada por los paneles solares foto-voltaicos, acumuladores o baterías, en corriente alterna y de esta manera poder ser inyectados en la red eléctrica o usados en instalaciones eléctricas aisladas.

No es la intención de este documento hacer un repaso exhaustivo de todos los tipos de inversores, pero se resumirán los casos más representativos para orientar el resto de las explicaciones.

1.1.1 Inversor dos niveles:

El convertidor DC-AC de dos niveles es el montaje más sencillo, que pretende hacer el acercamiento más básico a una señal alterna. La tensión de salida es totalmente cuadrada.

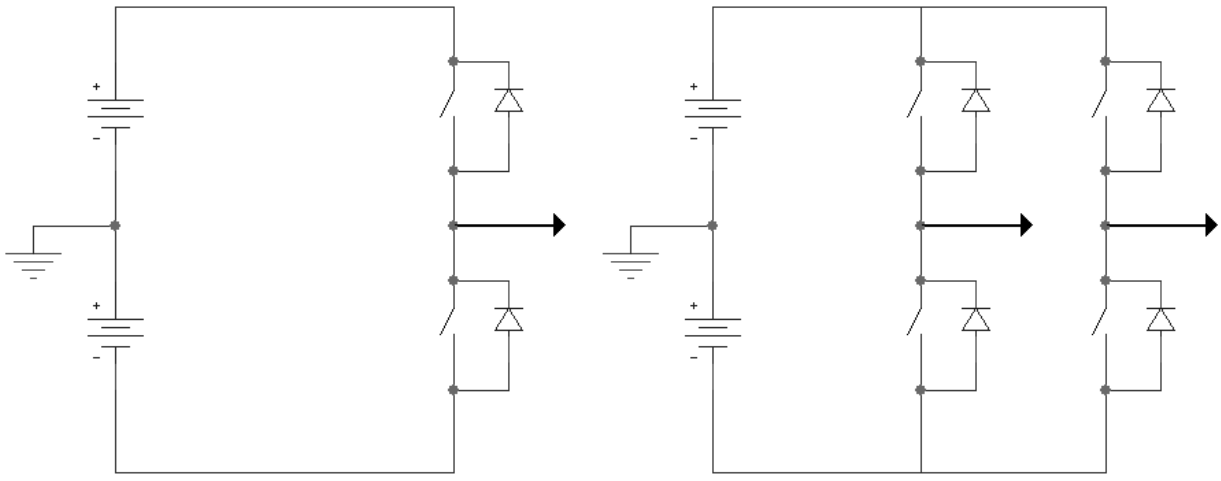

Figura 1-1: Inversores monofásicos de dos niveles en medio puente y en puente completo

1.1.2 Inversor trifásico de 2 niveles

Para conseguir un sistema trifásico, el montaje se basa en la Figura 1-2, donde las tensiones de cada rama (fase) son sólo dependientes de la tensión continua y del estado de los interruptores de la propia rama.

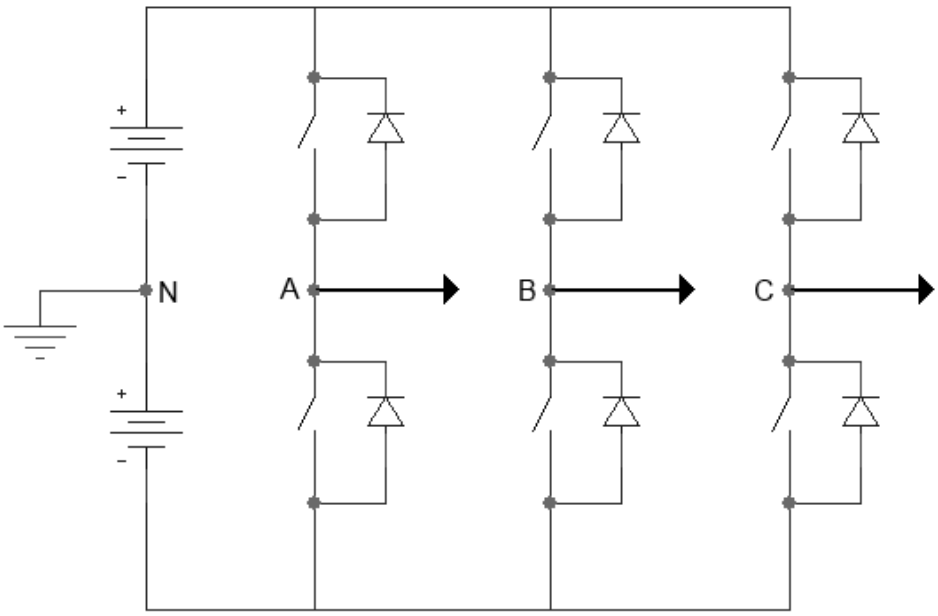

Figura 1-2: Inversor trifásico de dos niveles de puente completo

1.1.3 Inversores PWM

Las estrategias de conmutación de los interruptores marcarán el comportamiento del inversor. Una buena solución es PWM (Pulse-Width Modulation) o modulación por ancho de pulsos. La forma más elemental de controlar los disparos (conmutaciones) de los interruptores, y con ello la tensión de salida, se basa en comparar una señal portadora triangular de frecuencia fija y compararla con una referencia senoidal para decidir el instante y duración de los estados *cerrado* y *abierto* del interruptor (Figura 1-3).

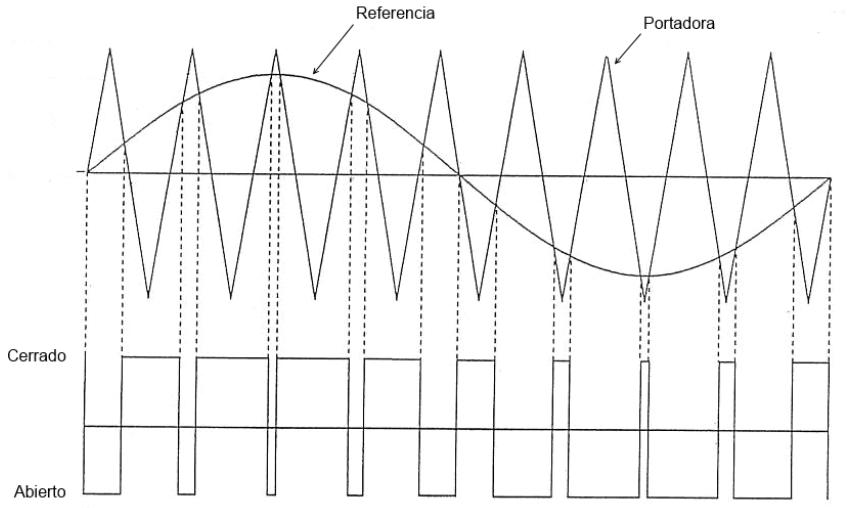

Figura 1-3: Tensiones de salida con PWM em Circuito monofásico

Si hacemos esto para cada fase, con señales senoidales convenientemente desfasadas, obtendremos los momentos de conmutación de cada interruptor y las tensiones de salida de Fase y de línea de la Figura 1-4:

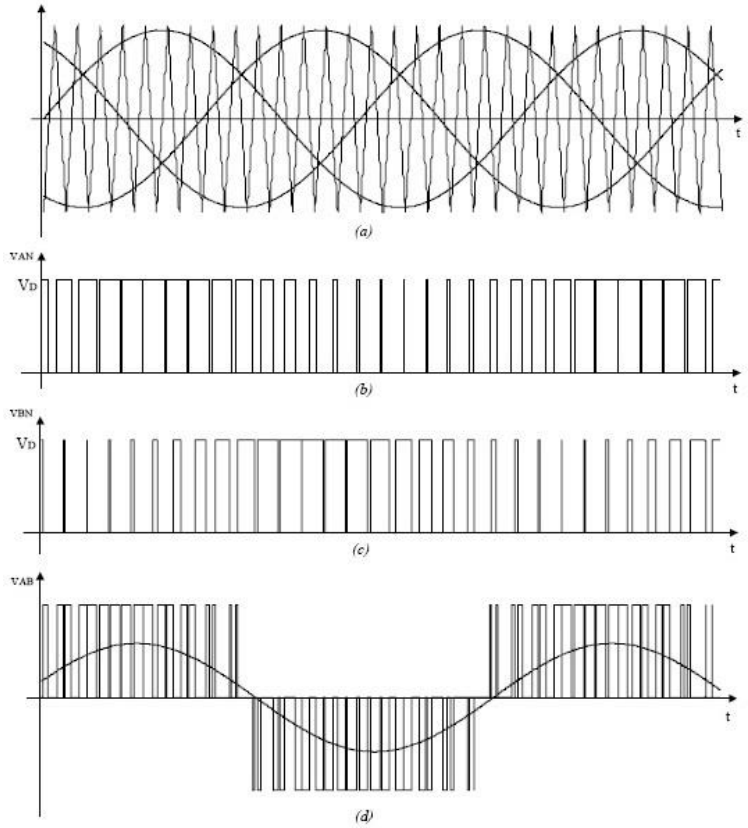

Figura 1-4: Tensiones de salida con PWM en Circuito trifásico

1.1.4 Inversor Trinivel y multinivel

Si queremos hilar más fino en la aproximación a la tensión senoidal, podemos pasar a un montaje trinivel, para conseguir tensiones intermedias de salida.

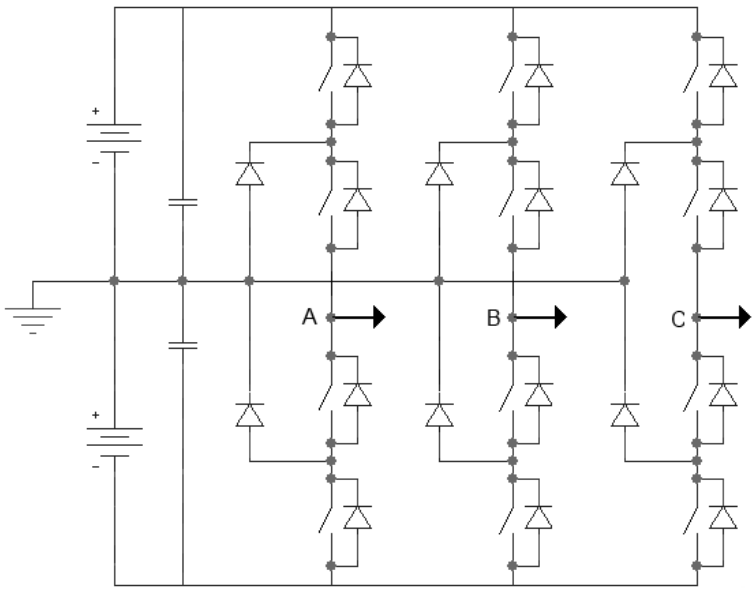

Figura 1-5: Inversor trinivel NPC

El Inversor trinivel de la Figura 1-5 es del tipo NPC (Neutral Point Clamped – Punto Neutro Fijo), por el bus común que une los puntos neutros de las 3 fases. Con este montaje se pueden conseguir tensiones de salida como la ilustrada en la Figura 1-6

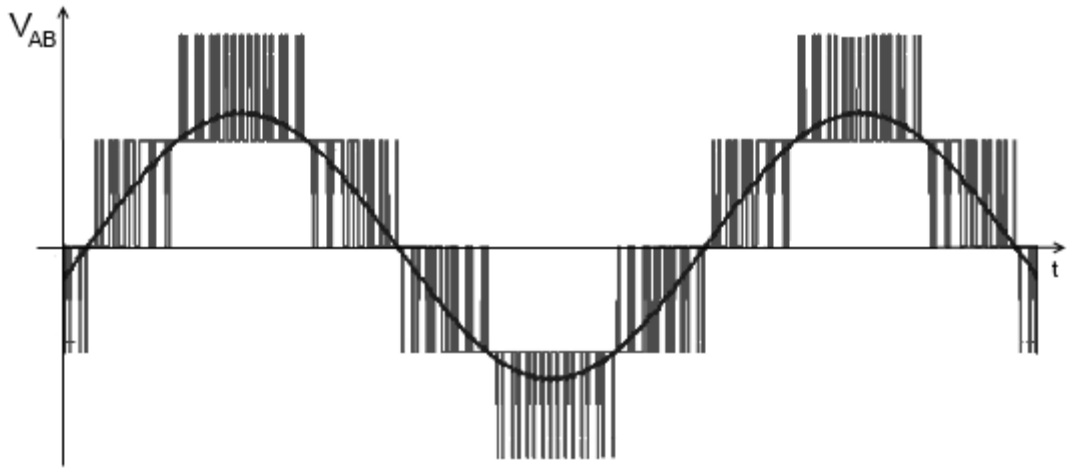

Figura 1-6: Tensiones de salida con PWM en Inversor trinivel NPC

Igualmente podemos seguir extrapolando los niveles del Inversor para conseguir mejores ajustes de la tensión de salida a la señal senoidal, pero debido al coste de componentes se convierte en un concepto más bien teórico puesto que las ventajas en un circuito real no son tantas como la complejidad y e coste de llevarlo a cabo.

La Tesis Doctoral de M. Ángeles Martín Prats, *Nuevas técnicas de modulación vectorial para conversores electrónicos de potencia multinivel* [4] abordan los inversores multinivel con modulaciones por vectores de espacio, generalmente utilizados para mejorar la distorsión armónica de la salida, entre otros objetivos.

Como veremos más adelante, las simulaciones que se desarrollarán en las sucesivas páginas necesitan adaptarse a la topología del circuito, pero son totalmente independientes del algoritmo de control utilizado. Eso sí, de estas simulaciones se sacarán importantes datos para ajustar y depurar dichos algoritmos teóricos para adaptarlos a la realidad de una bancada de componentes electrónicos de potencia.

1.1.5 Inversor Trinivel con carga R-L

Lógicamente, el sistema estudiado debe actuar sobre alguna carga. Se usará una típica carga trifásica R-L, a la que posteriormente daremos valores, e iremos dimensionando todo el circuito de potencia. Este esquema es válido tanto para el circuito ideal como para el real. Lo único que varía son los conmutadores, que en el caso teórico (Figura 1-7) son simples interruptores ideales y en el caso del circuito de potencia a simular serán interruptores electrónicos controlados del tipo IGBT.

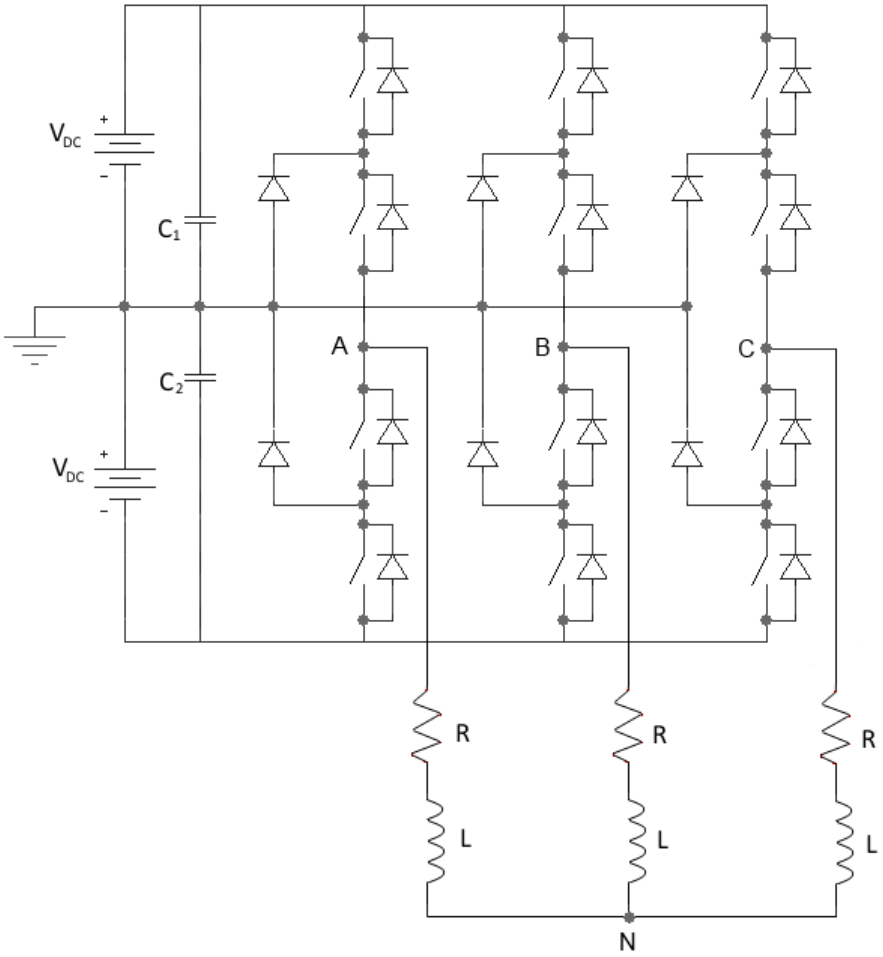

Figura 1-7: Inversor trinivel ideal con carga R-L

En este montaje, es relativamente sencillo conseguir un modelo de simulación válido de cada uno de los dispositivos. El caso más delicado es precisamente en interruptor. Cuanto más se asemeje el modelo del interruptor real al dispositivo que realmente se utilice, más cerca estaremos de unos resultados realmente comparables con el funcionamiento real, que es al fin y al cabo el objetivo final. Esto es especialmente crítico teniendo en cuenta que económicamente no tienen sentido sobredimensionar el sistema para asegurarse el buen funcionamiento y que intentaremos trabajar con unos valores cercanos a los límites de los dispositivos (frecuencia, intensidad y potencia). O lo que es más cierto, intentaremos dimensionar los dispositivos para ajustarse a los valores de trabajo que necesitamos para usarlos cerca de sus limitaciones físicas.

1.2 Punto de partida: simulaciones ideales

Precisamente, la finalidad del presente proyecto es convertir una serie de datos ideales y aplicarlos a una situación más real. Estos datos de partida se extraen de las simulaciones teóricas realizadas por Ramón Portillo Guisado [1] y MªÁngeles Prats [4] al investigar en el desarrollo de los distintos algoritmos de control. Los datos necesarios son exclusivamente los momentos exactos de conmutación de cada interruptor, independientemente de las tensiones e intensidades de salida que se generen. La primera hipótesis con la que trabajaremos es que las simulaciones del sistema real pueden llevarse a cabo independientemente de si el algoritmo original era en bucle abierto o en bucle cerrado, esto es, independientemente si en el control del sistema ideal se ha tenido en cuenta el estado de la salida o no.

Esto lleva a poder simular el sistema real más fácilmente (control en bucle abierto) y llegar a resultados que dependan sólo de las actuaciones sobre los interruptores, obviando las dependencias del sistema físico en sí. Así, se reducen los parámetros, las simulaciones son más rápidas y se hace más sencillo depurar el algoritmo original.

Los algoritmos de control sobre los sistemas ideales creados por Ramón Portillo Guisado [1] y MªÁngeles Prats [4] se desarrollaron en MATLAB (herramienta de software matemático de MathWorks). Podríamos simplificar al máximo el esquema de trabajo de estos algoritmos ideales en MATLAB trabajando sobre el trinivel de la siguiente manera:

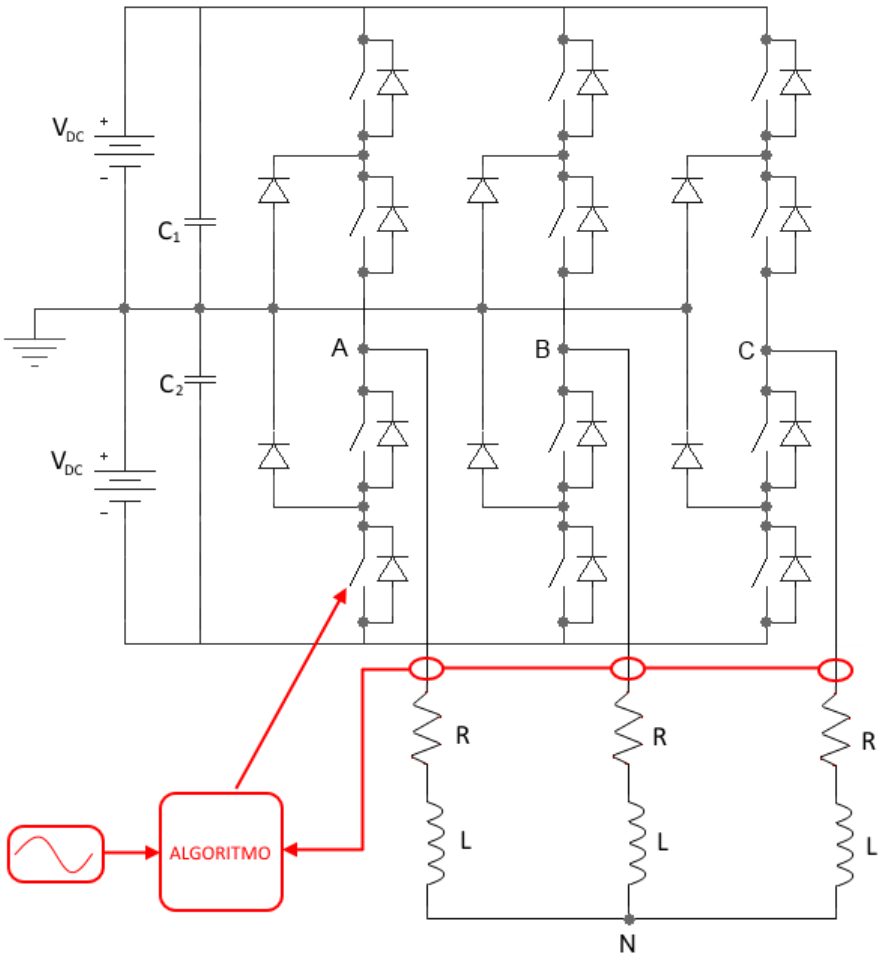

Figura 1-8: Algoritmo aplicado sobre inversor trinivel ideal con carga R-L

En el caso más complejo, el algoritmo de control trabaja a partir de una señal de referencia y con ciertos valores de tensión e intensidad a la salida. En estas simulaciones, ni el algoritmo ni las salidas están afectados por desviaciones de tensión e intensidad de los interruptores puesto que son interruptores ideales. Así, el montaje completo es ideal.

La simulación suele hacerse para una duración determinada, Esta duración necesaria dependerá de, por ejemplo, si se parte de valores iniciales distintos del régimen estacionario a alcanzar y queremos estudiar ese transitorio, o si estamos buscando estudiar la respuesta del sistema a cambios de un régimen permanente a otro, cambios en la carga, armónicos, etc...

Para poder sacar valores y comportamientos físicos reales del sistema dirigido por estos algoritmos se elige simular sobre OrCAD Capture CIS y PSPICE, de CADENCE. Dado que el algoritmo se está estudiando paralelamente en MATLAB y se sigue desarrollando continuamente, se decide intentar conectar de alguna manera abierta los dos sistemas (MATLAB y PSPICE) para poder simular en cualquier momento la respuesta real del trinivel cuando se llevan a cabo cambios en el algoritmo del sistema ideal.

La solución elegida para abordar esta situación se basa en coger los disparos (señales de control para los interruptores) que se generan en una simulación ideal (MATLAB) para un determinado periodo de tiempo e insertarlos en un montaje equivalente en PSPICE para conocer la respuesta (Intensidad) y poder compararla con la respuesta del sistema ideal. El esquema de trabajo se puede resumir de la siguiente manera:

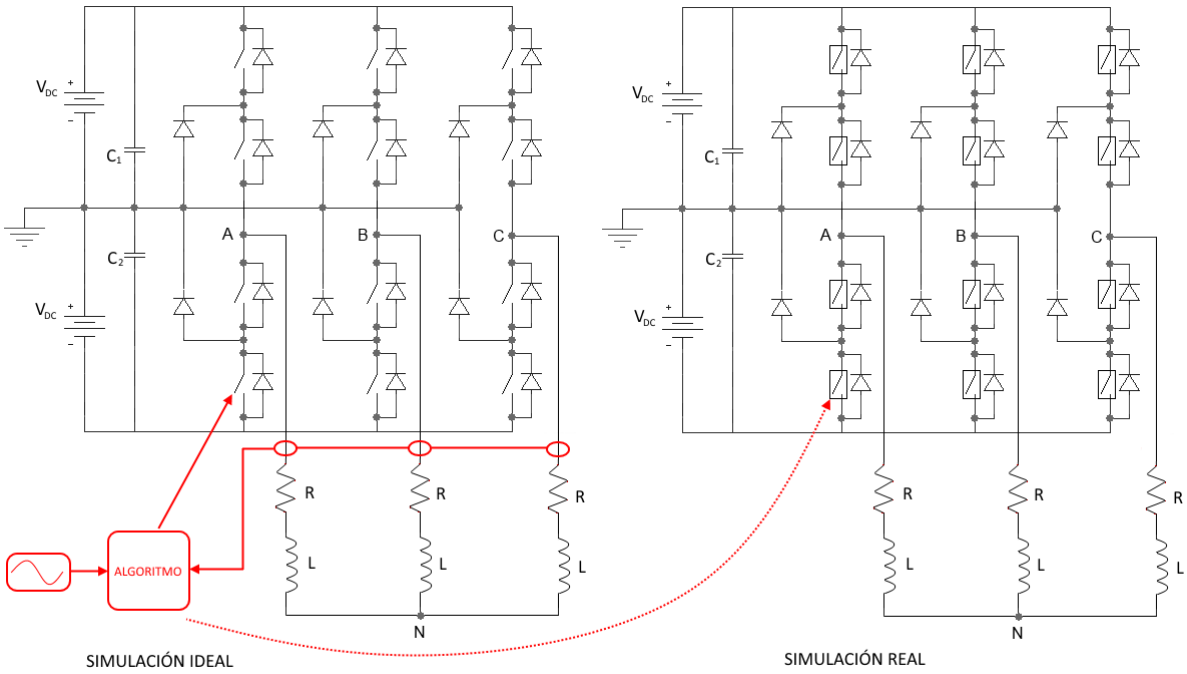

Figura 1-9: Control de la simulación real a partir de datos sistema ideal

La primera diferencia a destacar entre la simulación ideal y la real es que en el primer caso tenemos un bucle cerrado de control y en el segundo podemos decir que estamos aplicando los disparos al sistema real sobre un bucle abierto. En la simulación del sistema ideal cada decisión en cada instante de tiempo se toma teniendo en cuenta el estado del sistema, sus salidas en los instantes inmediatamente anteriores y una referencia. En la simulación del sistema real las decisiones no dependen de la salida del sistema real simulado; No hay dependencia de una referencia inicial ni del estado de las tensiones e intensidades de la salida, sino que ya están predefinidas por la simulación ya realizada anteriormente en el sistema ideal.

Esto significa que estamos formulando la hipótesis principal en la que se basa el proyecto completo. No se trata de suponer que la salida del sistema no va a depender del sistema en sí, lo cual sería totalmente erróneo. Se trata de suponer que las desviaciones de los datos al aplicar este bucle abierto en las simulaciones siempre serán lo suficientemente pequeñas como para dar por válidos los resultados, esto es, que la salida del sistema real simulado no varía apenas de la del sistema ideal anterior. De hecho, en la realidad sobre la bancada se seguirá aplicando un control en bucle cerrado que si respondería ante estas mínimas variaciones. De todas maneras, uno de los objetivos es conocer los tiempos muertos, consumos y ajustes del sistema, no la estabilidad del sistema en si.

La segunda diferencia entre los montajes de ambas simulaciones se basa en la utilización por primera vez del modelo del IGBT real que se quiere utilizar para montar físicamente el inversor trinivel. De esta manera podremos simular su comportamiento para sacar tanto los valores locales del IGBT (caídas de tensión, pérdidas de potencia/consumos, retrasos, ...), y adelantaremos el comportamiento global de la bancada real, pudiendo además dimensionar el resto de dispositivos necesarios para el control del sistema y la medida de los estados.

Cierto es que cuanto más cerca de la inestabilidad estemos en la simulación de Matlab, más posibilidades habrá que la simulación de PSPICE se haga inestable. De hecho, se comprueba que la inestabilidad de PSPICE es alta para este tipo de montajes de electrónica de potencia a frecuencias relativamente altas. Largas horas de estudio y de pruebas han llevado a delimitar los parámetros de simulación de PSPICE para los que la simulación tiene más posibilidades de ser estable, como veremos más adelante.

Como hemos comentado anteriormente, esto nos dará la oportunidad de corregir errores de funcionamiento que no se dan en el modelo ideal y dimensionar todos los elementos secundarios del prototipo. Con todo esto pretendemos ajustar los costes tanto en las compras iniciales como en las reparaciones que puedan evitarse, acortar el tiempo de puesta en marcha y lograr un funcionamiento estable y válido de la bancada lo antes posible.

Las simulaciones serán validas para depurar cualquier algoritmo nuevo o cualquier modificación de algoritmo existente. Basta con seguir estudiando y modificando el algoritmo en MATLAB y posteriormente tomar esas órdenes de control de los interruptores e introducirlos en la simulación en PSPICE para estudiar el comportamiento. Simulación real que no hay que adaptar al algoritmo porque sólo depende de los valores de estado (Figura 1-9) que se han generado a lo largo del periodo de simulación ideal. Con ello tendremos en marcha un sistema de pruebas en el que reducimos al mínimo los riesgos de accidentes por experimentación y podremos estudiar el sistema completo sin tener que recurrir físicamente a la bancada.

1.3 Diferencias entre el sistema real y el ideal

En los primeros pasos dados por Ramón Portillo [1] y MªÁngeles Prats [4] para crear y estudiar el algoritmo de control, el sistema es 100% ideal, sobre todo en lo referente a la velocidad de respuesta del sistema. Se necesita que el algoritmo funcione en un plano puramente teórico. Posteriormente se irán dando pasos para asemejar el sistema poco a poco a la respuesta real de los dispositivos electrónicos que se utilizarán. El primer ejemplo, y el más importante, es que un interruptor ideal responde sin ningún tipo de retrasos y sólo tiene 2 valores: PASA ó NO PASA. Pero los dispositivos electrónicos tienen su tiempo de respuesta y además no suelen ser elementos lineales.

Esta es la primera prueba a la que se enfrenta un algoritmo ideal en su camino poder convertirse en un algoritmo de control de un sistema físico real. Sin salir de las simulaciones ideales, se pueden empezar a introducir estos tiempos teóricos de respuesta de los dispositivos individuales para ver si el algoritmo ideal es capaz de conseguir los mismos resultados que con el sistema de interruptores 100% ideales. Una de las pruebas que tuvo que superar el equipo de trabajo del sistema ideal fue el hecho de poder reproducir de nuevo los resultados, una vez introducidos estos tiempos de respuesta de los dispositivos reales. Se estudiaron formas de reducir los armónicos, de mejorar el factor de potencia, de equilibrar las tensiones en el punto neutro (equilibrado de los condensadores), y de optimizar la secuencia de conmutación. A cada uno de esos problemas se le añadió una variable más cuando se introdujo el concepto de los tiempos de respuesta de los interruptores reales.

Pero no es sólo esa la importancia básica de introducir estos tiempos de respuesta en la simulación ideal. Dado la topología de montaje del circuito trinivel con el que se trabaja, hay una serie de estados de conmutación prohibidos. Por ejemplo, tener cerrados todos los interruptores de una rama significa estar cortocircuitando la fuente de tensión.

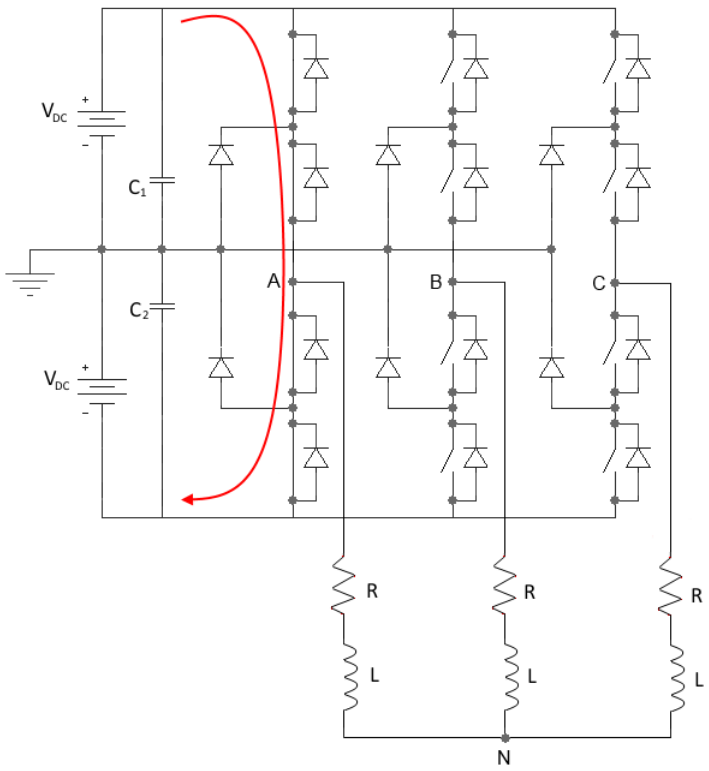

Figura 1-10: Cortocircuito producido por el cierre de interruptores de una rama

Este detalle, que en un principio puede parecer trivial es muy importante en este tipo de circuitos. Por mucho que se controle que los interruptores nunca reciban señales que correspondan con estados prohibidos, el tiempo de respuesta de los interruptores reales lleva con demasiada frecuencia a caer en cortocircuitos. No podemos suponer que un interruptor ha conseguido cortar la corriente instantáneamente y que tenemos la libertad de abrir uno contiguo sin riesgo a cortocircuito. Debido a este retraso físico que experimentan los interruptores reales, podemos provocar el problema incluso evitando estos estados prohibidos, simplemente por culpa de los transitorios reales.

1.3.1 Tiempos muertos

A estos espacios de tiempo que debemos cederle a los dispositivos electrónicos para llevar a cabo la conmutación de manera segura les llamamos tiempos muertos. Son instantes en los que no podemos cambiar el estado de otros interruptores hasta asegurarnos totalmente de que los dispositivos han conseguido abandonar el estado anterior.

Aún más, tampoco es 100% fiable usar directamente los valores de tiempos de conmutación facilitados por el fabricante para convertirlos directamente en tiempos muertos. Estos tiempos están medidos y comprobados en bancadas monofásicas simples, pensadas simplemente para sacar las curvas de funcionamiento del dispositivo, sin que otros dispositivos le afecten. Pero la realidad es que en cualquier circuito unos dispositivos se ven afectados por los circundantes y viceversa. Es importante acotar los tiempos muertos necesarios por cada dispositivo interruptor en la topología real a utilizar. De aquí que la simulación sea capaz de representar lo más fielmente posible el comportamiento de cada uno de los dispositivos trabajando en conjunto.

Debemos recordar que estamos intentado llevar a cabo un control a altas frecuencias relativas, comparado con las frecuencias a las que suele hacerse trabajar a un circuito de potencia. Aplicar simplemente la prudencia y utilizar unos tiempos muertos excesivamente amplios estaría limitando el margen de maniobra al algoritmo de control. Por otro lado, arriesgarse demasiado en usar tiempos muertos bajos, o simplemente utilizar un método de prueba y error es totalmente inaceptable dado el precio de cada uno de los dispositivos que se pretenden utilizar.

1.3.2 Rampa de subida

Por otro lado, se debe tener en cuenta el concepto de rampa de subida del circuito disparador de un IGBT. Como se ha introducido anteriormente, el IGBT será el dispositivo electrónico que utilizaremos a modo de interruptor de potencia. Pero al IGBT no pueden llegarle directamente señales tipo 1 ó 0 (Abierto ó cerrado) directamente del sistema de control, ni el paso de un estado a otro de la señal de control puede ser un escalón.

Se necesita un dispositivo (driver) que controle la conmutación del IGBT, al fin y al cabo lo que debemos hacer es estimular (alimentar eléctricamente) de manera correcta la puerta del IGBT para que cambie de estado de conmutación. Y este driver debe incluirse en un pequeño circuito que adapte las señales de encendido y apagado de control y el transitorio necesario entre ellas, sus niveles de tensión y las corrientes necesarias en la entrada del IGBT.

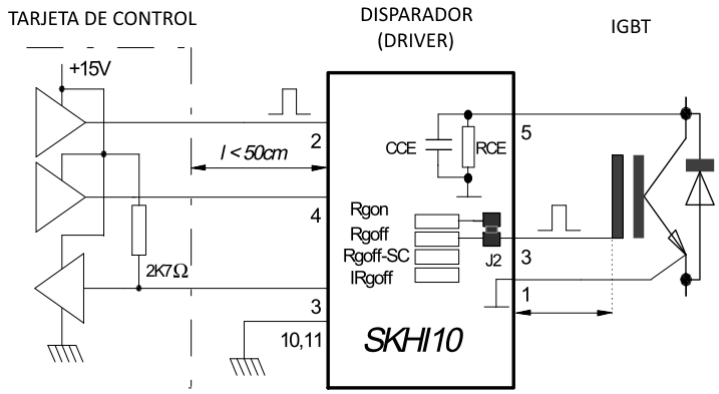

Figura 1-11: Tarjeta de control alimentando a dispositivo disparador de IGBT

Por ello, para poder simular el funcionamiento del IGBT debemos también simular correctamente las señales que le llegan, para que el comportamiento real del mismo quede perfectamente reflejado. No tiene mucho sentido intentar modelar cada uno los disparadores y su correspondiente tarjeta de control para cada uno de los 12 IGBTs. Bastará con simular convenientemente las señales que el driver le envía al IGBT. De hecho, se montará una simulación exclusivamente para probar que la respuesta del modelo del IGBT a las señales de disparo simuladas es correcta, comparándolas con las suministradas por el fabricante en las especificaciones.

En el caso del IGBT que se va a usar en esta bancada necesitamos un driver de SEMIKRON del modelo SKHI10. Este dispositivo envía señales de tensión que pasan de –8V a +15V y viceversa, y lo hacen a través de una rampa, y no un escalón como puede suponerse a un interruptor. Esto conlleva que la misma señal de conmutación sufre un retraso debido al tiempo de propagación de esta señal por los circuitos (~1.4 μs), sumada al necesario por esta rampa para pasar de un estado de tensión al otro, que es de 10 μs para llegar a la señal ON y de 6.8 μs para volver a la señal OFF.

2 CONVERSIÓN DE MATLAB A PSPICE

En lo sucesivo, llamaremos "Conversor MATLAB-PSPICE" o simplemente "Conversor M-P" al programa (conjunto de programas) creados para Matlab para convertir los disparos de los IGBTs generados por el algoritmo ideal en Matlab a ficheros de disparo utilizables en PSPICE.

La razón de este conversor es generar un archivo de señales de disparo para los IGBT con un formato que PSPICE pueda entender y de paso, simular las diferencias que hay entre las señales escalonadas de las conmutaciones ideales y las señales con rampas creadas por el dispositivo disparador. Aun cuando ambos sistemas manejan los datos de manera discreta (no son graficas continuas sino definidas por puntos) y usan datos de tensiones e intensidades en función del tiempo, la definición punto a punto de estas gráficas no tiene la misma validez en ambos sistemas. Por ejemplo, mientras en Matlab tenemos un escalón de tensión perfectamente cuadrado en las señales de conmutación, en la simulación en PSPICE necesitamos "traducir" estas transiciones a rampas de subida y bajada para dar el primer paso para convertir el sistema de ideal a real.

Por otro lado, en la transformación se aprovecha para reducir la cantidad de datos a tratar reduciendo los puntos que no aportan información.

El conversor se ha dividido en 5 sub-programas bien definidos que facilitan su depuración, modificación y posible adaptación a otros sistemas y/o situaciones. A estos sub-programas se les ha asignado el nombre según su objetivo:

1. REDUCE: Eliminar los puntos redundantes creados por el algoritmo en MATLAB
2. AJUSTA: Convierte escalones de conmutación a rampas (comportamiento real del driver)
3. DESPLAZA: Retrasar el encendido para evitar cortocircuitos y evitar conmutaciones inútiles
4. ESCALA: Modificar los niveles a tensiones de para el driver del IGBT (15V y -8V)
5. GRABA: Generar el fichero 'disparos.stl' necesario para ORCAD/PSPICE

.

2.1 Pasos previos a la aplicación del Conversor M-P.

El *conversor M-P* está programado en Matlab para ser ejecutado directamente en su ventana de comandos. Para aplicar la conversión, necesitamos que previamente se haya ejecutado una simulación de cualquier algoritmo trinivel (véase 1.-Razón de este proyecto), para tener así cargados en memoria los valores de los estados de los IGBT a lo largo de toda la simulación. No es necesario fijar una duración máxima de la simulación (tiempo de simulación), puesto que el conversor transformará todos los puntos existentes.

La simulación de MATLAB debe haber generado y mantenido en memoria los valores que necesitamos tratar. Generalmente tendremos una tabla de datos por cada elemento. El conversor leerá las 12 tablas que corresponden a los 12 interruptores (IGBT), que deben llamarse *disp1, disp2, ..., disp12*. En la Figura 2-2 se detalla a qué interruptor corresponde cada tabla de datos. Todas las tablas serán de tamaño (Nx2), con lo que tendremos N pares de valores (tiempo, estado) en cada tabla para cada IGBT.

Inicialmente las 12 tablas tienen longitud N, ya que para cada instante de tiempo se han capturado los estados de los 12 interruptores. Tras aplicar 'REDUCE', el primer sub-programa del conversor, obtendremos 12 nuevas tablas. Como veremos en detalle, las dimensiones de las nuevas tablas dejan de ser iguales. Sólo nos interesan los momentos en lo que el interruptor conmuta, con lo que obviaremos toda información intermedia, que no genera cambio de estado alguno. Posteriormente iremos llevando a cabo distintas actuaciones sobre los datos de estas tablas (AJUSTA, DESPLAZA y ESCALA) para adaptar la señal las necesidades de PSPICE y el IGBT. Todos estos pasos podrían perfectamente haberse diseñado en uno solo, pero dividir cada actuación en pasos distintos facilita el entendimiento, depuración y posterior modificación y adaptación para posteriores trabajos.

2.2 Conversor M-P: REDUCE

Los algoritmos implementados (en MATLAB y en el circuito de control de la bancada) trabajan con un paso de tiempo constante definido por la propia frecuencia de trabajo de cada algoritmo. Esta frecuencia de trabajo debe necesariamente ser al menos un orden de magnitud mayor que la frecuencia típica de conmutación para poder tener un buen control del sistema.

De esta manera, durante la ejecución del algoritmo Matlab generan tablas de pares de valores que corresponden al estado de cada interruptor frente al tiempo. Así, en el caso del trinivel tenemos 12 tablas que corresponden a los 12 IGBT. Las 12 tablas se rellenan con un valor nuevo cada una en cada nuevo instante de tiempo de muestreo. Si el estado de un IGBT no cambia durante una serie consecutiva de instantes, se genera una serie de puntos con el mismo valor (1 ó 0) dependiendo del estado. A posteriori esta información es redundante, y bastaría una tabla donde se indican los instantes en los que se tienen valores distintos, como se aclara en la figura 2-1.

Para cada cambio de estado sólo necesitamos el punto de inicio de la rampa y el punto de final de rampa. Todos los puntos intermedios no aportan información y podemos eliminarlos. Exactamente esto es lo que hace el algoritmo REDUCE.

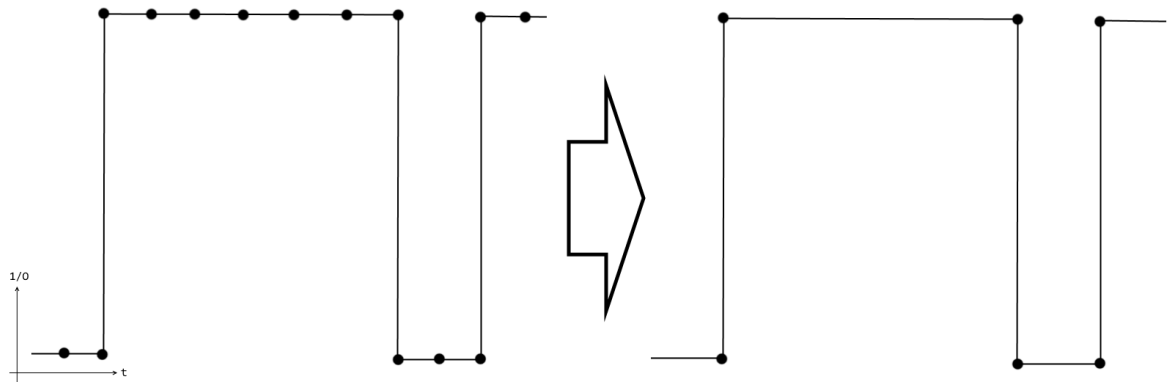

Figura 2-1: Eliminación de puntos redundantes en la señal de conmutación

En un primer momento, esta eliminación de puntos superfluos puede parecer innecesaria, pero para altas frecuencias de muestreo se generan una cantidad importante de puntos comparada con la frecuencia de conmutación. En bucle cerrado todo esto no es necesario puesto que pasados los pocos puntos anteriores que se usan en el control en cada instante dado, toda la información anterior se desecha porque carece de interés. Pero recordemos que en este caso hemos guardado todos y cada uno de los estados de los interruptores en Matlab para hacer luego una simulación en bucle abierto en PSPICE. La cantidad de datos puede ser importante y el tamaño de los ficheros de datos excesivo.

Por ejemplo, si necesitamos mantener cerrado el interruptor (IGBT) durante 1 milisegundo y estamos generando puntos cada 1 microsegundo, estamos generando 1000 puntos que simplemente dicen "mantener cerrado", cuando en realidad basta con saber cuándo de debe abrir y cuando cerrar cada uno de los IGBT.

De esta manera vamos a reducir el tamaño de los ficheros de disparo que se generarán, reduciendo así el tiempo de exportación/importación, manejo y computación posterior. Se ha comprobado que el ratio típico de reducción de datos en los algoritmos manejados ha sido del orden de 1000 a 1. Lógicamente, esta cantidad puede variar mucho dependiendo del algoritmo, su frecuencia de muestreo y la frecuencia máxima de conmutación.

El sub-programa creado es realmente sencillo. Lo único que hace es ir tomando puntos y compararlos con el anterior y con el siguiente. Si detectamos que no hay ningún cambio de estado, pues obviamos el punto y seguimos con el siguiente. Así nos quedamos exclusivamente con los puntos que aportan información de cambio de estado.

Las tablas que debemos encontrarnos en memoria deben ser de 2 columnas en las que tendremos la pareja de valores (tiempo, estado) y deben llamarse *disp1* hasta *disp12*, siendo *disp1* a *disp4* las tablas de estados de conmutación de los interruptores de la primera rama, ordenados de arriba abajo en el esquema del circuito (ver siguiente figura), y los vamos creando otras 12 tablas siguientes:

- *ar1*, *ar2*, *ar3* y *ar4* para la rama A, IGBTs ordenados de arriba a abajo
- *br1*, *br2*, *br3* y *br4* para la rama B
- *cr1*, *cr2*, *cr3* y *cr4* para la rama C

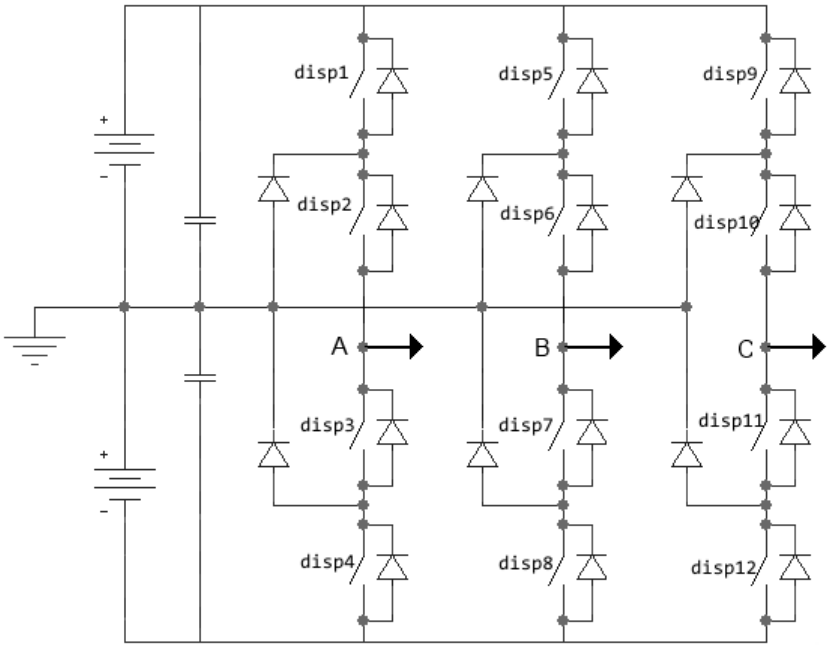

Figura 2-2: Correspondencia de las tablas originales de Matlab con los IGBT

Además, se aprovecha este barrido punto a punto para eliminar los picos en la señal que no aportan nada a la conmutación. Son puntos en los que se cambia 2 veces de estado en el mismo instante de tiempo.

Para el primer IGBT:

```
tam=size(tout);
long=tam(1);
disp('Rama 1');
ar1(1,1:2) = disp1(1,1:2);
ar1(2,1:2) = disp1(2,1:2);
n=3;
for k=3:long-1
 ar1(n-1,1:2) = disp1(k-1,1:2);
 ar1(n,1:2) = disp1(k,1:2);
 if (disp1(k-1,2) == disp1(k+1,2))&(disp1(k,2) ~= disp1(k+1,2)) %Detecta picos
  ar1(n+1,1:2) = disp1(k,1:2);
  n=n+3;
  k=k+2;
  end;
 if disp1(k,2) ~= disp1(k-1,2) %Detecta cambios de estado
  n=n+2;
  k=k+1;
 end;
end;
ar1(n,1:2) = disp1(long,1:2);
```

Igualmente, el código se repite para los otros 3 IGBT de la misma rama (variables *ar2*, *ar3* y *ar4*) y para los de las otras 2 ramas (*br1* a *br4* y *cr1* a *cr4*); (ver Anexo A)

2.3 Conversor M-P: AJUSTA

El siguiente paso es convertir el escalón en rampa. Para ello recorremos las tablas de "estado VS tiempo" y sumamos un tiempo prefijado en 1e-6 segundos al primer punto tras un cambio de estado.

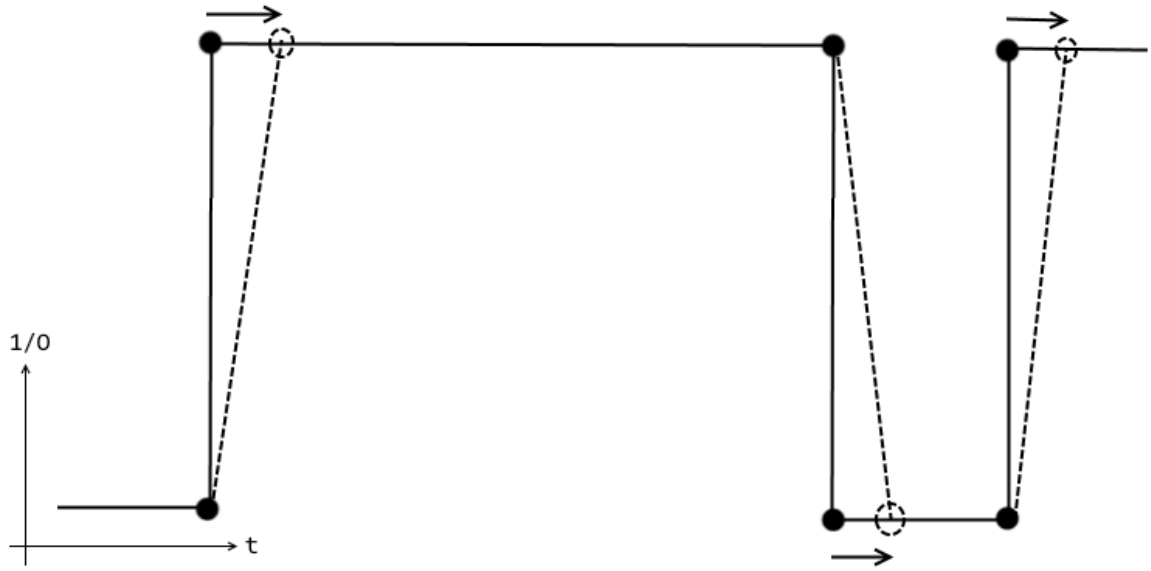

Figura 2-3: Conversión de escalones en rampas

Para ello, se cogen los valores de las tablas *ar1* hasta *cr4* y se crean otras tablas con los nuevos valores que ya incluyen los puntos extra: tablas *a1* hasta *c4*.

```
tam=size(ar1);
long=tam(1);
l=2;
a1(1,1:2)=ar1(1,1:2);
for k=2:long-1
 if (ar1(k-1,2)==ar1(k+1,2))&(ar1(k,2)~=ar1(k-1,2)) %Detecta picos residuales
    a1(l,1)=ar1(k-1,1)+1e-6;
    a1(l,2)=ar1(k,2);
    l=l+1;
    a1(l,1:2)=ar1(k,1:2);
 else
    if ar1(k,2) ~= ar1(k-1,2)          %Detecta cambio estado
        a1(l,1) = ar1(k-1,1)+1e-6;     %Introduce rampa de t=1e-6
    a1(l,2) = ar1(k,2);
 else
        a1(l,1:2)=ar1(k,1:2);
 end;
end;
l=l+1;
end;
a1(long,1:2)=ar1(long,1:2);
```

Igualmente, el código se repite para los otros 3 IGBT de la misma rama (variables *ar2*, *ar3* y *ar4*) y para los de las otras 2 ramas (*br1* a *br4* y *cr1* a *cr4*); (ver Anexo A)

2.4 Conversor M-P: DESPLAZA

Con esta acción, intentamos evitar el problema de cortocircuito que provoca el encendido de un IGBT antes de que el apagado de otro de la misma rama se haga efectivo. Se trata de añadir los tiempos muertos que se explicaron de manera cualitativa en el anterior capitulo.

La idea es buscar un escalón de bajada y esperar el tiempo necesario tras él para asegurarnos de que ese IGBT se ha apagado realmente. Sólo entonces podemos encender el otro IGBT implicado. Así, en cada bucle se busca el tiempo en el que la señal cambia de estado (de 1 a 0) y en ese mismo instante de tiempo se busca si hay algún encendido que pueda provocar un cortocircuito al ocurrir antes del total apagado del anterior. Si lo encontramos, retrasamos el comienzo de la rampa de encendido para evitar el cortocircuito.

Se incluye una comprobación que pretende localizar si en algún momento el tiempo transcurrido entre la orden de encendido de un IGBT y su posterior orden de apagado (y viceversa) es menor que el tiempo de retraso en el encendido (ó apagado) del IGBT. Si esto ocurre significa que el algoritmo de control está intentando hacer una conmutación que físicamente no es posible y que de hecho no se va a poder llevar a cabo. Por ello, eliminamos esta conmutación completa por no tener efecto en la salida de la bancada y si tener, en cambio, efecto en la potencia disipada, en la siguiente conmutación, etc...

Se ha comprobado que el IGBT *a1* sólo interfiere en la conmutación con el *a3* debido a la política de cambio de estado paso a paso del algoritmo de control. Esto es: en la conmutación de cada rama, desde el estado '+V' sólo se puede pasar a '0', y de éste último sólo se puede pasar a '-v' y viceversa. Igualmente con *a2* y *a4* y las otras dos ramas.

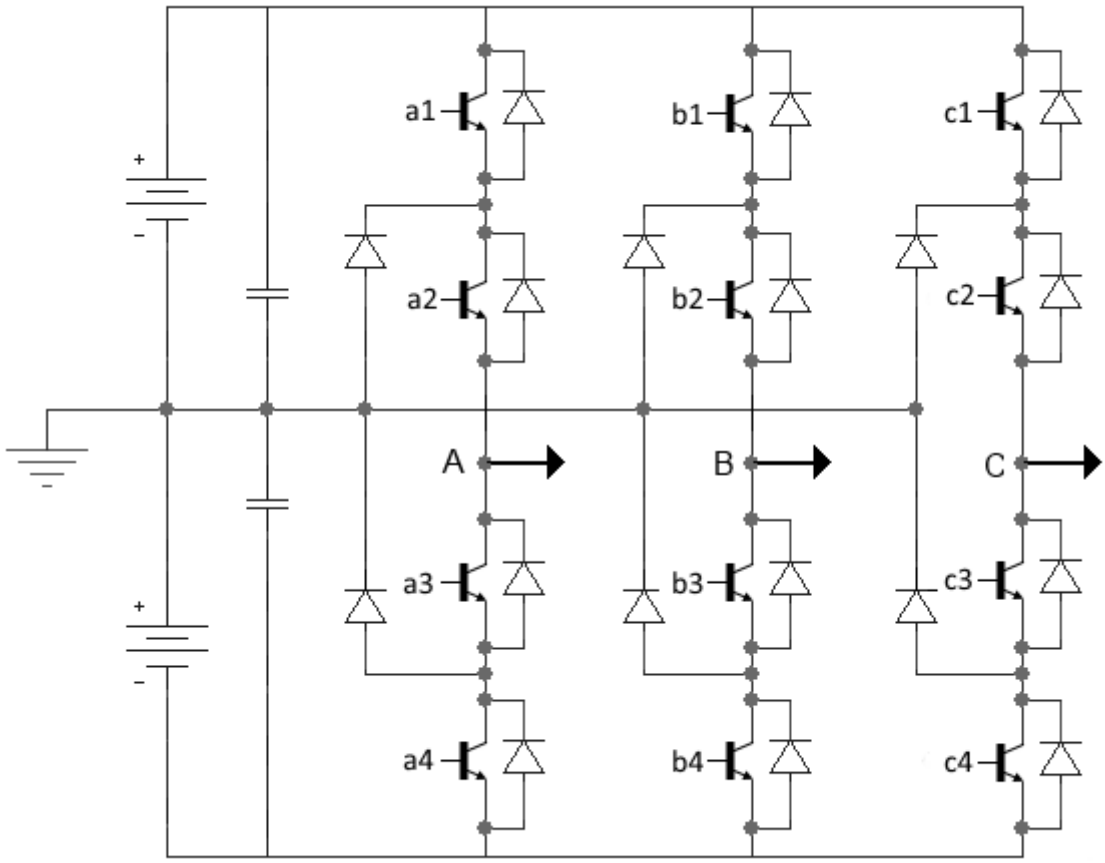

Figura 2-4: Nomenclatura de los distintos IGBT

El paso 'desplaza.m' está diseñado para funcionar según la anterior suposición. Así, se busca un escalón de bajada en, por ejemplo, a1 y (si existe) se desplaza en el tiempo el escalón de subida de a3, una cantidad definida por el *tiempo muerto* necesario. En realidad este tiempo muerto hay que ajustarlo por el método de prueba y error. Partiendo de los valores especificados por el fabricante del IGBT se deberá aumentar (posiblemente disminuir no sea posible) hasta que en la simulación real desaparezcan los problemas de cortocircuito.

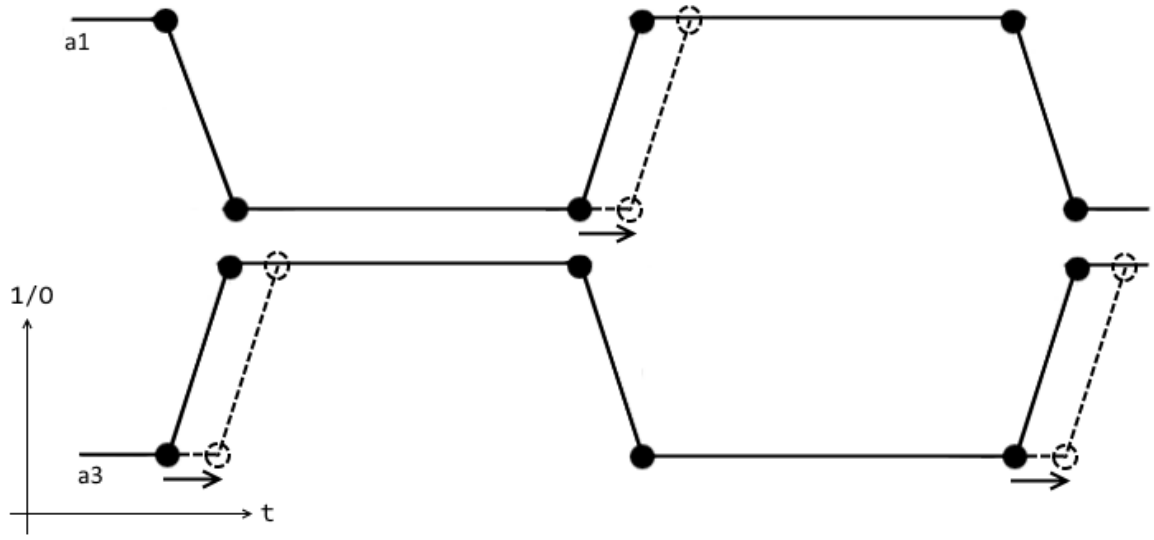

Figura 2-5: Aplicación de tiempos muertos

El código MATLAB primero detecta la existencia de 2 cambios de estado de un IGBT en un tiempo menor que el tiempo muerto para obviar directamente esa doble conmutación. Y en un segundo paso inserta estos tiempos muertos en las señales de conmutación de *a1* y *a3*.

```
minramp=0.000004; %Variable de Usuario: Tiempo mínimo de bajada de una rampa
elim=0;
tam=size(a1);
long=tam(1);
for k=1:long-2
 %Utilidad 1: Detecta si conmutación completa (0-1-0 ó 1-0-1) en t<'minramp'
 if a1(k+2,1)-a1(k,1)<=minramp
  a1(k+1,2)=a1(k,2);
  a1(k+2,2)=a1(k,2);
  a1(k+3,2)=a1(k,2);
  a3(k+1,2)=a3(k,2);
  a3(k+2,2)=a3(k,2);
  a3(k+3,2)=a3(k,2);
  elim=elim+1; %contador puntos eliminados
 end;
 %Utilidad 2: Detecta escalones bajada y los convierte a rampas de t='minramp'
 if (a1(k,2)==1)&(a1(k+1,2)==0) %Detección del escalon de bajada en 'a1'
  a3(k,1)=a3(k,1)+minramp;      %Desplazar punto inicial del escalon
  a3(k+1,1)=a3(k+1,1)+minramp;  %Desplazar punto final del escalon
  end;
 if (a3(k,2)==1)&(a3(k+1,2)==0) %Detección del escalon de bajada en 'a3'
  a1(k,1)=a1(k,1)+minramp;      %Desplazar punto inicial del escalon
  a1(k+1,1)=a1(k+1,1)+minramp;  %Desplazar punto final del escalon
 end;
end;
msg=sprintf('%d eliminaciones en a1/a3:',elim); %Mensaje meramente informativo
disp(msg)
```

Igualmente, para los IGBT 2 y 4 de la misma rama y los 1-3 y 2-4 de las otras dos ramas (ver Anexo A)

2.5 Conversor M-P: ESCALA

Cuando hablábamos de simular también directamente la señal de la salida del dispositivo disparador del IGBT comentábamos que, para este dispositivo, los valores de la salida son -8V y +15V para el apagado y encendido respectivamente. Así, tenemos que escalar nuestra señal para adaptarse a estos valores. Así, los valores 'CERO' se convertirán en '-8' y los valores '1' en '+15'.

Es una simple acción de escalado vertical de una gráfica, que punto a punto podemos hacer de la siguiente manera:

```
tam=size(a1);
long=tam(1);
for k=1:long
 if a1(k,2)==1
  a1(k,2)=15;
 else
  a1(k,2)=-8;
 end;
end;
```

Igualmente para las otras 11 tablas de datos correspondientes a las señales del resto de los IGBT (ver Anexo A)

2.6 Conversor M-P: GRABA

Una vez que ya tenemos todos los datos convertidos a un formato de señal que es reconocible por PSPICE, tenemos que volcar esta información sobre un fichero, que será el que le entregue la información a la simulación. El fichero que se necesita tiene extensión STL y el formato es el de un simple fichero de texto. Es el tipo de ficheros que genera el editor/generador de curvas STIMULUS EDITOR. Lo único que tiene de particular es que tenemos que añadir una cabecera con información del tipo de datos que estamos entregando.

Un ejemplo de contenido de este tipo de ficheros es:

```
   ;!Stimulus Get
   ;! a1 Analog a2 Analog a3 Analog a4 Analog b1 Analog b2 Analog b3 Analog b4
Analog c1 Analog c2 Analog c3 Analog c4 Analog
   ;!Ok
   ;!Plot Axis_Settings
   ;!Xrange 0s 40ms
   ;!Yrange -10 18
   ;!AutoUniverse
   ;!XminRes 1ns
   ;!YminRes 1
   ;!Ok

   .STIMULUS a1 PWL
   + TIME_SCALE_FACTOR = 1
   + VALUE_SCALE_FACTOR = 1
   + ( 0.000000000 , 15.0 )
   + ( 0.003588260 , 15.0 )
   + ( 0.003588261 , -8.0 )
   + ( 0.003600000 , -8.0 )
   + ( 0.003600001 , 15.0 )
   + ( 0.003770274 , 15.0 )
       . . .
```

Las primeras líneas, las que empiezan por ';!' (puntoycoma-exclamacion) sólo son necesarias si vamos a abrir el fichero de señales con el STIMULUS EDITOR para ver que curvas son las que tenemos y para comprobar cualquier detalle. PSPICE realmente no las necesita. Aunque en el caso de que la queramos abrir en el editor, necesitaremos adaptar las variables de rango `Xrange` y `Yrange` a los tamaños de nuestra señal; el resto de parámetros definen la resolución y las variables usadas.

La línea ".STIMULUS a1 PWL" es una declaración de variable, con la que le decimos a PSPICE que tipo de señal le vamos a entregar: Le llamaremos *a1* y es una PWL (PieceWise Linear / función lineal a trozos).

Así que lo que hace este código de programa es preparar un archivo para escribir en él con la línea *`fid = fopen('disparos.stl','w')`* y a continuación empieza a escribir la cabecera y luego los datos, declarando una variable por cada uno de los IGBT (a1 hasta c4). Tras ello, cerramos el fichero con *`fclose(fid);`*

Por defecto, el fichero 'disparos.stl' se graba en la carpeta MATLAB/BIN y debe llevarse a la carpeta del proyecto PSPICE a simular.

```
   % Cabecera del fichero PSPICE
   fid = fopen('disparos.stl','w');
   fprintf(fid,';!Stimulus Get\n');
   fprintf(fid,';! a1 Analog a2 Analog a3 Analog a4 Analog b1 Analog b2 Analog b3
Analog b4 Analog c1 Analog c2 Analog c3 Analog c4 Analog\n');
   fprintf(fid,';!Ok\n');
   fprintf(fid,';!Plot Axis_Settings\n');
   fprintf(fid,';!Xrange 0s 20ms\n');
   fprintf(fid,';!Yrange -1 6\n');
   fprintf(fid,';!AutoUniverse\n');
   fprintf(fid,';!XminRes 1ns\n');
   fprintf(fid,';!YminRes 1\n');
   fprintf(fid,';!Ok\n');


   % Escritura de los datos
   fprintf(fid,'.STIMULUS a1 PWL\n');
   fprintf(fid,'+ TIME_SCALE_FACTOR = 1\n');
   fprintf(fid,'+ VALUE_SCALE_FACTOR = 1\n');
   fprintf(fid,'+ ( %6.9f , %1.1f ) \n',a1.');
```

Este último bloque se repite para cada IGBT (ver Anexo A)

3 SIMULACIONES EN PSPICE

3.1 Estabilidad de PSPICE en electrónica de Potencia

Para afrontar la estabilidad de los sistemas simulados en PSPICE, debemos conocer el problema de fondo. Intentar resolver un transitorio para una simulación como las que se tratan en este documento, significa resolver problemas no lineales. La forma de llegar a la solución es mediante procesos iterativos. Precisamente es este proceso iterativo el que se puede hacer inestable, esto es, no converger hacia ninguna solución, finalizando así el problema sin poder llegar a la solución.

El hecho de haberle pedido la solución de un problema de electrónica de potencia en un software preparado inicialmente para microelectrónica nos da la primera razón de la posibilidad de no convergencia de la solución. Para empezar, no tiene sentido que usemos las mismas tolerancias absolutas cuando manejamos miliamperios que cuando manejamos Amperios. Estamos hablando de una diferencia de muchos órdenes de magnitud.

La estabilidad no depende sólo de la configuración de la simulación. La variación de la topología de un circuito puede llevarnos a tener que estar continuamente jugando con los valores de control de las iteraciones para poder llevar a buen puerto. Y por supuesto, los modelos utilizados para los distintos dispositivos también juegan un importante papel.

3.2 Parámetros de simulación.

En las posteriores simulaciones se ha ido depurando la configuración de las simulaciones hasta llegar a unos valores que son estables en la mayoría de las situaciones, con pequeñas variaciones:

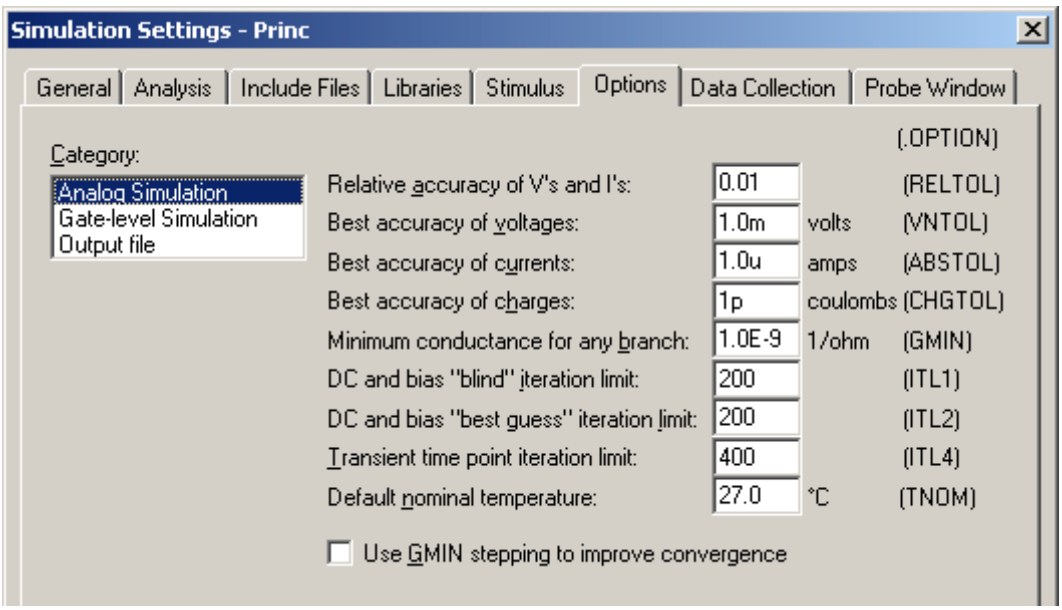

Figura 3-1: Esquema del Inversor monofásico PWM

3.3 Pruebas de validez del modelo del IGBT

Para poder verificar la validez del modelo final de simulación de un trinivel trifásico con IGBT reales, se han puesto a prueba distintas topologías distintas, desde monofásicas a trifásicas, y empezando por ideales e introduciendo posteriormente interruptores reales.

Ha sido muy importante empezar con modelos sencillos e ir aumentando la complejidad paso a paso. La inestabilidad de las simulaciones ha sido patente durante todo el proceso, y era necesario ir confirmando la validez de los dispositivos conforme se iban introduciendo.

3.3.1 Simulación de un Inversor monofásico PWM

Las primeras pruebas previas llevaron a intentar simular directamente montajes con modelos simples de IGBT, con una primera aproximación a los circuitos de control y disparo. Lo primero que se demostró es que la estabilidad podía ser un problema importante al aumentar la complejidad del circuito. Tras resolver esos primeros inconvenientes, se pusieron a prueba los IGBT y se comprobó la validez del modelo. Las intensidades de salida se corresponden con lo esperado.

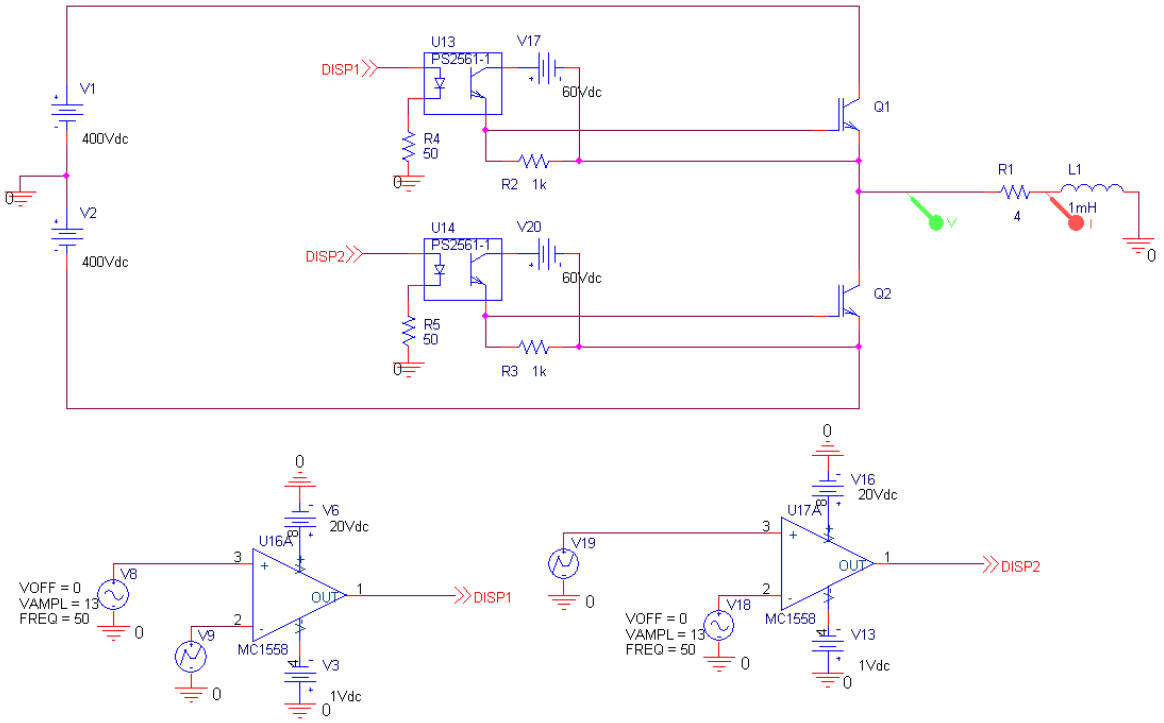

Figura 3-2: Esquema del Inversor monofásico PWM

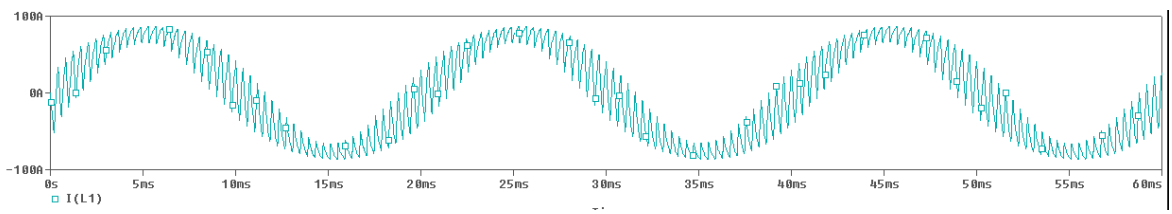

Figura 3-3: Intensidad de Rama del Inversor monofásico PWM

3.3.2 Simulación de un Inversor trifásico PWM

Al extrapolar el mismo tipo de montaje a un sistema trifásico se confirmaron las primeras sospechas. La estabilidad no era el punto fuerte cuando empezábamos a subir las tensiones e intensidades e incluir dispositivos de potencia. Aún así, al ser un "simple" PWM, los resultaron cuadraron con facilidad.

Pero tras varias pruebas sobre esta topología, se decide que no tiene sentido complicar los circuitos disparadores del driver del IGBT, si se consigue una alternativa suficientemente válida.

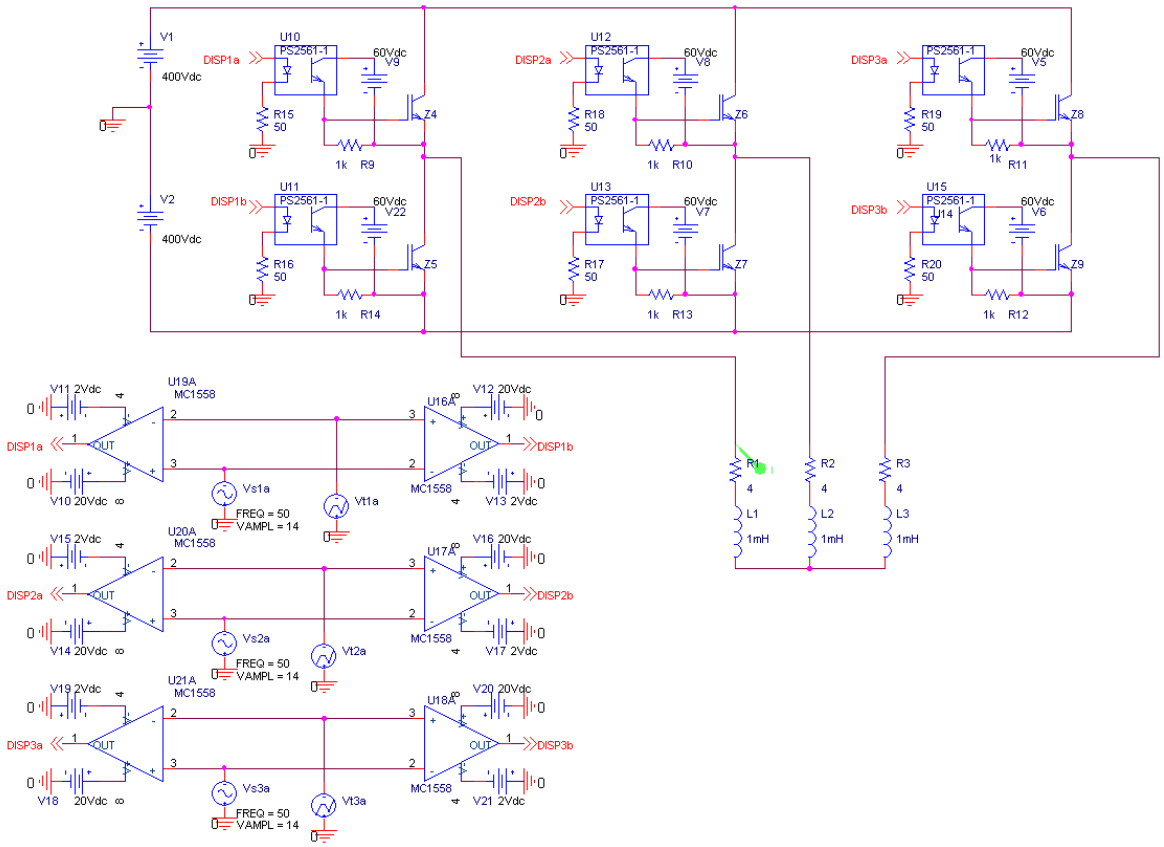

Figura 3-4: Esquema del Inversor trifásico PWM

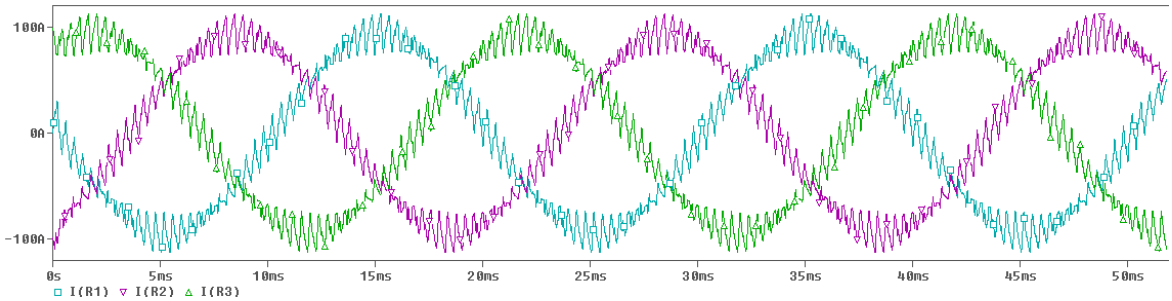

Figura 3-5: Intensidad del Inversor trifásico PWM

3.3.3 Simulación de un Inversor trifásico trinivel ideal sin carga

Para comprobar la validez de las señales de disparo de los IGBT que han sido transformadas desde la simulación teórica en Matlab, se empieza aplicando dichos estímulos sobre un montaje trifásico trinivel ideal. La idea es tener referencias validas para comparar las salidas cuando sustituyamos los interruptores ideales por IGBT de potencia.

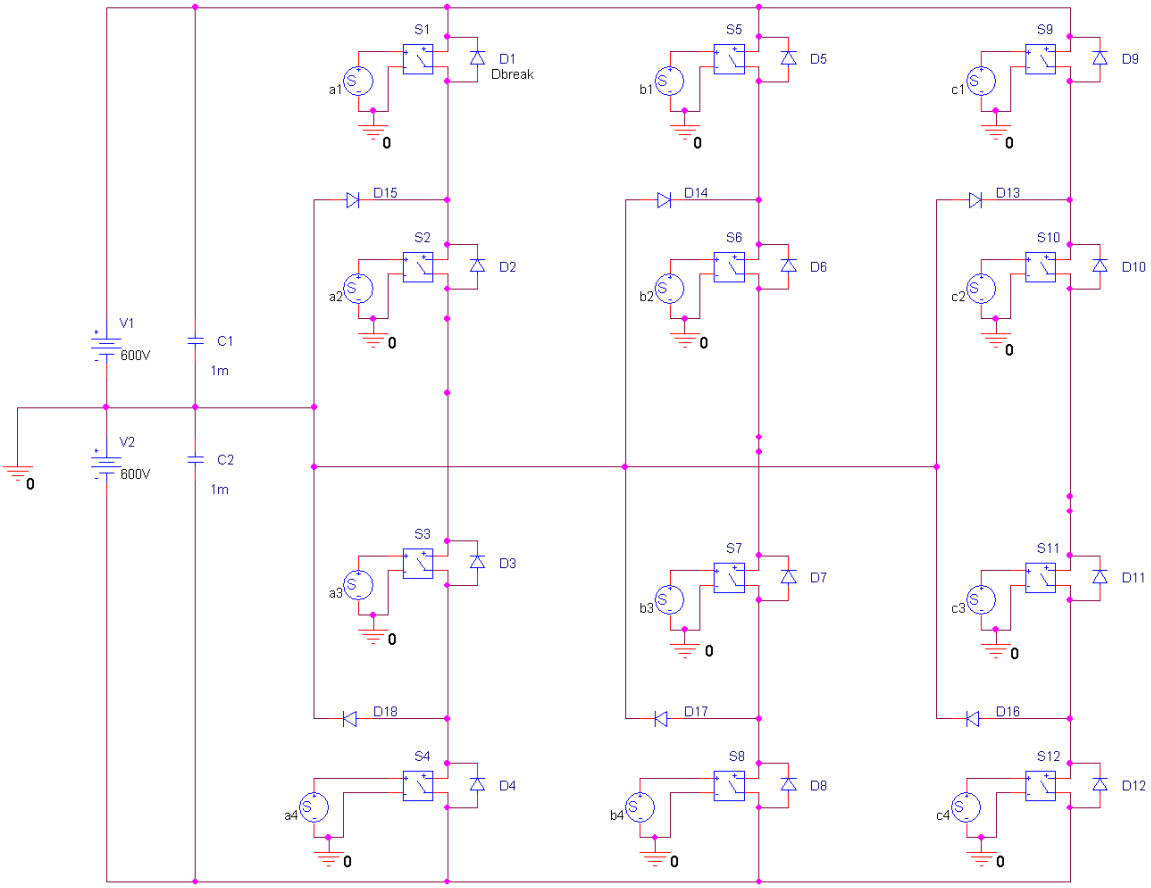

Figura 3-6: Esquema del Inversor trifásico trinivel sin carga

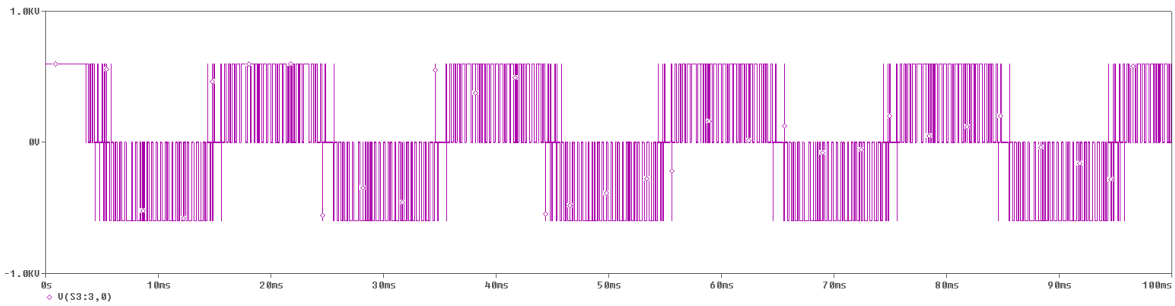

Figura 3-7: Tensión de Fase del Inversor trifásico trinivel sin carga

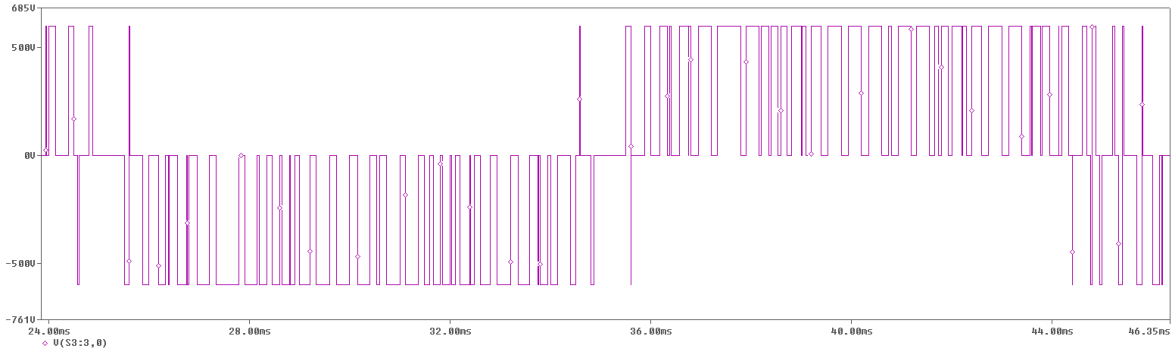

Figura 3-8: Detalle de la tensión de Fase del Inversor trifásico trinivel sin carga

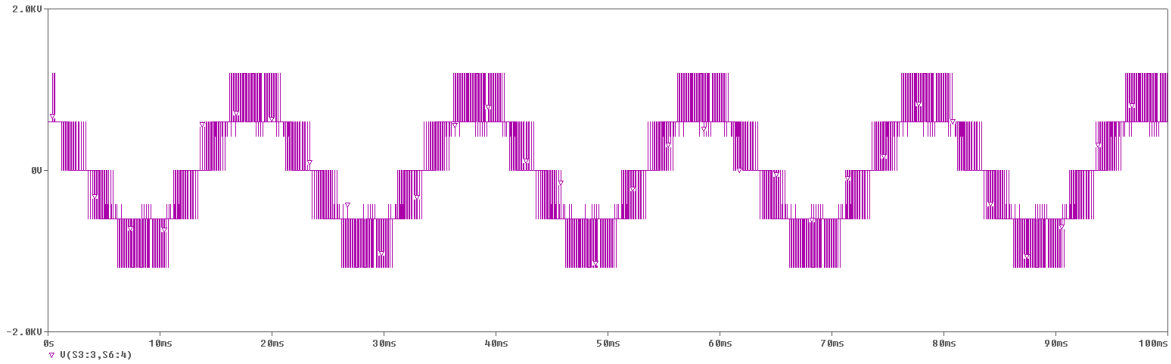

Figura 3-9: Tensión de Línea del Inversor trifásico trinivel sin carga

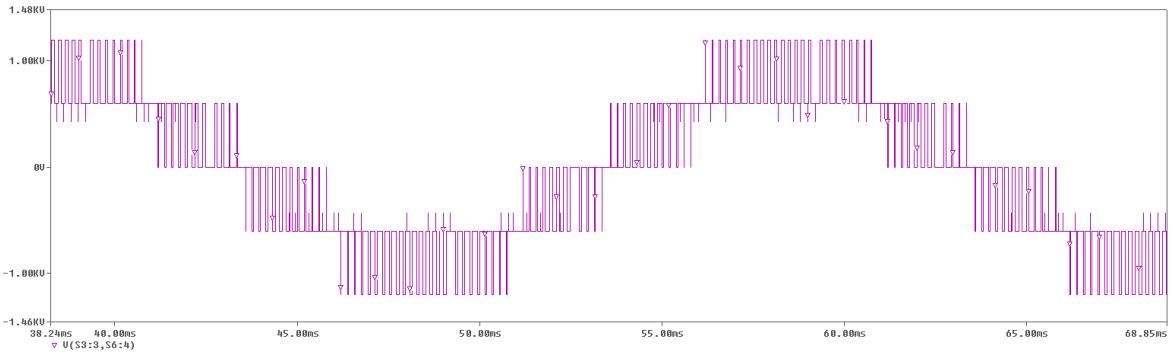

Figura 3-10: Detalle de la tensión de Línea del Inversor trifásico trinivel sin carga

Inmediatamente se demuestra la necesidad de un ajuste fino de las rampas de conmutación. Todavía no podemos hablar estrictamente de tiempos muertos porque los tiempos muertos aparecen por culpa de los transitorios de los interruptores reales y todavía estamos manejando interruptores ideales.

3.3.4 Simulación de un Inversor trifásico trinivel ideal con carga

Al aplicar carga al circuito ideal volveremos a tomar referencias para comparar con el caso ideal, al mismo tiempo que se hacen distintas pruebas variando levemente la topología y estudiando los transitorios de los distintos casos.

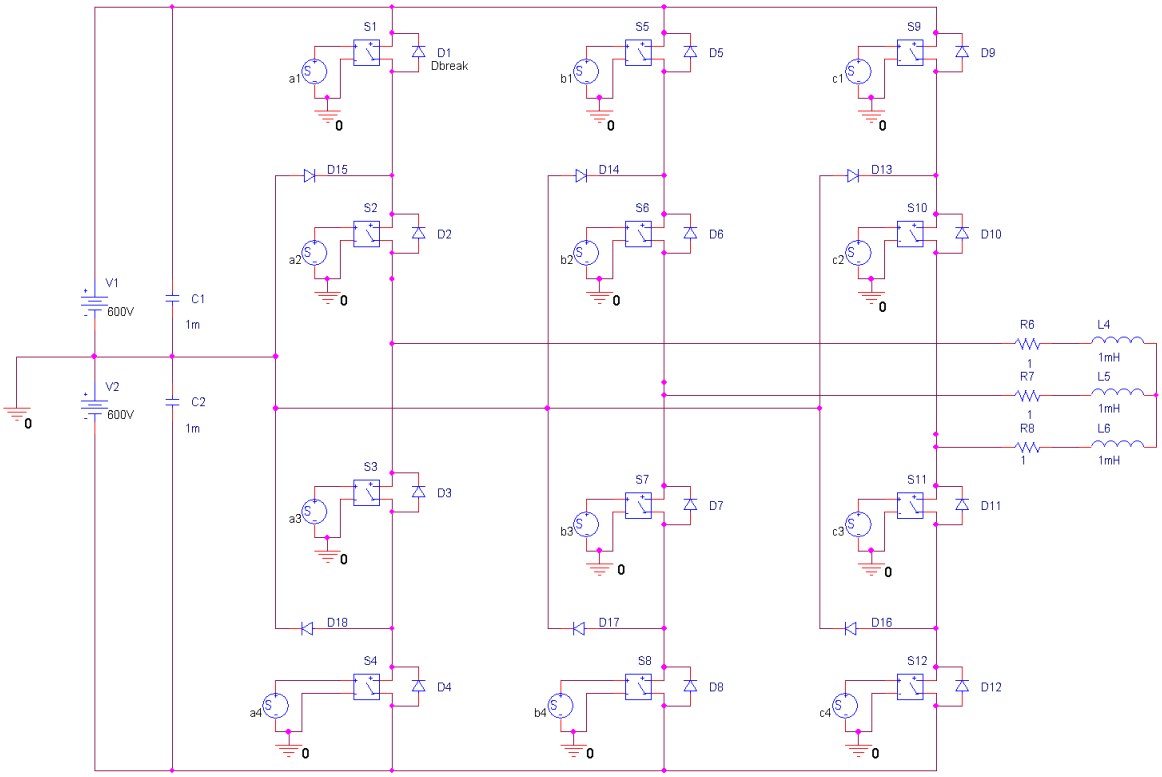

Figura 3-11: Esquema del Inversor trifásico trinivel con carga

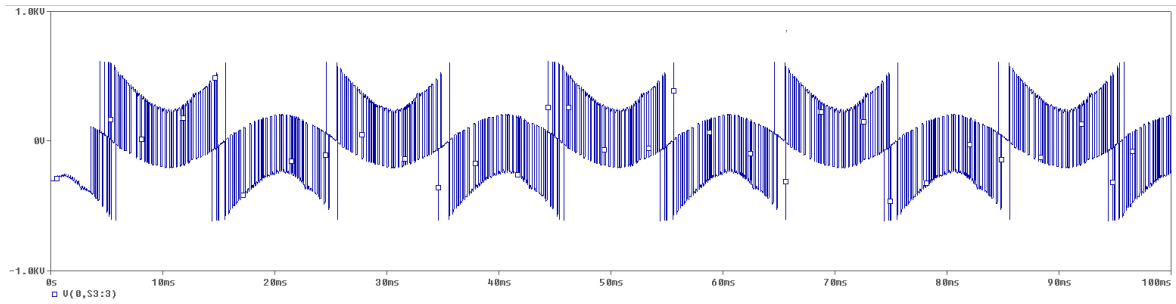

Figura 3-12: Tensión de Fase del Inversor trifásico trinivel con carga

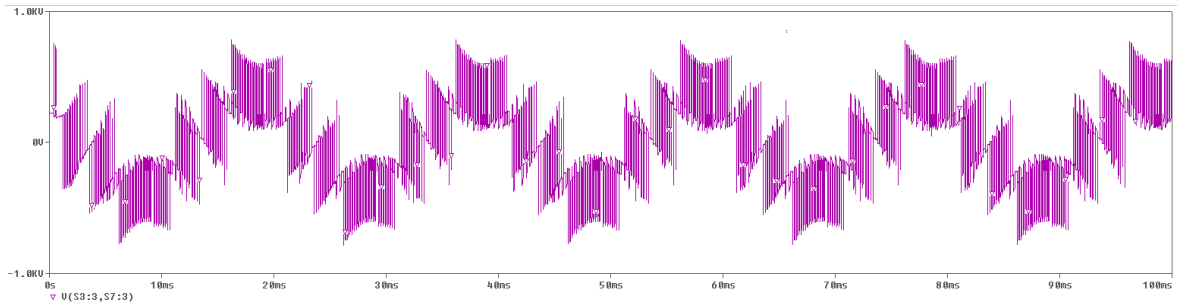

Figura 3-13: Tensión de Línea del Inversor trifásico trinivel con carga

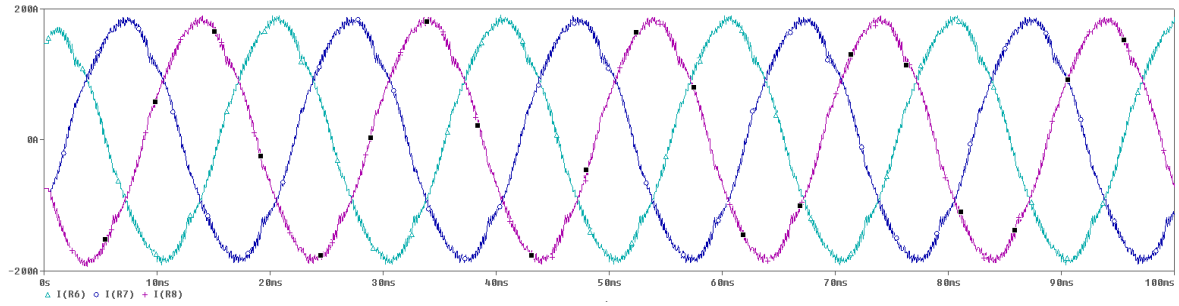

Figura 3-14: Intensidades de Rama del Inversor trifásico trinivel con carga

3.3.5 Simulación TEST de un IGBT

Cuando se toma la decisión final sobre el IGBT elegido para el montaje de la bancada en laboratorio, lo primero es hacerse con un modelo del mismo para poder llevar a cabo las simulaciones. Las librerías necesarias para PSPICE con los IGBT de SEMIKRON se consiguen en foros de electrónica Alemanes y se consideran todavía modelos experimentales. Por ello, lo primero que se hace es realizar una completa batería de pruebas para comparar los valores del modelo con las especificaciones del fabricante.

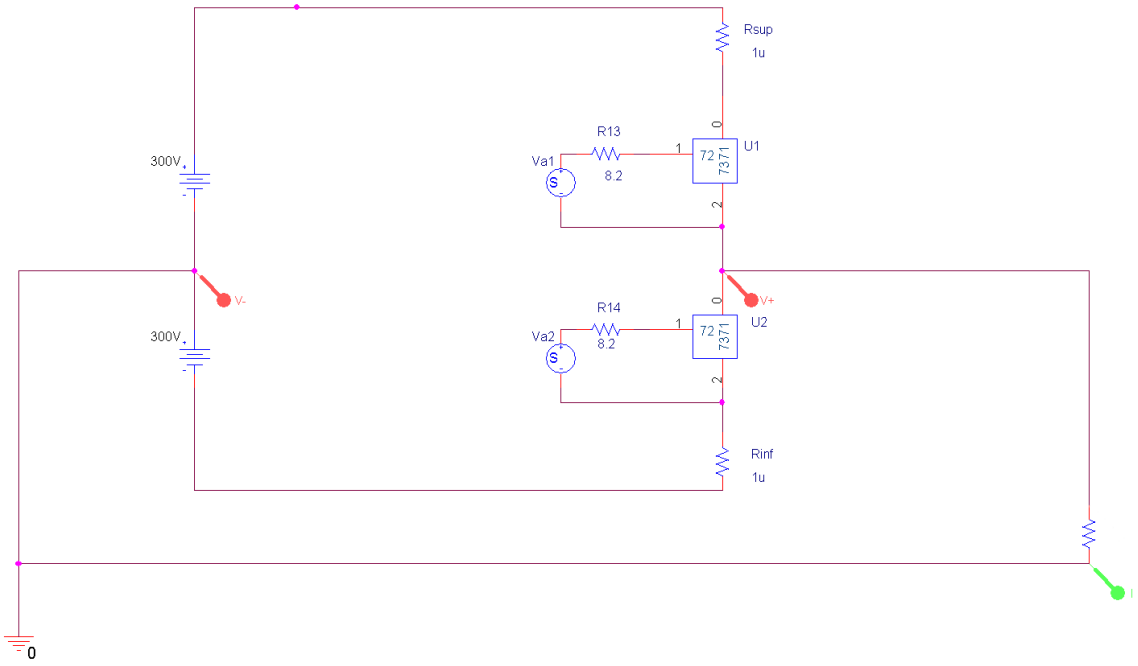

Figura 3-15: Esquema del Inversor de 2 niveles con IGBT SKM300

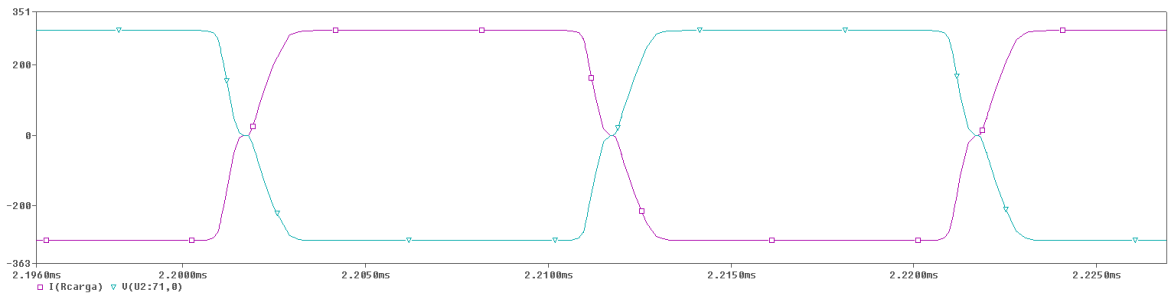

Figura 3-16: Tensión – Intensidad del Inversor de 2 niveles con IGBT SKM300

Se comprueba que la validez del modelo es muy buena, por lo menos en montajes simples. Se aprovecha para hacer las primeras pruebas de simulación del disparador del IGBT.

3.3.6 Simulación Inversor Trinivel Trifásico con IGBT SKM300

Finalmente se llevan a cabo las pruebas del sistema Inversor Trinivel trifásico con los IGBT SKM300 que se montarán en la bancada y con disparos generados por los algoritmos teóricos de control.

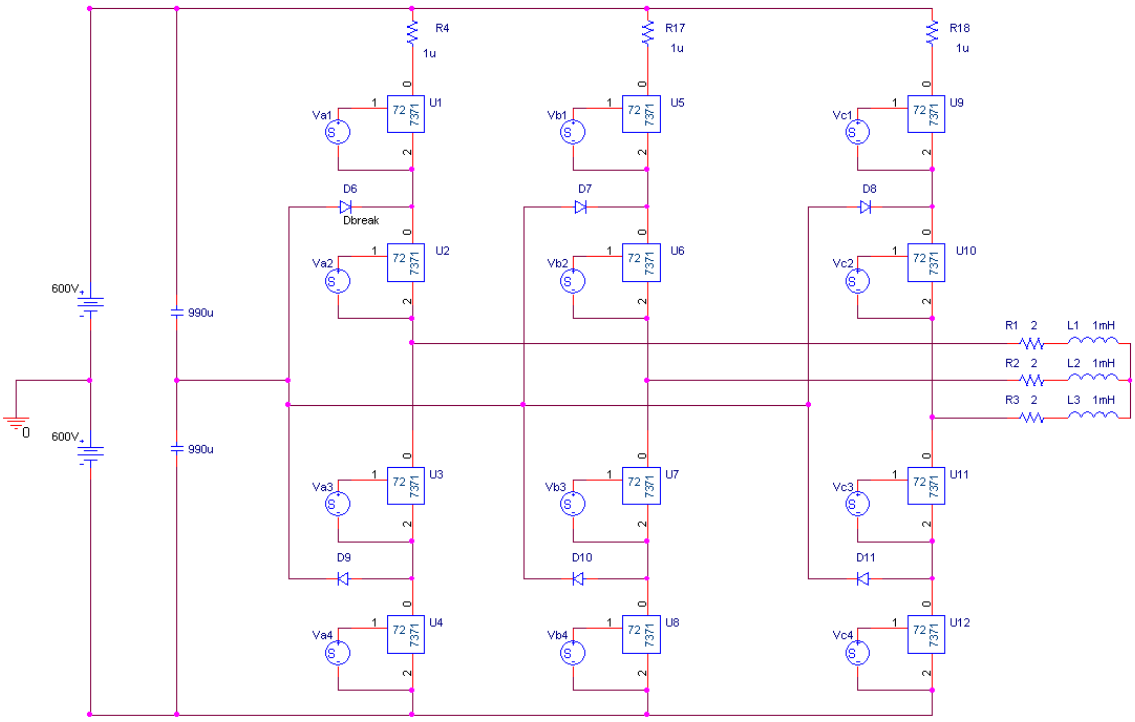

Figura 3-17: Esquema del Inversor Trinivel Trifásico con IGBT SKM300

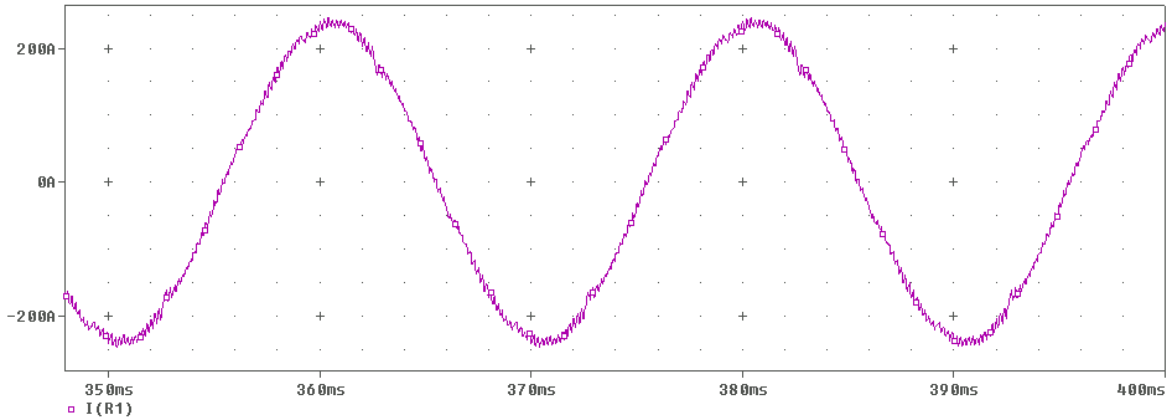

Figura 3-18: Intensidad en régimen estacionario del Inversor Trinivel Trifásico con IGBT SKM300

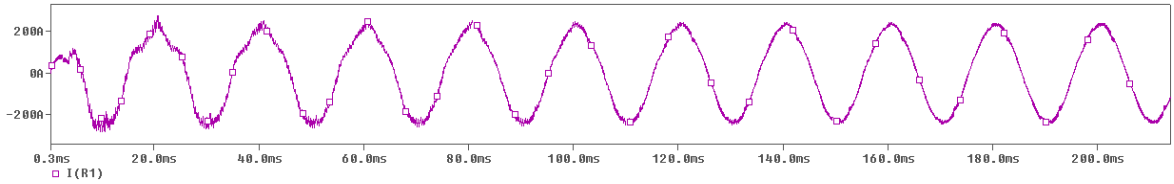

Figura 3-19: Evolución de Intensidad de Rama del Inversor Trinivel Trifásico con IGBT SKM300

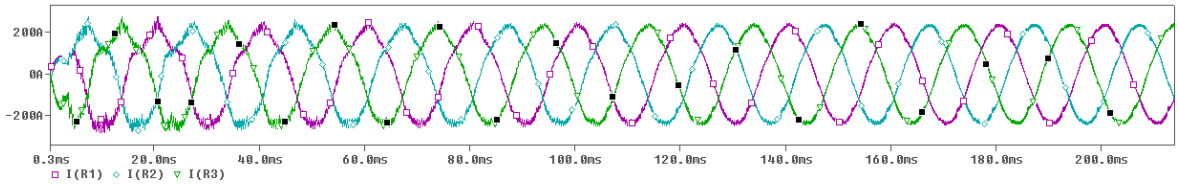

Figura 3-20: Intensidades de Rama del Inversor Trinivel Trifásico con IGBT SKM300

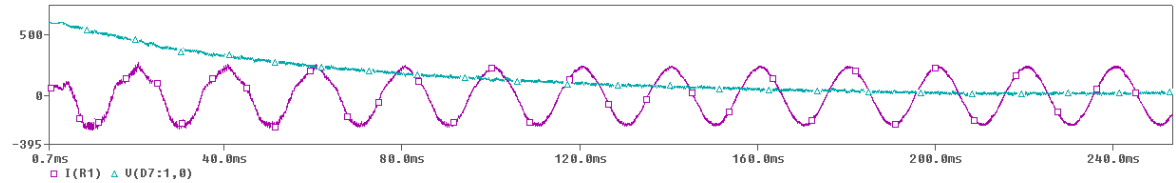

Figura 3-21: Transitorio de estabilización de curva de intensidad

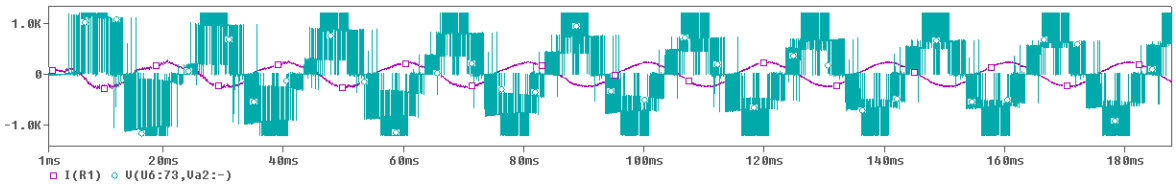

Figura 3-22: Transitorio de estabilización de la tensión

De nuevo, al introducir los modelos de los dispositivos reales, el sistema se hace bastante inestable y se deben ajustar los parámetros de tolerancias de Tensiones e intensidades en cuanto se varía cualquiera de los elementos del circuito. Aún así, se sacan importantes datos de tensiones e intensidades que servirán para dimensionar los dispositivos auxiliares de la bancada.

4 SLPS: INTEGRACION COMPLETA

SLPS es una interfaz de conexión entre SimuLink y PSpice A/D que presenta Cadence en el año 2012. Pretende unir la potencia matemática y algorítmica de MATLAB con las herramientas de simulación de circuitos electrónicos de PSpice.

Lo que esta solución conjunta aporta es intentar evitar tener que postular la primera hipótesis de este proyecto, que se basaba en suponer como punto de partida que "las simulaciones del sistema real pueden llevarse a cabo independientemente de si el algoritmo original era en bucle abierto o en bucle cerrado, esto es, independientemente si en el control del sistema ideal se ha tenido en cuenta el estado de la salida real o no".

Con SLPS podremos llegar a un sistema de control cerrado en el que las siguientes decisiones a tomar siguen dependiendo del estado en la salida, pero en este caso el estado no es ideal, sino que podremos basarnos en el estado real simulado en cada instante.

A lo largo de todo este documento, entre los sistemas ideal (MATLAB) y real (PSpice) se han intercambiado los datos en bucle abierto. Se ha ejecutado una simulación ideal completa en MATLAB (esta si, en bucle cerrado) y posteriormente las matrices de disparo de los IGBT para un determinado periodo de tiempo se han introducido directamenten el PSpice para ejecutar de nuevo la simulación del sistema real (Figura 4-1). En ningún momento hay una vuelta atrás de los datos desde PSpice a Matlab para que el algoritmo tomase decisiones basándose en la salida del sistema real simulado.

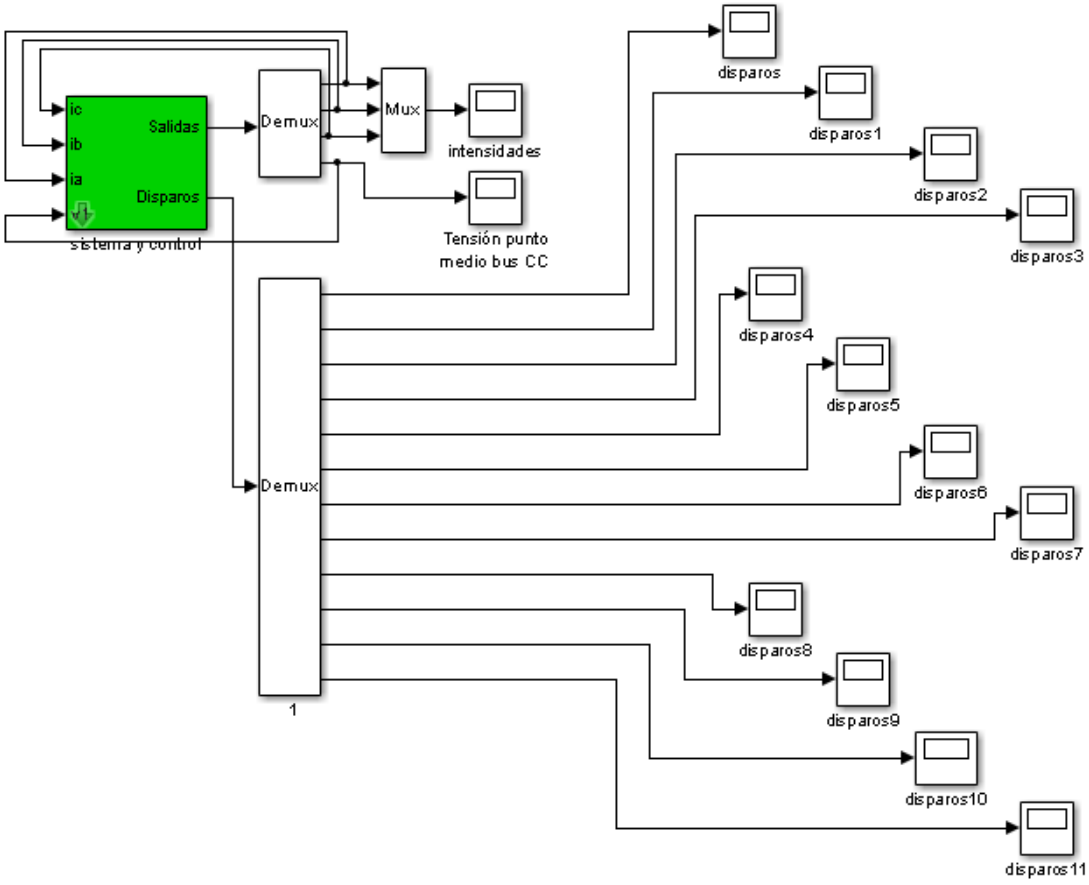

Figura 4-1: Bucle cerrado en simulación ideal, con salida de datos para PSpice

Con la llegada de SLPS se abre la opción de ejecutar en paralelo las simulaciones ideal en Matlab y real en PSpice habiendo una comunicación en tiempo real entre las dos. En Matlab se calcularán las conmutaciones necesarias para los IGBT, información que se pasará en cada momento a PSpice a través del bloque SLPS, donde se actualizará el estado del sistema y se irá pasando información de vuelta a Matlab en los instantes que se requieran, para que el algoritmo tenga conocimiento del estado real y pueda actuar en conseguiencia. Se ha creado de esta manera un bucle cerrado entre los dos sistemas de simulaciones (Figura 4-2).

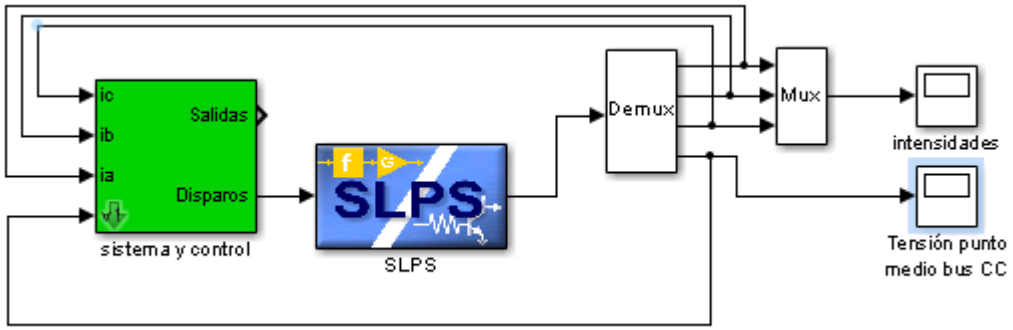

Figura 4-2: Bucle cerrado en simulación ideal, con salida de datos para PSpice

Podemos comprobar los datos de salida tanto en PSpice como en Simulink y comprobar que, no es que sean simulaciones equivamente, sino que son los mismos datos y que realmente son integraciones en tiempo real de las dos simulaciones.

Viendo la figura anterior, salta a la vista que no estamos utilizando la información “salidas” del sistema de control, ya que estos nos darían los estados ideales. En su lugar estamos utilizando las salidas del bloque SLPS, que nos dan los estados de la simulación real. Por supuesto, dentro del sistema de control pueden utilizarse los estados ideales para un mejor seguimiento, comparar estados ideal y real, predicción de estados, etc…

Las 2 anteriores figuras son capturas de los sistemas en bucle abierto (nos referimos al intercambio de datos con PSpice) y en bucle cerrado, en Simulink. Para ver cómo manejamos los parámetros de la simulación de Pspice desde Simulink, abrimos la configuración del bloque (doble click ó botón derecho – abrir) y vemos (Figura 4-3) que necesitamos configurar cuál es el proyecto real de PSpice que corresponde a esta simulación ideal. Desde esta ventana de Simulink también se elegirá cuál de los circuitos se está tratando.

Y lo más importante para el intercambio de datos, tenemos que elegir cuáles de los datos de la simulación de Pspice son los que va a suministrar Simulink. En este caso son disparos de los los IGBT, que se representan como valores de tensión a la entrada del driver. Igualmente, tenemos que indicar qué valores cogeremos de vuelta para el algoritmo, entre todos los valores de tensión e intensidad disponibles en el ciruito. En este caso se eligen los valores de intesidad de cada rama, así como la tensión del punto neutro de la carga.

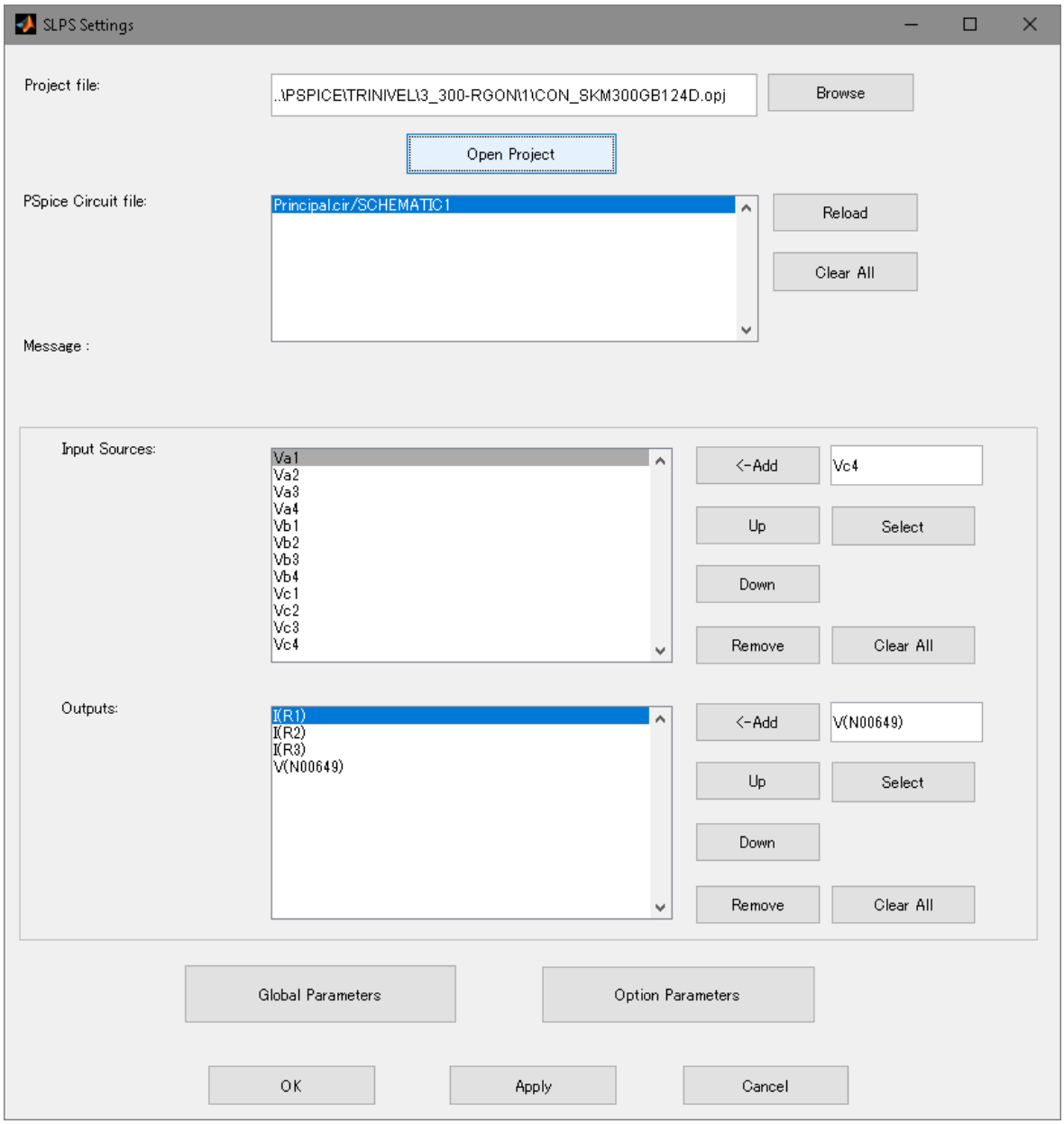

Figura 4-3: Configuración de datos del bloque SLPS.

El bloque SLPS controla también los parámetros de la simulación de PSpice. E n el botón “Option Parameters” podemos abrir la ventana (Figura 4-4) en la que podemos elegir si usar el modo de auto-convergencia de SLPS, el de PSpice (disponible desde la versión 16.6), o mantener en todo momento los parámetros que nosotros indiquemos. Se ha comprobado que las simulaciones de los circuitos en PSpice se hacen ligeramente más inestables cuando entra en juego SLPS. Esto es debido en su mayoría a que los tiempos de ejecución de los dos sistemas son discretos y no corren en la misma frecuencia, por lo que a veces PSpice tiene que recalcular estados anteriores cercanos para poder suministrar los estados que le pide Simulink.

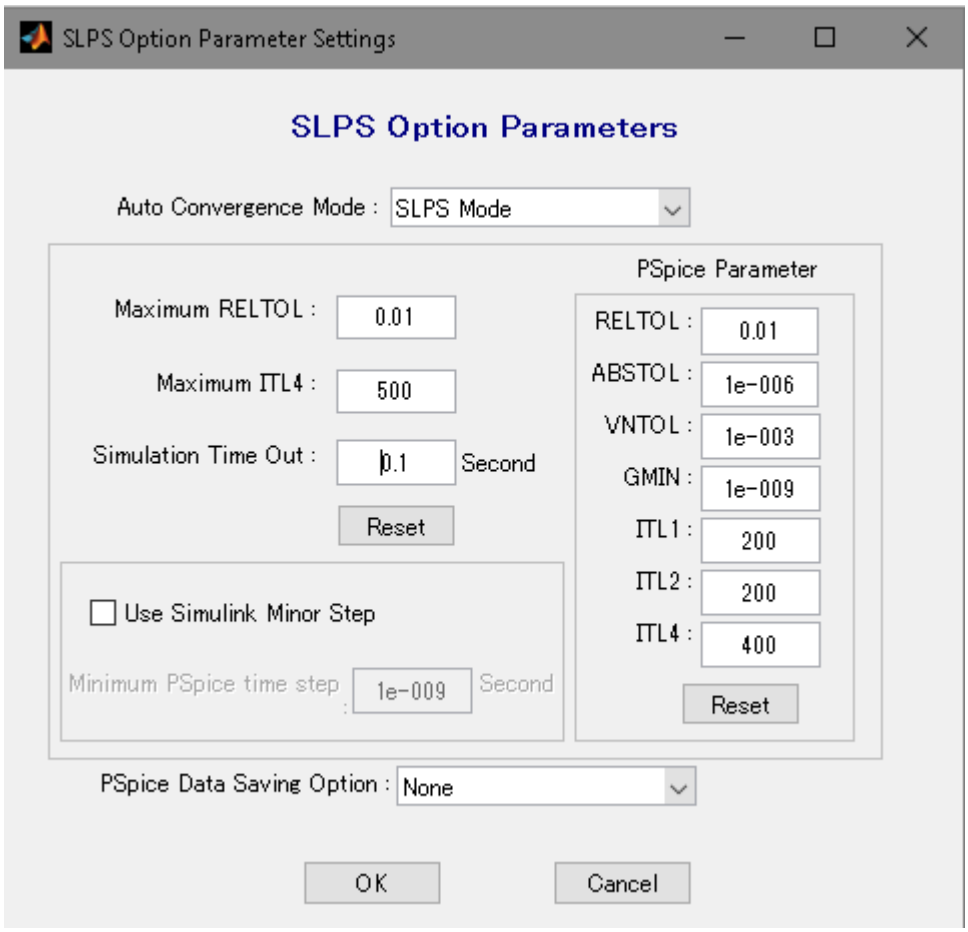

Figura 4-4: Parámetros de simulación de PSpice dentro del bloque SLPS.

En la Figura 4-5 podemos ver la coordinación del intercambio de datos teniendo en cuenta la diferencia en las frecuencias de simulación de uno y otro sistema.

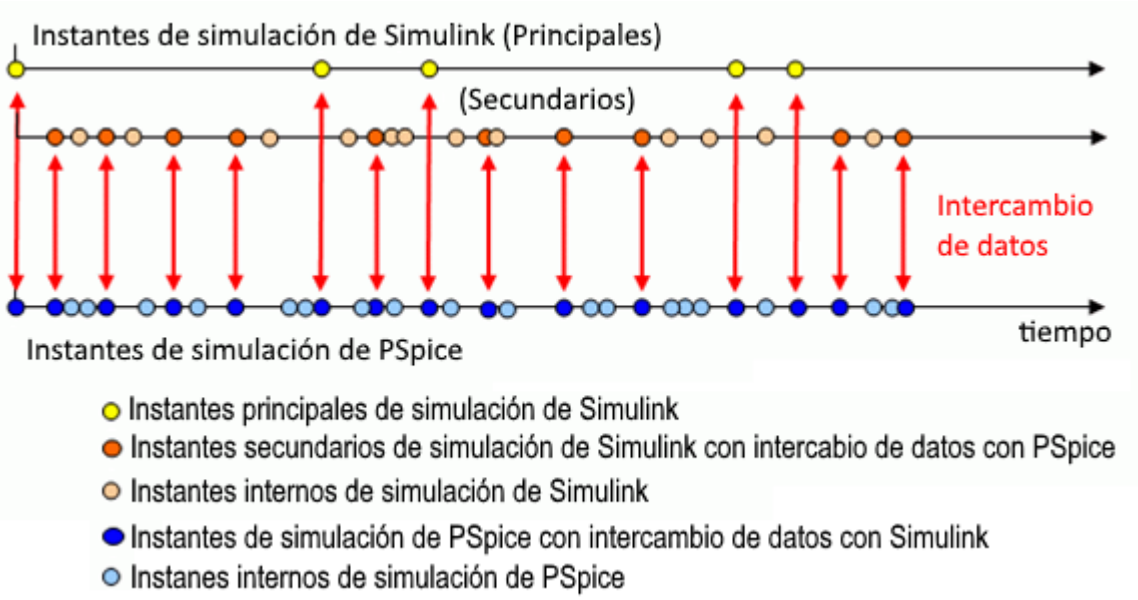

Figura 4-5: Coordinación en el intercambio de datos entre Simulink y PSpice.

Como hemos visto conceptualmente, la introducción de este bloque SLPS nos sirve para fusionar dos sitemas a priori independientes, al menos en lo que a las simulaciones en si se refiere. Pero la fusión de los dos sistemas no es automática, no basta con coger los dos sistemas existentes (ideal y real) y conectarlos con SLPS. El problema planteado en los anteriores capítulos sigue existiendo. No deja de ser necesario adaptar la señal de conmutación teórica a la tipología de señal que requiere el IGBT, o en el caso del circuito completo, la señal necesaria por el driver del IGBT.

Pero las funciones tratadas en el punto 2, las que componen el Convertidor M-P (*Reduce*, *Ajusta*, *Desplaza*, *Escala* y *Graba*) no pueden aplicarse directamente. Algunas de ellas no son necesarias, como *Graba*, puesto que ya no tenemos que generar un fichero externo, ó *Reduce* puesto que la conexión es en tiempo real y no hace falta eliminar puntos redundantes. Otras como *Ajusta* y *Escala* pueden implementarse con relativa facilidad. La atención se centra en la tarea que realizaba la función *Desplaza*, que se encargaba de retrasar las conmutaciones de los IGBT para evitar cortocircuitos y también filtraba conmutaciones más rápidas que la respuesta propia de los Drivers ó IGBT.

Evidentemente, puede complicarse el algoritmo de control, pero al mismo tiempo lo hará más potente, estable y real.

.

CONCLUSIONES.

Las simulaciones sobre sistemas ideales son un perfecto campo de trabajo para la experimentación de los algoritmos teóricos de control de sistemas. De hecho, los tiempos necesarios para completar cada simulación para los sistemas ideales son drásticamente menores que aquellos que se necesitan para simulaciones con modelos de dispositivos reales, lo cual es importante si finalmente se llevan a cabo muchas pruebas consecutivas.

Pero una vez que los algoritmos consiguen su primer objetivo, se tienen que enfrentar a la realidad de los sistemas y se hace necesario ponerlos a prueba en sistemas reales. Pero generalmente, las plataformas de simulación que se utilizan para uno y otro sistema no suelen ser iguales. Contar con una herramienta que adapte los trabajos de uno a otro sistema se convierte en una ventaja comparativa importante. Si además se consigue que la herramienta no depende del algoritmo en sí, es suficientemente flexible para su posterior adaptación a cambios de topología y se ha creado modularmente para poder cambiar o eliminar partes de la misma fácilmente, se convierte en una herramienta que tiene poca dependencia de su creador y abre la puerta a futuras mejoras y adaptaciones por otras personas que puedan necesitarla.

Del lado puramente experimental, intentar llegar de un solo paso a modelo del sistema completo real puede conllevar dificultades para depurar los problemas y comprender las desviaciones que se produzcan en los resultados. Preparar una serie de circuitos distintos en los que en cada uno se introduzca un nuevo dispositivo o una tipología más complejas hasta llegar al sistema final hace mucho más sencillo la depuración de responsabilidades de los problemas que aparezcan, así como abre el camino a ir ajuntando los parámetros de simulación para asegurar una estabilidad en la convergencia de las soluciones iterativas utilizadas para resolver los transitorios de las simulaciones.

El tiempo invertido en el estudio ordenado y simulación de todas las topologías hasta llegar al esquema final que modela el montaje que realmente se llevará a cabo en la bancada de laboratorio, se compensa con creces los resultados obtenidos. Se pretendía tener un sistema sobre el que poder adelantar acontecimientos, consiguiendo así ajustar valores que de otra manera sólo podrían ajustarse mediante el método de prueba y error.

Gracias a los datos aportados se ha conseguido un mejor nivel de dimensionamiento de los elementos secundarios, perdidas a disipar, protecciones del sistema, así como acotar con bastante precisión los niveles de tiempos muertos a partir de los cuales empiezan a aparecer problemas para este montaje en particular.

Esto significa una reducción importante de tiempo y dinero necesarios para completar el proyecto de puesta en marcha de la bancada física.

ANEXO A: CÓDIGO MATLAB COMPLETO.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                  REDUCE.m                   %
% REDUCE EL TAMAÑO DE LOS FICHEROS DE DISPARO %
%      ELIMINANDO PUNTOS NO SECESARIOS        %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('Comenzando sub-programa REDUCE');

tam=size(tout);
long=tam(1);

disp('Rama 1');
ar1(1,1:2) = disp1(1,1:2);
ar1(2,1:2) = disp1(2,1:2);
n=3;
for k=3:long-1
   ar1(n-1,1:2) = disp1(k-1,1:2);
        ar1(n,1:2) = disp1(k,1:2);
   if (disp1(k-1,2) == disp1(k+1,2))&(disp1(k,2) ~= disp1(k+1,2)) %Detecta picos
        ar1(n+1,1:2) = disp1(k,1:2);
      n=n+3;
      k=k+2;
        end;
   if disp1(k,2) ~= disp1(k-1,2)
      n=n+2;
      k=k+1;
   end;
end;
ar1(n,1:2) = disp1(long,1:2);

ar2(1,1:2) = disp2(1,1:2);
ar2(2,1:2) = disp2(2,1:2);
n=3;
for k=3:long-1
   ar2(n-1,1:2) = disp2(k-1,1:2);
        ar2(n,1:2) = disp2(k,1:2);
   if (disp2(k-1,2) == disp2(k+1,2))&(disp2(k,2) ~= disp2(k+1,2))
        ar2(n+1,1:2) = disp2(k,1:2);
      n=n+3;
      k=k+2;
        end;
   if disp2(k,2) ~= disp2(k-1,2)
      n=n+2;
      k=k+1;
   end;
end;
ar2(n,1:2) = disp2(long,1:2);

ar3(1,1:2) = disp3(1,1:2);
ar3(2,1:2) = disp3(2,1:2);
n=3;
for k=3:long-1
   ar3(n-1,1:2) = disp3(k-1,1:2);
        ar3(n,1:2) = disp3(k,1:2);
   if (disp3(k-1,2) == disp3(k+1,2))&(disp3(k,2) ~= disp3(k+1,2))
        ar3(n+1,1:2) = disp3(k,1:2);
      n=n+3;
      k=k+2;
        end;
   if disp3(k,2) ~= disp3(k-1,2)
      n=n+2;
      k=k+1;
   end;
end;
ar3(n,1:2) = disp3(long,1:2);

ar4(1,1:2) = disp4(1,1:2);
ar4(2,1:2) = disp4(2,1:2);
n=3;
for k=3:long-1
   ar4(n-1,1:2) = disp4(k-1,1:2);
        ar4(n,1:2) = disp4(k,1:2);
   if (disp4(k-1,2) == disp4(k+1,2))&(disp4(k,2) ~= disp4(k+1,2))
        ar4(n+1,1:2) = disp4(k,1:2);
```

```
      n=n+3;
      k=k+2;
        end;
   if disp4(k,2) ~= disp4(k-1,2)
      n=n+2;
      k=k+1;
   end;
end;
ar4(n,1:2) = disp4(long,1:2);

disp('Rama 2');
br1(1,1:2) = disp5(1,1:2);
br1(2,1:2) = disp5(2,1:2);
n=3;
for k=3:long-1
   br1(n-1,1:2) = disp5(k-1,1:2);
        br1(n,1:2) = disp5(k,1:2);
   if (disp5(k-1,2) == disp5(k+1,2))&(disp5(k,2) ~= disp5(k+1,2))
        br1(n+1,1:2) = disp5(k,1:2);
      n=n+3;
      k=k+2;
        end;
   if disp5(k,2) ~= disp5(k-1,2)
      n=n+2;
      k=k+1;
   end;
end;
br1(n,1:2) = disp5(long,1:2);

br2(1,1:2) = disp6(1,1:2);
br2(2,1:2) = disp6(2,1:2);
n=3;
for k=3:long-1
   br2(n-1,1:2) = disp6(k-1,1:2);
        br2(n,1:2) = disp6(k,1:2);
   if (disp6(k-1,2) == disp6(k+1,2))&(disp6(k,2) ~= disp6(k+1,2))
        br2(n+1,1:2) = disp6(k,1:2);
      n=n+3;
      k=k+2;
        end;
   if disp6(k,2) ~= disp6(k-1,2)
      n=n+2;
      k=k+1;
   end;
end;
br2(n,1:2) = disp6(long,1:2);

br3(1,1:2) = disp7(1,1:2);
br3(2,1:2) = disp7(2,1:2);
n=3;
for k=3:long-1
   br3(n-1,1:2) = disp7(k-1,1:2);
        br3(n,1:2) = disp7(k,1:2);
   if (disp7(k-1,2) == disp7(k+1,2))&(disp7(k,2) ~= disp7(k+1,2))
        br3(n+1,1:2) = disp7(k,1:2);
      n=n+3;
      k=k+2;
        end;
   if disp7(k,2) ~= disp7(k-1,2)
      n=n+2;
      k=k+1;
   end;
end;
br3(n,1:2) = disp7(long,1:2);

br4(1,1:2) = disp8(1,1:2);
br4(2,1:2) = disp8(2,1:2);
n=3;
for k=3:long-1
   br4(n-1,1:2) = disp8(k-1,1:2);
        br4(n,1:2) = disp8(k,1:2);
   if (disp8(k-1,2) == disp8(k+1,2))&(disp8(k,2) ~= disp8(k+1,2))
        br4(n+1,1:2) = disp8(k,1:2);
      n=n+3;
      k=k+2;
        end;
   if disp8(k,2) ~= disp8(k-1,2)
      n=n+2;
      k=k+1;
   end;
```

```
end;
br4(n,1:2) = disp8(long,1:2);

disp('Rama 3');
cr1(1,1:2) = disp9(1,1:2);
cr1(2,1:2) = disp9(2,1:2);
n=3;
for k=3:long-1
   cr1(n-1,1:2) = disp9(k-1,1:2);
        cr1(n,1:2) = disp9(k,1:2);
   if (disp9(k-1,2) == disp9(k+1,2))&(disp9(k,2) ~= disp9(k+1,2))
        cr1(n+1,1:2) = disp9(k,1:2);
      n=n+3;
      k=k+2;
       end;
   if disp9(k,2) ~= disp9(k-1,2)
      n=n+2;
      k=k+1;
   end;
end;
cr1(n,1:2) = disp9(long,1:2);

cr2(1,1:2) = disp10(1,1:2);
cr2(2,1:2) = disp10(2,1:2);
n=3;
for k=3:long-1
   cr2(n-1,1:2) = disp10(k-1,1:2);
        cr2(n,1:2) = disp10(k,1:2);
   if (disp10(k-1,2) == disp10(k+1,2))&(disp10(k,2) ~= disp10(k+1,2))
        cr2(n+1,1:2) = disp10(k,1:2);
      n=n+3;
      k=k+2;
       end;
   if disp10(k,2) ~= disp10(k-1,2)
      n=n+2;
      k=k+1;
   end;
end;
cr2(n,1:2) = disp10(long,1:2);

cr3(1,1:2) = disp11(1,1:2);
cr3(2,1:2) = disp11(2,1:2);
n=3;
for k=3:long-1
   cr3(n-1,1:2) = disp11(k-1,1:2);
        cr3(n,1:2) = disp11(k,1:2);
   if (disp11(k-1,2) == disp11(k+1,2))&(disp11(k,2) ~= disp11(k+1,2))
        cr3(n+1,1:2) = disp11(k,1:2);
      n=n+3;
      k=k+2;
       end;
   if disp11(k,2) ~= disp11(k-1,2)
      n=n+2;
      k=k+1;
   end;
end;
cr3(n,1:2) = disp11(long,1:2);

cr4(1,1:2) = disp12(1,1:2);
cr4(2,1:2) = disp12(2,1:2);
n=3;
for k=3:long-1
   cr4(n-1,1:2) = disp12(k-1,1:2);
        cr4(n,1:2) = disp12(k,1:2);
   if (disp12(k-1,2) == disp12(k+1,2))&(disp12(k,2) ~= disp12(k+1,2))
        cr4(n+1,1:2) = disp12(k,1:2);
      n=n+3;
      k=k+2;
       end;
   if disp12(k,2) ~= disp12(k-1,2)
      n=n+2;
      k=k+1;
   end;
end;
cr4(n,1:2) = disp12(long,1:2);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                  AJUSTA.m                    %
%    Modifica las rampas de subida y bajada    %
% para adecuarlas a un comportamiento más real %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('Comenzando sub-programa AJUSTA');

tam=size(ar1);
long=tam(1);
l=2;
a1(1,1:2)=ar1(1,1:2);
for k=2:long-1
        if (ar1(k-1,2)==ar1(k+1,2))&(ar1(k,2)~=ar1(k-1,2))
                a1(l,1)=ar1(k-1,1)+1e-6;
                a1(l,2)=ar1(k,2);
                l=l+1;
                a1(l,1:2)=ar1(k,1:2);
        else
                if ar1(k,2) ~= ar1(k-1,2)
                        a1(l,1) = ar1(k-1,1)+1e-6;
        a1(l,2) = ar1(k,2);
        else
          a1(l,1:2)=ar1(k,1:2);
      end;
        end;
l=l+1;
end;
a1(long,1:2)=ar1(long,1:2);

tam=size(ar2);
long=tam(1);
l=2;
a2(1,1:2)=ar2(1,1:2);
for k=2:long-1
        if (ar2(k-1,2)==ar2(k+1,2))&(ar2(k,2)~=ar2(k-1,2))
                a2(l,1)=ar2(k-1,1)+1e-6;
                a2(l,2)=ar2(k,2);
                l=l+1;
                a2(l,1:2)=ar2(k,1:2);
        else
                if ar2(k,2) ~= ar2(k-1,2)
                        a2(l,1) = ar2(k-1,1)+1e-6;
        a2(l,2) = ar2(k,2);
        else
          a2(l,1:2)=ar2(k,1:2);
      end;
        end;
l=l+1;
end;
a2(long,1:2)=ar2(long,1:2);


tam=size(ar3);
long=tam(1);
l=2;
a3(1,1:2)=ar3(1,1:2);
for k=2:long-1
        if (ar3(k-1,2)==ar3(k+1,2))&(ar3(k,2)~=ar3(k-1,2))
                a3(l,1)=ar3(k-1,1)+1e-6;
                a3(l,2)=ar3(k,2);
                l=l+1;
                a3(l,1:2)=ar3(k,1:2);
        else
                if ar3(k,2) ~= ar3(k-1,2)
                        a3(l,1) = ar3(k-1,1)+1e-6;
        a3(l,2) = ar3(k,2);
        else
          a3(l,1:2)=ar3(k,1:2);
      end;
        end;
l=l+1;
end;
a3(long,1:2)=ar3(long,1:2);


tam=size(ar4);
long=tam(1);
l=2;
a4(1,1:2)=ar4(1,1:2);
```

```
for k=2:long-1
       if (ar4(k-1,2)==ar4(k+1,2))&(ar4(k,2)~=ar4(k-1,2))
              a4(l,1)=ar4(k-1,1)+1e-6;
              a4(l,2)=ar4(k,2);
              l=l+1;
              a4(l,1:2)=ar4(k,1:2);
       else
              if ar4(k,2) ~= ar4(k-1,2)
                     a4(l,1) = ar4(k-1,1)+1e-6;
       a4(l,2) = ar4(k,2);
       else
         a4(l,1:2)=ar4(k,1:2);
      end;
       end;
l=l+1;
end;
a4(long,1:2)=ar4(long,1:2);


tam=size(br1);
long=tam(1);
l=2;
b1(1,1:2)=br1(1,1:2);
for k=2:long-1
       if (br1(k-1,2)==br1(k+1,2))&(br1(k,2)~=br1(k-1,2))
              b1(l,1)=br1(k-1,1)+1e-6;
              b1(l,2)=br1(k,2);
              l=l+1;
              b1(l,1:2)=br1(k,1:2);
       else
              if br1(k,2) ~= br1(k-1,2)
                     b1(l,1) = br1(k-1,1)+1e-6;
       b1(l,2) = br1(k,2);
       else
         b1(l,1:2)=br1(k,1:2);
      end;
       end;
l=l+1;
end;
b1(long,1:2)=br1(long,1:2);


tam=size(br2);
long=tam(1);
l=2;
b2(1,1:2)=br2(1,1:2);
for k=2:long-1
       if (br2(k-1,2)==br2(k+1,2))&(br2(k,2)~=br2(k-1,2))
              b2(l,1)=br2(k-1,1)+1e-6;
              b2(l,2)=br2(k,2);
              l=l+1;
              b2(l,1:2)=br2(k,1:2);
       else
              if br2(k,2) ~= br2(k-1,2)
                     b2(l,1) = br2(k-1,1)+1e-6;
       b2(l,2) = br2(k,2);
       else
         b2(l,1:2)=br2(k,1:2);
      end;
       end;
l=l+1;
end;
b2(long,1:2)=br2(long,1:2);


tam=size(br3);
long=tam(1);
l=2;
b3(1,1:2)=br3(1,1:2);
for k=2:long-1
       if (br3(k-1,2)==br3(k+1,2))&(br3(k,2)~=br3(k-1,2))
              b3(l,1)=br3(k-1,1)+1e-6;
              b3(l,2)=br3(k,2);
              l=l+1;
              b3(l,1:2)=br3(k,1:2);
       else
              if br3(k,2) ~= br3(k-1,2)
                     b3(l,1) = br3(k-1,1)+1e-6;
       b3(l,2) = br3(k,2);
       else
```

```
         b3(l,1:2)=br3(k,1:2);
      end;
       end;
l=l+1;
end;
b3(long,1:2)=br3(long,1:2);


tam=size(br4);
long=tam(1);
l=2;
b4(1,1:2)=br4(1,1:2);
for k=2:long-1
       if (br4(k-1,2)==br4(k+1,2))&(br4(k,2)~=br4(k-1,2))
              b4(l,1)=br4(k-1,1)+1e-6;
              b4(l,2)=br4(k,2);
              l=l+1;
              b4(l,1:2)=br4(k,1:2);
       else
              if br4(k,2) ~= br4(k-1,2)
                     b4(l,1) = br4(k-1,1)+1e-6;
       b4(l,2) = br4(k,2);
       else
         b4(l,1:2)=br4(k,1:2);
      end;
       end;
l=l+1;
end;
b4(long,1:2)=br4(long,1:2);


tam=size(cr1);
long=tam(1);
l=2;
c1(1,1:2)=cr1(1,1:2);
for k=2:long-1
       if (cr1(k-1,2)==cr1(k+1,2))&(cr1(k,2)~=cr1(k-1,2))
              c1(l,1)=cr1(k-1,1)+1e-6;
              c1(l,2)=cr1(k,2);
              l=l+1;
              c1(l,1:2)=cr1(k,1:2);
       else
              if cr1(k,2) ~= cr1(k-1,2)
                     c1(l,1) = cr1(k-1,1)+1e-6;
       c1(l,2) = cr1(k,2);
       else
         c1(l,1:2)=cr1(k,1:2);
      end;
       end;
l=l+1;
end;
c1(long,1:2)=cr1(long,1:2);


tam=size(cr2);
long=tam(1);
l=2;
c2(1,1:2)=cr2(1,1:2);
for k=2:long-1
       if (cr2(k-1,2)==cr2(k+1,2))&(cr2(k,2)~=cr2(k-1,2))
              c2(l,1)=cr2(k-1,1)+1e-6;
              c2(l,2)=cr2(k,2);
              l=l+1;
              c2(l,1:2)=cr2(k,1:2);
       else
              if cr2(k,2) ~= cr2(k-1,2)
                     c2(l,1) = cr2(k-1,1)+1e-6;
       c2(l,2) = cr2(k,2);
       else
         c2(l,1:2)=cr2(k,1:2);
      end;
       end;
l=l+1;
end;
c2(long,1:2)=cr2(long,1:2);


tam=size(cr3);
long=tam(1);
l=2;
```

```
c3(1,1:2)=cr3(1,1:2);
for k=2:long-1
       if (cr3(k-1,2)==cr3(k+1,2))&(cr3(k,2)~=cr3(k-1,2))
              c3(l,1)=cr3(k-1,1)+1e-6;
              c3(l,2)=cr3(k,2);
              l=l+1;
              c3(l,1:2)=cr3(k,1:2);
       else
              if cr3(k,2) ~= cr3(k-1,2)
                     c3(l,1) = cr3(k-1,1)+1e-6;
       c3(l,2) = cr3(k,2);
       else
         c3(l,1:2)=cr3(k,1:2);
      end;
       end;
l=l+1;
end;
c3(long,1:2)=cr3(long,1:2);


tam=size(cr4);
long=tam(1);
l=2;
c4(1,1:2)=cr4(1,1:2);
for k=2:long-1
       if (cr4(k-1,2)==cr4(k+1,2))&(cr4(k,2)~=cr4(k-1,2))
              c4(l,1)=cr4(k-1,1)+1e-6;
              c4(l,2)=cr4(k,2);
              l=l+1;
              c4(l,1:2)=cr4(k,1:2);
       else
              if cr4(k,2) ~= cr4(k-1,2)
                     c4(l,1) = cr4(k-1,1)+1e-6;
       c4(l,2) = cr4(k,2);
       else
         c4(l,1:2)=cr4(k,1:2);
      end;
       end;
l=l+1;
end;
c4(long,1:2)=cr4(long,1:2);
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                              DESPLAZA.m                                       %
% Con esta acción, intentamos evitar el problema de cortocircuito que provoca  %
%  el encendido de un IGBT cuando todavía no ha apagado el de la misma rama    %
%       (vease 'desplaza.m_explicaciones.txt' para mayor información)          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('Comenzando sub-programa Desplaza.m');

minramp=0.000004;    %Variable de Usuario: Tiempo mínimo de subida de una rampa

eliminados=0;
tam=size(a1);
long=tam(1);
for k=1:long-2

 %Utilidad 1: Detectar si hay conmutacion completa (0-1-0 ó 1-0-1) en t<'minramp'
   if a1(k+2,1)-a1(k,1)<=minramp
      a1(k+1,2)=a1(k,2);
      a1(k+2,2)=a1(k,2);
      a1(k+3,2)=a1(k,2);
      a3(k+1,2)=a3(k,2);
      a3(k+2,2)=a3(k,2);
      a3(k+3,2)=a3(k,2);
      eliminados=eliminados+1; %contador puntos eliminados (por conmutar rápido)
   end;

 %Utilidad 2: Detecta escalones bajada y los convierte a rampas de t='minramp'
   if (a1(k,2)==1)&(a1(k+1,2)==0)    % Detección del escalon de bajada en 'a1'
      a3(k,1)=a3(k,1)+minramp;       % Desplazar punto inicial del escalon
      a3(k+1,1)=a3(k+1,1)+minramp;   % Desplazar punto final del escalon
   end;

   if (a3(k,2)==1)&(a3(k+1,2)==0)    % Detección del escalon de bajada en 'a3'
      a1(k,1)=a1(k,1)+minramp;       % Desplazar punto inicial del escalon
      a1(k+1,1)=a1(k+1,1)+minramp;   % Desplazar punto final del escalon
   end;

end;
msg=sprintf('%d eliminaciones en a1/a3:',eliminados);
disp(msg)

eliminados=0;
tam=size(a2);
long=tam(1);
for k=1:long-2

   if a2(k+2,1)-a2(k,1)<=minramp
        a2(k+1,2)=a2(k,2);
      a2(k+2,2)=a2(k,2);
      a2(k+3,2)=a2(k,2);
      a4(k+1,2)=a4(k,2);
      a4(k+2,2)=a4(k,2);
      a4(k+3,2)=a4(k,2);
      eliminados=eliminados+1;
   end;

   if (a2(k,2)==1)&(a2(k+1,2)==0)
      a4(k,1)=a4(k,1)+minramp;
      a4(k+1,1)=a4(k+1,1)+minramp;
   end;

   if (a4(k,2)==1)&(a4(k+1,2)==0)
      a2(k,1)=a2(k,1)+minramp;
      a2(k+1,1)=a2(k+1,1)+minramp;
   end;
end;
msg=sprintf('%d eliminaciones en a2/a4:',eliminados);
disp(msg)

% Repetimos para la rama 2

eliminados=0;
tam=size(b1);
long=tam(1);
for k=1:long-2

   if b1(k+2,1)-b1(k,1)<=minramp
      b1(k+1,2)=b1(k,2);
      b1(k+2,2)=b1(k,2);
```

```
      b1(k+3,2)=b1(k,2);
      b3(k+1,2)=b3(k,2);
      b3(k+2,2)=b3(k,2);
      b3(k+3,2)=b3(k,2);
      eliminados=eliminados+1;
   end;

   if (b1(k,2)==1)&(b1(k+1,2)==0)
      b3(k,1)=b3(k,1)+minramp;
      b3(k+1,1)=b3(k+1,1)+minramp;
   end;

   if (b3(k,2)==1)&(b3(k+1,2)==0)
      b1(k,1)=b1(k,1)+minramp;
      b1(k+1,1)=b1(k+1,1)+minramp;
   end;

end;
msg=sprintf('%d eliminaciones en b1/b3:',eliminados);
disp(msg)

eliminados=0;
tam=size(b2);
long=tam(1);
for k=1:long-2

   if b2(k+2,1)-b2(k,1)<=minramp
               b2(k+1,2)=b2(k,2);
      b2(k+2,2)=b2(k,2);
      b2(k+3,2)=b2(k,2);
      b4(k+1,2)=b4(k,2);
      b4(k+2,2)=b4(k,2);
      b4(k+3,2)=b4(k,2);
      eliminados=eliminados+1;
   end;

   if (b2(k,2)==1)&(b2(k+1,2)==0)
      b4(k,1)=b4(k,1)+minramp;
      b4(k+1,1)=b4(k+1,1)+minramp;
   end;

   if (b4(k,2)==1)&(b4(k+1,2)==0)
      b2(k,1)=b2(k,1)+minramp;
      b2(k+1,1)=b2(k+1,1)+minramp;
   end;
end;
msg=sprintf('%d eliminaciones en b2/b4:',eliminados);
disp(msg)

% Repetimos para la rama 3


eliminados=0;
tam=size(c1);
long=tam(1);
for k=1:long-2

   if c1(k+2,1)-c1(k,1)<=minramp
      c1(k+1,2)=c1(k,2);
      c1(k+2,2)=c1(k,2);
      c1(k+3,2)=c1(k,2);
      c3(k+1,2)=c3(k,2);
      c3(k+2,2)=c3(k,2);
      c3(k+3,2)=c3(k,2);
      eliminados=eliminados+1;
   end;

   if (c1(k,2)==1)&(c1(k+1,2)==0)
      c3(k,1)=c3(k,1)+minramp;
      c3(k+1,1)=c3(k+1,1)+minramp;
   end;

   if (c3(k,2)==1)&(c3(k+1,2)==0)
      c1(k,1)=c1(k,1)+minramp;
      c1(k+1,1)=c1(k+1,1)+minramp;
   end;

end;
msg=sprintf('%d eliminaciones en c1/c3:',eliminados);
disp(msg)
```

```
eliminados=0;
tam=size(c2);
long=tam(1);
for k=1:long-2

   if c2(k+2,1)-c2(k,1)<=minramp
               c2(k+1,2)=c2(k,2);
      c2(k+2,2)=c2(k,2);
      c2(k+3,2)=c2(k,2);
      c4(k+1,2)=c4(k,2);
      c4(k+2,2)=c4(k,2);
      c4(k+3,2)=c4(k,2);
      eliminados=eliminados+1;
   end;

   if (c2(k,2)==1)&(c2(k+1,2)==0)
      c4(k,1)=c4(k,1)+minramp;
      c4(k+1,1)=c4(k+1,1)+minramp;
   end;

   if (c4(k,2)==1)&(c4(k+1,2)==0)
      c2(k,1)=c2(k,1)+minramp;
      c2(k+1,1)=c2(k+1,1)+minramp;
   end;
end;
msg=sprintf('%d eliminaciones en c2/c4:',eliminados);
disp(msg)
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%                  ESCALA.m                       %
%  Cambia la escala de tensiones a los niveles    %
%     requeridos por el driver y el IGBT          %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('Comenzando sub-programa ESCALA');

tam=size(a1);
long=tam(1);
for k=1:long
   if a1(k,2)==1
      a1(k,2)=15;
   else
      a1(k,2)=-8;
   end;
end;

tam=size(a2);
long=tam(1);
for k=1:long
   if a2(k,2)==1
      a2(k,2)=15;
   else
      a2(k,2)=-8;
   end;
end;

tam=size(a3);
long=tam(1);
for k=1:long
   if a3(k,2)==1
      a3(k,2)=15;
   else
      a3(k,2)=-8;
   end;
end;

tam=size(a4);
long=tam(1);
for k=1:long
   if a4(k,2)==1
      a4(k,2)=15;
   else
      a4(k,2)=-8;
   end;
end;

tam=size(b1);
long=tam(1);
for k=1:long
   if b1(k,2)==1
      b1(k,2)=15;
   else
      b1(k,2)=-8;
   end;
end;

tam=size(b2);
long=tam(1);
for k=1:long
   if b2(k,2)==1
      b2(k,2)=15;
   else
      b2(k,2)=-8;
   end;
end;

tam=size(b3);
long=tam(1);
for k=1:long
   if b3(k,2)==1
      b3(k,2)=15;
   else
      b3(k,2)=-8;
   end;
end;

tam=size(b4);
long=tam(1);
```

```
for k=1:long
   if b4(k,2)==1
      b4(k,2)=15;
   else
      b4(k,2)=-8;
   end;
end;

tam=size(c1);
long=tam(1);
for k=1:long
   if c1(k,2)==1
      c1(k,2)=15;
   else
      c1(k,2)=-8;
   end;
end;

tam=size(c2);
long=tam(1);
for k=1:long
   if c2(k,2)==1
      c2(k,2)=15;
   else
      c2(k,2)=-8;
   end;
end;

tam=size(c3);
long=tam(1);
for k=1:long
   if c3(k,2)==1
      c3(k,2)=15;
   else
      c3(k,2)=-8;
   end;
end;

tam=size(c4);
long=tam(1);
for k=1:long
   if c4(k,2)==1
      c4(k,2)=15;
   else
      c4(k,2)=-8;
   end;
end;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% GRABA.m / Escribe el fichero de disparos para PSPICE %
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

disp('Comenzando sub-programa GRABA');
fid = fopen('disparos.stl','w');                  % Cabecera del fichero PSPICE
fprintf(fid,';!Stimulus Get\n');
fprintf(fid,';! a1 Analog a2 Analog a3 Analog a4 Analog b1 Analog b2 Analog b3 Analog b4
Analog c1 Analog c2 Analog c3 Analog c4 Analog\n');
fprintf(fid,';!Ok\n');
fprintf(fid,';!Plot Axis_Settings\n');
fprintf(fid,';!Xrange 0s 20ms\n');
fprintf(fid,';!Yrange -1 6\n');
fprintf(fid,';!AutoUniverse\n');
fprintf(fid,';!XminRes 1ns\n');
fprintf(fid,';!YminRes 1\n');
fprintf(fid,';!Ok\n');

fprintf(fid,'.STIMULUS a1 PWL\n');                % Escritura de los datos
fprintf(fid,'+ TIME_SCALE_FACTOR = 1\n');
fprintf(fid,'+ VALUE_SCALE_FACTOR = 1\n');
fprintf(fid,'+ ( %6.9f , %1.1f ) \n',a1.');

fprintf(fid,'.STIMULUS a2 PWL\n');
fprintf(fid,'+ TIME_SCALE_FACTOR = 1\n');
fprintf(fid,'+ VALUE_SCALE_FACTOR = 1\n');
fprintf(fid,'+ ( %6.9f , %1.1f ) \n',a2.');

fprintf(fid,'.STIMULUS a3 PWL\n');
fprintf(fid,'+ TIME_SCALE_FACTOR = 1\n');
fprintf(fid,'+ VALUE_SCALE_FACTOR = 1\n');
fprintf(fid,'+ ( %6.9f , %1.1f ) \n',a3.');

fprintf(fid,'.STIMULUS a4 PWL\n');
fprintf(fid,'+ TIME_SCALE_FACTOR = 1\n');
fprintf(fid,'+ VALUE_SCALE_FACTOR = 1\n');
fprintf(fid,'+ ( %6.9f , %1.1f ) \n',a4.');

fprintf(fid,'.STIMULUS b1 PWL\n');
fprintf(fid,'+ TIME_SCALE_FACTOR = 1\n');
fprintf(fid,'+ VALUE_SCALE_FACTOR = 1\n');
fprintf(fid,'+ ( %6.9f , %1.1f ) \n',b1.');

fprintf(fid,'.STIMULUS b2 PWL\n');
fprintf(fid,'+ TIME_SCALE_FACTOR = 1\n');
fprintf(fid,'+ VALUE_SCALE_FACTOR = 1\n');
fprintf(fid,'+ ( %6.9f , %1.1f ) \n',b2.');

fprintf(fid,'.STIMULUS b3 PWL\n');
fprintf(fid,'+ TIME_SCALE_FACTOR = 1\n');
fprintf(fid,'+ VALUE_SCALE_FACTOR = 1\n');
fprintf(fid,'+ ( %6.9f , %1.1f ) \n',b3.');

fprintf(fid,'.STIMULUS b4 PWL\n');
fprintf(fid,'+ TIME_SCALE_FACTOR = 1\n');
fprintf(fid,'+ VALUE_SCALE_FACTOR = 1\n');
fprintf(fid,'+ ( %6.9f , %1.1f ) \n',b4.');

fprintf(fid,'.STIMULUS c1 PWL\n');
fprintf(fid,'+ TIME_SCALE_FACTOR = 1\n');
fprintf(fid,'+ VALUE_SCALE_FACTOR = 1\n');
fprintf(fid,'+ ( %6.9f , %1.1f ) \n',c1.');

fprintf(fid,'.STIMULUS c2 PWL\n');
fprintf(fid,'+ TIME_SCALE_FACTOR = 1\n');
fprintf(fid,'+ VALUE_SCALE_FACTOR = 1\n');
fprintf(fid,'+ ( %6.9f , %1.1f ) \n',c2.');

fprintf(fid,'.STIMULUS c3 PWL\n');
fprintf(fid,'+ TIME_SCALE_FACTOR = 1\n');
fprintf(fid,'+ VALUE_SCALE_FACTOR = 1\n');
fprintf(fid,'+ ( %6.9f , %1.1f ) \n',c3.');

fprintf(fid,'.STIMULUS c4 PWL\n');
fprintf(fid,'+ TIME_SCALE_FACTOR = 1\n');
fprintf(fid,'+ VALUE_SCALE_FACTOR = 1\n');
fprintf(fid,'+ ( %6.9f , %1.1f ) \n',c4.');

fclose(fid);
```

REFERENCIAS

[1] R. Portillo Guisado, Modelado y Simulación de Inversores Multinivel, 2002.

[2] F. Cortés Ponce, Desarrollo de Técnicas de Control y su Implementación en DSP para Inversores Multinivel, 2002.

[3] F. J. Benítez Camacho, Diseño de un prototipo de inversor trinivel e implementación del hardware de control, 2002.

[4] M. Á. Martín Prats, Nuevas técnicas de modulación vectorial para conversores electrónicos de potencia multinivel, 2003.

Índice de Conceptos

DC-AC 11
disparos 12
driver 20
Estabilidad 31
NPC 13, 14
PWM 12, 13, 60
tiempos muertos 20

Glosario

Control en bucle abierto y cerrado

Convertidor M-P

Disparo

Distorsión armónica

Estados

NPC

PWM

Bibliografía

- Leopoldo García Franquelo. "Apuntes de electrónica de potencia". Departamento de ingeniería electrónica, Universidad de Sevilla.

- Ned Mohan, Tore M. Undeland, William P. Robbins, "Power Electronics: Converter, Applications, and Desing".

- R. Teodorescu, F. Blaabjerg, J.K Pedersen, E. Cengelci, S.U. Sulistijo, B.O. Woo, P. Enjeti. "Multilevel Converters – A Survey".

- Hoy L. Liu, Nam S. Choi and Gyu H. Cho. "DSP based space vector PWM for three-level inverter with DC-LINK voltage balancing".

- Nikola Celanovic, Dushan Boroyevich. "A comprehensive study of neutralpoint voltaje balancing problem in three-level neutral-point-clamped voltaje source PWM inverters". IEEE transactions on power electronics, Vol. 15. Nº 2, Marzo 2000.

- Joachim Holtz, Fellow, IEEE. "Pulsewidth Modulation Techniques for Controlled AC Drives".

- Jousep Pou, Rafael Pindado, "Nueva técnica de modulación de conversores DCI de tres niveles". SAAEI'2000.

- Nikola Celanovic, Dushan Boroyevich. "A comprehensive study of neutralpoint voltaje balancing problem in three-level neutral-point-clamped voltaje source PWM inverters". IEEE transactions on power electronics, Vol. 15. Nº 2, Marzo 2000.

- J. Bordonau, M. Cosan, D. Borojevic, H. Mao, F.C. Lee. " A state-space model for the comprehensive dynamic análisis of three-level voltaje-source inverters". IEEE'1997.

- Akira Nabae, Isao Takahashi, Hirofumi Akagi. "A new neutral-point-clamped PWM inverter. IEEE transactions on industry applications, Vol. IA-17, Nº. 5, Septiembre/Octubre 1981.

- Jih-Sheg Lai, Fang Zheng Peng. "Multilevel converters. A new breed of power converters". IEEE transactions on industry applications, Vol. 32, Nº. 3, Mayo/Junio 1996.

- Geoff Walker, Gerard Ledwich. "Bandwith considerations for multilevel converters". IEEE transactions on power electronics, Vol. 14, Nº. 1, Enero 1999.

- Martin Veenstra, Prof. Alfred Rufer. "PWM-Control of multi-level voltagesource inverters". IEEE'2000.

- Leon M. Tolbert, Thomas G. Habetler. "Novel Multileves inverter carrier-based PWM method". IEEE transactions on industry applications, Vol. 35, Nº. 5 Septiembre/Octubre 1999.