Departamento de Tecnología Electrónica

TRABAJO DE FIN DE GRADO

CONTROL DIGITAL DE SISTEMAS DE CONVERSIÓN DE ENERGÍA

AUTOR: JAVIER CALERO GÓMEZ

TUTORA: CELIA LÓPEZ ONGIL

TÍTULO: CONTROL DIGITAL DE SISTEMAS DE CONVERSIÓN DE ENERGÍA

AUTOR: JAVIER CALERO GÓMEZ

TUTORA: CELIA LÓPEZ ONGIL

EL TRIBUNAL

Presidente: Francisco José Rodríguez Urbano

Vocal: Luis Alfonso Entrena Arrontes

Secretario: Miguel Ángel Montaner López

Realizado el acto de defensa y lectura del Trabajo de Fin de Grado el día 12 de JULIO de 2013 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

AGRADECIMIENTOS

No hay nada mejor en esta vida, que tener mucha gente a la que agradecer cosas públicamente. Por ello me siento feliz de poder hacerlo, ya que el número de personas a las que tienes que agradecer es directamente proporcional al número de personas que te han apoyado.

Por eso me gustaría agradecer a estas personas de un modo personal:

Papá y mamá: Sois los que me disteis la vida, es obvio que sin vosotros no hubiera podido hacer nada. Vosotros me lo enseñasteis todo, y es algo de lo que sentirme orgulloso. Espero ahora que vosotros podáis sentiros tan orgullosos de mí, como yo de vosotros. Gracias.

Jorge: contigo he compartido buenos y malos momentos, discusiones, juegos… simplemente gracias.

Prihers: Teníais que estar aquí, mis amigos de toda la vida. Siempre hemos caminado juntos y aún seguimos haciéndolo. En especial quiero mandar un saludo a Porku allá donde esté. Gracias.

Colegas de la uni: Vosotros habéis tenido que aguantarme durante los años en clase, y sólo por eso tengo mucho que agradeceros. Ha sido un placer compartir tantas y tantas horas con vosotros. Gracias.

Como no, tengo que agradecer a mí tutora de proyecto, Celia López Ongil, el apoyo prestado en todo momento, también a Pablo Zumel, Antonio Lázaro, Mario García Valderas y todos los que en todo momento me han prestado ayuda para llevar a buen puerto este trabajo. Gracias por el apoyo y ayuda prestada.

Contenido

ÍNDICE DE FIGURAS

RESUMEN

Es evidente que el control digital en los sistemas electrónicos de potencia está imponiéndose al control analógico. Para ello el uso de herramientas de diseño de circuitos en alto nivel se hace muy necesario.

Sin embargo estas herramientas tienen ciertas taras en las bibliotecas de componentes disponibles que, de solucionarlas, las harán mucho más potentes.

El presente Trabajo de Fin de Grado se dedica al diseño de un divisor de números en formato de complemento A2 que sea sintetizable, bloque del que carecen las actuales bibliotecas de las herramientas de diseño de circuitos.

En este documento se detalla el diseño de la arquitectura hardware digital, así como el desarrollo del algoritmo seleccionado para la división de los números. Seguidamente se expone y presenta la descripción de *hardware* (VHDL) que implementa dicho algoritmo. El hardware obtenido de la síntesis del mencionado código se ha prototipado en un dispositivo programable de alta capacidad, como prueba de concepto, integrada en una placa provista de varios *displays* de 7 segmentos para la comprobación del resultado, así como así como botones e interruptores para la interfaz de usuario durante las pruebas. Finalmente, como aplicación industrial en el marco del Grado en Ingeniería Electrónica Industrial y Automática, se plantea y aplica la utilización de este bloque funcional en un lazo de control de un convertidor reductor para dotarle de una realimentación "Feed-Forward". Esta aplicación pretende mostrar la mejora de prestaciones en el control de sistemas electrónicos de potencia cuando se incluyen elementos digitales, además de demostrar la viabilidad de una metodología de trabajo conjunta analógica-digital, propuesta y desarrollada en el departamento de Tecnología Electrónica, con el uso de herramientas CAD comerciales para el diseño mixto analógico-digital.

PALABRAS CLAVE: divisor, algoritmo, VHDL, FPGA, simulación, PSIM, ModelSim, realimentación Feed-Forward.

ABSTRACT

It is evident that the digital control in the electronic systems of power is imposing on him to the analogical control. For it the use of tools of design of circuits in high level becomes very necessary.

Nevertheless these tools have certain tares in the libraries of available components that, of solving them, will make them much more powerful.

The present Work of End of Degree devotes itself to the design of a divisor of numbers in format of complement A2 that is possible to elaborate, block which there lack the current libraries of the tools of design of circuits.

In this document the design of the architecture details digital hardware, as well as the development of the algorithm selected for the division of the numbers. Immediately afterwards one exposes and presents the description of hardware (VHDL) that implements the above mentioned algorithm. The hardware obtained of the synthesis of the mentioned code has done a prototype in a programmable device of high capacity, in proof of concept, integrated a plate provided with several displays of 7 segments for the checking of the result, as well as as well as buttons and switches for the user's interface during the tests. Finally, like industrial application in the frame of the Degree in Electronic Industrial and Automatic Engineering, raises and applies to itself the utilization of this functional block in a bow of control of a convertor reducer to provide him with a feedback "Feed-Forward". This application tries to show the improvement of presentations in the control of electronic systems of power when digital elements are included, beside demonstrating the viability of a methodology of work combines analogical - digital, proposed and developed in the department of Electronic Technology, with the use of tools commercial CAD for the mixed analogical - digital design.

KEY WORDS: divisor, algorithm, VHDL, FPGA, simulation, PSIM, ModelSim, feedback Feed-Forward.

INTRODUCCIÓN

En la actualidad la conversión de energía es un tema tan extendido que sus aplicaciones se pueden ver en el ámbito cotidiano: un cargador de móvil, un ordenador, cualquier sistema de iluminación…. Todos llevan convertidores de energía para adaptar los 230 en alterna que obtenemos en las tomas de corriente a la necesidad de cada uno de los aparatos.

Para que estos equipos operen correctamente y puedan reaccionar ante posibles variaciones de tensión de la red se requieren sistemas de control integrados que sean capaces de responder con la suficiente rapidez para proteger los convertidores y los equipos a los que vayan conectadas dichos sistemas de control.

Este sistema de control consiste en el sensado de ciertos parámetros dentro del conversor de potencia y que interactuando con un regulador y un modulador se adapta el ciclo de trabajo para corregir las posibles variaciones de la fuente.

Típicamente los sistemas de control se han construido con componentes electrónicos analógicos; sin embargo el abaratamiento de los costes de los componentes para la electrónica digital, como microprocesadores o conversores analógicos-digitales, así como el desarrollo de la misma en términos de capacidad de procesamiento y de mejora de la velocidad y consumo permiten que se esté imponiendo la digitalización de los sistemas de control.

Uno de los avances en éste tipo de circuitos, del que nos serviremos en la realización de todo el Trabajo de Fin de Grado, es la generalización del uso de lenguajes de descripción de *hardware*.

Debido a la complejidad cada vez mayor de los circuitos digitales se ha hecho necesaria una nueva forma de diseñar circuitos. En concreto el uso de los lenguajes de descripción de *hardware* junto con las herramientas comerciales de síntesis lógica constituyen una potentísima herramienta para el prototipado rápido en dispositivos programables de alta capacidad y bajo coste, capaces emular el comportamiento de un circuito integrado (ASIC), con las grandísimas ventajas de la simplificación en el proceso de prototipado del circuito y de las pruebas para comprobar el funcionamiento. Pero, sobre todo, los lenguajes de descripción de *hardware* permiten el diseño de circuitos digitales de muy alta complejidad de forma eficiente, rápida y barata, desde un nivel de abstracción muy alto (algorítmico).

Sin embargo no todo es tan sencillo. Hay ciertos procesos que son irrealizables en la descripción de *hardware*; algunas instrucciones concebidas en el nivel algorítimico no

son sintetizables ya que un circuito no puede ser dinámico: el circuito no puede cambiar según cambien las condiciones.

Por lo tanto, el conseguir aprender a usar de forma eficiente este tipo de lenguajes de descripción de *hardware* es muy beneficioso para futuros diseños de bloques digitales. Esto supone un proceso largo y difícil de aprendizaje de diseño y desarrollo de circuitos digitales, distinguiéndolos del desarrollo de programas *software* y del diseño de circuitos analógicos, pero que produce resultados muy provechosos.

OBJETIVO

El objetivo del Trabajo de Fin de Grado será resolver uno de los principales retos del procesamiento digital *hardware*: la división de dos números enteros.

Éste objetivo surge como trabajo futuro de la realización de un Proyecto de Fin de Carrera previo titulado: “Diseño y Validación del Control Digital de un Inversor de Potencia en Ejes de Referencia Síncronos Conectado a Red” donde se planteó la implementación del código para la realización del control de un inversor trifásico y su posterior simulación. Sin embargo no se pudo llevar a cabo su síntesis ya que varias de las ecuaciones que debían implementarse contenían funciones de síntesis no inmediata en las herramientas CAD comerciales actuales.

FASES DE DESARROLLO DEL TRABAJO

A continuación paso a detallar las fases en las que se dividió el Trabajo para llevarlo a cabo.

FASE DE PLANIFICACIÓN.

Se llevó a cabo el estudio del Proyecto de Fin de Carrera mencionado en el apartado anterior para ver cuáles eran exactamente las necesidades que debía cubrir el presente Trabajo de Fin de Grado.

Posteriormente se pasó a la búsqueda de un método para dividir números binarios naturales utilizando operaciones aritméticas sintetizables: sumas, restas y/o multiplicaciones.

En este apartado debo también incluir la adquisición de los conocimientos de descripción en lenguaje VHDL necesarios para la implementación del circuito. Éstos los adquirí cursando la asignatura "Diseño de Circuitos Integrados", optativa de 4º curso del grado estudiado (Electrónica y Automática).

FASE DE DESARROLLO.

El desarrollo ha consistido en el diseño de la arquitectura que implementará el algoritmo de división, hallado en la fase de planificación, y en la descripción en lenguaje VHDL de dicha arquitectura.

Para ello se ha utilizado la herramienta de la empresa Xilinx "*Ise Design Suite*", disponible tanto en las aulas informáticas de la universidad como en la página Web de la empresa para su descarga gratuita, por lo que pude trabajar tanto en la universidad como en mi ordenador personal.

A medida que se iba realizando la descripción del diseño la propia herramienta te permite realizar simulaciones para la comprobación del código.

A continuación, una vez obtenido un código definitivo y comprobado funcionalmente en las simulaciones, se realizó una comprobación de prototipo. En este caso se ha elegido una placa FPGA de altas prestaciones con interruptores y botones para las entradas y 4 *displays* de 7 segmentos para mostrar las salidas de las operaciones. Obviamente es necesario añadir lógica adicional al bloque divisor para realizar una interfaz que permita la introducción de operandos y la visualización del resultado.

Por último, pero no menos importante, se ha incluido el bloque divisor sintetizable realizado en una etapa de control de un convertidor de electrónica de potencia, de tipo *Feed-Forward*. Esta inclusión se ha enmarcado en una metodología de diseño

mixto, planteada y desarrollada en el departamento de Tecnología Electrónica, que implica la utilización de cosimulación de bloques analógicos y digitales, entre las herramientas CAD comerciales *PSIM®* y *ModelSim®*.

FASE DE DOCUMENTACIÓN.

En la última fase del proyecto se pasó a la realización de la presente memoria así como de la presentación.

MEDIOS EMPLEADOS

Para la consecución de los objetivos del Trabajo han sido utilizados unas veces los ordenadores de las aulas informática y otras mi ordenador personal, según dónde me encontrara. En cualquier dispositivo el programa utilizado ha sido el "*Ise Design Suite*" de Xilinx en distintas versiones.

Además se han empleado los laboratorios del departamento para hacer las comprobaciones en la FPGA modelo "Spartan 3 XC3S200-FT256" ayudándome de los ordenadores con puertos paralelos que se requieren para la conexión de la tarjeta con el ordenador y así poder cargar el programa, además de la herramienta "Impact", también de Xilinx.

Para la cosimulación del diseño mixto se empleó el simulador analógico *PSIM®* versión 9.2, con los módulos *ModCoupler* y *SmartCtrl* para implementar el circuito donde va integrado el divisor y *ModelSim®* de *Mentor Graphics®* para trabajar la parte de VHDL del mismo. Todos estos programas están instalados en el laboratorio del grupo de Sistemas Electrónicos de Potencia.

ESTRUCTURA DE LA MEMORIA

El presente documento se ha estructurado de la misma forma en que se fue desarrollando el proyecto.

En primer lugar se presenta la búsqueda del algoritmo a utilizar en el divisor, así como el esquema del circuito.

Más adelante se presenta la síntesis del circuito digital en VHDL, explicando pormenorizadamente todos los elementos del mismo.

Tras esto, se atiende a las pruebas de prototipado realizadas, así como el desarrollo de la interfaz necesaria para las mismas.

Seguidamente se ha incluido un ejemplo práctico donde se incluyen las imágenes de la simulación de conjunto del circuito del divisor con la interfaz para la FPGA.

Por último se desarrolla una aplicación sugerida por el Grupo de Sistemas Electrónicos de Potencia donde se introduce el divisor digital en el lazo de control de un convertidor *Buck.*

BÚSQUEDA DEL ALGORITMO DEL DIVISOR

El primer paso a la hora de desarrollar el trabajo de Fin de Grado fue investigar qué funciones descritas y desarrolladas en el Proyecto de Fin de Carrera realizado por Beatriz Brogeras García en2011 titulado "Diseño y Validación del Control Digital de un Inversor de Potencia en Ejes de Referencia Síncronos Conectado a Red" presentaban problemas de síntesis.

Al acudir a la memoria de dicho proyecto encuentro descrito en el apartado de "Líneas de trabajo futuro" la necesidad de obtener un divisor capaz de integrarse en el sistema de control desarrollado por Beatriz que pueda ser sintetizable.

Quiero hacer un pequeño inciso donde voy a explicar para qué va a servir el divisor en esta aplicación el concreto, aunque puede implementarse en otras aplicaciones.

En el control de los inversores trifásicos utilizados en las cargas de baterías de los coches eléctricos o en los sistemas de alimentación ininterrumpida (S.A.I.) es esencial realizar divisiones de dos valores variantes a lo largo del tiempo. El método de control, seleccionado por el grupo de Sistemas Electrónicos de Potencia y desarrollado en el citado proyecto fin de carrera, tiene en cuenta las 3 fases del inversor simultáneamente tendiendo a atender a cada fase por separado dejando de lado posibles desequilibrios entre ellas.

El control de las fases utilizan las transformadas de Park y su inversa, la transformada de Clark, con las que se consigue reflejar los fasores de la tensión de las tres fases en un vector referenciado en unos ejes que giran con una velocidad angular igual a la pulsación de la red, con lo cual se obtiene la posibilidad de emplear las técnicas de control de los convertidores de continua a continua en los inversores trifásicos.

Dichas transformaciones consisten en la multiplicación matricial de ciertos parámetros del inversor por una serie de constantes. Para realizar estas operaciones se requieren tanto productos como divisiones de los parámetros del convertidor que van variando en el tiempo.

Una vez que estuvo claro el problema pasé a buscar un algoritmo con el que pudiese realizar la división de dos números en binario mediante un circuito digital. A partir de ahí iría aplicando diferentes mejoras hasta obtener un resultado eficiente, barato y preciso.

Para ello busqué tanto en libros de la biblioteca como en documentación publicada en Internet hasta que di con ello.

A pesar de las descripciones de algunos libros, en YouTube encontré videos tutoriales explicativos sobre cómo hacer la división de números binarios. Concretamente http://www.youtube.com/watch?v=YBhBJSyaGTk fue el que seguí para obtener el algoritmo que paso a detallar:

1) En primer lugar se comprueba el número de cifras de bits del divisor y se atiende al número de cifras del dividendo. Para que se pueda llevar a cabo la operación se deben contar las cifras de cada uno de los números (dividendo y divisor), las primeras cifras significativas deben estar alineadas una bajo la otra, y seguidamente se añaden los '0' necesarios a la derecha del divisor hasta que tenga el mismo tamaño que el dividendo. Una vez hecho esto paso a restar las cifras del dividendo menos el divisor.

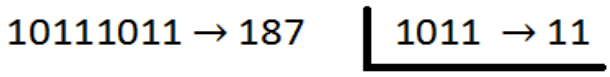

FIGURA 1. ALGORITMO DE DIVISIÓN

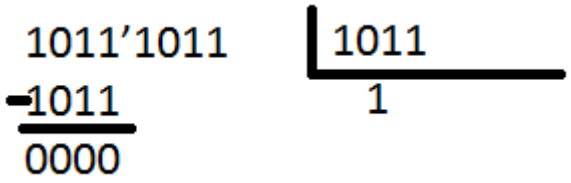

FIGURA 2. ALGORITMO DE DIVISIÓN

2) Una vez hecho esto puedo tener dos opciones: que el acarreo de la resta sea 1 o que sea 0.

   Si fuese 1 significa que el divisor extendido es mayor que el dividendo, por lo tanto se debe desplazar el divisor una posición a la derecha de forma que se haga coincidir el final del mismo con la siguiente cifra del dividendo y anotar un '0' en el cociente.

   Sin embargo si fuese '0' significa que la resta de las dos cifras ha dado un número positivo y se ha obtenido un resto intermedio, por lo tanto ahora los acontecimientos transcurren de distinta forma.

   - En primer lugar se deba anotar un '1' en el cociente en vez de un '0'.

   - Se debe sustituir en el dividendo los dígitos originales por las cifras obtenidas en el resto.

   - Se debe desplazar el divisor un lugar a la derecha para hacer coincidir el final del mismo con la siguiente cifra del dividendo.

Se repite esto hasta que el final del divisor coincide con el del dividendo, siendo esta vez el resto el resto final de la división.

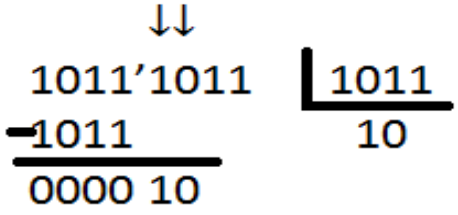

FIGURA 3. ALGORITMO DE DIVISIÓN

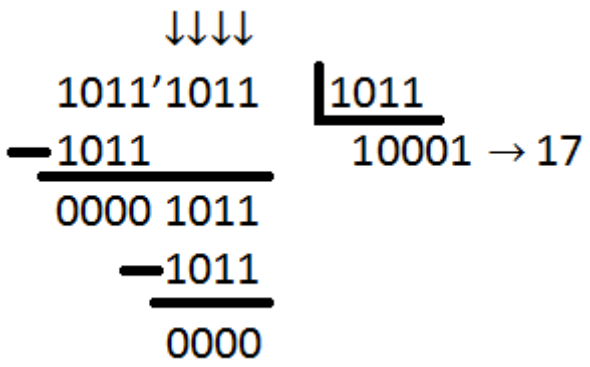

FIGURA 4. ALGORITMO DE DIVISIÓN

Por lo tanto he sido capaz de obtener el cociente de una división de números binarios mediante un procedimiento que emplea solamente restas y desplazamientos de cifras; elementos que, esta vez sí, son sintetizables por el compilador.

ESQUEMA DEL CIRCUITO

El siguiente paso fue el diseño de la arquitectura del circuito digital que implementa el algoritmo de división seleccionado. Esta arquitectura se realizó en el nivel de transferencia de registros, que incluye bloques sumadores, restadores, comparadores, etc. así como elementos de memoria donde se almacenan temporalmente los datos procesados. Además, es frecuente la inclusión de un bloque de control (máquina de estados, decodificador de instrucciones, etc.) que gestione la ruta de los datos y su procesamiento.

En primer lugar se establecieron cuatro registros; dos donde se van a insertar tanto numerador como denominador, aparte otro adicional donde se van a insertar los '1' y los '0', dependiendo de los valores que se vayan obteniendo en la división y un cuarto donde se va a almacenar el resultado de cada sustracción para que sea, o no, sustituido por el numerador, según el acarreo. Ya que el divisor va a tener que moverse una posición a la derecha hasta completar la división el registro del mismo debe ser un registro con desplazamiento.

De igual forma, ya que es necesario el almacenamiento del acarreo para tomar las acciones pertinentes, se estableció que se debía tener 3 señales adicionales que ejecutaran los cambios según el acarreo:

-Una que establezca si se anota un '1' o un '0' en el cociente.
-Otra que indique si se debe cambiar el numerador por los valores obtenidos en cada sustracción.
-Una última que indique se debe desplazar el divisor.

Obviamente también es necesaria una máquina de estados ya que no deben ocurrir todos los procesos al mismo tiempo, sino secuencialmente:

-En primer lugar se cargan los números en los registros.
-Más tarde se llevan a cabo las restas oportunas.
-Una vez se terminado la resta, se evalúa el acarreo.
-Se toman las acciones pertinentes acorde con el valor del acarreo.
-Se repite el proceso hasta que se llega al final de la división.

En la siguiente figura se representan las señales adicionales mencionadas como "enable_1", "enable_2" y "D" respectivamente. Así mismo los 4 biestables nombrados como "N_x" y los 4 nombrados como "D_x" serán las representaciones de los 4 dígitos menos significativos del dividendo y el divisor. Así mismo "S_x" atenderá a las necesidades de desplazamiento del divisor.

La señal del cociente irá almacenada en un registro aparte.

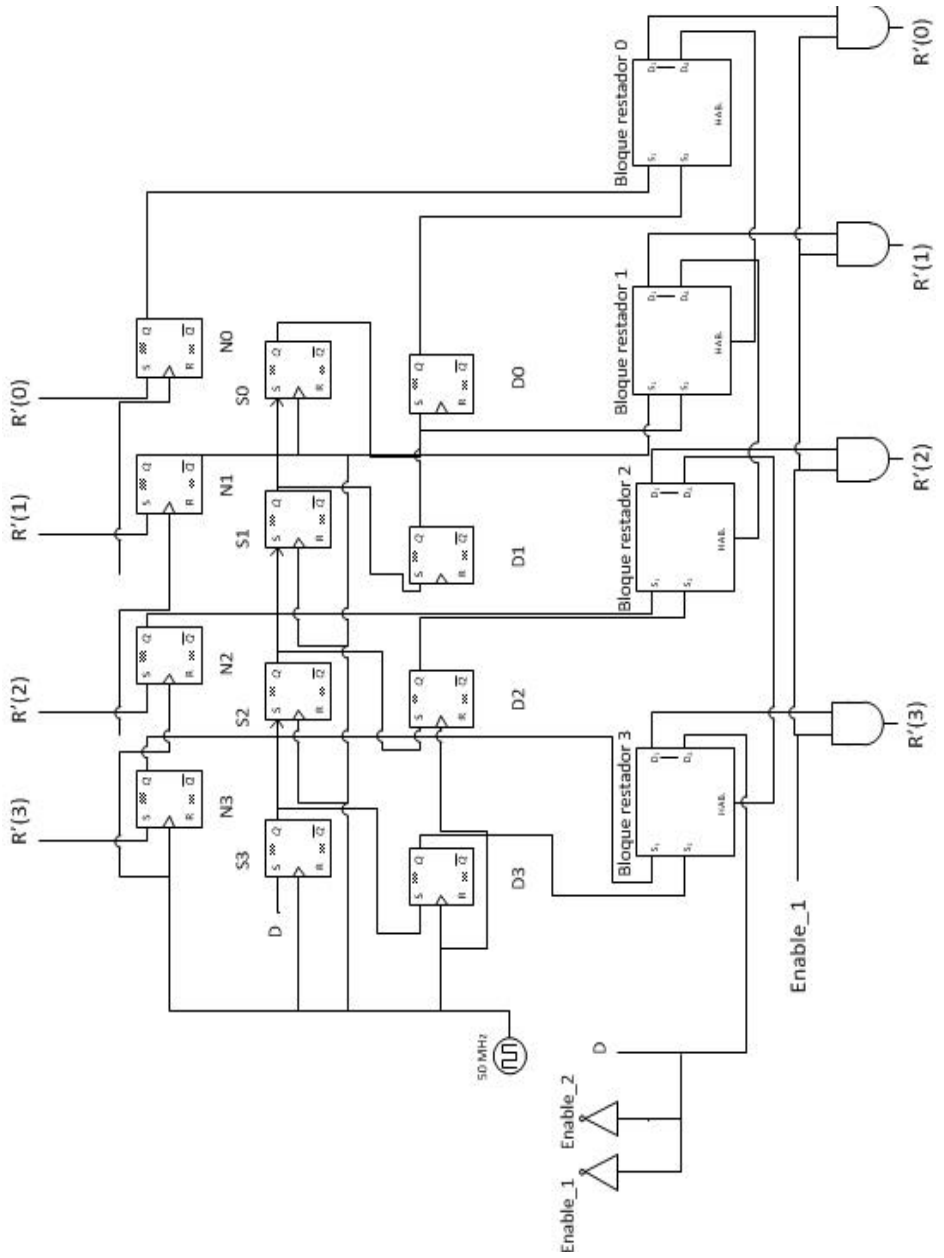

FIGURA 5. PRIMER ESQUEMA

DISEÑO DE LA ARQUITECTURA

DIVISOR

Una vez que se tuvo claro el funcionamiento del divisor así como la arquitectura del mismo procedí a describir el código VHDL.

El diseño de la arquitectura, implica también prever la frecuencia de reloj de sincronismo para los elementos de control y de memoria. Atendiendo a que la red eléctrica tiene una frecuencia de 50 Hz llegué a la conclusión de que con un reloj de 50 MHz tendré un 1.000.000 de ciclos de reloj por ciclo de red eléctrica, tiempo más que suficiente para realizar múltiples operaciones.

Lo siguiente fue establecer el número de bits con los que va a trabajar el divisor. La elección fue trabajar con números de 8 bits ya que me van a proporcionar números lo suficientemente grandes. Sin embargo la ampliación a datos con mayor número de bits no es problemático, pues el código se puede modificar fácilmente.

Se comienza a trabajar con la premisa de realizar operaciones de números codificados en complemento A2.

El complemento A2 es un sistema por el cual se establece que los números cuyo bit más significativo sea ‘0’ son números positivos, sin embargo si son negativos, este bit valdrá ‘1’.

Si se desea llegar al complemento A2 de un número a partir de uno binario tenemos que atender al signo del mismo: si el número es positivo su equivalente en complemento A2 es ese mismo número. Por el contrario si es negativo se deberá sustituir los dígitos de ese número por su contrario (‘1’ por ‘0’ y viceversa) y sumarle 1.

Con esta forma de codificación de los números se obtienen distintas ventajas, sin embargo para lo que respecta a este Trabajo de Fin de Grado se hace necesario esta conversión porque es el método de trabajo que utilizan los procesadores y el resto de circuitería periférica al divisor, por lo tanto es un paso lógico.

La duración de cada división será variable, ya que dependiendo del tamaño de los operandos que entren en el componente éste tardará más o menos iteraciones en resolver la división.

El componente ha ido ideado para que se pueda ejecutar una sola división cada vez y que sea necesario ejecutar un pulso de “reset” cada vez que sea necesario volver a realizar otra.

A la hora de establecer la interfaz del divisor se determinaron las entradas y salidas con las que tenía que contar el componente. Son las siguientes:

ENTRADAS:

-Reset: como la mayoría de los circuitos digitales, se estableció un señal de activa a nivel alto que pone a '0' el valor de los biestables del circuito, de forma asíncrona, para tener la capacidad de restaurar instantáneamente el sistema a un estado conocido.

-CLK: es la señal de sincronismo, reloj de 50 MHz de la que se ha hablado con anterioridad. Ésta señal es común para todos los biestables.

-Typeestados: se definió un nuevo tipo de señal, las señales tipo "estados"; con ellas voy a diseñar la máquina de estados sin interferencias de los otros tipos de señales.

-'A' y 'B': vectores de 8 bits en los que se va a cargar el dividendo y el divisor respectivamente.

SALIDAS:

-Fallo: como requisito de diseño se estableció que el dividendo no puede ser menor que el divisor, en caso contrario se abortará el proceso de la división y se activará esta señal.

-Cociente: salida de 8 bits donde va ir indicarse el resultado de la división.

-Fin: señal de un bit que indica que el proceso del componente ha finalizado y que la salida del componente es el resultado último y válido.

-Signo: otra salida de un bit que marca si el cociente es positivo o negativo.

Tras quedar fijada la interfaz, que se traduce directamente como una entidad en código VHDL, paso a describir las señales internas con las que cuenta el circuito para llevar a cabo los procesos:

-Test: es una señal asociada a la máquina de estados. En ella, según se iba desarrollando el código, se iban añadiendo más estados de acuerdo con las necesidades del circuito.

A la hora de hacer las simulaciones resultaba más conveniente comprobar los resultados de los estados en los que se encontraba la máquina en cada momento a través de un vector que tenía una correspondencia con cada uno de los estados que con el nombre de los mismos.

Por lo tanto esta señal no es necesaria pero se estableció para facilitar el trabajo. En el diseño final se puede eliminar, aunque siempre se puede dejar para realizar *test-on-line* durante la operación normal del bloque divisor.

-Numerador y denominador: en ellas se cargan los valores de las entradas y se hacen las respectivas modificaciones de ellos según transcurre la división.

-Cociente1: al igual que las anteriores, en ella se van a cargar los valores del cociente que se vayan obteniendo en el proceso.

-Diferencia: es el registro donde se va a guardar cada resta que se hace para obtener los dígitos de la división. Dependiendo del resultado este registro se cargará en sustitución de "Numerador" o no.

-Tamanio_num, tamanio_den y resta_tam: son enteros de 8 bits de vital importancia. Las señales que más trabajo llevó obtener y trabajar con ellas, con respecto a las otras.

Hay que tener en cuenta que dependiendo del número de bits significativos de que dispongan tanto el dividendo como el divisor tendremos un número de restas u otro. Para que el circuito controle cuantas iteraciones debe hacer se establecieron estas señales. Más adelante en otro punto de la memoria explicaré el funcionamiento del diseño con un ejemplo y quedará claro cómo funcionan las señales.

-Enable_1 y enable_2: estas señales de un bit son las que se modifican según el acarreo de las restas.

-División_terminada: señal de un bit que se cargará en la salida "fin" del componente una vez que se haya acabado el proceso de la división.

-Numerador1 y denominador1: estas señales de 9 bits son la respuesta encontrada a la problemática de que el circuito no tuviese en cuenta el bit de acarreo en cada diferencia. Se carga en ellos los valores de "numerador" y "denominador" concatenados con un '0' a su izquierda.

-Error1: es la señal que variará si se produce el error establecido como requisito de diseño de que el dividendo sea menor que el divisor. Se cargará en la salida "Fallo".

-'X' e 'Y': son valores que van a analizar el signo de las entradas para establecer el de la salida.

-Signo1: es la señal que va a ir a la salida "Signo".

Los procesos de los que consta el divisor son 3. El primero rige el funcionamiento del programa en cuanto a los cálculos se refiere. Acorde con la máquina de estados se van a producir ciertas operaciones o ciertos eventos que van englobados en un único proceso síncrono.

Los otros 2 describen la máquina de estados (proceso secuencial y combinacional). Según se produzcan los eventos adecuados en el primer proceso, la máquina los recogerá y hará los cambios oportunos para que se produzca la secuencia de estados adecuada para el proceso.

El diagrama de estados y transiciones de la máquina de estados es el siguiente:

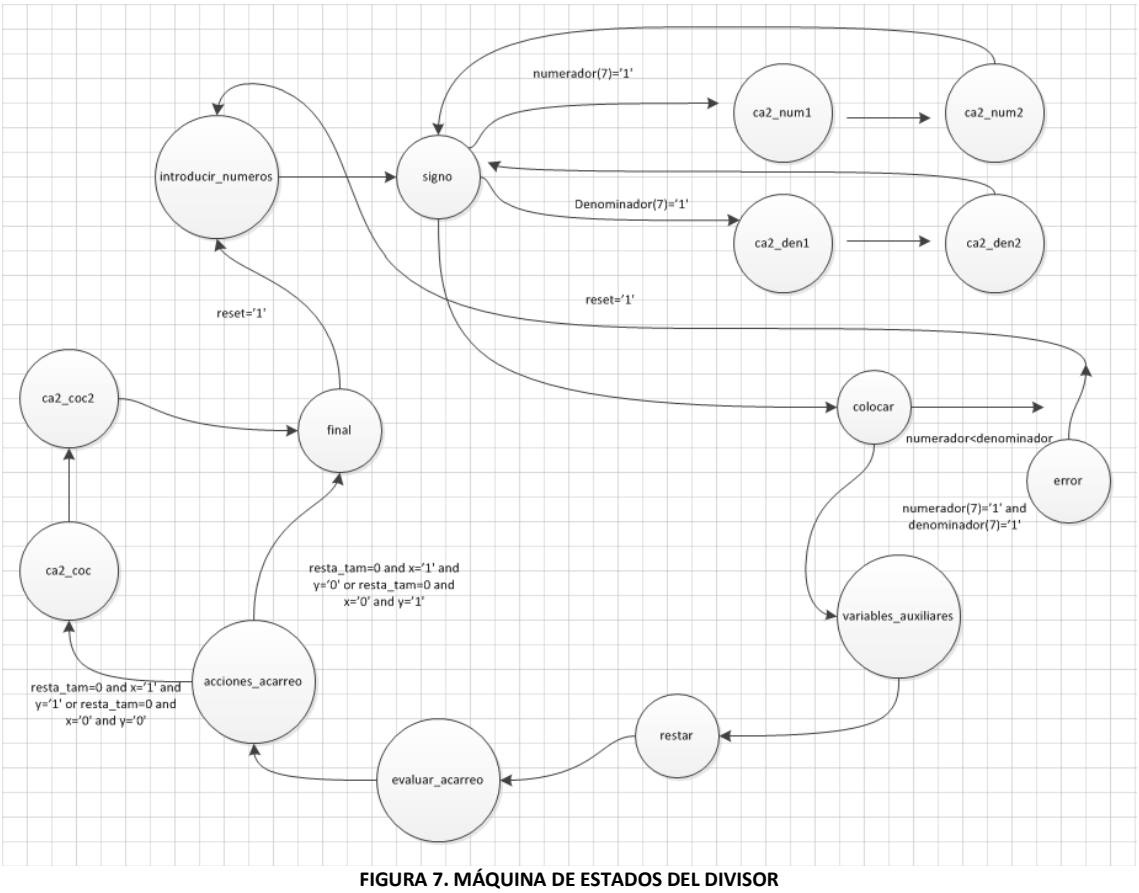

FIGURA 7. MÁQUINA DE ESTADOS DEL DIVISOR

En ella se especifican las condiciones necesarias para que se pase de un estado a otro y se consiga que el proceso de división esté ordenado y sea efectivo.

TEST

Para comprobar que las salidas que va a generar el divisor son correctas se generó un archivo "Test-bench" llamado "test_comp.vhd".

En éste archivo se va a generar un sistema artificial de entradas y salidas del componente virtual, con lo que se va comprobar si el circuito diseñado cumple los requisitos de diseño planteados y ver dónde se falla para corregirlo.

Esta clase de comprobaciones es lo que hace que la descripción de circuitos digitales mediante lenguajes de alto nivel, proporciones notables ventajas frente al prototipado directo. Así, dado que el sistema es enteramente virtual se puede ver los valores de todas las señales del circuito en cada instante para verificar dónde se encuentran los

fallos; así sólo es necesario cambiar el código allí donde se haya errado hasta conseguir un resultado óptimo.

En el software empleado en este Trabajo de Fin de Grado tenemos la posibilidad de que se genere en "*test-bench*" de forma automática, con lo cual sólo hay que indicar los números de entrada para que genere el cronograma donde se va a comprobar los resultados.

PRUEBAS EN LA FPGA

El siguiente paso es la comprobación física del dispositivo. Para hacerla se va a utilizar una tarjeta FPGA modelo XC3S200.

FIGURA 8. FPGA SPARTAN XC3S200

Se ha escogido este modelo porque, como puede verse en la imagen, posee 4 *displays* de 7 segmentos en los que mostrar los números del cociente.
Sin embargo el uso de la tarjeta plantea la problemática de añadir la lógica adicional que implemente el interfaz de las entradas y salidas de la placa. Seguidamente voy a detallar el nuevo circuito que debe realizarse para validar funcionalmente el prototipo.

En primer lugar se crea un diseño de mayor nivel que englobe al divisor y a los nuevos subcomponentes que van a hacerse.
Llamo a dicho bloque "componente" y se incluye el bloque "divisor" como subcomponente de ella.

INTERFAZ DE SALIDA: *DISPLAYS* DE 7 SEGMENTOS

CONVERSOR DE DATOS A FORMATO 7 SEGMENTOS

Este bloque realiza la conversión de los resultados para su visualización en los displays de 7 segmentos, disponibles en la placa de desarrollo. El objetivo es transformar el cociente que sale del subcomponente "divisor" en señales de activación de cada uno de los displays. Este bloque tendrá las siguientes entradas y salidas:

- "CLK" y "Reset": son las mismas que van a ir a todos los subcomponentes y el componente de más alto nivel. Así todo el circuito tendrá la misma señal de reset y el mismo reloj y para evitar posibles errores de sincronización.

- “Fin”: esta señal de un bit enlazará el conversor con el divisor, así el conversor sabrá qué número de los que marque la salida del divisor es el que debe cambiar para mostrarse en el display.

- “Cociente_BCD”: es la entrada del cociente que proviene del divisor.
- “Centenas”, “Decenas” y “Unidades”: las salidas del componente serán vectores de 7 elementos en los que se marque qué elementos del display se van a iluminar y cuáles no para mostrar el número adecuado.

En el bloque se identifican dos funciones con el cometido de realizar la conversión de los elementos:

- La primera de ellas llamada “Bcd” tendrá como entrada un vector de 4 elementos que va a llevar el dígito en concreto que debe mostrar uno de los displays, ya sean las centenas, las decenas o las unidades, y se su salida será el vector de 7 elementos que lleve la información de qué segmento del display iluminar y cuál no.
  Hay que tener muy en cuenta que los segmentos se iluminan cuando llega una señal a nivel bajo en la placa con la que se trabaja.

- La segunda se la llama “To_bcd” y será la encargada de transformar las 3 entradas provenientes del divisor en un vector de 12 elementos con los 3 dígitos decimales pasados a binario natural.

Aparte se ha descrito una máquina de estados que controle si el componente está en funcionamiento o en estado de reposo.

También se ha implementado un conformador de pulsos para la entrada “Fin”.

Con lo cual el subcomponente funcionará de la siguiente forma: cuando llegue la entrada “Fin” se pasará del estado “Reposo” a “Convertir” y se ejecutarán las acciones pertinentes. Tras esto el circuito pasará sin ningún estímulo externo a “Reposo de nuevo”.

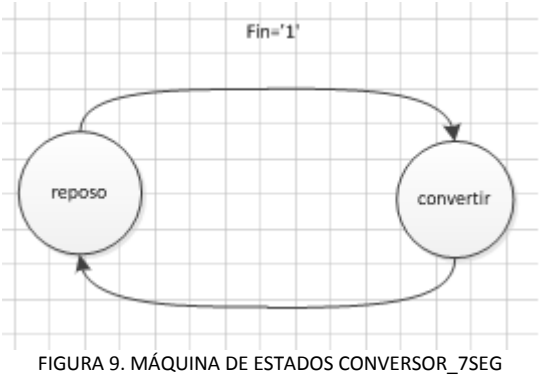

FIGURA 9. MÁQUINA DE ESTADOS CONVERSOR_7SEG

CONTROL DE SALIDA DE DATOS A LOS DISPLAYS DE 7 SEGMENTOS

El bloque llamado “contador_display” tiene como función que vean las señales apropiadas en los *displays* apropiados. Los *displays* de la placa de prototipado que se ha utilizado, tienen sus entradas de 7 segmentos (más el bit de punto decimal) cortocircuitados. La selección de un *display* en concreto se hace mediante una señal de 4 bits que conecta el cátodo común (de ese *display*) a la máxima tensión, haciendo que los LEDs de los segmentos se iluminen. Sólo se puede iluminar un *display* cada vez, pero si se cambia la señal de selección suficientemente rápido, el ojo humano no será capaz de distinguir este hecho y percibirá todos los los *displays* iluminados a la vez. Para conseguir esto debe realizarse un contador que cuente a una frecuencia media (superior a 50 Hz) y que su salida seleccione el dígito a mostrar a la vez que el *display* donde mostrarlo.

Del subcomponente anterior hemos obtenido 3 salidas que son 3 vectores de 8 elementos donde se guardan los valores de los dígitos de las centenas, las decenas y las unidades resultado de la división. Ahora lo que vamos a hacer es indicar en qué *display* va a ir cada uno de estos vectores para que se muestren los valores en los lugares adecuados.

Para este fin declaro una entidad con las siguientes entradas y salidas:

- “CLK” y “RESET”: son el reloj y el reset comunes al resto de subcomponentes.

- “Pin_display”: es un vector de salida de 4 términos donde se va a seleccionar en qué *display* se va a mostrar el dígito seleccionado.

- Mux: vector de salida de 2 elementos que va a indicar qué *display* se va a mostrar y, por lo tanto, qué número se debe cargar en él (si son las centenas, las decenas o las unidades). Para lograrlo se utiliza un “case” donde asigno a la salida “BCD” del componente principal el valor que corresponda

Aparte también declaro señales para asignar a las salidas ya mencionadas y otra que se llama “contador”.

Esta señal, como su propio nombre indica, va a servir de contador para que se vayan cambiando los valores de las salidas y así poder mostrar los 3 dígitos en los *displays*.

Debido al tamaño del vector y al reloj de 50 MHz utilizado los cambios se harán a una velocidad suficiente para que no sea perceptible por el ojo humano y así tendremos la sensación de que todos los *displays* están fijos y encendidos a la vez.

BLOQUE ANTIRREBOTES

Debido a posibles vibraciones en la parte mecánica de los botones que se utilizan en la placa de prototipado se hace necesario para su eliminación añadir al circuito un sistema antirrebotes.

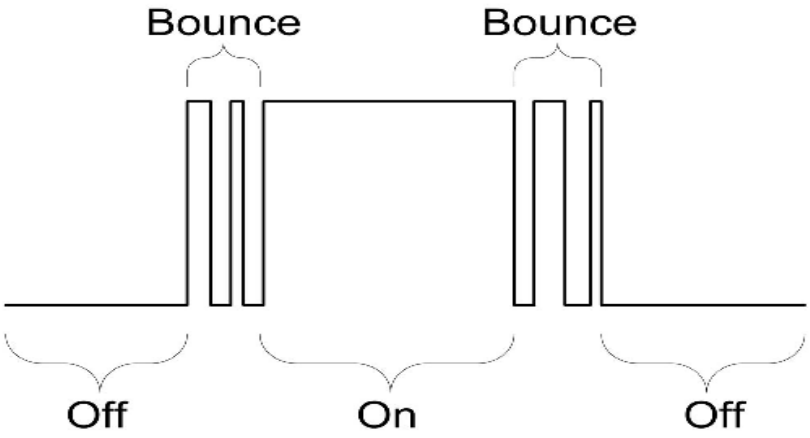

FIGURA 10. EFECTO DE LAS PIEZAS MECÁNICAS – REBOTES

La figura 8 muestra los efectos de esas vibraciones: todos los componentes mecánicos están sujetos a posibles vibraciones debidas a múltiples causas. Esto puede hacer creer a la FPGA que se está pulsando el botón de paso de estado en más ocasiones de las que realmente ocurren. Al trabajar la placa a 50 MHz estos rebotes se pueden detectar como pulsos de activación de los botones.

Para evitar el de los rebotes de los botones se va a diseñar un bloque que genera una señal de un solo pulso de un ciclo de reloj de duración y que durante un cierto tiempo no se pueda volver a generar otro para así estar seguros que sólo se va a hacer caso a las pulsaciones reales de los botones y no de los rebotes.

El bloque diseñado tendrá en su interfaz las siguientes salidas y entradas:

- “Clk” y “Reset”: son las entradas de un bit correspondientes a la señal de reloj y al reset que utilizan todos los componentes del diseño.
- “Boton: es la entrada de un bit que corresponde al accionamiento del botón físico de la FPGA con la que se realizan las pruebas.
- “Boton_anti”: es la salida de un bit que va a llevar un único pulso cada vez que se accione el botón de la placa, a su vezse provocará una inhibición de esta salida de un cierto número de ciclos de reloj cada vez que sea activada.

Las señales empleadas en la arquitectura son:

- “Contador”: es un vector de bits; este contador se activa cuando llega una entrada “Botón” y cuenta hasta desbordarse para, según sea el tamaño de éste, inhibir o no la salida del componente; esto es a mayor tamaño más tiempo estará el componente sin emitir pulsos aunque entre “Botón”.
- “Boton_anti_1”: es la señal que se va a cargar a la salida “Boton_anti” del componente.

- “Enable”: esta señal de un bit sirve para habilitar o no la salidadel componente. Permanecerá a ‘0’ mientras el contador no esté en funcionamiento.

CIRCUITO COMPLETA PARA PRUEBAS EN PROTOTIPO

El circuito completo para su prototipado en la FPGA de la placa de pruebas, presenta una interfaz con las siguientes entradas y salidas:

- “CLK” y “Reset”: son las entradas del reloj y el reset que se van a utilizar en el propio componente y en los subcomponentes que engloban.

- “Entrada”: es un vector de 8 bits donde se va a cargar tanto el dividendo como el divisor en los que se esté interesado hacer una comprobación. Éste proceso va a estar controlado por una máquina de estados de la que se hablara en profundidad más adelante.

- “Boton”: entrada de un bit que permite manejar la máquina de estados del componente a través de la FPGA, con lo cual voy a ser capaz de controlar los tiempos de carga de los datos a voluntad.

- “Cociente_cA2”: es un vector de salida de 9 bits que va a llevar el resultado que salga del subcomponente “divisor”, se puso para comprobar rápidamente en los “test-bench” si los resultados del divisor tras las conexiones al componente de alto nivel y al resto de subcomponentes eran correctas.

- “BCD”: es una salida de 8 bits que en la que va la secuencia de ‘1’ y ‘0’ para que se enciendan los displays de modo que se muestren los números que resulten de la división.

- “Signo”: salida proveniente del divisor que, dependiendo del signo del cociente valdrá ‘1’ o ‘0’.

- “Fallo”: es la salida de 1 bit que proviene del divisor y tiene valor ‘1’ cuando el divisor es mayor que el dividendo.
- “Pin_display”: salida donde se alberga el vector de 4 bits que marca qué display de los 4 debe iluminarse para cada salida “BCD”.

Además se han declarado varias señales para llevar a cabo distintas funciones. Éstas son:

-“Fin”, “fin1”, “fin2”, ”fin_conf”: señal que proviene de “divisor” que marca cuándo se ha llegado al resultado una vez que se han introducido el dividendo y el divisor.
Una vez que cambia de ‘0’ a ‘1’ al terminar la operación se diseña dentro del componente un conformador de pulsos ayudándome de las señales auxiliares “fin1” y

“fin2” de manera que “fin_conf” llevará un único pulso a “conversor_7seg” para que comience a trabajar. Se ha diseñado de esta manera para que la máquina de estados que controla a “conversor_7seg” no entrase en un bucle cuando, tras realizar la conversión, se encuentre en el estado “reposo” y la señal “fin” a ‘1’, que es la condición para avanzar al siguiente estado. Sólo se busca que se haga una conversión cada vez que hay una división.

-“Boton1”, “boton2” y “boton_conf”: estas 3 señales van a trabajar de la misma forma que con la señal “fin”: se establece un conformador de pulsos para que cada vez que se pulse el botón que cambia de estado a la máquina del componente “componente” sólo se produzca un pulso por cada accionamiento y así se tendrá un mayor control sobre ella.

-“num” y “den”: vectores de 8 bits que van a guardar los valores del dividendo y del divisor que se establezcan en las placas. Estos valores serán los que se pasen a “divisor” para que haga la operación adecuada.

-“cociente_BCD”: es la señal que conectará la salida del divisor con la entrada del conversor a BCD.

-“go”: es una señal que irá a la entrada “enable” de “divisor”. Ésta valdrá ‘1’ sólo cuando la máquina de estados de “componente” está en el estado donde se deban realizar las operaciones. Así indicamos a “divisor” que se han cargado los valores de dividendo y divisor y que debe comenzar a operar.

-mux1: señal que sale de “contador_display” y que indica, con un “case” establecido en el componente si el número que se debe cargar en la salida “BCD” es el de las centenas, las decenas o las unidades.

-“centenas1”, “decenas1”, “unidades1”: son las señales en las que se cargan los resultados de la conversión y que posteriormente se van alternando a la salida “BCD” para mostrarse en los displays.

Para hacer este cambio se utiliza una función que se encontró en internet que hace justo esta función. La página tiene como enlace:

http://embdev.net/topic/278160 siendo la función obtenida:

```
function to_bcd ( bin : std_logic_vector(7 downto 0) ) return std_logic_vector is
variable i    : integer:=0;
variable bcd  : std_logic_vector(11 downto 0) := (others => '0');
variable bint : std_logic_vector(7 downto 0) := bin;
begin
   for i in 0 to 7 loop  -- repeating 8 times.
   bcd(11 downto 1) := bcd(10 downto 0);  --shifting the bits.
   bcd(0) := bint(7);
   bint(7 downto 1) := bint(6 downto 0);
   bint(0) :='0';
      if(i < 7 and bcd(3 downto 0) > "0100") then --add 3 if BCD digit is greater than 4.
      bcd(3 downto 0) := bcd(3 downto 0) + "0011";
      end if;
      if(i < 7 and bcd(7 downto 4) > "0100") then --add 3 if BCD digit is greater than 4.
      bcd(7 downto 4) := bcd(7 downto 4) + "0011";
      end if;
      if(i < 7 and bcd(11 downto 8) > "0100") then  --add 3 if BCD digit is greater than 4.
      bcd(11 downto 8) := bcd(11 downto 8) + "0011";
      end if;
end loop;
return bcd;
end to_bcd;
```

FIGURA 11. FUNCIÓN “TO_BCD”

Para que se pueda utilizar ha sido declarado el vector “bcd_binary” donde se cargará la salida de la función.

También se implementó otra función que tiene el cometido de activar las 7 señales de los displays en función de los números que se quiera sacar.

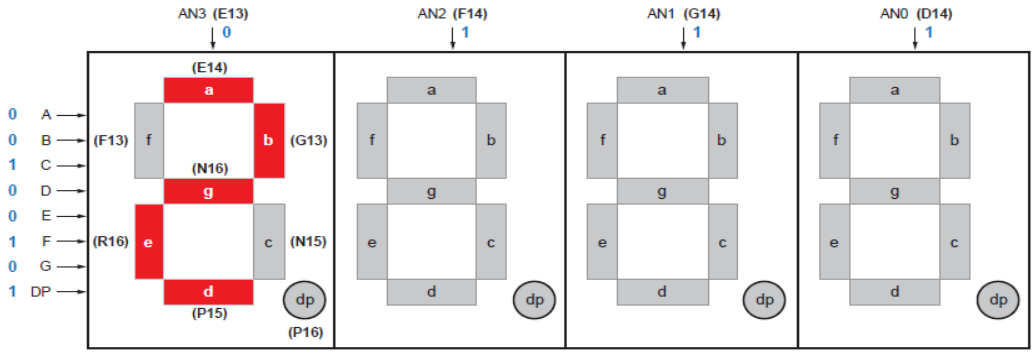

FIGURA 12. PINES DE LA FPGA

Esta función tiene como entradas un vector de 4 cifras que lleva la centena, decena o unidad del cociente a convertir y devuelve otro vector de 7 cifras que lleva la información de cada una de las franjas de los displays a iluminar.

Para que todos los procesos lleven un orden adecuado para la realización del proceso se establece una máquina de estados regulada por una entrada llamada “botón” y por la señal de “fin” del divisor.

Se establecen los estados “reposo”, que será el que tenga el componente una vez que se le hace un “reset”, “numerador” y “denominador” que serán los de carga de los valores del dividendo y el divisor, todos ellos regidos por la señal “botón_conf” que está relacionada con la entrada “botón”, y un último estado llamado “división” en el que estará la máquina mientras el divisor está haciendo las operaciones adecuadas para la obtención del cociente. Tras la emisión de la señal “fin” por parte de “divisor” el componente “componente” volverá al estado de reposo.

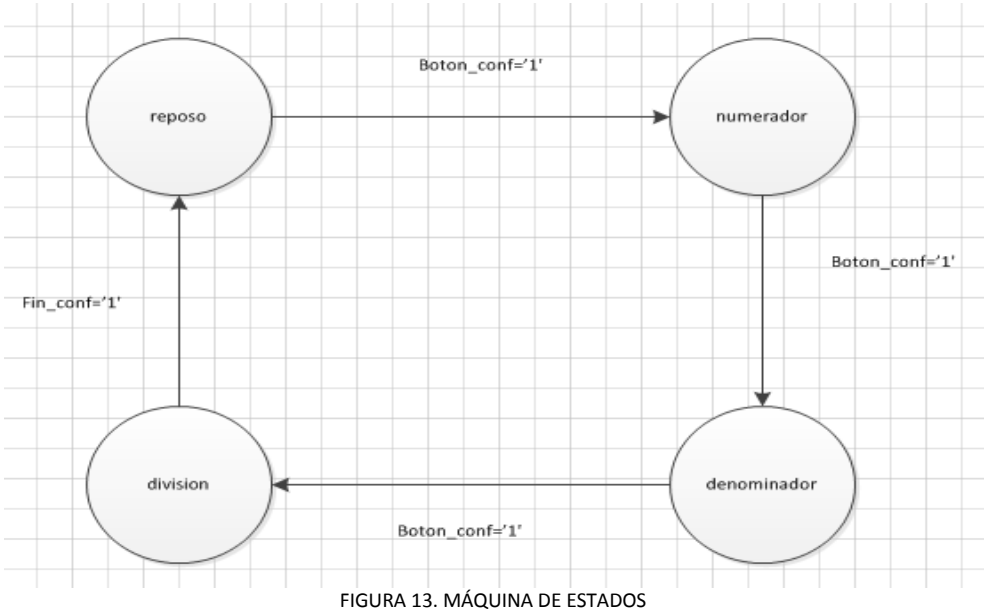

FIGURA 13. MÁQUINA DE ESTADOS

ASIGNACIONES DE PINES DE LA FPGA A LAS ENTRADAS Y SALIDAS DEL CIRCUITO. ARCHIVO ".UCF".

Es un archivo donde se indica a la FPGA cuáles de sus pines van a ser utilizados y a qué entradas o salidas del código van a ser asignadas.

Fue generado siguiendo las instrucciones de la placa para averiguar el número de cada pin a utilizar.

SÍNTESIS

Los resultados de tiempo y área tras el proceso de síntesis del circuito son:

```
Clock to Setup on destination clock clk
---------------+---------+---------+---------+---------+
               | Src:Rise| Src:Fall| Src:Rise| Src:Fall|
Source Clock   |Dest:Rise|Dest:Rise|Dest:Fall|Dest:Fall|
---------------+---------+---------+---------+---------+
clk            |    7.021|         |         |         |
---------------+---------+---------+---------+---------+
```

FIGURA 14. RESULTADOS DE SÍNTESIS EN FRECUENCIA Y TIEMPO

Estos 7,021 ns equivalen a una frecuencia de 142,429 MHz, que es una frecuencia de funcionamiento lo suficientemente rápida para las aplicaciones en el campo de la electrónica de potencia.

```
Design Summary
--------------
Number of errors:      0
Number of warnings:    0
Logic Utilization:
  Number of Slice Flip Flops:           153 out of   3,840    3%
  Number of 4 input LUTs:               321 out of   3,840    8%
Logic Distribution:
  Number of occupied Slices:            201 out of   1,920   10%
    Number of Slices containing only related logic:     201 out of     201 100%
    Number of Slices containing unrelated logic:          0 out of     201   0%
      *See NOTES below for an explanation of the effects of unrelated logic.
  Total Number of 4 input LUTs:         329 out of   3,840    8%
    Number used as logic:               321
    Number used as a route-thru:          8

  The Slice Logic Distribution report is not meaningful if the design is
  over-mapped for a non-slice resource or if Placement fails.

  Number of bonded IOBs:                 33 out of     173   19%
  Number of BUFGMUXs:                     1 out of       8   12%

Average Fanout of Non-Clock Nets:                3.95

Peak Memory Usage:  212 MB
Total REAL time to MAP completion:  3 secs
Total CPU time to MAP completion:   1 secs
```

FIGURA 15. RESULTADOS DE SÍNTESIS EN ÁREA

Dado que se ha utilizado una FPGA Spartan XC3S200, que es una placa de capacidad media-baja, y que de ella se utilizan un 3% de los *Slice Flip Flop* y un 8% de las *LUTS* se concluye que el resultado obtenido del algoritmo utilizado es óptimo en área.

EJEMPLO PRÁCTICO DE FUNCIONAMIENTO

A continuación se va a hacer una descripción del funcionamiento global del sistema mediante un ejemplo para que así se pueda observar la utilización de todas las señales e instrucciones establecidas en el código.

El ejemplo elegido es la división (-27)/3; con ella se pueden demostrar todas las acciones que lleva a cabo el componente: el coger los números, comprobar si son positivos o negativos según la codificación de "complemento a2", dividir utilizando el algoritmo de sucesivas diferencias y finalmente emitir los pulsos de salida adecuados que contengan la información en complemento a2 y el pulso que indique que se ha terminado el proceso.

La prueba se va a hace con todos los elementos diseñados; esto es con los elementos para la prueba en la FPGA, para así mostrar todo el proceso que llevan a cabo los distintos componentes al completo.

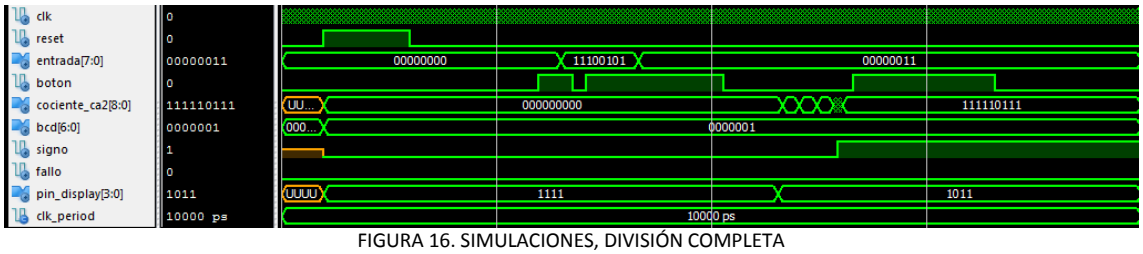

FIGURA 16. SIMULACIONES, DIVISIÓN COMPLETA

Lo primero es hacer llegar al componente los números que queremos dividir, para ello se escribirá un "*Test-bench*" con el siguiente código:

```
process
  begin
    reset   <= '0';
    entrada <= (others => '0');
    boton   <= '0';
    wait for 100 ns;
    reset <= '1';
    wait for 200 ns;
    reset <= '0';
    wait for 300 ns;
    boton <= '1';
    wait for 50 ns;
    entrada <= "11100101";
    wait for 30 ns;
    boton   <= '0';
    wait for 30 ns;
    boton   <= '1';
    wait for 120 ns;
    entrada <= "00000011";
    wait for 200 ns;
    boton   <= '0';
    wait for 300 ns;
    boton   <= '1';
    wait for 330 ns;
    boton   <= '0';
    wait;
end process;
```

FIGURA 17. TEST-BENCH

Donde se puede ver que en un primer momento se tienen todos los valores igualados a ‘0’, a los 100 ns se va a activar la señal de “Reset” que durará 200 ns para que elementos de memoria del circuito se inicialicen en valores conocidos, se dejan pasar 300 ns y se accionará el botón adecuado para que la máquina de estados del componente de alto nivel “sepa” que se le va a introducir el dividendo, a los 50 ns y se introduce el valor “11100101” en la entrada “entrada” del circuito que es el valor correspondiente a “-27” en complemento a2, tras dejar pasar 30 ns para que el circuito adquiera el valor mencionado se volverá a pulsar el botón que señalará que se va a poner el divisor, que a los 120 ns se indica que valdrá ”00000011”, ‘3’ en complemento a2. Las últimas líneas de código indican que se volverá a pulsar el botón para que el componente comience el proceso de la división.

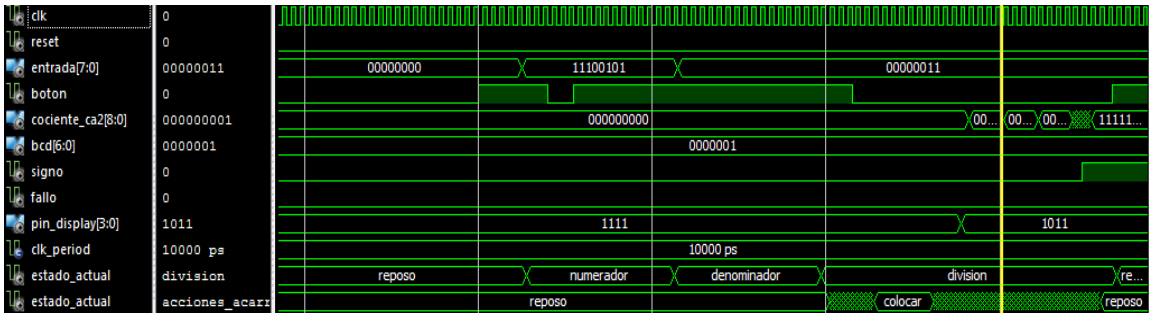

FIGURA 18. SIMULACIONES, CAMBIO DE FUNCIONAMIENTO DE COMPONENTES

Hecho esto en el componente de alto nivel “componente” se va a ver cómo trabaja el subcomponente “divisor” que es el que entra a trabajar, más tarde se volverá a “componente”.

En cuanto a “divisor”, se pasó del estado “reposo” a “introducir número tras desactivar la señal de “reset”, por lo tanto ya se tienen cargadas las señales “num” y “den” con los valores a dividir (“11100101” y ”00000011”). El siguiente paso según la máquina de estados es “ev_signo”, que tiene el cometido de comprobar si los números introducidos son positivos o negativos.

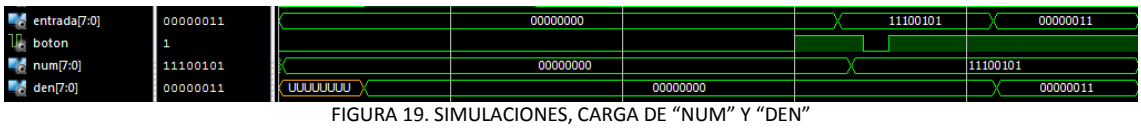

FIGURA 19. SIMULACIONES, CARGA DE “NUM” Y “DEN”

En primer lugar se va a atender al dividendo, si es o no positivo. Ya que una de las características de la codificación en complemento a2 es que la cifra más significativa los números negativos es ‘1’ y de los positivos es ‘0’, ese es el elemento diferenciador al que se va a atender. En nuestro ejemplo el número que se comprueba es “11100101”, cuya cifra más significativa es ‘1’, por lo tanto negativo.

Dado que es negativo se va a pasar al estado “ca2_num1” para comenzar a pasar el número de complemento a2 a binario natural para poder realizar la división buscada. Para convertir la cifra de que disponemos en la cifra deseada se ha de restar ‘1’ a “11100101”, quedando “11100100”, más tarde se pasa al estado “ca2_num2” que tiene el cometido de invertir cada uno de los dígitos del dividendo. Por lo tanto al final de estos dos estados se va a obtener que el dividendo para a tener el valor “00011011”, que en decimal es “27”, así que se ha conseguido el valor absoluto del dividendo.

Tras “ca2_num2” se pasa de nuevo a “ev_signo, donde la condición del dividendo se cumple y se pasa a comprobar el divisor. Dado que éste es “00000011” es positivo, por lo tanto no es necesario pasarlo a binario natural y no se va a pasar a los estados “ca2_den1” y “ca2_den2”, cuyo cometido es realizar lo mismo que se hizo con el dividendo. Por consiguiente ya que se cumplen las condiciones de “ev_signo” se pasa al estado “colocar”.

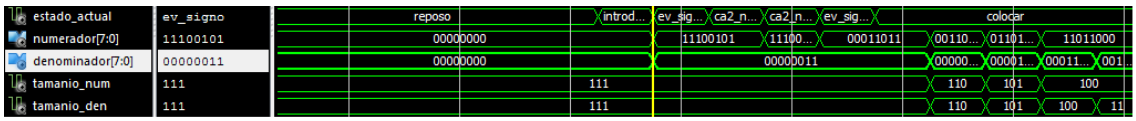

FIGURA 20. SIMULACIONES, SIGNO.

Es en “colocar” donde realmente va a empezar el proceso de la división. Como ya se explicó con anterioridad en la presente memoria el algoritmo a utilizar para resolver la operación consiste en las sucesivas diferencias de dividendo y divisor estando las cifras colocadas adecuadamente. Para llevar a cabo esta colocación el circuito va a comprobar la cifra másа la izquierda de cada uno de los números (dividendo y divisor) y va a modificarlos para que las cifras significativas más altas estén alineadas a la izquierda de los 2 vectores.

Para conseguir esto se van a apoyar en las señales “tamanio_num” y “tamanio_den”, que son capitales a lo largo del proceso por dos razones:

- Son las encargadas de controlar los desplazamientos iniciales de los vectores para llevar a cabo las diferencias. No de permitirlas o no sino de llevar la cuenta de cuántos desplazamientos se han hecho de las cifras de cada vector. Para ello se efectúa una comprobación de las cifras más altas de los números que se tienen, en nuestro caso “00011011” y “00000011”.

  Comenzando por “0011011” se hará la comprobación de la cifra más alta, en este caso es ‘0’. El desplazamiento que se lleva a cabo es eliminar ese ‘0’, desplazar un lugar a la izquierda el resto de cifras y concatenar un ‘0’ al final de la siguiente forma:

  “0011011” ⇨ “X011011” ⇨ “11011X” ⇨ “110110”
  ⇧

  A su vez el valor de “tamanio_num” que fue inicializado en el pulso de “reset” pasará de tener el valor ‘7’ a ‘6’, de tal forma que el circuito llevará la cuenta del total de desplazamientos de dividendo y divisor.

- La segunda razón viene a colación de la primera: si se tienen en cuenta los desplazamientos que se han llevado a cabo en el estado “colocar” se puede prever cuantas diferencias va a hacer falta para realizar la división, por lo tanto con ambas señales se puede establecer una condición para conseguir el resultado final. Se mostrará más adelante esta condición.

Volviendo al estado colocar y atendiendo a nuestros dos números se ve que serán necesarios 2 desplazamientos para adecuar “numerador” a la forma que debe adoptar para la resta, quedando “numerador” con el valor “1101100”, que a pesar de que equivale a ‘108’ en decimal no se tendrá en cuenta ya que es “tamanio_num”, que valdrá ‘4’, quien controlará junto con “tamanio_den” el número de restas que se harán.

Así mismo la señal del divisor de valor “00000011” pasará a ser “11000000”, y “tamanio_den” ‘1’.

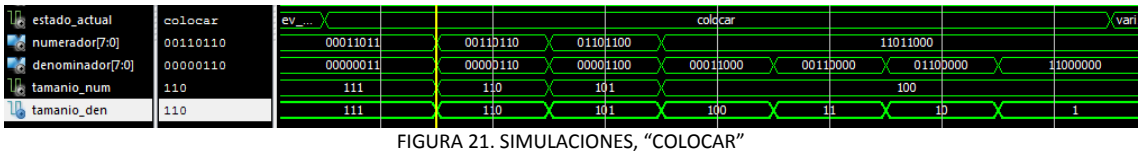

FIGURA 21. SIMULACIONES, “COLOCAR”

Una vez que tenemos colocados adecuadamente “numerador” y “denominador” se pasa al estado “variables_auxiliares”, donde los cargamos en los vectores

“numerador1” y “denominador1”. Al ser vectores una cifra mayor que “numerador” y “denominador” se concatenarán con un ‘0’ por la izquierda.

La razón para hacerlo es que se debe tener en cuenta el acarreo de resta, ya sea ‘0’ o ‘1’ para realizar las operaciones oportunas en el proceso. Así es 100% seguro que el compilador no lo desechará. “numerador1” vale en el ejemplo “01101100” y “denominador1” “011000000”.

Tras “variables_auxiliares” se pasa al estado “restar”, donde se efectúa la primera diferencia cuyo resultado se guarda en la señal “diferencia”. En el caso del ejemplo “diferencia” valdrá “000011000”.

El estado siguiente, llamado “evaluar_acarreo” va a ser el encargado de comprobar cual ha sido el acarreo de la resta para posteriormente llevar a cabo los procesos pertinentes. En este caso el acarreo es ‘0’. Por lo tanto se se establecerá el valor de las señales “enable_1” y “enable_2” según el valor del acarreo para que se lleven a cabo las acciones pertinentes. Dado que el acarreo es ‘0’, “enable_1” y “enable_2” toman el valor ‘1’.

Tras esto se pasa al estado “acciones_acarreo”; en él se realizan las acciones adecuadas según el valor de cada una de las señales “enable”.

“Enable_1” es la encargada de hacer cambios en el dividendo. Como ya se vio en la sección “Búsqueda del algoritmo del divisor” si se tiene el acarreo ‘0’ el dividendo debe tomar el valor de la diferencia para seguir operando. En caso de que fuese ‘1’ el numerador se mantiene tal como está. En el caso actual se sustituyen mediante un bucle “for” los valores de “numerador1” por los de “diferencia”.

“Enable_2”tiene como cometido añadir a la señal “cociente1” un ‘1’ o un ‘0’ según el acarreo ya que de ello depende el resultado: si el acarreo ha sido ‘1’ significa que el dividendo es más pequeño que el divisor, por lo tanto no cabe, hay que añadir un ‘0’ al cociente y correr el divisor un lugar a la derecha para seguir operando, en cambio si es ‘0’, como en el ejemplo, el valor que se añade es ‘1’ ya que si que se puede hacer la división.

Por lo tanto en el proceso actual se sustituyen los valores de “numerador1” por los de “diferencia” y a la señal “cociente1” se desplazan sus cifras un lugar a la izquierda y se concatena un ‘1’ a la derecha.

Aparte se debe desplazar el divisor un lugar hacia la izquierda para continuar el proceso. Para esto se tiene que concatenar un ‘0’ con las 7 primeras cifras de “denominador1”, que pasa a ser el nuevo valor de la señal. Además se le suma ‘1’ al valor de “tamanio_den” para tener controlados los desplazamientos que ha sufrido el divisor con respecto a su posición inicial.

Una vez aquí la máquina de estados tiene varias opciones: las dos primeras vienen dadas por la señal “resta_tam”, que se define como la diferencia entre los valores de

“tamanio_num” menos “tamanio_den”. Si el resultado de ésta es ‘0’ significa que se ha llegado al final de las diferencias posibles, por lo tanto se ha acabado el proceso de la división y se ha llegado al cociente buscado.

Sin embargo en el actual ejemplo no es el caso, ya que “tamanio_num” sigue valiendo ‘4’ y “tamanio_den” ha pasado a valer ‘2’, así que el estado siguiente es de nuevo “variables_auxiliares”.

FIGURA 22. SIMULACIONES, DIVISIÓN

Ya que lo interesante de esta descripción es ver cómo funciona el proceso, voy a saltar la reiteración de diferencias, que van a llevar un proceso equivalente al descrito, y me voy a centrar en relatar el funcionamiento del sistema a partir de la última diferencia en el estado “acciones_acarreo”.

Una vez que se ha añadido a “cociente1” la última cifra queda como “000001001”, lo que equivale en decimal a ‘9’, que es el módulo del cociente que queremos obtener.

Tras esto se vuelve a comprobar el valor de “resta_tam”; esta vez sí vale ‘0’, con lo cual se pasa a evaluar otra condición de nivel más bajo.

Si ya se ha acabado la división se debe comprobar el signo que tenían tanto el dividendo como el divisor para ver el que tendrá el cociente a su salida. De ello se encargan las señales ‘x’ e ‘y’.

Las señales ‘x’ e ‘y’ tomaron su valor en los estados de cambio de complemento a2 a binario. Al estar inicializados en ‘0’ con el pulso de “reset”, si no han pasado por los estados de cambio significa que ambos eran positivos, por lo tanto no necesitan pasar por una serie de estados que produzcan esta vez el cambio de binario natural a complemento a2; se puede quedar el número en binario ya que el binario y el complemento a2 de los números positivos es equivalente. Lo mismo hubiese pasado si ambos números hubiesen sido negativos.

Sin embargo nuestras entradas estaban compuestas por un número positivo yuno negativo. Por lo tanto hubo un paso por el estado “ca2_den1” donde la señal ‘y’ quedó fijada a ‘1’. Lo que viene a significar que es necesario el paso por dos nuevos estados que hagan que el número obtenido se pase a negativo. Por lo tanto la señal “signo1” queda a ‘1’ y es asignada a la salida “signo”

Estos dos estados son “ca2_coc” y “ca2_coc2” donde en el primero se van a negar sus cifras (se cambia ‘1’ por ‘0’ y viceversa) y en el siguiente se va a sumar ‘1’, quedando “cociente1” como “111110111”, que buscando el equivalente en decimal es ‘-9’, con lo que se ha acabado satisfactoriamente el proceso de la división.

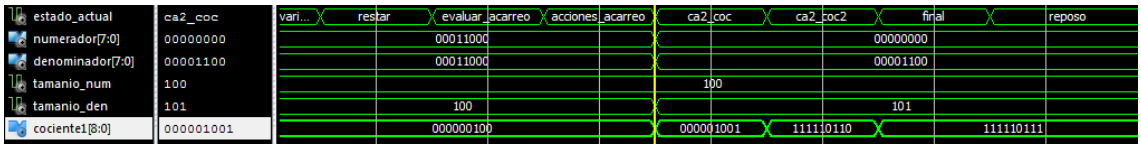

FIGURA 23. SIMULACIONES, SIGNO DEL COCIENTE

Otra de las variables de la comprobación del signo es la señal “cocienteBCD1”, que sólo va a llevar el valor del cociente a los módulos que se van a comunicar con la FPGA. Ya que estos módulos no llevan el número en codificación a2 debido a que el convertidor de las señales de los displays de 7 segmentos está hecho para binarios naturales. Con lo cual si el número es positivo se asignará “cociente” a “cocienteBCD1” en el estado “final” del divisor, pero si fuese negativo se asignaría en “ca2_coc” para que fuese en binario natural.

Aún queda por demostrar el funcionamiento del resto de subcomponentes que ahora comienzan a trabajar.

En primer lugar cabe destacar que la máquina de estados de la entidad principal, “componente”, sufre un nuevo cambio, ya que al terminar la división pasa a valer ‘1’ la señal “división_terminada”, que se asigna a la salida “fin”, que es la condición para que la máquina cambie de “división” a “reposo” a y se mantiene así a la espera de que se vuelvan a accionar el botón pertinente.

En cuanto al conversor de binario a las señales apropiadas para los BCD, ya se está recibiendo la entrada “000001001”, con lo que se puede comenzar a trabajar.

De primeras lo que se debe hacer es pasar el binario natural a BCD, o lo que es lo mismo, obtener un vector de 12 términos que lleve la información de cada uno de los 3 posibles dígitos que compongan el cociente pero por separado; esto es que las 4 primeras cifras serán las centenas, las 4 siguientes las decenas y las 4 restantes las unidades.

La entrada “cociente_BCD”, que es la que lleva la información del módulo del cociente, llama a las funciones “to_BCD” y “BCD” que pasarán éste módulo en binario natural a un vector de 12 elementos a código BCD, de “00001001” se obtiene el vector “bcd_binary” “0000000001001”. Y este vector llama a la función “BCD” donde se le introduce en 3 segmentos.

Por lo tanto las salidas de “centenas”, “decenas” y “unidades” quedan cargadas con las señales “centenas1”, “decenas1” y “unidades1” con los valores "0000001", "0000001" y "0001100", que corresponden a ‘0’, ‘0’ y ‘9’.

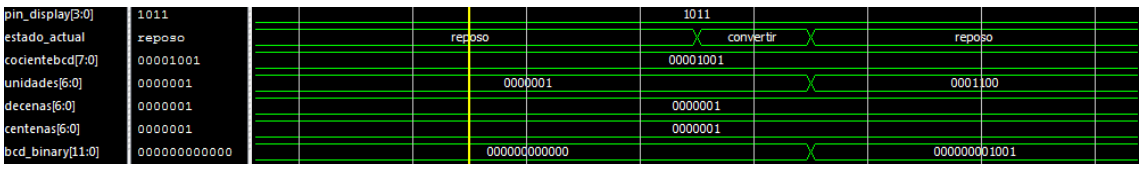

FIGURA 24. SIMULACIONES, CONVERSOR DE 7 SEGMENTOS

“Contador *display”* funciona independientemente de los valores introducidos en la simulación, ya que su única misión es generar una señal que vaya pasando los resultados de “conversor_7seg” ordenados en los distintos *displays*, dando igual el valor de los mismos.

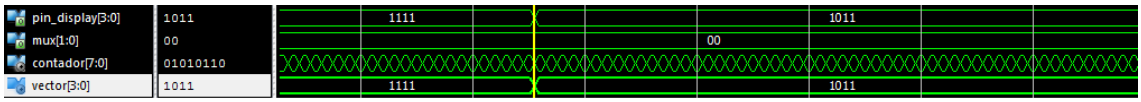

FIGURA 25. SIMULACIONES, CONTADOR-DISPLAY

Por último, “Antirrebotes” ejerce su función a la hora de pulsar el botón cambio en la máquina de estados del componente principal y también es independiente de los valores con los que se trabaje.

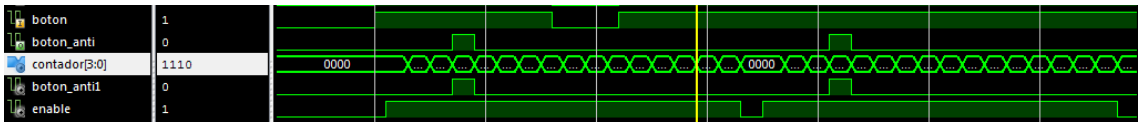

FIGURA 26. SIMULACIONES, ANTIRREBOTES

Así que como conclusión se ha obtenido como resultado del proceso de dividir (-27) entre 3, ambos en complemento a2, (-9), que es el que se debe obtener.

Queda demostrado cómo es el proceso de división del componente diseñado y la función de las señales internas del mismo, así como la interconexión entre los distintos subcomponentes para mostrar el resultado en los *displays* de una FPGA.

APLICACIÓN DEL DIVISOR EN EL LAZO DE CONTROL DE UN CONVERTIDOR "BUCK"

Ya que el fin del desarrollo del divisor digital fue la aplicación en la simulación de un convertidor, se procede a describir la aplicación de la utilidad del mismo: integrar el diseño en el lazo de control de un reductor para aplicar la realimentación "Feed-Forward".

CONVERTIDOR REDUCTOR

Un convertidor reductor es aquel que tiene una entrada de un cierto nivel de tensión que necesita bajarse y mantenerse estable para alimentar una carga.

Consta de una fuente continua de tensión a la entrada, que puede ser variable, un interruptor, que se abrirá y cerrará para permitir el paso de la corriente cada cierto tiempo según la necesidad de la salida; este intervalo es conocido como "ciclo de trabajo". Además trabaja con una bobina, que almacena corriente cuando el interruptor está cerrado y la libera cuando está abierto, un condensador que filtra la tensión a la salida almacenando la parte de alterna y una carga modelada como una resistencia que es lo que permite que la tensión de salida sea continua. Aparte tiene un diodo para asegurar el retorno de la corriente de la bobina cuando el interruptor está abierto y la bobina se descarga.

El esquema del circuito y la disposición de los elementos es la siguiente:

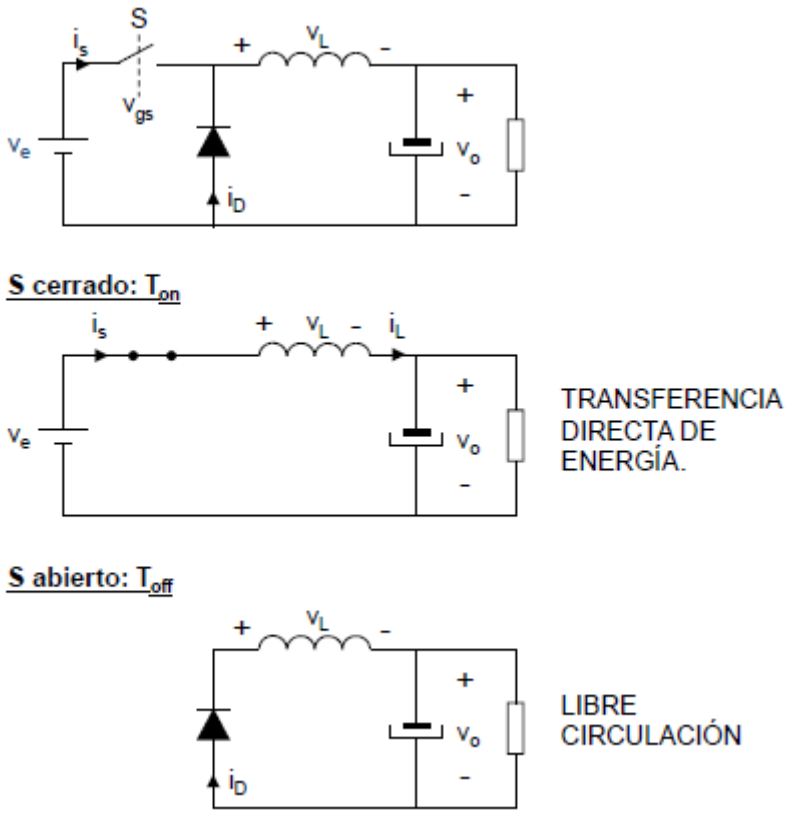

FIGURA 27. REDUCTOR EN MCC

Las gráficas que representan los parámetros del circuito son:

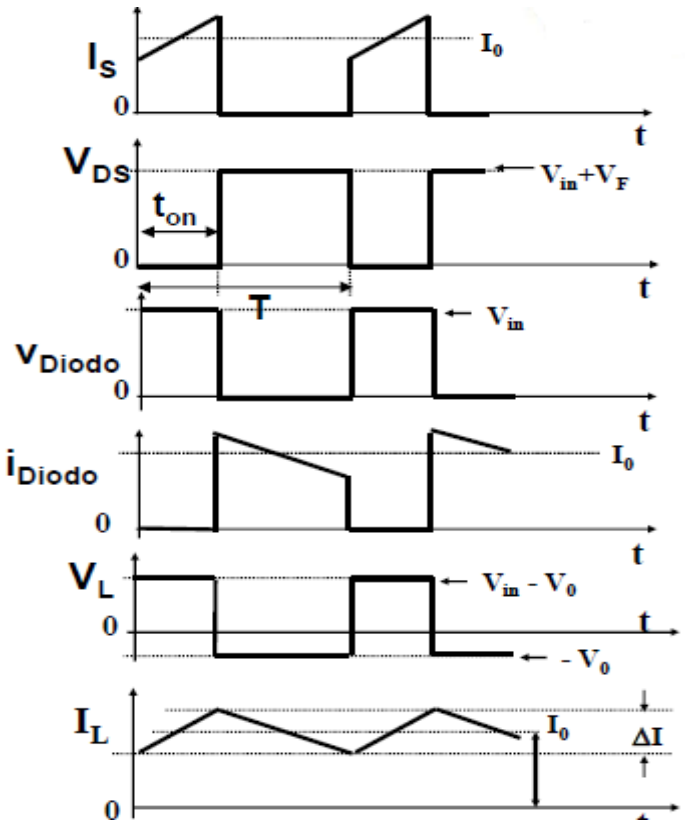

FIGURA 28. GRÁFICAS EN MCC

Si la bobina, posiblemente el elemento central del convertidor, llegara a descargarse completamente en el tiempo que el interruptor está abierto, la circulación de la corriente sería distinta:

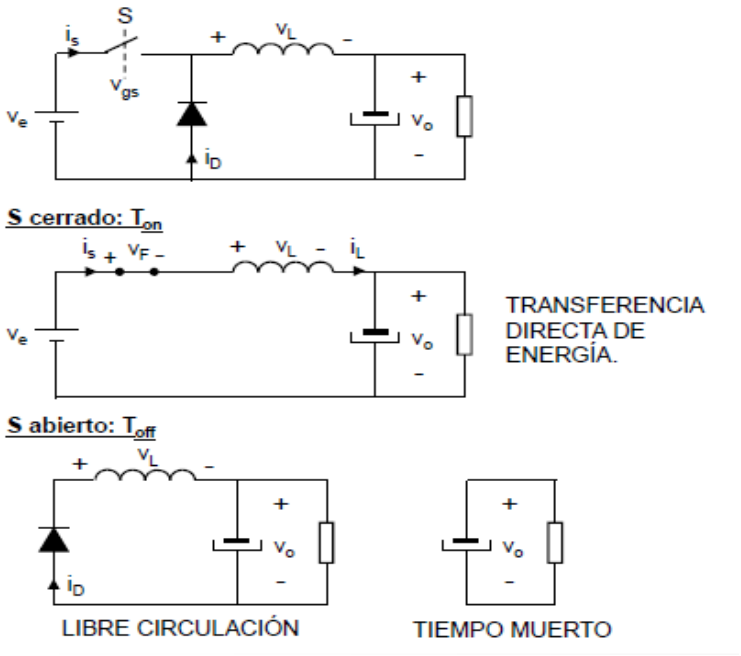

FIGURA 29. REDUCTOR EN MCD

Las gráficas son:

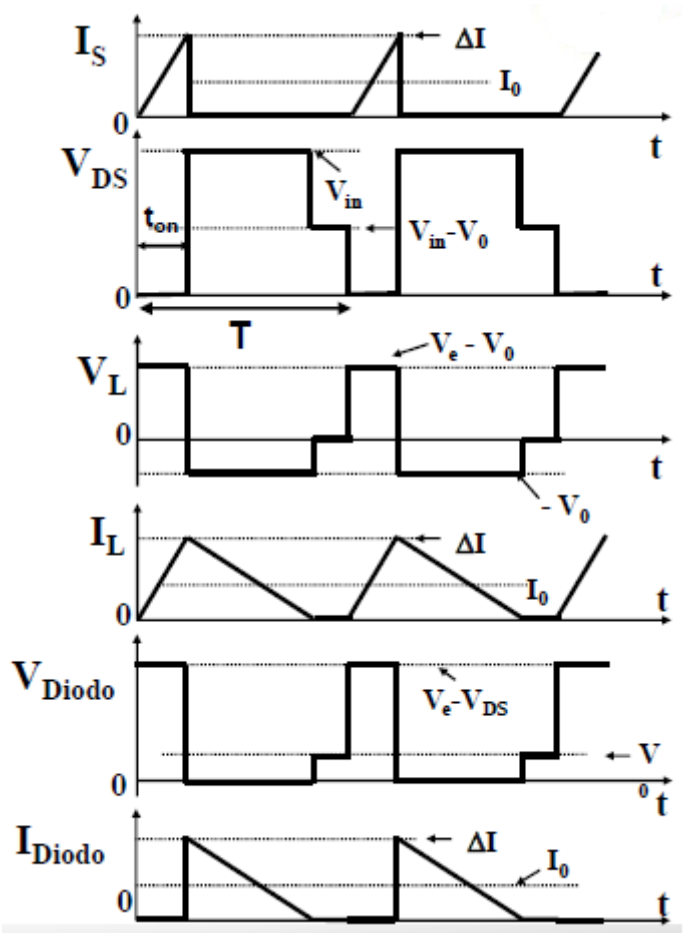

FIGURA 30. GRÁFICAS EN MCD

Para controlar la apertura y cierre del interruptor, que en este caso es un transistor, se utiliza un lazo simple de tensión.

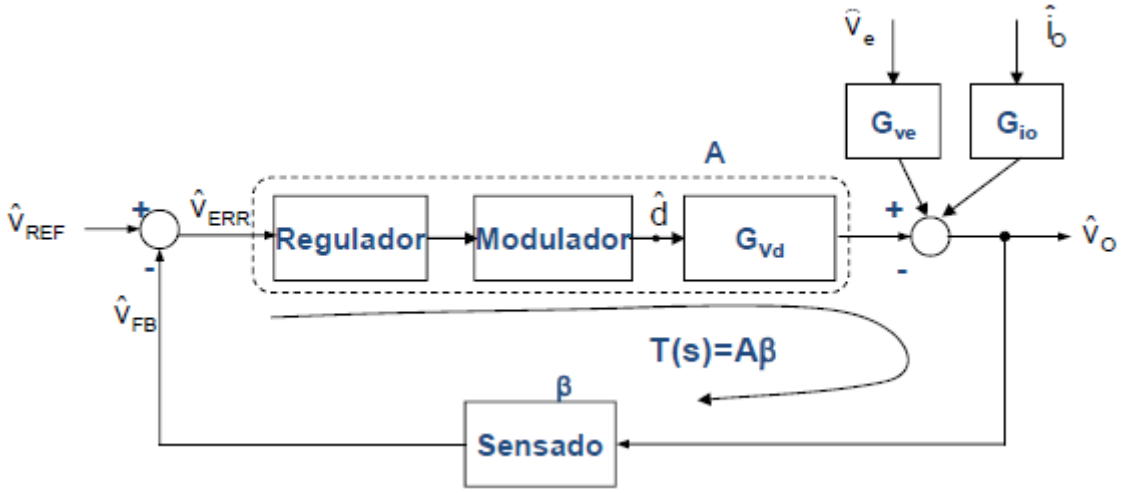

FIGURA 31. LAZO DE CONTROL

El lazo mide mediante un divisor de tensión el voltaje a la salida del convertidor Vo, que queda multiplicada por la ganancia β del sensor, en este caso será un divisor de tensión.

Una vez que se tiene una referencia de la tensión de la salida del convertidor se compara con una referencia de cómo debería ser esta tensión si fuese ideal, con ello se obtiene una señal que refiere al error que hay entre la tensión real y la ideal.

Seguidamente esa señal llega al regulador, que se encarga de generar una señal de salida de acuerdo con las especificaciones de diseño (cómo se quiere que se comporte el convertidor) y con el nivel de error de la entrada que lo corrija.

Más tarde esta señal del regulador llega al modulador, que es el encargado de generar el ciclo de trabajo del interruptor del convertidor a través de la comparación de la señal generada por el regulador y una referencia, que suele ser una señal triangular periódica.

Si hubiese, durante el funcionamiento normal del convertidor, cualquier tipo de perturbación que modificara la tensión de salida se volvería a tener éste proceso para corregirlo.

REALIMENTACIÓN FEED FORWARD

La realimentación Feed Forward tiene como misión eliminar de la función de transferencia de la planta el efecto de la tensión de entrada, que se presenta multiplicando, con lo cual se atenuará el efecto de las perturbaciones en la misma tantas veces como grande sea la tensión de entrada:

$$G_{Vd} = Ve \times (1/1+\frac{L}{R}s + LCs^2)$$

Este efecto se consigue dividiendo la salida del regulador entre el valor de la tensión de entrada y es ahí donde entrará en funcionamiento el divisor digital diseñado.

RESOLUCIÓN DEL PROBLEMA: USO DE PSIM, SMARTCTRL, MODELSIM Y MODCOUPLER

Para desarrollar el sistema de la aplicación que incluye el bloque divisor realizado en el convertidor reductor se utilizan varios programas CAD comerciales:

PSIM

PSIM es un programa que permite simular los circuitos eléctricos y electrónicos a través de una sencilla interfaz gráfica, además se tiene la opción colocar la aparamenta de medida virtual en cualquier parte del circuito que interese, por lo que se convierte en una herramienta capital a la hora de desarrollar el sistema.

En PSIM se implementará el reductor y el lazo de control que lo acompaña para regular el ciclo de trabajo.

Sin embargo en este caso no se utiliza la herramienta PSIM como tal sino una de sus aplicaciones: Smartctrl.

SMARTCTRL

Smartctrl es una herramienta de PSIM que encarga principalmente de seleccionar el lazo de control adecuado a cada convertidor de acuerdo a las especificaciones que se requieran del mismo.

Creando un sencillo modelo de un convertidor en PSIM y exportándolo o creándolo directamente en Smartctrl seleccionando el tipo de convertidor, el sensado y el tipo de regulador, Smartctrl exporta a PSIM el modelo del sistema con todos los componentes y el valor de los mismos.

Consta de un reductor típico de 12V de entrada a lo que se suman dos escalones de 7 y -9 V que se accionan a los 2 y 3 ms respectivamente que son los que valdrán para verificar que efectivamente el Feed Forward atenúa el efecto de éstos en la tensión de salida.

El circuito El circuito proporcionado por el grupo de Sistemas Electrónicos de Potencia es:

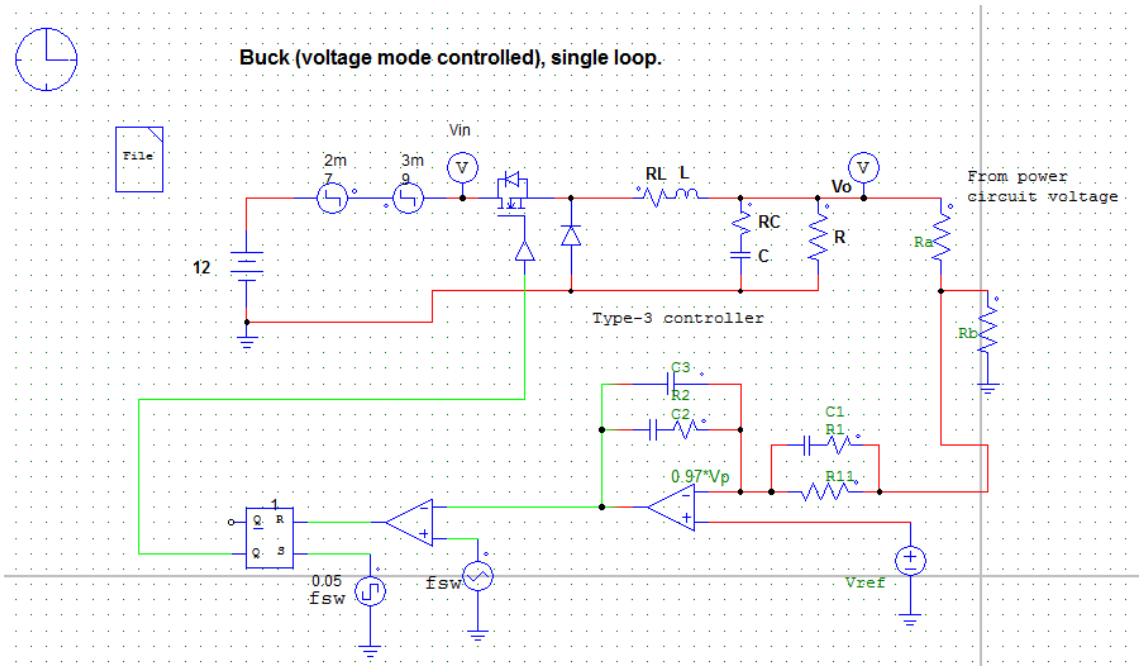

FIGURA 32. REDUCTOR CON LAZO DE CONTROL

Siendo estos los parámetros obtenidos de Smartctrl:

```
//SmartCtrl parameters
//Outer Regulator parameters
R1 = 1.9417k Ohm
C1 = 2.06574n F
C3 = 1.05022n F
R2 = 4.56084k Ohm
C2 = 5.40876n F
Vref = 2.5 V
Vp = 1 V
R11 = 10k Ohm
//Outer Sensor parameters
Ra = 82.2894 Ohm
Rb = 82.2894 Ohm
//Power Stage parameters
R = 1 Ohms
RC = 50m Ohms
C = 470u F
IC_C = 5 V
RL = 1n Ohms
L = 3u H
IC_L = 5 A
Vin = 12 V
//Modulator parameters
Vpp = 1 V
fsw = 250k Hz
Dramp = 975m
Vv = 0 V
//Other parameters
fdc = 16k Hz
fp1 = 39.6791k Hz
fp2 = 39.6791k Hz
fz1 = 6.45176k Hz
fz2 = 6.45176k Hz
```

FIGURA 33. PARÁMETROS DE SMARTCTRL

Simulando con PSIM se puede obtener la tensión de salida con el valor de sus parámetros:

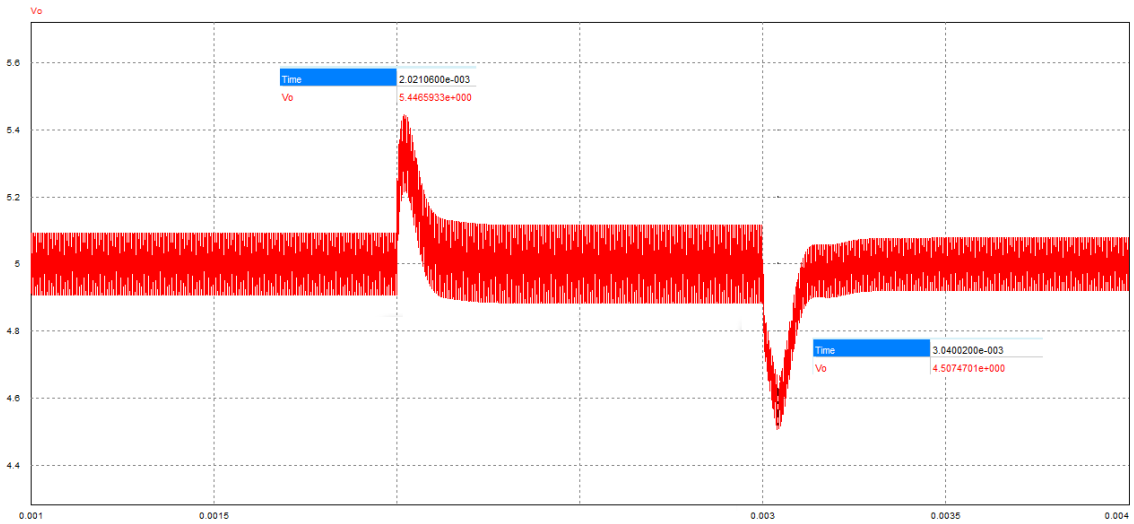

FIGURA 34. TENSIÓN DE SALIDA DEL REDUCTOR

Se ve que los picos de tensión en el momento de los escalones en la entrada son cercanos a 0.5 V

A continuación lo que se ha hecho es colocar la realimentación Feed-forward, o lo que es lo mismo, se divide la señal que sale del regulador entre la señal de entrada.

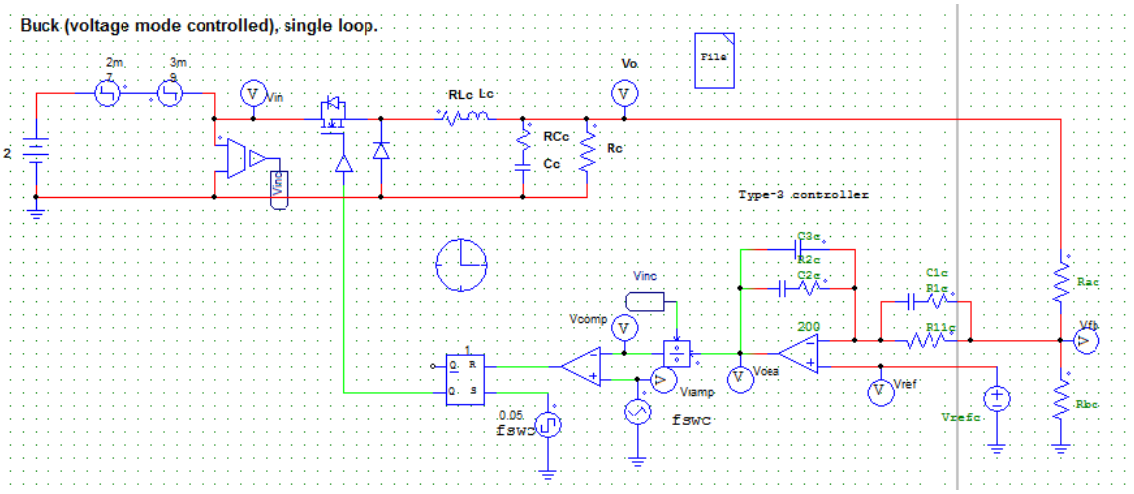

FIGURA 35. REDUCTOR CON FEED-FORWARD

Teniendo la siguiente gráfica de salida:

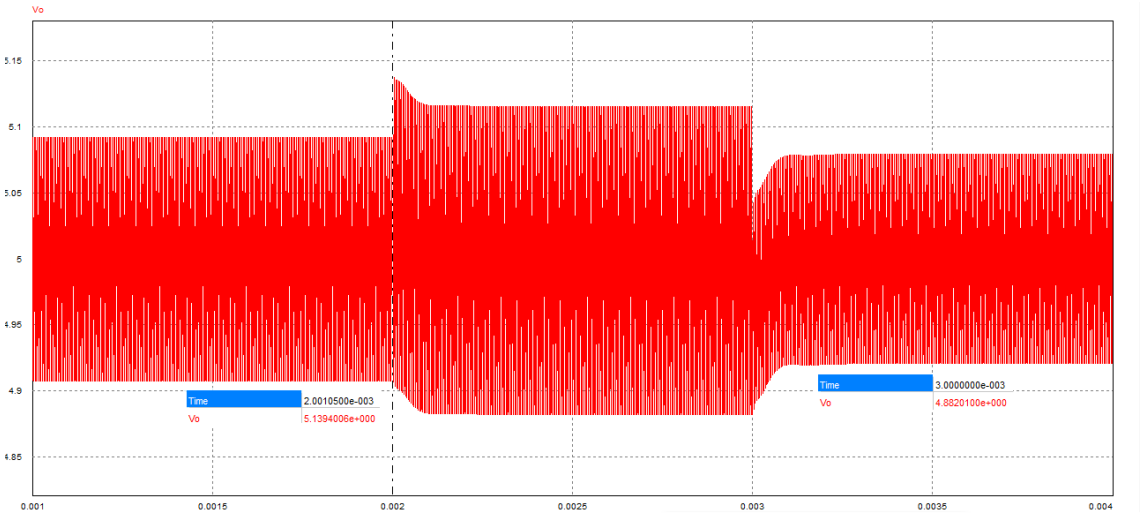

FIGURA 36. TENSIÓN DE SALIDA DEL REDUCTOR CON FEED-FORWARD

Donde se aprecia claramente que ahora los picos de tensión que reflejan los escalones de entrada son del orden de 0.15 V.

Una vez comprobado que la realimentación se ha colocado bien y que su efecto es visible se pasa a introducir en el esquema el divisor digital diseñado. Para ello se van a necesitar dos nuevas herramientas; ModeSim y ModCoupler.

MODELSIM

ModelSim es un simulador circuitos digitales desarrollados en lenguajes de descripción de *hardware* (VHDL u otros). Su utilización viene motivada por la siguiente herramienta: ModCoupler.

MODCOUPLER

ModCoupler es una herramienta de cosimulación que permite realizar pruebas en esquemas de PSIM que lleven elementos digitales en su circuito de control en VHDL y Verilog, que son simuladas por ModelSim.

En el caso actual la simulación analógica arranca con el reductor, pasa la medida del error a través del lazo de control hasta que llega a la conexión entre el regulador y el modulador, que es donde actúa el divisor digital, que toma la señal y, mediante la conexión con el simulador ModelSim, se realizará la división pertinente, se devuelve el cociente al simulador analógico PSIM que se carga en el modulador para que se compare con la señal triangular que genera el ciclo de trabajo.

Antes de realizar el modelo final en PSIM hay que hacer algunos cambios para que sea efectiva la simulación.

El primer cambio tiene que ver con la naturaleza del divisor: como el divisor trabaja con un dividendo siempre mayor que el divisor es necesaria una modificación en el esquema.

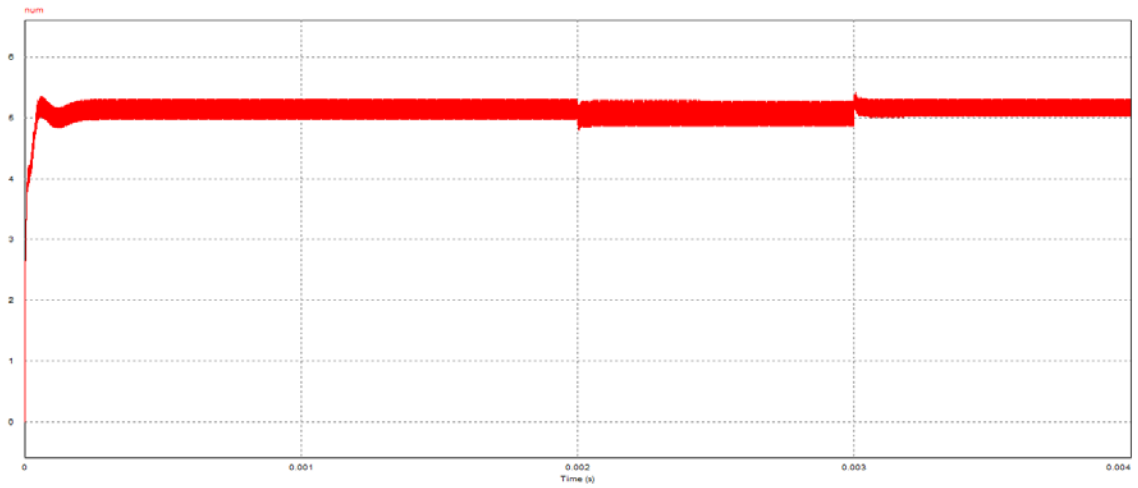

FIGURA 37. DIVIDENDO DEL COMPONENTE

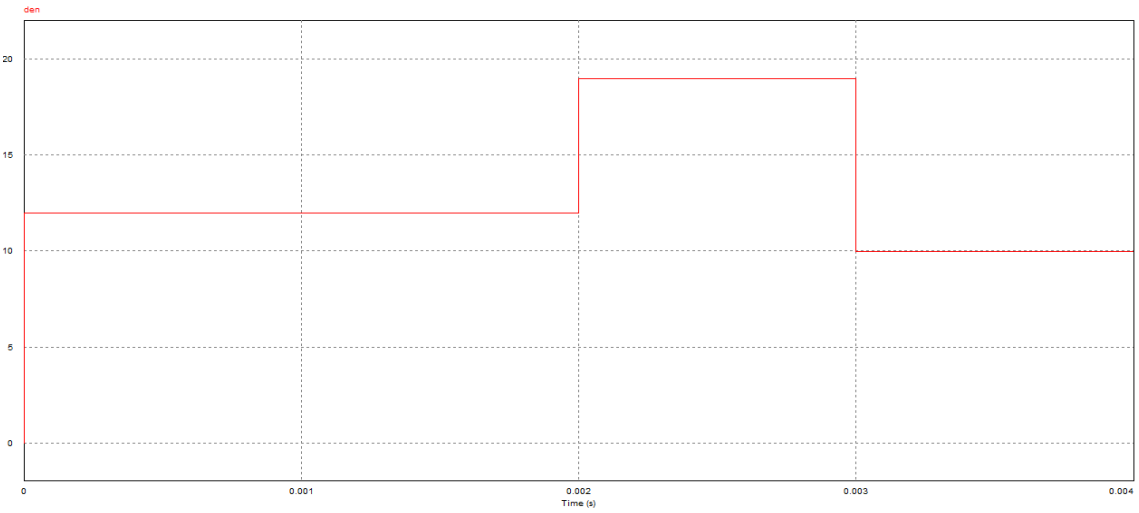

FIGURA 38. DIVISOR DEL COMPONENTE

La solución encontrada es multiplicar la señal de salida del regulador por 10, de manera que se tiene una señal en el dividendo en torno a 50V y una de divisor que se mantiene entre 10 y 19V, con lo cual el componente diseñado debe funcionar.

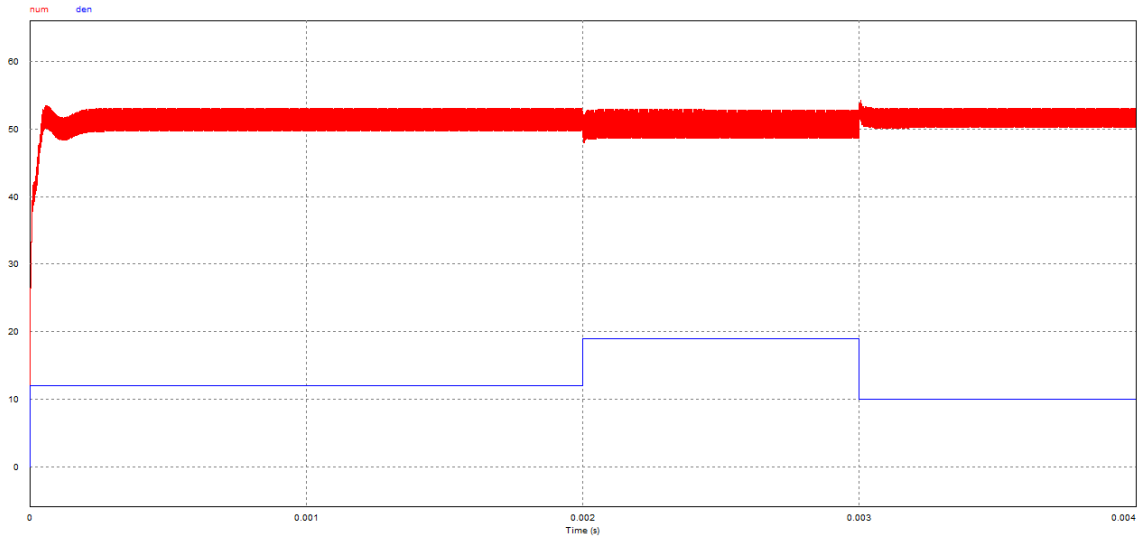

FIGURA 39. NUEVO DIVIDENDO COMPARADO CON EL DIVISOR

El siguiente problema es la necesidad de pasar la señal analógica que salga del regulador a un número digital que "entienda" el divisor y la posterior conversión del cociente a una señal analógica con la que pueda trabajar el modulador. Por lo tanto se hace necesaria la utilización de convertidores analógico-digitales y digital-analógicos en PSIM.

En pos de conseguir la máxima precisión en el circuito se utilizan los convertidores de 10 bits disponibles en PSIM (el resto son de 8 bits). Se dispone de la fórmula para sacar la ganancia en la pestaña "*Help*" de PSIM:

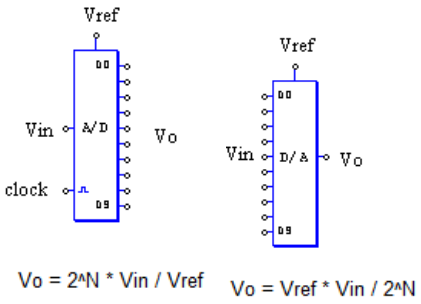

FIGURA 40. CONVERSORES A/D Y D/A CON SUS FÓRMULAS

Para conseguir esta máxima precisión se establecen como ganancias de los convertidores D/A 54.22 V y para el A/D 5.6 V.

Estos valores vienen de las señales de la simulación del sistema utilizando el divisor de PSIM, ya que con estas ganancias son las mínimas posibles para que las señales sean procesadas, y con ellas se utilizan los 10 bits de los convertidores internamente, con lo que se consigue la máxima resolución.

Como los cocientes que se obtienen internamente en el componente digital están entre 2 y 6 se necesitan aún más bits para trabajar con unas garantías mínimas de que los resultados serán aceptables.

Para ello se lleva a cabo una nueva modificación del diseño digital que consiste en:

-Eliminar la posibilidad de trabajar en “complemento a2”: ya que las señales de tensión con las que se trabajan son siempre positivas se elimina el trabajar en “complemento a2”, así no se da la posibilidad de que las entradas sean negativas y se tendrá el doble de rango de entrada.

- Multiplicar por 200 el numerador: para poder obtener una serie de cocientes que aprovechen los 10 bits de salida se añade un nuevo estado a la máquina del componente que es “multi”, donde se lleva a cabo el producto de la señal “numerador” por 200 y se carga en una nueva señal llamada “numerador_multi” que es un vector de 18 bits. Además se carga el denominador en una señal también de 18 bits llamada “denominador_multi”. Tras esto se modifica el código para operar con los nuevos tamaños de entrada.

Con este cambio se consigue que los cocientes lleguen a tener tres cifras en base decimal, con lo que se aumenta la resolución del componente. Más tarde con la ganancia del convertidor D/A se ajustan estos valores a los adecuados para la comparación con la señal del modulador.

- Además se debía modificar la máquina de estados ya que fue concebida para que cada vez que se hiciese una división se necesitase un pulso de “reset” para volver a hacer otra, sin embargo esto no es posible para esta aplicación ya que las divisiones deben hacerse de manera consecutiva. Por eso se cambiaron los estados “fin” y “error” para que acudiesen directamente a un nuevo estado llamado “reposo”, donde se reasignan los valores de las señales de la forma que si se hubiese producido un “reset”, tras esto se va directamente al estado “introducir_numeros” y se realiza otra división.

Tras estos cambios el circuito queda de la siguiente manera:

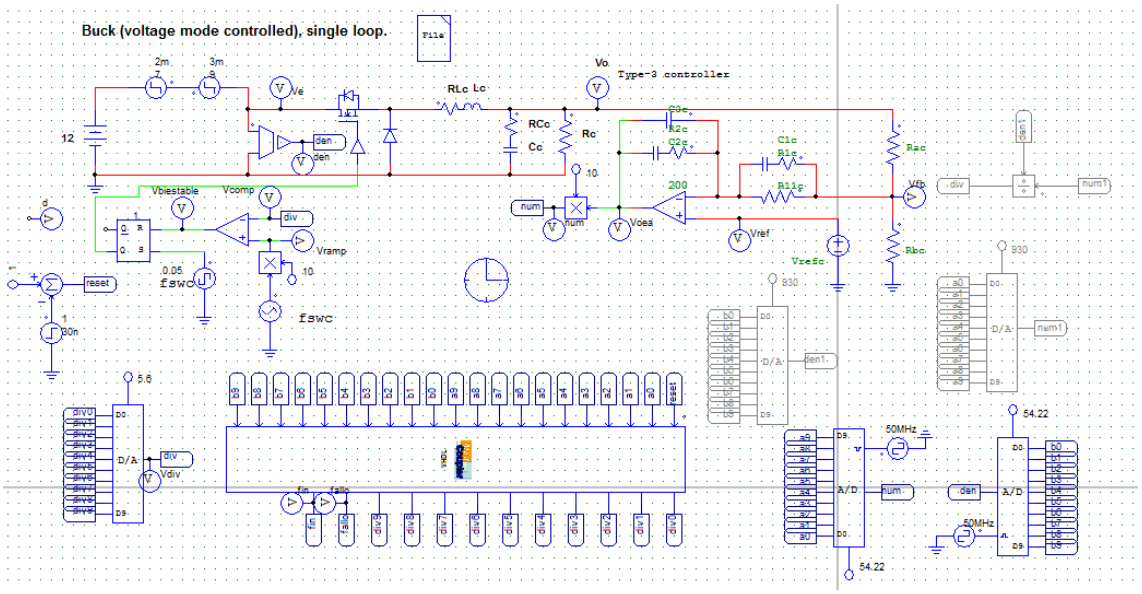

FIGURA 41. ESQUEMA FINAL – REDUCTOR CON FF Y DIVISOR DIGITAL

En la imagen se puede apreciar debajo del biestable una fuente escalón restando a una fuente de tensión de 1V que se utiliza para generar un pulso de “reset” al inicio de la simulación.

También en la parte derecha del esquema se ven 3 módulos comentados que se utilizaron para comprobar que no había problemas con los módulos de conversión de datos y así facilitar la depuración de errores. Además el módulo de mayor tamaño es el bloque de conexión con el simulador Modelsim. Dentro de él se configuran los parámetros para el correcto funcionamiento de la cosimulación:

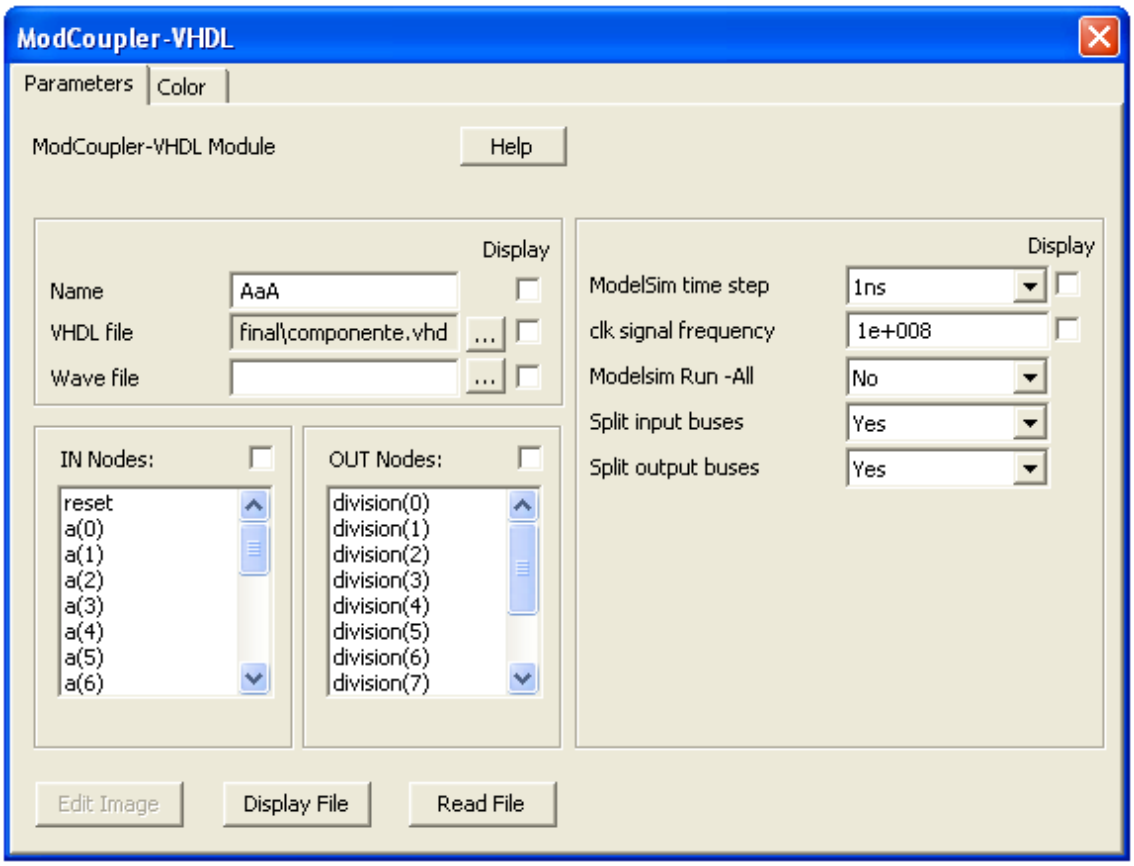

FIGURA 42. PARÁMETROS DE MODCOUPLER

En la figura 28 se muestra el cuadro de diálogo donde se especifica la frecuencia a la que trabaja el divisor, 100MHz, el *time step,* 1ns, se debe mostrar en PSIM un pin para cada uno de los bits de entrada y salida, ya que se debe trabajar con números en binario, si se debe seleccionar el botón "*Run-All*" en ModelSim antes de comenzar la simulación para permitir, si se desea, configurar una forma de onda de los procesos del circuito en digital durante la simulación para comprobar que se realizan correctamente. Además se le indica cuál es el archivo ".vhd" donde se encuentra el código, que previamente debe ser compilado en ModelSim.

Tras configurar los parámetros necesarios para asegurar la correcta interacción entr los simuladores analógico y digital, comienza la simulación, de la que se obtienen los siguientes resultados, que se van a comparar con los obtenidos en la simulación con el componente divisor que proporciona PSIM, que no es digital:

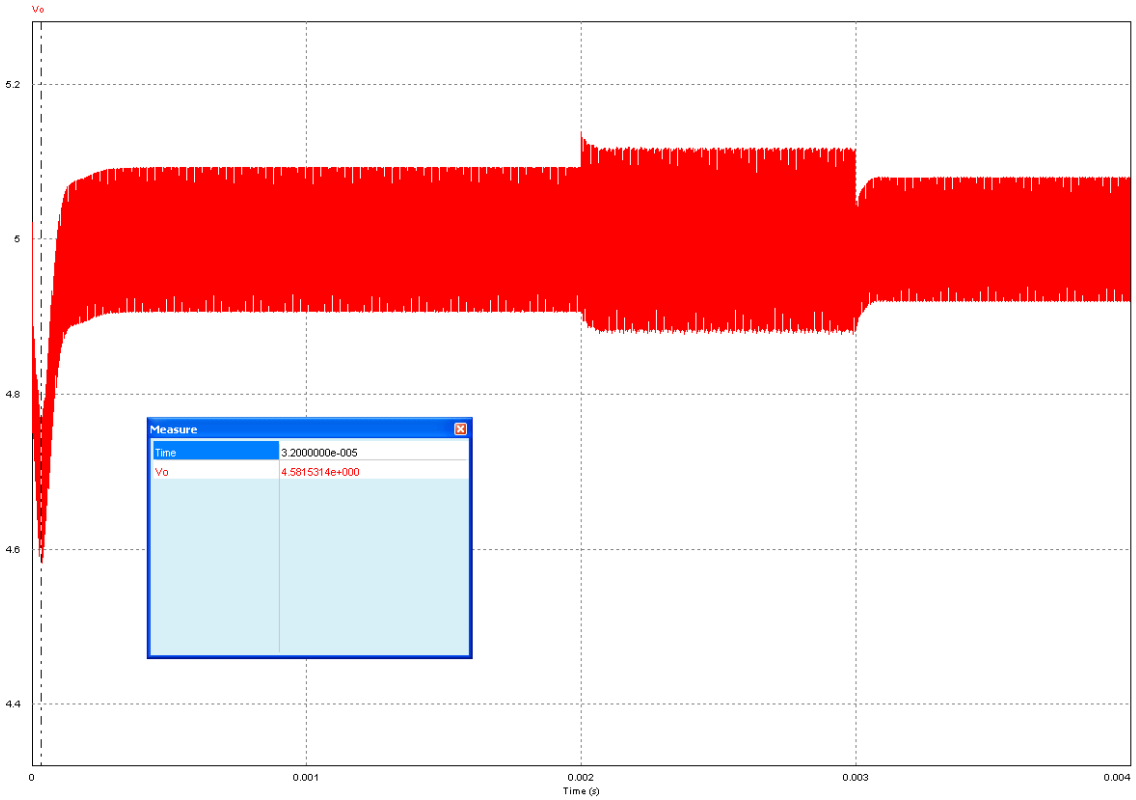

FIGURA 43. TENSIÓN DINÁMICA MÍNIMA CON EL COMPONENTE DISEÑADO

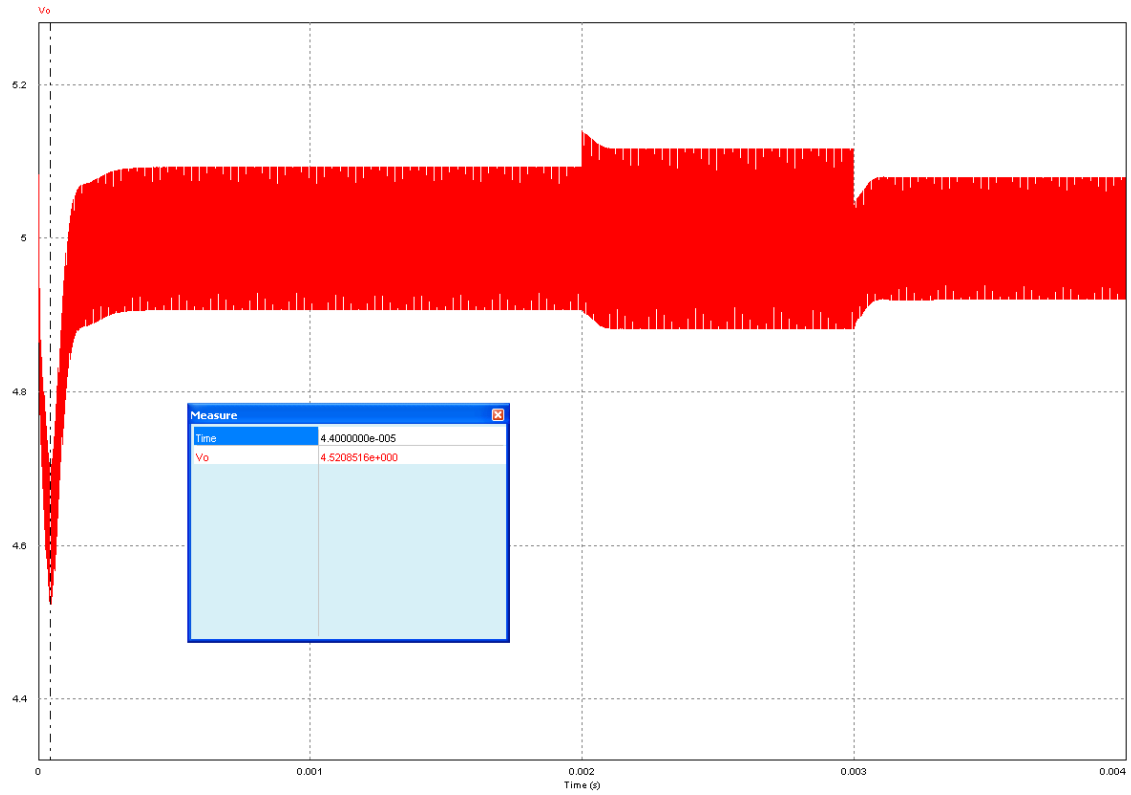

FIGURA 44. TENSIÓN DINÁMICA MÍNIMA CON EL COMPONENTE DE PSIM

En este primer par de gráficas se observan los valores de los picos de tensión en la dinámica del convertidor, que son los valores que marca la salida del convertidor desde el arranque

hasta que se estabiliza. Se ve que en la imagen utilizando el componente digital diseñado en VHDL el valor mínimo de la tensión es 4.5815V mientras que utilizando el de PSIM se tienen 4.5206V. Esto quiere decir que la utilización del divisor digital no afecta a la parte dinámica del convertidor, siendo un punto importante a tener en cuenta ya que unos picos de tensión excesivamente altos o bajos podrían afectar al circuito llegando incluso a dañar sus componentes.

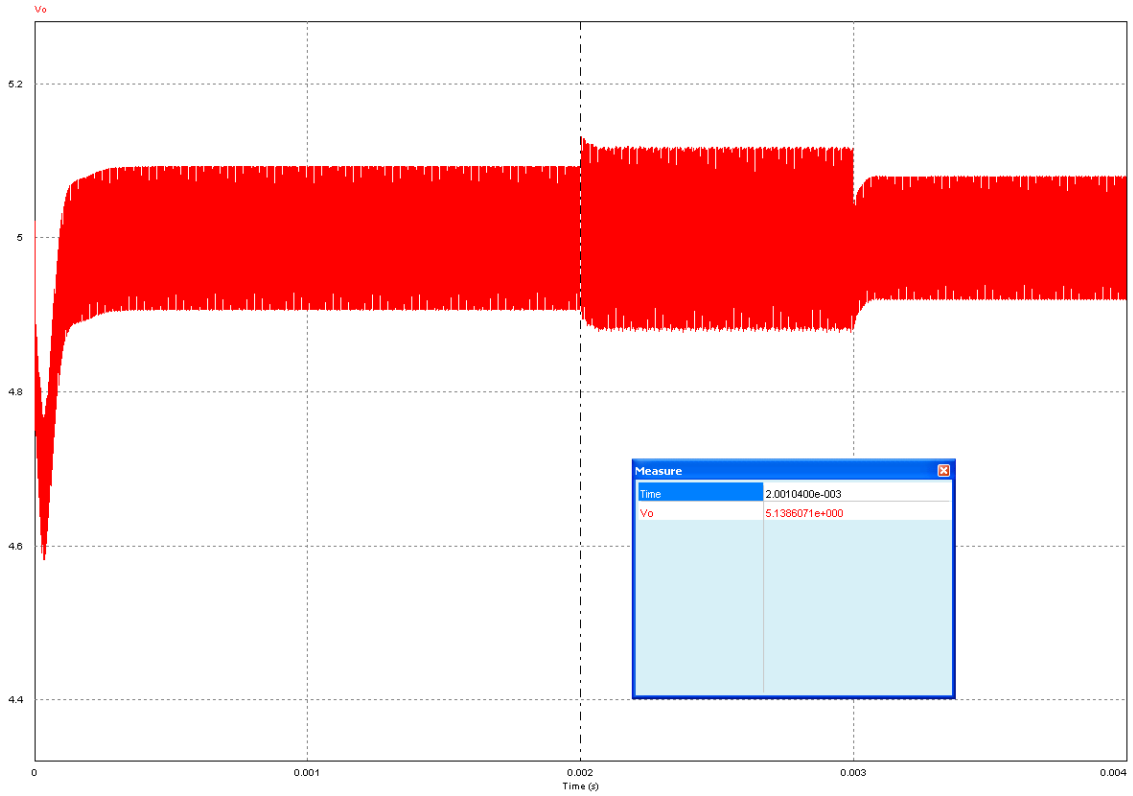

FIGURA 45. TENSIÓN MÁXIMA CON EL COMPONENTE DISEÑADO

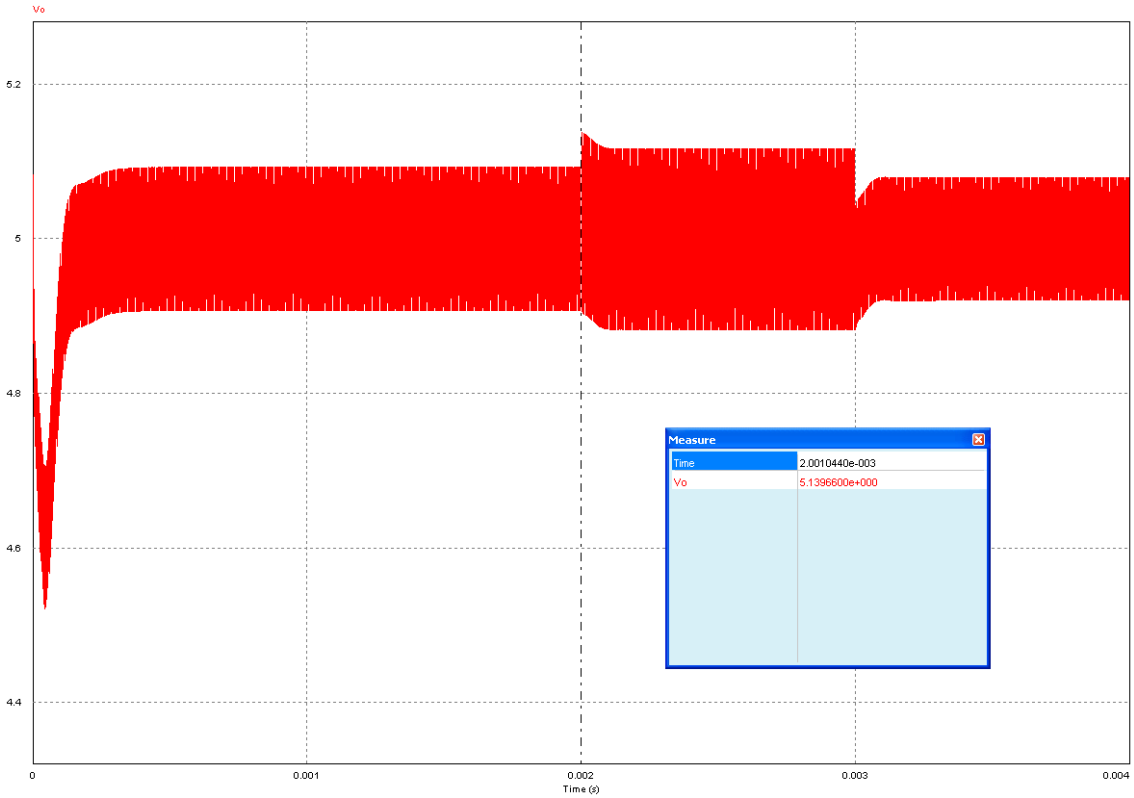

FIGURA 46. TENSIÓN MÁXIMA CON EL COMPONENTE DE PSIM

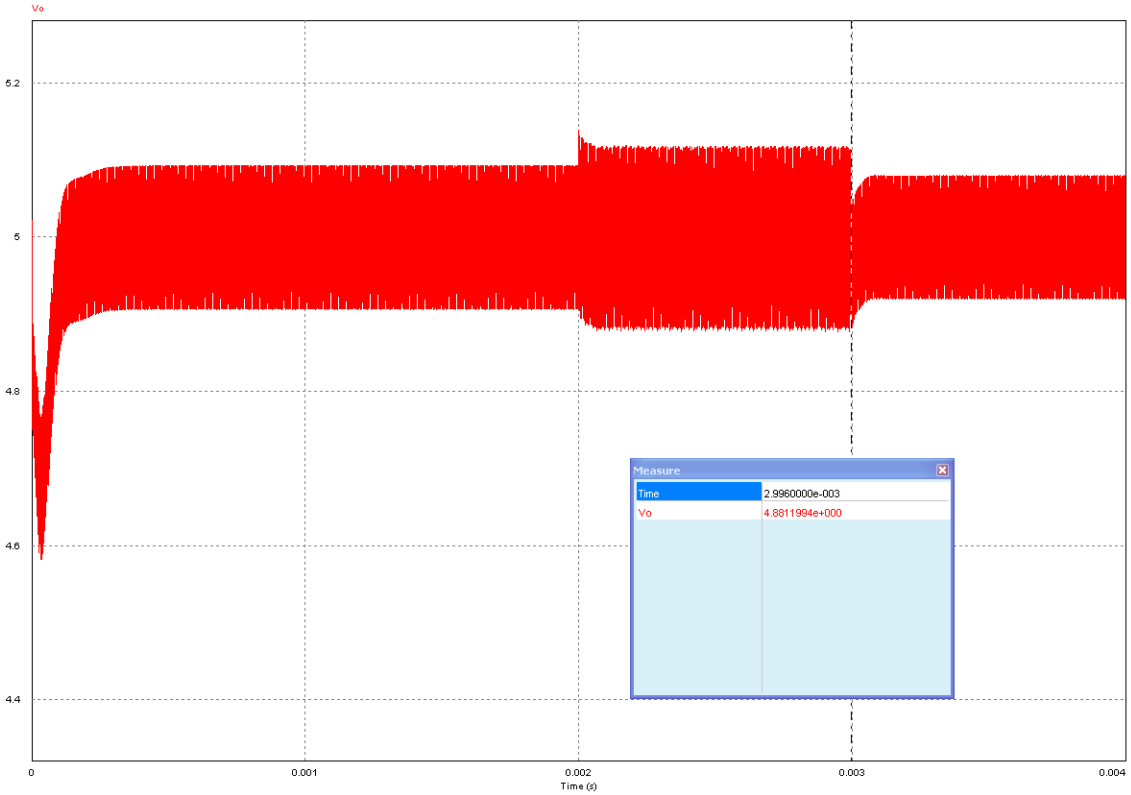

FIGURA 47. TENSIÓN MÍNIMA CON EL COMPONENTE DISEÑADO

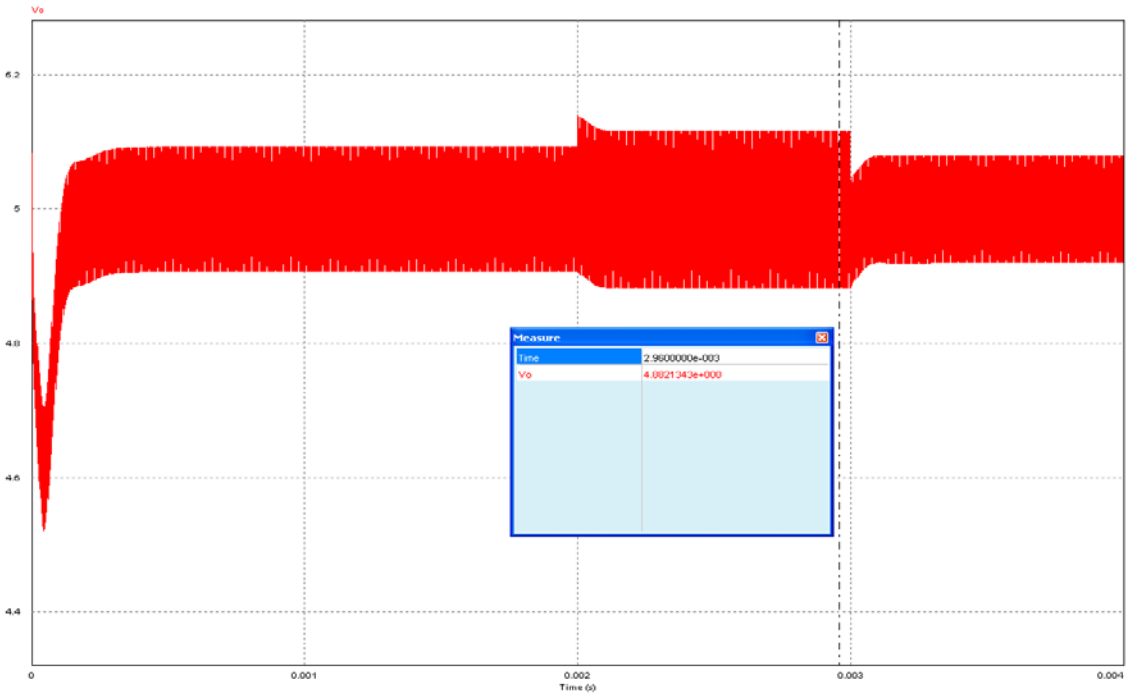

FIGURA 48. TENSIÓN MÍNIMA CON EL COMPONENTE DE PSIM

En estas últimas 4 imágenes se comparan los valores tanto máximos como mínimos alcanzados por la tensión de salida en los escalones. Son:

Vo_{max} con el divisor digital: 5.1386V

Vo_{min} con el divisor digital: 4.8811V

Vo_{max} con el divisor de PSIM: 5.1396V

Vo_{min} con el divisor de PSIM: 4.8821 V

Queda por tanto demostrado que los valores a los que llega la tensión de salida tanto máximos como mínimos son prácticamente iguales en ambos casos.

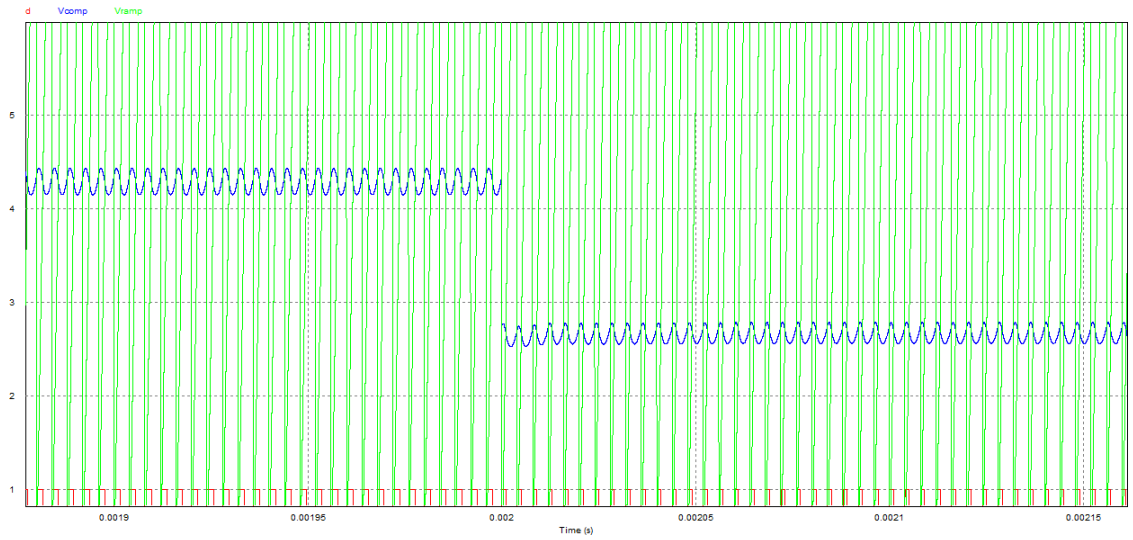

FIGURA 49. CICLO DE TRABAJO, SEÑAL MODULADORA Y SALIDA DEL DIVISOR DE PSIM

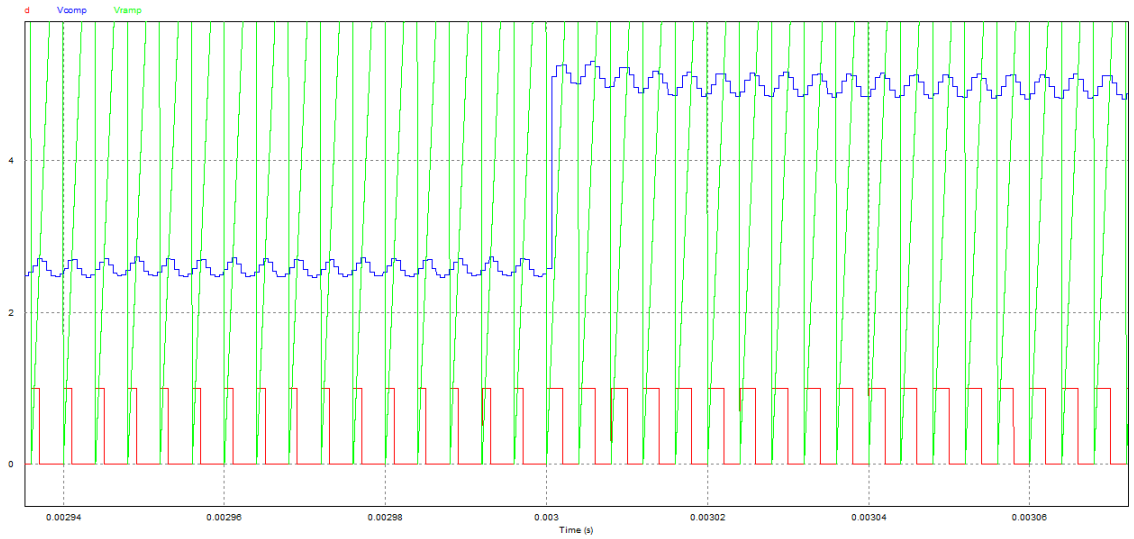

FIGURA 50: CICLO DE TRABAJO, SEÑAL MODULADORA Y SALIDA DEL COMPONENTE DISEÑADO

En estas dos gráficas se ha representado al mismo tiempo la señal de salida del divisor “Vcomp”, la señal triangular con la que se compara en el modulador “Vramp” y el ciclo de trabajo resultante de la comparación de ambos “d”.

En ellas se pude ver que la diferencia entre utilizar el divisor de PSIM y el digital: con el de PSIM se hacen divisiones inmediatas de números reales, con lo que no hay pérdida alguna de información y “Vcomp” es exacta, por lo tanto se obtiene el ciclo de trabajo óptimo. Sin embargo en el componente digital se ven los escalones de los que está compuesta la señal “Vcomp”.

Debido a las modificaciones hechas en el código para ganar precisión se obtuvo un componente más lento ya que eran necesarias un mayor número de iteraciones. Con un reloj de 50MHhz se pasó de una división cada 300ns aproximadamente a una cada 1000ns. Sin embargo la comparación de la salida del divisor es con la señal triangular de 250kHz de frecuencia, o 4us de periodo; por lo tanto da tiempo a realizar aproximadamente 4 divisiones por cada comparación con la señal triangular, aun así se decidió incrementar la frecuencia del divisor a 100MHz ya que con 50MHz se obtenía una mala respuesta a la salida en el momento del segundo escalón (la tensión caía por debajo de 4V antes de recuperarse). Por lo tanto la precisión obtenida con el trabajo dentro del divisor digital multiplicando el dividendo por 200 y el doblar la frecuencia del reloj del mismo han sido suficientes para obtener una respuesta a la altura de un requisito exigente.

CONCLUSIONES Y FUTURAS LÍNEAS DE TRABAJO

CONCLUSIONES

En este trabajo se planteó como objetivo el desarrollo de un divisor digital para integrarlo en el control de un convertidor.

Un divisor digital no es un componente de biblioteca en los circuitos digitales actuales, hay que realizarlo utilizando componentes más sencillos como sumadores, restadores, multiplexores, etc. El diseño de este divisor, su síntesis y su implementación eficiente es un trabajo que puede resultar muy beneficioso para la aceleración de funciones de control y de cálculo de complejidad media-alta

El trabajo hecho engloba el diseño de la arquitectura digital de un divisor que trabaja a partir de diferencias, con lo cual es sintetizable, la descripción del código VHDL de la arquitectura diseñada, así como la síntesis del mismo en una tecnología programable, FPGA, de Xilinx.

Para asegurar su funcionamiento se ha realizado una prueba en una FPGA incluida en una placa de desarrollo con elementos externos que ayudan a la depuración. Para ello se ha realizado el diseño de un bloque interfaz de entrada y salida para la captura de operandos y la visualización de resultados. Este bloque puede ser utilizado para futuras pruebas de otros diseños.

Por último se atiende a la utilidad desde la que se partía, integrar el divisor digital en un lazo de control de un convertidor, lo que se consigue aplicando la realimentación Feed-Forward en un convertidor reductor y comparando los resultados con los componentes de PSIM, llegando a la conclusión de que el componente implementado es perfectamente capaz de llevar a cabo las funciones que se le requieren.

TRABAJO FUTURO

Como propuesta de trabajo futuro se propone mejorar el diseño actual utilizando un número de bits mayor para mejorar aún más la precisión. Así mismo la utilización de distintos algoritmos que puedan aumentar la rapidez de funcionamiento, el área utilizada, la potencia de consumo, etc. Y así poder tener varios dispositivos que se adecúen a las necesidades de cada diseño.

PRESUPUESTO

Coste de personal			
Concepto	**Coste unitario**	**Cantidad**	**Importe**
Ing. Industrial	50 $^{€}/_{h}$	490 h	24.500,00 €
Coste total de personal			24.500,00 €

Recursos informáticos		
Concepto	**Cantidad**	**Importe**
Hardware: Ordenador	1	800,00 €
Hardware: FPGA Spartan XC3S200	1	200,00 €
Software: Licencia PSIM	1	2.000,00 €
Software: Paquete Smartcrtl	1	1.000,00 €
Software: Paquete ModCoupler	1	1.000,00 €
Software: Licencia ModelSim	1	2.000,00 €
Coste total		7.000,00 €

Resumen del presupuesto	
Concepto	**Importe total**
Personal	24.500,00 €
Recursos infomáticos	7.000,00 €
Impuestos (IVA 21%)	
Total presupuesto	**38.115,00€**

El presupuesto total de este proyecto asciende a un total de TREINTA Y UN OCHO MIL CIENTO QUINCE EUROS.

Getafe, a 12 de Junio de 2013.

Firmado: el ingeniero proyectista

Javier Calero Gómez

ANEXOS

CÓDIGO DEL DIVISOR EN VHDL PARA NÚMEROS CON CODIFICADOS EN COMPLEMENTO A2

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

entity divisor is
Port(clk  : in std_logic; --reloj
     Reset: std_logic;    --reset asincrono
     A    : in std_logic_vector (7 downto 0); --entrada para el
dividendo
     B    : in std_logic_vector (7 downto 0); --entrada para el
divisor
     resta: out std_logic_vector (8 downto 0); --salida del
cociente
     fallo: out std_logic; --salida que indica que
dividendo>divisor
     fin  : out std_logic); --salida de fin de operacion
end divisor;
architecture Behavioral of divisor is
signal test              : std_logic_vector(4 downto 0); --apoyo
TYPE estado IS (introducir_numeros,variables_auxiliares, restar,
evaluar_acarreo,acciones_acarreo,error,final,colocar,ca2_num1,ca
2_num2,ca2_den1,ca2_den2,ca2_coc,ca2_coc2,signo);
signal estado_actual, estado_siguiente: estado; --maquina de
estados
signal numerador         : std_logic_vector (7 downto 0);--
almacena el numerador (registro)
signal denominador       : std_logic_vector (7 downto 0);--
almacena el denominador (registro con desplazamiento)
signal cociente          : std_logic_vector (8 downto 0); --
almacena el cociente
signal diferencia        : std_logic_vector (8 downto 0); --
almacena las sucesivas diferecias
signal tamanio_num       : integer range 0 to 8; --cuenta del
tamaño del dividendo
signal tamanio_den       : integer range 0 to 8; --cuenta del
tamaño del divisor
signal enable_1          : std_logic; --es el enable que permite
que la diferencia pase a ser el nuevo numerador.
signal enable_2          : std_logic; --es el enable que añade
al cociente el acarreo.
signal resta_tam         : integer range 0 to 8; --para hacer la
resta de los tamaños que si no no me deja compararlos
signal division_terminada: std_logic;
signal numerador1        : std_logic_vector (8 downto 0); --
'0'+dividendo
```

```
signal denominador1      : std_logic_vector (8 downto 0); --
‘0’+divisor
signal error1: std_logic; --señal de error
signal x                 : std_logic; --para ver si el cociente
es negativo y hacer el cambio cA2
signal y                 : std_logic; --para ver si el cociente
es negativo y hacer el cambio cA2


begin
process (clk, reset)
begin
if reset = '1' then
numerador          <="00000000";     --valores de inicializacion
denominador        <="00000000";
numerador1         <="000000000";
denominador1       <="000000000";
cociente           <="000000000";
diferencia         <="000000000";
tamanio_den        <= 7;
tamanio_num        <= 7;
enable_1           <='0';
enable_2           <='0';
resta_tam          <= 0;
division_terminada <='0';
error1             <='0';
x                  <='0';
y                  <='0';
  elsif clk'event and clk = '1' then
    resta_tam<= tamanio_num-tamanio_den;
     if test = "00000" then  --introducir_numeros
       numerador   <= A;
       denominador <= B;
     end if;
     if test = "01000" then   --ca2_num1
       numerador<=numerador-1;
       x<='1';
     end if;
     if test ="01001" then    --ca2_num2
       for i in 0 to 7 loop
          numerador(i)<= not numerador(i);
       end loop;
     end if;
     if test = "01010" then   --ca2_den1
       denominador<=denominador-1;
       y<='1';
     end if;
     if test="01011" then     --ca2_den2
       for j in 0 to 7 loop
         denominador(j)<= not denominador(j);
       end loop;
     end if;
     if test = "00010" then              --colocar
       if numerador(7) ='0' then
         numerador   <= numerador(6 downto 0) & '0';
         tamanio_num <= tamanio_num-1;
```

```
        end if;
        if denominador(7) ='0' then
          denominador <= denominador(6 downto 0) & '0';
          tamanio_den <= tamanio_den-1;
         end if;
      end if;
      if test = "00011" then
        numerador1<='0' & numerador;
        denominador1<='0' & denominador;
      end if;
      if test = "00100" then --restar
        diferencia <= numerador1 - denominador1;
      end if;

      if test = "00101" then      --evaluar_acarreo
        if diferencia(8) = '1' then
          enable_1 <='0';
          enable_2 <='0';
        elsif diferencia(8) = '0' then
          enable_1 <='1';
          enable_2 <='1';
        end if;
      end if;
      if test = "00110" then          --acciones_acarreo
        if enable_1 = '1' then
          for i in 0 to 7 loop
           numerador (i) <= diferencia (i);
          end loop;
        end if;
        if enable_2 = '1' then
          cociente <= cociente (7 downto 0) & '1' ;
        else
          cociente <= cociente (7 downto 0) & '0' ;
        end if;
        denominador <= '0' & denominador (7 downto 1);
        tamanio_den <= tamanio_den+1; --para ver lo de si sigue
haciendo la resta o no
      end if; -- end if del estado 00110
      if test = "00111" then  --fin
        division_terminada <= '1';
      end if;    --se cierra el if de los ciclos de reloj.
      if test = "10000" then  --estados de cambio a complement A2
        for l in 0 to 8 loop
          cociente(l) <= not cociente(l);
        end loop;
      end if;
      if test = "10001" then
        cociente <= cociente+1;
      end if;
      if test = "01111" then     --error
                error1 <= '1';
           else
                error1 <= '0';
           end if;

end if;  --se cierra el if del reset.
```

```
end process;
fallo <= error1;             --se cargan las señales a la salida
fin <= division_terminada;
resta <= cociente;

process (clk,reset)          --maquina de estados
begin
     if reset = '1'  then
          estado_actual <= introducir_numeros;
     elsif clk'event and clk='1' then
          estado_actual <= estado_siguiente;
     end if;
end process;

process (estado_actual,estado_siguiente, resta_tam, numerador,
denominador, cociente)
begin
  case estado_actual is
    when introducir_numeros => test <= "00000";  --
introducir_numeros
      estado_siguiente <= signo;
    when signo => test <= "00001";         --signo
       if numerador(7)='1' then
         estado_siguiente <= ca2_num1;
       elsif denominador(7)='1' then
         estado_siguiente <= ca2_den1;
       else
         estado_siguiente <= colocar;
       end if;
    when ca2_num1 => test <= "01000";          --CA2
       estado_siguiente <= ca2_num2;
    when ca2_num2 => test <= "01001";
      estado_siguiente <= signo;
    when ca2_den1 => test <="01010";
      estado_siguiente <= ca2_den2;
    when ca2_den2 => test <= "01011";
      estado_siguiente <= signo;
    when colocar => test <= "00010";       --colocar
       if numerador<denominador then
         estado_siguiente <=error;
       elsif numerador(7)='1' and denominador(7)='1' then
         estado_siguiente <= variables_auxiliares;
       else
         estado_siguiente <= colocar;
       end if;
    when variables_auxiliares => test <= "00011";
--variables_auxiliares
      estado_siguiente <= restar;
    when restar => test <= "00100";                  --restar
       estado_siguiente <= evaluar_acarreo;
    when evaluar_acarreo => test <= "00101";  --evaluar_acarreo
       estado_siguiente <= acciones_acarreo;
    when acciones_acarreo => test <= "00110"; --acciones_acarreo
       if resta_tam = 0 then
         if (x='1' and y='0') or (x='0' and y='1') then
           estado_siguiente <= ca2_coc;
```

```
         else
           estado_siguiente <= final;
         end if;
       elsif resta_tam > 0 then
         estado_siguiente <= variables_auxiliares;
           end if;
      when ca2_coc => test <="10000";
        estado_siguiente <=ca2_coc2;
      when ca2_coc2 => test <="10001";
        estado_siguiente <=final;
      when final => test <= "00111";              --final
        if reset ='1' then
          estado_siguiente <= introducir_numeros;
        else
         estado_siguiente <= final;
      end if;
     when error => test <= "01111";                  --error
        if reset ='1' then
          estado_siguiente <= introducir_numeros;
            else
           estado_siguiente <= error;
            end if;
  end case;
end process;
end Behavioral;
```

CÓDIGO DEL TEST-BENCH

```
--generado automáticamente except la declaracion de las acciones
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
ENTITY tb IS
END tb;
ARCHITECTURE behavior OF tb IS
-- Component Declaration for the Unit Under Test (UUT)
COMPONENT divisor
PORT(
clk   : IN  std_logic;
reset : IN  std_logic;
A     : IN  std_logic_vector(7 downto 0);
B     : IN  std_logic_vector(7 downto 0);
resta : OUT  std_logic_vector(8 downto 0);
fallo : OUT  std_logic;
fin   : OUT  std_logic
       );
END COMPONENT;
--Inputs
signal clk   : std_logic := '0';
signal reset : std_logic := '0';
signal A     : std_logic_vector(7 downto 0) := (others => '0');
signal B     : std_logic_vector(7 downto 0) := (others => '0');
--Outputs
signal resta : std_logic_vector(8 downto 0);
```

```
signal fallo : std_logic;
signal fin : std_logic;
-- Clock period definitions
constant clk_period : time := 10 ns;
BEGIN
-- Instantiate the Unit Under Test (UUT)
uut: divisor PORT MAP (
clk   => clk,
reset => reset,
A     => A,
B     => B,
resta => resta,
fallo => fallo,
fin   => fin
);


reset<='1','0' after 100 ns; --declaracion de las acciones del
test bench
A<="00011001";
B<="11111011";
-- Clock process definitions
process (clk)
begin
clk <= not clk after 10 ns;
end process;
-- Stimulus process
stim_proc: process
begin
- hold reset state for 100 ns.
wait for 100 ns;
wait for clk_period*10;
-- insert stimulus here
wait;
end process;
END;
```

CÓDIGO DE "CONTADOR_DISPLAY"

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
entity contador_display is

Port(clk   : in std_logic;  --reloj
Reset      : in std_logic;  --reset asincrono
pin_display: out std_logic_vector (3 downto 0);  --se elige el
display a iluminar
mux        : out std_logic_vector(1 downto 0)); -- indica qué
display se va a mostrar
end contador_display;
architecture Behavioral of contador_display is
```

```
signal contador: std_logic_vector(7 downto 0); --sirve para
dejar iluminado un cierto tiempo cada display
signal vector: std_logic_vector(3 downto 0);
signal display: std_logic_vector(1 downto 0);
begin
process(clk, reset)
begin
if reset='1' then
  contador<="00000000";         --valores de inicializacion
  vector  <="1111";
  display <="00";
  elsif clk'event and clk='1' then
    if contador = "11111111" then
     contador<= "00000000";
    else
     contador<=contador+1;
    end if;
    if contador="01010101" then
     vector<="1011";
     display<="00";
     elsif contador="10101010" then
       vector<="1101";
       display<="01";
     elsif contador="11111111" then
       vector<="1110";
       display<="10";
     end if;
  end if;
end process;
mux        <=display;    -asignacion a las salidas
pin_display<=vector;
end Behavioral;
```

CÓDIGO DE "ANTIRREBOTES"

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

entity antirrebotes is
Port(clk         : in std_logic;   --reloj
     reset       : in std_logic;   --reset asincrono
     boton       : in std_logic;   --entrada active cuando se
pulsa el boton de cambio de estado del component de alto nivel
     boton_anti  : out std_logic); --salida conformada
end antirrebotes;
architecture Behavioral of antirrebotes is
signal contador   : std_logic_vector (3 downto 0); --tiempo que
se debe esperar hasta que se acepte un nuevo pulso de entrada
signal boton_anti1: std_logic;   --señal del antirrebotes
signal enable     : std_logic;   --permite bloquear la
aceptacion de un nuevo pulso o pulsos de entrada
```

```
process(clk, reset)                --el proceso permite que solo
se genere un pulso de salida cuando se produzca una entrada, sea
cual sea, durante cierto tiempo
begin
if reset = '1' then
  contador   <=(others=>'0');
    boton_anti1 <='0';
     enable     <='0';
     elsif clk'event and clk='1' then
       if boton ='1' and contador="0000" then
         enable<='1';
       end if;
       if enable='1' then
         if contador="1111" then
           contador<="0000";
           enable<='0';
        else
          contador<=contador+1;
        end if;
       end if;
       if contador="0010" then
         boton_anti1<='1';
       else
         boton_anti1<='0';
        end if;
   end if;
end process;
boton_anti <= boton_anti1;
end Behavioral;
```

CÓDIGO DE "CONVERSOR_7SEG"

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

entity conversor_7seg is
Port(clk        : in std_logic;  --reloj
     reset      : in std_logic;  --reset asincrono
     fin        : in std_logic;  --entrada de activacion del
componente
     cocienteBCD: in std_logic_vector (7 downto 0); --entrada
del cociente que proviene del divisor
     centenas   : out std_logic_vector (6 downto 0); --centenas
del cociente en BCD
     decenas    : out std_logic_vector (6 downto 0); --decenas
del cociente en BCD
     unidades   : out std_logic_vector (6 downto 0)); --unidades
del cociente en BCD
end conversor_7seg;
architecture Behavioral of conversor_7seg is
```

```
function BCD (a: in std_logic_vector(3 downto 0)) return
std_logic_vector is
  variable v: std_logic_vector (6 downto 0);
  begin
    case a is
     when        "0000"=>v:="0000001";
     when        "0001"=>v:="1001111";
     when        "0010"=>v:="0010010";      --funcion para
iluminar los pines adecuados de los displays
     when        "0011"=>v:="0000110";
     when        "0100"=>v:="1001100";
     when        "0101"=>v:="0100100";
     when        "0110"=>v:="1100000";
     when        "0111"=>v:="0001111";
     when        "1000"=>v:="0000000";
     when        "1001"=>v:="0001100";
     when others     =>v:="1111111";
    end case;
  return v;
end BCD;
function to_bcd ( bin : std_logic_vector(7 downto 0) ) return –
funcion del pado a BCD
std_logic_vector is
variable i    : integer:=0;
variable bcd  : std_logic_vector(11 downto 0) := (others =>
'0');
variable bint : std_logic_vector(7 downto 0) := bin;
begin
     for i in 0 to 7 loop  -- repeating 8 times.
     bcd(11 downto 1) := bcd(10 downto 0);  --shifting the bits.
     bcd(0) := bint(7);
     bint(7 downto 1) := bint(6 downto 0);
     bint(0) :='0';
       if(i < 7 and bcd(3 downto 0) > "0100") then --add 3 if
BCD digit is greater than 4.
       bcd(3 downto 0) := bcd(3 downto 0) + "0011";
       end if;
       if(i < 7 and bcd(7 downto 4) > "0100") then --add 3 if
BCD digit is greater than 4.
       bcd(7 downto 4) := bcd(7 downto 4) + "0011";
       end if;
       if(i < 7 and bcd(11 downto 8) > "0100") then  --add 3 if
BCD digit is greater than 4.
       bcd(11 downto 8) := bcd(11 downto 8) + "0011";
       end if;
end loop;
return bcd;
end to_bcd;
signal int_cociente: integer;
signal centenas1   : integer range 0 to 9;
signal decenas1    : integer range 0 to 9;
signal unidades1   : integer range 0 to 9;
signal bcd_binary  : std_logic_vector (11 downto 0);
type estado is(reposo,convertir);
signal estado_actual, estado_siguiente: estado;
begin
```

```
process(clk,reset)
begin
  if reset='1' then
    centenas1    <=0;
    decenas1     <=0;
    unidades1    <=0;
    int_cociente <=0;
       bcd_binary   <="000000000000";
         else
           if clk'event and clk ='1' then
             if estado_actual = convertir then
                 bcd_binary <= to_bcd(cocienteBCD);
             end if;
           end if;
     end if;
end process;
process (clk,reset)          --maquina de estados
begin
  if reset = '1'  then
    estado_actual <= reposo;
  elsif clk'event and clk='1' then
    estado_actual <= estado_siguiente;
  end if;
end process;
process(fin, estado_actual,estado_siguiente)
begin
  case estado_actual is
    when reposo =>
      if fin='1'  then
        estado_siguiente <= convertir;
      else
        estado_siguiente <= reposo;
      end if;
      when convertir =>
        estado_siguiente <= reposo;
      end case;
end process;
centenas <=BCD(bcd_binary(11 downto 8));
decenas  <=BCD(bcd_binary(7 downto 4));
unidades <=BCD(bcd_binary(3 downto 0));
end Behavioral;
```

CÓDIGO DEL ARCHIVO .UCF

```
NET Clk TNM_NET = Clk;
TIMESPEC TS_Clk = PERIOD Clk 20 ns HIGH 50 %;
NET clk LOC = T9;
NET reset LOC = L14;
NET entrada<7> LOC = K13;  --asignacion de pines de la FPGA
NET entrada<6> LOC = K14;
NET entrada<5> LOC = J13;
NET entrada<4> LOC = J14;
NET entrada<3> LOC = H13;
NET entrada<2> LOC = H14;
```

```
NET entrada<1> LOC = G12;
NET entrada<0> LOC = F12;
NET boton LOC = M13;
NET pin_display<3> LOC = E13;
NET pin_display<2> LOC = F14;
NET pin_display<1> LOC = G14;
NET pin_display<0> LOC = D14;
NET BCD<6> LOC = E14;
NET BCD<5> LOC = G13;
NET BCD<4> LOC = N15;
NET BCD<3> LOC = P15;
NET BCD<2> LOC = R16;
NET BCD<1> LOC = F13;
NET BCD<0> LOC = N16;
NET signo LOC = K12;
NET fallo LOC = P14;
```

CÓDIGO DEL DIVISOR MODIFICADO PARA EL LAZO DE CONTROL

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
--se diferencia del primero en las partes referidas en la
memoria en la sección “resolución del problema”
entity divisor is
Port(clk: in std_logic;
reset   : in std_logic;
A       : in std_logic_vector (9 downto 0);
B       : in std_logic_vector (9 downto 0);
division: out std_logic_vector (9 downto 0);
fallo   : out std_logic;
fin     : out std_logic);
end divisor;
architecture Behavioral of divisor is
TYPE estado IS (introducir_numeros,variables_auxiliares, restar,
evaluar_acarreo,acciones_acarreo,error,final,colocar,reposo,colo
car2,multi);
signal estado_actual, estado_siguiente: estado;
signal numerador         : std_logic_vector (9 downto 0);--
almacena el numerador (registro)
signal denominador       : std_logic_vector (9 downto 0);--
almacena el denominador (registro con desplazamiento)
signal cociente          : std_logic_vector (18 downto 0);
signal diferencia        : std_logic_vector (18 downto 0);
signal tamanio_num       : integer;-- range 0 to 8;
signal tamanio_den       : integer;-- range 0 to 8;
signal enable_1          : std_logic; --es el enable que permite
que la diferencia pase a ser el nuevo numerador.
signal enable_2          : std_logic; --es el enable que añade
al cociente el acarreo.
signal resta_tam         : integer;-- range 0 to 8; --para hacer
la resta de los tamaños que si no no me deja compararlos
```

```
signal division_terminada: std_logic;
signal numerador_multi   : std_logic_vector (17 downto 0);
signal denominador_multi : std_logic_vector (17 downto 0);
signal numerador1        : std_logic_vector (18 downto 0);
signal denominador1      : std_logic_vector (18 downto 0);
signal error1            : std_logic;
signal cociente1         : std_logic_vector (9 downto 0);
begin
process (clk, reset)
begin
  if reset = '1' then
    numerador          <="0000000000";
    denominador        <="0000000000";
    numerador1         <="00000000000000000000";
    denominador1       <="00000000000000000000";
    cociente           <="00000000000000000000";
    diferencia         <="00000000000000000000";
    numerador_multi    <="00000000000000000000";
    denominador_multi  <="00000000000000000000";
    tamanio_den        <= 17;
    tamanio_num        <= 17;
    enable_1           <='0';
    enable_2           <='0';
    resta_tam          <= 0;
    division_terminada <='0';
    error1             <='0';
    cociente1          <="0000000000";
  elsif clk'event and clk = '1' then
    resta_tam<= tamanio_num-tamanio_den;
     if estado_actual = introducir_numeros then  --
introducir_numeros
      numerador   <= A;
      denominador <= B;
      end if;
    if estado_actual=multi then
      numerador_multi <= (numerador*"11001000");
      denominador_multi <= "00000000"&denominador;
      end if;
    if estado_actual = colocar2 then              --colocar2
      if numerador_multi(17) ='0' then
        numerador_multi   <= numerador_multi(16 downto 0) & '0';
        tamanio_num       <= tamanio_num-1;
      end if;
      if denominador_multi(17) ='0' then
      denominador_multi <= denominador_multi(16 downto 0) & '0';
      tamanio_den <= tamanio_den-1;
      end if;
     end if;
     if estado_actual = variables_auxiliares then --
variables_auxiliares
       numerador1<='0' & numerador_multi;
       denominador1<='0' & denominador_multi;
     end if;
     if estado_actual = restar then --restar
       diferencia <= numerador1 - denominador1;
     end if;
```

```
      if estado_actual = evaluar_acarreo then --evaluar_acarreo
        if diferencia(18) = '1' then
          enable_1 <='0';
          enable_2 <='0';
        elsif diferencia(18) = '0' then
          enable_1 <='1';
          enable_2 <='1';
         end if;
       end if;
       if estado_actual = acciones_acarreo then--acciones_acarreo
         if enable_1 = '1' then
           for i in 0 to 17 loop
             numerador_multi (i) <= diferencia (i);
           end loop;
         end if;
       if enable_2 = '1' then
         cociente <= cociente (17 downto 0) & '1' ;
       else
         cociente <= cociente (17 downto 0) & '0' ;
       end if;
      denominador_multi <= '0' & denominador_multi (17 downto 1);
      tamanio_den <= tamanio_den+1; --para ver lo de si sigue
haciendo la resta o no
      end if; -- end if del estado 00110
      if estado_actual = final then
        division_terminada <= '1';
          if cociente(10) = '1' then
            cociente1 <= cociente (10 downto 1);
          else
            cociente1 <= cociente (9 downto 0);
          end if;
      end if;    --se cierra el if de los ciclos de reloj.
      if estado_actual = error then
        error1 <= '1';
          else
       error1 <= '0';
      end if;
      if estado_actual = reposo then
        numerador          <="0000000000";
        denominador        <="0000000000";
        numerador1         <="00000000000000000000";
        denominador1       <="00000000000000000000";
        cociente           <="00000000000000000000";
        diferencia         <="00000000000000000000";
        numerador_multi    <="00000000000000000000";
        denominador_multi  <="00000000000000000000";
        tamanio_den        <= 17;
        tamanio_num        <= 17;
        enable_1           <='0';
        enable_2           <='0';
        resta_tam          <= 0;
        division_terminada <='0';
        error1             <='0';
      end if;
  end if;  --se cierra el if del reset.
end process;
```

```
fallo <= error1;
fin <= division_terminada;
division <= cociente1;
process (clk,reset)
begin
  if reset = '1'  then
    estado_actual <= reposo;
  elsif clk'event and clk='1' then
    estado_actual <= estado_siguiente;
  end if;
end process;
process (estado_actual,estado_siguiente, resta_tam, numerador,
denominador, cociente)
begin
  case estado_actual is
      when reposo =>                                    --reposo
        estado_siguiente<= introducir_numeros;
      when introducir_numeros =>      --introducir_numeros
        estado_siguiente <= multi;
      when multi =>                                    --multi
        estado_siguiente<= colocar;
      when colocar =>                                  --colocar
        if numerador_multi<denominador_multi then
          estado_siguiente <=error;
        else
          estado_siguiente <= colocar2;
        end if;
      when colocar2 =>                                --colocar2
        if numerador_multi(17)='1' and denominador_multi(17)='1'
then
          estado_siguiente <= variables_auxiliares;
        else
          estado_siguiente <= colocar2;
        end if;
      when variables_auxiliares =>      --variables_auxiliares
        estado_siguiente<= restar;
      when restar =>                             --restar
        estado_siguiente <= evaluar_acarreo;
      when evaluar_acarreo =>          --evaluar_acarreo
        estado_siguiente <= acciones_acarreo;
      when acciones_acarreo =>          --acciones_acarreo
        if resta_tam = 0 then
          estado_siguiente <= final;
        else
          estado_siguiente <= variables_auxiliares;
        end if;
      when final =>                         --final
        estado_siguiente <= reposo;
      when error =>                         --error
        estado_siguiente <= reposo;
  end case;
end process;
end Behavioral;
```