Universidad Nacional de La Plata

Facultad de Ciencias informáticas

Contribuciones a la Verificación Funcional Moderna

Tesis presentada por Oscar Enrique Goñi
para obtener el grado de Doctor en Ciencias Informáticas

Director: Elias Todorovich - Co Director: Javier Diaz

2015

Agradecimientos

Este trabajo no habría sido posible sin el apoyo y el estímulo de mi familia, cuya contención se hizo presente en más de un momento. A mis amigos y compañeros, testigos del esfuerzo y cómplices de mis delirios de investigador principiante. A todo el grupo de Sistemas Digitales de la UNCPBA por su constante aliento y por no permitirme bajar los brazos en los momentos más duros. En particular a Elías, por su infiníta paciencia, dedicación y comprensión. A las empresas que mediante proyectos de transferencia pusieron su voto de confianza en mi trabajo y reconocieron la importancia de la tarea de verificación funcional.

Resumen

El incesante desarrollo tecnológico en la industria del silicio y la necesidad de cumplir con tiempos de mercado competitivos han llevado al desarrollo de nuevas técnicas de diseño y verificación funcional. En este trabajo de tesis se presentan diferentes aportes a la verificación funcional moderna partiendo de casos de estudio reales y en donde se plantean diversos problemas a resolver.

El trabajo se divide implícitamente en dos partes: La primera, se enfoca en realizar aportes a la verificación funcional proponiendo métodos para generación de entradas, chequeo de salidas y evalucación de cobertura funcional en diseños. En particular, se considera como caso de estudio la verificación funcional de unidades aritméticas sustentándose en la utilización y desarrollo de los frameworks para verificación funcional OVM y Teal and Truss. Del análisis y desarrollo se obtiene un conjunto de reglas para la generación aleatoria de casos de prueba para la verificación de unidades de punto flotante. Además, se realiza la evaluación de, no solo el comportamiento de los frameworks propuestos, sino también se analizan parámetros de calidad como su usabilidad, extensibilidad y nivel de abstracción. Como aporte se describen los pasos para diseñar e implementar monitores de verificación aplicables a diversos dominios. El monitor presentado, no sólo realiza el chequeo de resultados comparados con modelos de referencia, sino que además realiza el análisis de cobertura considerando los casos ejercitados. En la evaluación de los frameworks se verifican dos diseños diferentes de unidades aritméticas donde al menos se detectan dos errores en casos no considerados por parte del diseñador.

La segunda parte del trabajo propone un nuevo enfoque para la verificación funcional. El enfoque propuesto parte del hecho que las pruebas orientadas a la verificación deben realizarse indefectiblemente en todo proyecto y propone preparar a los

entornos de verificación de manera tal que sus resultados sean de utilidad para el diseñador. Como caso de estudio se presenta la verificación de un modelo digital de un convertidor de potencia para emulación HIL (Hardware in the Loop). En particular, se hace enfásis en la aceleración obtenida al utilizar las técnicas de HIL en la verificación de diseños analógicos-digitales.

Continuando con la línea de aritmética presentada en la primer sección, primero se compara el comportamiento de un modelo que utiliza punto fijo para la representación de variables de estado con otro que utiliza representación en punto flotante. La evaluación se realiza considerando diversos escenarios representativos y reales donde se aplicaría el convertidor. Luego de comprobar la funcionalidad del convertidor utilizando técnicas descriptas en la primer parte de la tesis, se utilizan los resultados recopilados para el análisis de resolución de dos variables de estados en el modelo de convertidor de punto fijo. La motivación principal de este análisis es que partiendo de resoluciones óptimas es posible obtener modelos que minimizan el área ocupada en el diseño final y la máxima velocidad de reloj del circuito.

A partir de los resultados obtenidos, el diseñador puede apreciar que el aumento de resolución a partir de cierto umbral sólo incrementa el área utilizada y reduce la frecuencia de reloj sin obtener mejoras significativas en la presición. Para el caso de estudio evaluado, es posible apreciar que existe un pequeño intervalo de resolución crítico de las variables de estado donde los errores se reducen drásticamente a cero, sin reducción de la performance o aumentar el consumo de área. Si bien en este trabajo, los resultados completos de las pruebas se presentan en un apéndice, el enfoque propuesto resume un gran volumen de información en gráficas de rápida interpretación por parte del diseñador.

Abstract

The steady technological growth of the silicon industry and the need to fulfill competitive time-to-market have driven new design and functional verification techniques. This work presents several contributions to modern functional verification based on real scenarios.

This work is implicitly divided in two parts: One of them focusses on contributions for input generation, output checking alternatives and functional coverage evaluation. In particular, arithmetic units are verified by means of two verification environments based on the frameworks OVM and Teal and Truss. A set of rules is obtained for random constrained inputs generation targeted to floating point arithmetic units. In addition, both environments are evaluated in terms of not only their behavior, but also their quality attributes like usability, extensibility, and abstraction level. The contributions involve a list of steps to design and develop functional verification monitors towards many domains. These monitors are meant to check the correct design output based on a reference design, and also to provide a functional coverage analysis. The developed environments instantiate the monitors to check two arithmetic units. At least two errors caused by cases not considered by the designer were detected on the tests.

The other part of this work presents a non-studied approach to functional verification. Considering that test for the functional verification should be carried out anyway, this part of the study proposes to adjust the verification environments so that functional verification results may be useful for the designer. The case study involves verifying a digital model of a power converter intended for HIL emulation (Hardware in the Loop). Particular emphasis is given to the speed-up obtained by using this techniques on mixed digital-analog designs.

By following the arithmetic context proposed in the first part, a fixed point model of a power converter is compared against a floating-point converter model. The models are evaluated considering several real representative scenarios, where the converter might be applied. The techniques presented in the first section are used to check the correct behavior of the converter model. Then, the functional verification results are used on the resolution analysis of two state variables of the fixed-point model. The motivation of this analysis is that by using an optimized resolution, it is possible to obtain models that minimize the final desing area and maximize the clock frequency.

From the verification results, the designer can see that the resolution increment from a certain threshold number only increases the occupied area and decreases the clock frequency without legible benefits on the precision of the design. In particular for this case of study, there exists a small interval of the resolution of state variables where errors drastically decrease without compromising design performance or area. Although complete results of the tests are presented on an appendix, the proposed approach graphically summarizes a large volume of information for a fast designer interpretation.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

La tecnología actual del silicio ha permitido empaquetar millones de transistores en una sola pastilla. Este hecho es consecuencia, no solo de avances tecnológicos en la fabricación, sino también de nuevas técnicas para su diseño. En la actualidad, resulta impensable el disenño de un chip complejo desde cero. Por el contrario, un SoC (System-on-Chip) actual, se desarrolla reutilizando componentes ya diseñados y verificados obteniendo diseños de calidad aceptables y con tiempos y presupuestos razonables. Sin embargo, este crecimiento casi constante de la complejidad, provoca que los diseñadores hayan visto excedidas sus capacidades para verificar diseños de tal magnitud. Esto se debe a que es común encontrar que diseños de millones de compuertas necesiten un gran número de casos de pruebas para su verificación. La verificación funcional de sistemas digitales, tal como indica su nombre, es el proceso usado para demostrar que un diseño digital descripto en algún HDL (Hardware Description Language) cumple o no con sus especificaciones.

Hoy en día entre el 60 % y el 80 % del esfuerzo de desarrollo es invertido en verificación funcional [41]. En algunos casos, este hecho puede resultar un cuello de botella para el desarrollo del proyecto. Por esta razón, se ha intentado reducir los tiempos de verificación basásndose en:

- Inicio de la verificación en etapas tempranas: Aún cuando el diseño no haya sido finalizado, el equipo de verificación cuenta con las especificaciones funcionales y puede comenzar a diseñar una estructura de pruebas para ir verificando partes del diseño a medida que sean finalizadas.

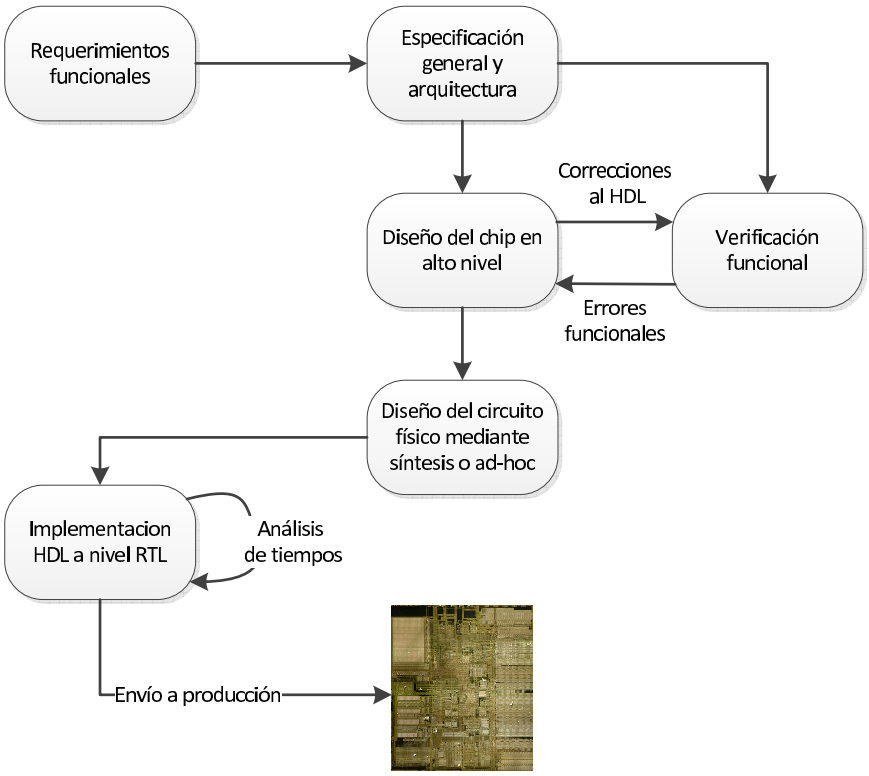


Figura 1.1: Proceso de desarrollo

Debe considerarse que la verificación funcional es un proceso que en realidad no termina nunca. La verificación funcional en sí, pretende exponer los errores presentes en un diseño y no demostrar su ausencia. Esto indica, que si un error está oculto en el diseño, en algún momento se manifestará.

1.1. El proceso de verificación funcional

Al igual que en otros procesos, la verificación funcional debe ser planificada cuidadosamente. El resultado de esa planificación se lo conoce como *Plan de verificación*. Este artefacto, es un documento revisado y seguido por el equipo de verificación. En el se describen:

- Estrategias de testeo: Se indican que metodologías o técnicas se aplicarán a cada parte del diseño

- Entorno de simulación: Arquitectura general del entorno software que estimulará y evaluará el comportamiento del diseño.
- Herramientas de verificación: Un listado de simuladores o analizadores necesarios para las pruebas.
- Pruebas específicas: Listado de pruebas imprescindibles para cumplir ciertas normas.
- Mapeo: Dada una serie de especificaciones funcionales, se indica que caso o casos de prueba son destinados a la misma.

En el presente trabajo, se hace hincapié en las estrategias y en los entornos de verificación y no en el seguimiento del proceso. A lo largo de los Capítulos, se presentan en profundidad diferentes estrategias utilizadas para abordar diferentes problemáticas comunes en la verificación industrial actual. La verificación funcional comparte técnicas y herramientas de gestión y seguimiento de procesos relacionadas con el desarrollo de software lo suficientemente probadas y actualizadas, razón por la cual no son abordadas en este trabajo. Por otra parte, en el ámbito del proceso de desarrollo de hardware, los esfuerzos se han concentrado en las estrategias y entornos.

Puede considerarse que el desarrollo de una estructura de verificación confiable, flexible y reutilizable es, en sí, una de las estrategias globales más importantes.

1.2. Interacción dentro del desarrollo de un proyecto

Anteriormente se mencionó la necesidad de la verificación funcional dentro del desarrollo de un proyecto. Aun así, el proceso de verificación funcional permite mejorar el rendimiento de los demás procesos del proyecto. La colaboración más inmediata aparece con el proceso de diseño. El equipo de diseño alimenta al de verificación con partes del diseño para verificar. Los resultados de la verificación se resumen en un conjunto de casos ejecutados de los cuales pueden o no surgir errores funcionales. En el caso de presentarse errores, se realimenta al diseño con una serie de inconsistencias (o issues en inglés) y descripciones precisas del escenario donde se manifiesta

el error. En caso de no encontrarlos, esa parte se marca como verificada para su posterior reutilización.

Otro tipo de interacción se presenta durante el análisis de las especificaciones funcionales y su mapeo en casos de prueba. Dado que el equipo de verificación trabaja de manera independiente al de diseño, es posible detectar ambigüedades en las especificaciones o bien la ausencia de consideraciones sobre las mismas de manera temprana. De esta forma, el equipo encargado de la captura de requerimientos puede reformular o completar las especificaciones. Si bien este no es uno de los objetivos de la verificación funcional, es una situación que ocurre con frecuencia.

Sin embargo, si bien se consideran los errores resultantes de la verificación para corregir el diseño, poco se han considerado los resultados para llevar a cabo optimizaciones o mejoras sobre este. Este trabajo presenta entre sus aportes la verificación funcional de diversos diseños, pero contrastando el enfoque colaborativo para hallar o detectar partes del mismo que podrían ser optimizadas como valor agregado. Esta colaboración en forma de sugerencias surgen de la utilización y análisis de los resultados propios de la simulación con fines de verificar el diseño. En el Capítulo 6 se presenta la aplicación de la verificación funcional en el análisis de resolución, área y frecuencia máxima de un diseño industrial.

1.3. Organización del trabajo

El objetivo principal del presente trabajo es proveer un conjunto de aportes a la verificación funcional moderna. Estos aportes aparecen como soluciones a problemáticas reales originadas en el sector industrial y académico. El trabajo sustenta sus pilares en el desarrollo de entornos de verificación, estrategias para la generación y verificación automática y en aportes de resultados para la optimización del proceso de diseño.

El contexto del trabajo de tesis elegido presenta el estudio de la verificación funcional de sistemas digitales que utilizan sistemas de representación de punto fijo y de punto flotante. En términos generales, se enfoca a la verificación funcional como herramienta complementaria y de asistencia en la fase de diseño. El trabajo comienza relevando las herramientas utilizadas en la verificación funcional moderna conside-

rando en todo momento las necesidades que presenta la industria en este proceso. Luego, el trabajo se enfoca en el análisis estructural basándose en dos infraestructuras (o *Frameworks* en inglés) para verificación funcional. Sobre estos se proponen y aplican métodos para la verificación de sistemas digitales que utilizan operaciones aritméticas simples (sumas, restas y productos). Inicialmente, se aborda la generación automática de casos de prueba. La generación involucra el estudio del dominio de entrada, en este caso operadores de punto fijo y de punto flotante (con representación binaria y decimal) y análisis de casos extremos (o *corner cases* en ingés) para luego convertirlos en casos de pruebas de un test aleatorio con restricciones. Esta fase de la verificación es presentada como el Capítulo 2 y presenta como aporte una mejora en las pruebas de diseños con punto flotante decimal más una comparación cualitativa de ambos frameworks.

Para completar el proceso de verificación, el trabajo continúa con las técnicas utilizadas para la comprobación de resultados y el análisis de cobertura funcional en este tipo de diseños. Entre estas técnicas se destacan: La construcción de modelos de referencia, la definición de aserciones generales y para cada operación aritmética en particular, y la definición de modelos de cobertura. La aplicación de este conjunto de técnicas se implementan en lo que se denomina Monitor de verificación y es el foco de desarrollo del Capítulo 3.

En la segunda parte del informe, se provee una visión distinta de la verificación funcional. En particular, se propone enfatizar las tareas de la verificación funcional de manera tal que sus resultados sirvan de apoyo al proceso de diseño. Esta asistencia, no solo resulta útil al diseñador en el proceso de depuración, sino que también puede ser usada como herramienta de análisis para la selección de ciertos parámetros de diseño que pueden ser difíciles de estimar a priori o bien aquellos cuya selección tiene impacto directo sobre la performance o el área del diseño. De esta manera, el Capítulo 4, aborda las herramientas que provee la verificación funcional moderna para la ejecución automatizada de pruebas, complementadas con técnicas de generación de escenarios y herramientas para el análisis y reporte de resultados. Para la generación de escenarios se propone la implementación de scripts para la ejecución de pruebas así como también la construcción de componentes de verificación genéricos para utilizarse, por ejemplo, en entornos basados en frameworks. En este Capítulo

se aplican los aportes de los Capítulos anteriores pero dentro de un contexto más complejo. Para demostrar las características y problemas que se abordan en diseños con los dos sistemas de representación, se presenta como caso de estudio la verificación funcional de un modelo de convertidor de potencia para la prueba de sistemas HIL (del inglés *Hardware in the Loop*). En particular, se aborda la verificación de un modelo implementado con aritmética de punto fijo y considerando como referencia un modelo en punto flotante. El estudio involucra la generación de diferentes escenarios o configuraciones estáticas emulados digitalmente en cada prueba, por ejemplo, capacidad de los condensadores, inductividad de la bobina, tensión de entrada, entre otros parámetros que incluiría el modelo real. Además, se utilizan herramientas para generar dinámicamente condiciones de entrada y salida, como por ejemplo corrientes de entrada y cargas que simulan diversos dispositivos conectados a la salida del convertidor. Las pruebas poseen dos niveles de análisis, el primero es durante la ejecución de la prueba, recolectando valores de tensión y corriente, errores y desviaciones paso a paso. En el segundo nivel se aplican herramientas como el análisis de cobertura funcional, análisis de cobertura de código, profiling y análisis estadístico de los resultados, una vez finalizada la prueba. Para las primeras tareas se utilizan herramientas comúnmente integradas en el simulador mientras que para el análisis se utilizan como Matlab u Octave. Tanto la generación de pruebas como el análisis de resultados son presentados de manera tal que puedan ser interpretados rápidamente por el diseñador.

En el Captulo 5, se presenta una situación concreta donde la verificación funcional es utilizada como herramienta para el análisis de resolución. La motivación de este análisis se basa en las ventajas que posee la aplicación de aritmética de punto fijo sobre la aritmética de punto flotante y las consecuencias aparejadas por la incorrecta selección de parámetros de resolución: Alta ocupación de área y baja frecuencia de funcionamiento. La bibliografa estudiada propone diversos métodos para la selección de parámetros de diseño asociados a la resolución. Sin embargo, coinciden que la simulación permanece como la herramienta más eficaz para esta tarea. El caso de estudio propuesto es el mismo al propuesto en el Captulo 4. En particular se aborda el análisis de resolución en los registros para el cálculo de corriente y tensión. Para corroborar o refutar las afirmaciones de la bibliografa, se presenta un método analti-

co para el análisis de resolución que considera únicamente la fórmula del modelo del convertidor. La ejecución de las pruebas demuestra que una incorrecta selección de parámetros de diseño tienen impacto directo sobre la velocidad del reloj así como en el área ocupada. Más aun, con los resultados de la simulación se demuestra que existe una delgada zona del espacio de posibles resoluciones la cual permiten la resolución óptima. Esto significa que si ésta se aumenta, se incrementa innecesariamente el área y se disminuye la frecuencia de reloj. Por otro lado, si la resolución se disminuye, se provoca una abrupta pérdida de precisión en el modelo. El segundo método es evaluado, y si bien su aplicación insume menos tiempo que la simulación, sus resultados son sobredimensionados y no consideran el posible error provocado ciclo tras ciclo. Esto refuerza la hipótesis y confirma los resultados de otros trabajos que proponen a la simulación como herramienta para el análisis de resolución. Finalmente en el Captulo 6 se presentan las conclusiones del trabajo y se propone el trabajo futuro continuando con esta línea.

Capítulo 2

Fundamentos

La revolución del silicio ha generado la demanda constante de mejoras en los diseños electrónicos. Aunque en los últimos años las mejoras se hayan visto enfocadas a disminuir consumo de potencia y aumentar el nivel de integración, desde sus comienzos, los principales objetivos fueron el aumento en la velocidad de cálculo, la reducción de costos de fabricación y la reducción en los tiempos de desarrollo. Esto ha permitido que los sistemas digitales sean cada vez más complejos. Sin embargo, una limitante del crecimiento de la complejidad de estos sistemas se encuentra en la etapa de verificación. Es frecuente que en la industria del hardware existan pérdidas millonarias y retardos en los proyectos de varios meses producto de un error en el diseño, en particular en los circuitos integrados de aplicación específica (ASICs). Aun cuando se utilizan arquitecturas reconfigurables como FPGAs (field-programmable gate arrays) y CPLDs (Complex Programmable Logic Device), resulta imprescindible que el diseño funcione tal como fue especificado desde la versión inicial. Es por eso que la Verificación Funcional (VF) ha surgido como un esfuerzo en equipo para asegurarse que el chip o el sistema en desarrollo se comporte tal como fuera especificado. Uno de los textos referentes del área *Comprehensive Functional Verification the Complete Industry Cycle* [58] define a la verificación como *"El proceso de determinar si los productos de un ciclo de vida dado cumplen una serie de requerimientos establecidos"*. En el ámbito de los sistemas digitales, se refiere al proceso de asegurarse que los requerimientos son correctamente mapeados a una implementación RTL (Register-Transfer Logic). En la etapa de diseño, el diseñador generalmente realiza su

trabajo basándose en casos representativos que responden a las especificaciones. Sin embargo, la tarea del verificador es asegurarse que la implementación sea correcta en todos los casos.

Para cumplir con este objetivo planteado, el medio más directo sería la inspección exhaustiva donde todas las características del diseño fueran ejercitadas en todos los posibles escenarios. Sin embargo, sucede que la búsqueda exhaustiva es prácticamente imposible en diseños reales. Si de todas formas, esto fuera posible, su aplicación carecería de sentido ya que existen conjuntos significativos de casos de pruebas cuya evaluación obtiene los mismos resultados.

En general, no existe una única metodología de verificación que sea aplicable a todos los posibles diseños. En contraposición, existen diferentes metodologías que se adaptan mejor a la verificación de diseños con determinadas carcterísticas. Si el equipo de verificación sigue el mismo estilo de diseño o evalúa los mismos casos de referencia, ambos equipos cometerán idénticos errores y muy pocos errores funcionales podrán ser encontrados [58]. Es por eso que la etapa de verificación debe ser independiente a la de diseño ya que existe el riesgo de que la verificación sea realizada sobre la implementación y no sobre las especificaciones. Este hecho, sumado a la complejidad de los sistemas actuales, implica que la verificación sea una tarea lenta y costosa. Sin embargo, utilizar las técnicas apropiadas pueden ahorrar tanto presupuesto como tiempo [35]. Otro error frecuente en los equipos de verificación es invertir todo el potencial humano disponible en la especialización para el trabajo sobre una única metodología. Esto puede llevar a forzar la aplicación de metodologías que no se adaptan a la verificación de cierto tipo de diseño. Es por eso que es deseable contar con un amplio conocimiento de las tecnologías de verificación existentes así como también poseer capacidad de reconocer que metodología aplicar en cada caso.

2.1. Verificación funcional tradicional

Desde los comienzos en la industria elctrónica, la complejidad de los dispositivos ha aumentado a ritmo constante. Si bien la verificación funcional ha intentado seguir este ritmo de crecimiento, nunca ha logrado alcanzarlo, por lo contrario, la diferencia

es cada vez mayor. Es de vital importancia dotar al equipo de verificación de un conjunto actualizado de herramientas, procesos y lenguajes para una verificación eficaz.

2.1.1. Verificación basado en integración

Tanto al momento de diseñar como de verificar un diseño digital, resulta imposible realizarlo a nivel de compuertas básicas. En general, las tareas se realizan en diferentes niveles de abstración previamente establecidos. Los niveles superiores brindarán una idea global de las tareas de esa etapa. A medida que los niveles bajan su abstracción, estos se dividirán en una o más tareas. Esta división en diferentes niveles es válida tanto para el diseño como para la verificación. En el diseño, el nivel más bajo está compuesto por numerosas descripciones de componentes simples, mientras que en verificación contienen monitores para verificar cada uno de esos componentes.

En la literatura [56], esta división en niveles se establece en:

- Verificación de bloque: Contiene un único módulo RTL o bien varios módulos agrupados. En este nivel, las entradas y las salidas del módulo se conectan a modelos funcionales de buses o BFM (del inglés bus functional model). Las características de este nivel son la *flexibilidad*(para adaptarse a módulos con diferentes parámetros) y la *observabilidad*(capacidad de poder analizar valores de señales de entrada/salida/internas).

- Verificación del sub-sistema: En este nivel se evalúa la interacción entre diferentes bloques. Los modelos funcionales de bus son reemplazados por descripciones reales RTL. El tiempo de la ejecución de las pruebas es mayor, no solo porque la simulación sea más compleja, sino también por que la cantidad de módulos participantes es mayor. Dado que los módulos pueden provenir de diferentes diseñadores, en este nivel suelen aparecer bugs relacionados con suposiciones o interpretaciones del diseñador.

- Verificación del chip completo: Todo el chip es verificado. La emulación se realiza a partir de RTL únicamente y sin BFMs (sólo aquellos aquellos que estimulan entradas y recolectan salidas). El hecho de que toda la simulación sea a partir

de RTL provoca que la verificación del chip completo sea considerablemente más lenta.

2.1.2. Verificación de microprocesadores

Su objetivo es lograr el máximo nivel de confianza en un diseño que incorpora microprocesadores. En este enfoque, se utilizan programas codificados en C o lenguaje ensamblador utilizando un generador de instrucciones específico para determinada arquitectura. Debe considerarse que la simulación de un microprocesador es una tarea computacionalmente costosa si se realiza una simulación a nivel de compuertas. Es por eso que el DUV se simula a nivel de BFM. La secuencia de instrucciones coherentes generadas estimulan al DUV mientras que un conjunto de monitores embebidos en el diseño y en el generador recolectan información. El proceso de chequeo puede realizarse durante la simulación o posterior a la misma. Básicamente, el chequeo consiste en contrastar las salidas del DUV con las de un modelo de referencia.

2.1.3. Emulación y aceleración

Dada la creciente complejidad de los cálculos que realizan los diseños, es necesario contar con medios eficientes para simulación. Resulta indispensable disponer de herramientas que rápidamente simulen el diseño RTL y que cuenten con una interfaz hardware-software para comprobar que el diseño se comporta tal como fue especificado. Sin embargo, los diseños que requieren grandes cantidades de ciclos de simulación comprometen a la velocidad de la verificación. En estos casos, es necesario contar con herramientas diferentes al software de simulación. Dos enfoques utilizados para resolver esta problemática son la emulación y la aceleración (Fig. 2.1).

La emulación mapea el diseño a un dispositivo hardware especializado para acelerar la simulación RTL. Luego, mediante una interfaz software-hardware, es posible otorgar flexibilidad al sistema. Esta interfaz es la encargada de inyectar valores de señales en las entradas así como la recolección de las señales a la salida. En general, los emuladores están formados por una matriz de FPGAs. El software de emulación mapea las diferentes partes del diseño en diversas FPGAs. Luego, los resultados son devueltos al software por medio de alguna interfaz con gran ancho de banda (ej. PCI

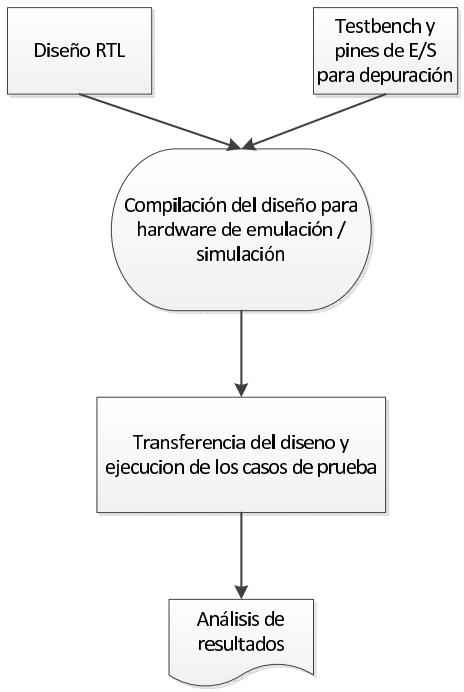


Figura 2.1: Flujo de emulación y aceleración

o PCI-e) logrando de esta manera, resultados en el orden de 1-10 Mhz.

Por otro lado, la aceleración compila el diseño HDL en un programa ejecutable. Cada parte del diseño es traducida a instrucciones específicas de un procesador. Este procesa la instrucción simulando su comportamiento. De esta manera, se pueden disponer, en varios procesadores para ejecución en paralelo, diversas partes del diseño RTL. Este enfoque provee un rendimiento superior a la simulación tradicional que se ejecuta en micro procesadores de uso general.

Debe considerarse que estos enfoques requieren de hardware externo específico y generalmente costoso, lo cual puede ser, en ciertos proyectos, una limitante para la aplicación. Por otro lado, la visibilidad de señales debe ser especificada a priori, esto significa que si el ingeniero desea conocer el comportamiento de alguna señal interna adicional no prevista, deberá recompilar y volver a cargar el diseño. Además, al aumentar la visibilidad con nuevas señales internas, se aumenta el tráfico de datos en la interfaz hardware-software, disminuyendo el ancho de banda.

2.2. Elementos de la verificación funcional moderna

Se consideran técnicas de funcional moderna a aquellas que aparecieron después del año 2005. El objetivo de las técnicas actuales no es reemplazar a las tradicionales. Por el contrario, su objetivo es complementarlas, proveyendo de diversos métodos y elementos que aumentan el nivel de confiabilidad de la verificación.

Tradicionalmente, la verificación funcional se realizaba estimulando al DUV dentro de un módulo de hardware conocido como *Testbench*. En general, este módulo estaba escrito en el mismo lenguaje de descripción que el DUV. Este módulo, no era sintetizable ya que contenía primitivas orientadas a simulación (por ejemplo, generación de retardos). La aparición de la verificación funcional moderna está muy relacionada con el surgimiento de los lenguajes de Verificación de Hardware (HVL). A diferencia de los HDLs, los HVLs son super conjuntos de HDLs que permiten escribir entornos de verificación en mayor nivel de abstracción que los HDLs. Los HVLs incorporan elementos existentes en los lenguajes de programación de software modernos como son la programación orientada a objetos y estructuras y tipos de datos complejos. Además incorporan construcciones para descripción de propiedades que posteriormente se utilizarán tanto para crear estímulos como así también para verificar comportamientos. Más adelante se detallan con mayor profundidad las aplicaciones más relevantes.

2.2.1. Generación aleatoria con restricciones

Tradicionalmente, la verificación de un DUV se realizaba estimulándolo con un conjunto finito de entradas. Las salidas eran conocidas a-priori y generalmente cubrían los mismos casos que el diseñador había utilizado como referencia. Esto provocaba, no solo que la verificación no fuese completa, sino que respondiera a los requerientos del diseñador y no de las especificaciones. La generación de estímulos aleatorios aparece como una alternativa para evitar esa generación dirigida. Este enfoque ha ganado popularidad en los últimos años. Su éxito se debe a la ventaja en la utilización de escenarios aleatorios que describen situaciones o casos no contemplados expresamente en la descripción del plan de verificación. Un generador de estímulos aleatorios

se ejecuta en paralelo con la simulación del DUV y produce una serie de estímulos que alimentan al entorno de verificación. Sin embargo, los casos generados no son completamente aleatorios ya que contienen restricciones estratégicamente ajustadas para probar una característica interesante del diseño. Si la generación de valores fuese totalmente aleatoria, se producirían combinaciones de entradas inválidas o bien semánticamente incorrectas que no serían de utilidad para la verificación. La calidad de la generación que provee un test aleatorio depende de dos factores importantes. El primero está directamente ligada a la capacidad del ingeniero verificador en describir situaciones precisas mediante restricciones. El segundo, tiene relación al soporte que provee el entorno de verificación para ejecución y depuración de tests aleatorios. La generación aleatoria es lograda mediante elementos de los HVLs conocidas como *contraints*. El ingeniero puede describir escenarios utilizando la sintaxis declarativa del HVL. Por ejemplo en SystemVerilog:

constraint Bank2Addresses $\{addr \geq 1024; addr \leq 2047;\}$

El ejemplo define al constraint de nombre *Bank2Addresses* que exige que la generación aleatoria produzca valores de *addr* en el rango de 1024 a 2047. En el caso del ejemplo, la relación es simplemente mayor y menor, pero pueden De esta manera, pueden describirse una o más constraints que restrinjan los intervalos de valores de uno o más datos. En un nivel de sofisticación mayor, puede exigirse que los límites (1024 y 2047 del ejemplo) sean variables de manera tal de poder restringir, durante la simulación, el intervalo de posibles valores.

Debe considerarse que la generación aleatoria debe proveer un soporte para seguimiento de los casos ya generados. Es por eso que la verficación aleatoria con restricciones está fuertemente relacionada con la verificación dirigida por la cobertura como se explica más adelante.

2.2.2. Programación orientada a objetos en verificación funcional

Las técnicas de OOP han demostrado ser de utilidad en equipos de programación en abordar la codficación de problemas complejos de software. En particular, la característica clave para enfrentar esa complejidad es la capacidad de expresar la intención del código mediante objetos. Este paradigma permite a los programadores

desarrollar de manera individual diferentes piezas de código de forma más efectiva.

Los problemas a los que se enfrentan hoy en verificación son similares a los que se presentaban cuando la POO fue adoptada en los sistemas software. Códigos de miles de líneas y cientos de módulos que deben ser compilados, instanciados, controlados y ejecutados. En este sentido, la POO viene a relajar estos problemas separando y flexibilizando características y funcionalidades en objetos abstractos que serán luego especializados.

2.2.3. Frameworks para verificación funcional

A lo largo del proceso de construir entornos de verificación para cada diseño hardware en particular, los ingenieros reconocen patrones en la estructura de los testbenches que aparecen una y otra vez. En algunos casos, esa similitud no existe a nivel estructural sino a nivel comportamental. En este punto, los ingenieros de verificación utilizan algunas de las características de la POO para implementar estructuras altamente reusables, configurables y extensibles a nuevas características no previstas al momento de la especificación. En particular, la *herencia de clases* permite, entre otras funcionalidades, crear esqueletos de objetos con propiedades en común para luego ser instanciados en entornos específicos donde adquieran su forma final. Una *clase abstracta* es capaz de definir solo las propiedades del objeto pero no necesariamente sus métodos. De hecho, es posible definir su comportamiento de manera abstracta por medio de métodos *template*. Por otro lado, el *polimorfismo de objetos* permite a un objeto cambiar su tipo a otro otorgando alta flexibilidad al entorno que lo contiene. En contraposición a la POO en sistemas software, las clases no son utilizadas para el *ocultamiento de información* y por lo tanto las propiedades del objeto son accedidas de manera directa desde el código fuente. Esto se debe a que en verificación funcional es deseable obtener máxima visibilidad de los objetos con la menor cantidad de indirecciones posibles. Finalmente, la creación de módulos y posteriormente entornos de verificación complejos es impulsada por la composición de objetos. De esta manera un entorno complejo puede ser creado a partir de módulos más simples, también creados a partir de objetos simples.

En general, un framework para verificación funcional es visto como una infraestructura abstracta de módulos de verificación que utiliza todas esas características

de la POO. En las siguientes subsecciones se describen brevemente tres frameworks comúnmente usados para verificación funcional.

OVM

OVM (Open Verification Methodology [25]) es un framework de código abierto para verificación funcional de sistemas digitales en un entorno de simulación. Está totalemte escrito en SystemVerilog, aunque existe una versión adaptada a SystemC. La filosofía de OVM es reemplazar el enfoque tradicional de escribir testbenches por una metodología más robusta. Esta robustez se logra utilizando componentes reusables y optimizados para cumplir una tarea dentro de una estructura de verificación cudiadosamente diseñada. OVM provee al equipo de verificación, una colección de clases abstractas que modelan componentes para estimular el DUV. La flexibilidad estructural de OVM se logra implementando clases abstractas para crear items de datos, drivers, monitores, scoreboards a medida de la aplicación. Por otro lado, la flexibilidad comportamental de un entorno OVM se logra describiendo funcionalides específicas en las fases predefinidas (Fig. 2.3a). A continuación se describen algunos de los compontentes abstractos de OVM que serán especializados a medida de la aplicación:

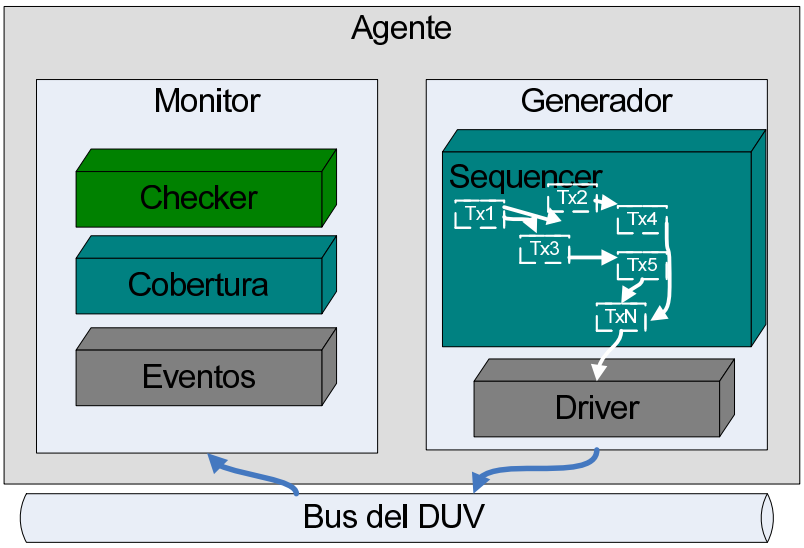


Figura 2.2: Esqueleto del un entorno basado en OVM

SystemVerilog Wrapper es un simple módulo SystemVerilog que solamente instancia al diseño que se está verificando. Este mecanismo se utiliza para homogeneizar el lenguaje ya que éste puede estar descrito también en cualquier HDL, tal como VHDL o Verilog.

OVM_Agent es la mínima unidad funcional de OVM. Un agente es el responsable de estimular y evaluar una parte específica del diseño. Cada agente puede ser pasivo (monitorea solo las señales provenientes del diseño) o activo (además de monitorear el diseño estimula entradas específicas).

OVM_Sequencer o secuenciador es el encargado de generar secuencias de transacciones válidas que serán posteriormente inyectadas en el diseño. El sequencer conoce a priori el tipo de transacción que debe generar.

OVM_Driver recibe una a una las transacciones generadas por el sequencer y convierte la información de estas en un formato y protocolo reconocible por el diseño. Por ejemplo, un Driver podría encargarse de recibir arreglos con datos (transacciones) y emitirlas a un pin RX del diseño respetando el baudrate y codificación del protocolo UART1650.

OVM_Monitor es el encargado de recolectar señales desde la interfaz del dispositivo y convertirla en información a nivel de transacción. Esta transacción puede ser posteriormente chequeada o analizada.

OVM_Environment es el conjunto de unidades funcionales que integran el entorno de verificación. El entorno puede variar según las características del diseño que se desee verificar, es decir puede tener algunos agentes activos y otros pasivos, poseer o no recolección de cobertura, etc.

OVM_Test En la bibliografía [25] se considera a los objetos de tipo test fuera del entorno de verificación. Un test es una combinación específica de entorno y tipos de transacciones formada para evaluar una característica de interés del diseño. Es posible codificar una serie de tests diferentes y posteriormente ejecutarlos de manera independiente o bien de forma consecutiva e invocados desde un script TCL.

UVM

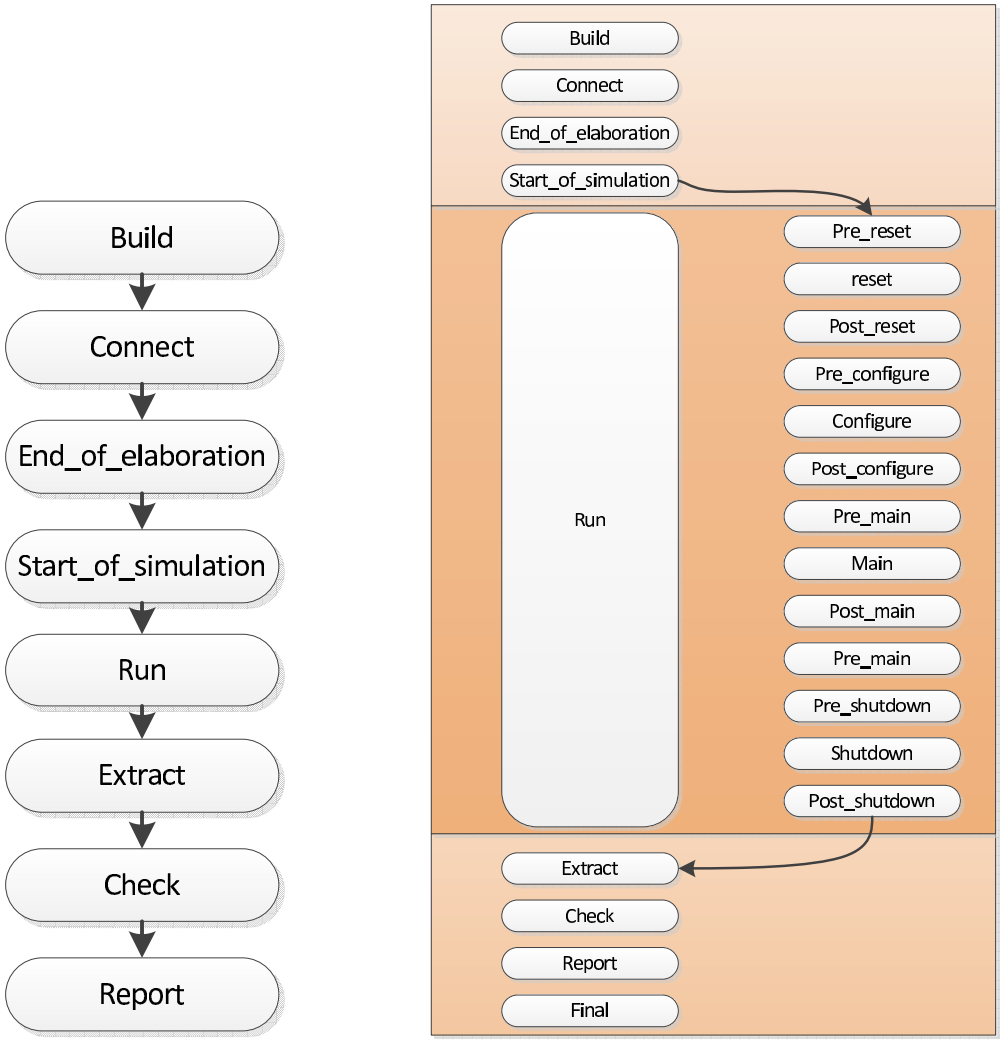
UVM (Universal Verification Methodology) es una metodología híbrida para verificar diseños complejos. Es de código abierto y soportada por Cadence y Mentor. UVM es el sucesor directo de OVM. De hecho, un entorno existente en OVM es fácilmente portable a UVM. La estructura de UVM es similar a la de OVM, basado en la composición de objetos, que a lo largo de la simulación, pasan por diferentes fases (Fig 2.3b): Creación, construcción, ejecución y reporte. La composición de objetos simples para generar objetos complejos y la división de su comportamiento en fases permiten a ingeniero verificador ajustar diferentes comportamientos a medida de sus necesidades. Está codificado en gran parte en SystemVerilog y utiliza una librería escrita en C++ para la aceleración de ciertas funcionalidades (UVM_DPI).

Teal and Truss

Teal & Truss es una combinación de librerías de software orientadas a la verificación funcional de sistemas digitales. Una de las características más importantes de Truss es la simplicidad de uso. El diseñador es provisto de un simple esqueleto de clases el cual debe completar o refinar para la verificación de su diseño. Sus componentes están altamente acoplados lo cual restringe como el entorno de verificación debe ser estructurado. Está basado en un diseño por capas donde la capa de nivel inferior interactúa con el DUV mientras que la de nivel superior provee funcionalidades como la de creación de escenarios y la de generación de reportes. Por otro lado, Teal es una librería de soporte que provee una interfaz entre diseños HDL y C++ y provee elementos fundamentales para el desarrollo en alto nivel de un sistema de verificación como Truss. Aunque T&T es usado especialmente en entornos académicos y de hardware abierto, su aplicación es factible en la industria de desarrollo de cores IP.

2.2.4. Aserciones

Desde el punto de vista del software, una aserción es una expresión lógica que, si su evaluación resulta falsa, es una situación de error. En particular, se utilizan para describir situaciones que no deben suceder. Dentro de un HDL, una aserción es una



(a) OVM (b) UVM

Figura 2.3: Fases en la ejecución de los entornos de verificación

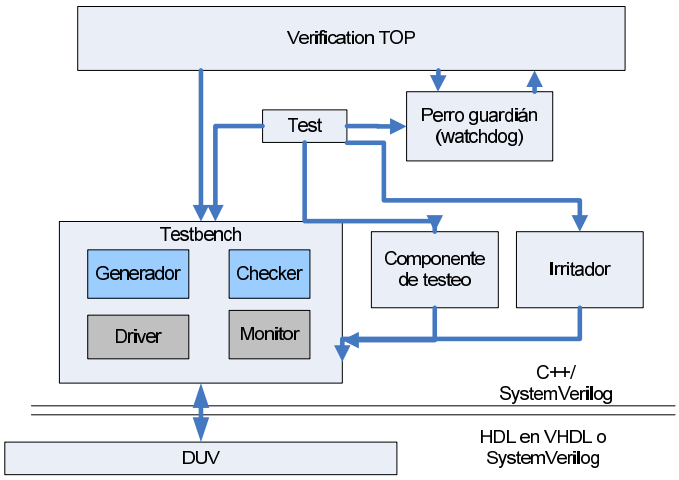


Figura 2.4: Estructura de un entorno basado en Teal & Truss

sentencia condicional que evalúa comportamientos específicos. Dentro de la verificación funcional, una aserción es una especificación ejecutable de los requerimientos del diseño. En particular se pretende que esa especificación siempre se cumpla. La verificación funcional se basa en cuatro pilares fundamentales:

- Chequeo automático de aserciones: Un conjunto de reglas predefinidas se embebe dentro del código RTL. Así como las herramientas de *linting* analizan la sintáxis y semántica del código, el chequeo automático de aserciones sintetiza el diseño y utiliza métodos formales para analizar las estructuras internas del diseño. El chequeo automático de aserciones puede reconocer errores de sincronización de clocks, estructuras de buses y máquinas de estados finitos. Estas funcionalidades son difíciles de verificar solamente con la simulación. Dado que las reglas de aserciones forman parte de un conjunto predefinido, el proceso puede ser automatizado completamente. El chequeo automático se adecua para la verificación a nivel de bloque antes que el código RTL sea evaluado. Su uso también es adecuado luego de la integración a nivel de chip. Por otro lado, no son adecuadas para chequear estilos de código o chequeo de sintáxis.

- Verificación formal estática: Los diseñadores agregan las aserciones dentro de su código. Mediante la verificación formal, el diseñador puede obtener inmediatamente realimentación sobre sus suposiciones. El análsis es estático, por

lo tanto necesita un único estado inicial. Generalmente el estado **after reset** es suficiente. La verificación formal utiliza técnicas matemáticas para probar aserciones verdaderas (según un conjunto de suposiciones) o aserciones falsas (descubriendo contra-ejemplos). Una Prueba significa que la verificación formal estática ha explorado exhaustivamente todos los comportamientos posibles con respecto a la aserción y esta no ha podido ser violada. Las ventajas que posee la verificación formal estática son: el análisis exhaustivo de ciertas áreas del diseño y la independencia de un entorno de verificación. En general, la verificación formal estática es aplicable en la depuración de esquemas de arbitración, interconexionados, lógicas complejas y diseño de memorias.

- Simulación con aserciones: Las aserciones especificadas por el usuario son simuladas junto al DUV y al entorno de verificación. Durante la simulación, las aserciones monitorean pasivamente las señales en las interfaces y dentro del diseño. Una característica interesante es que al detectarse una violación, no resulta necesario esperar que el error se propague a la salidas para ser detectado. Las aserciones son utilizadas para recolectar información de cobertura e información estadística para luego poder evaluar que proporción de la funcionalidad ha sido ejercitada como se explicará más adelante. Entre las ventajas en la utilización de esta metodología se puede destacar la adaptación transparente al entrono de simulación y la mejora en la observabilidad del entorno de verificación. La aserciones dentro de la simulación son las herramientas adecuadas para probar interfaces estándar y buses. Además, pueden utilizarse aserciones para asegurar la cobertura de puntos de verificación críticos o bien para el guiado en la generación de entradas en un test aleatorio, tal como se muestra más adelante. Por otro lado, las aserciones no son adecuadas para realizar cobertura de código o de máquinas de estado. En ese caso, las herramientas de simulación proveen esta información de manera más eficiente.

- Verificación formal dinámica: A diferencia de la verificación formal estática, ésta se ejecuta en paralelo con la simulación. La verificación formal hace uso del conocimiento encapsulado en los vectores de simulación para explorar posibles violaciones. En el caso de detectarse una violación, el contraejemplo que ha descubierto dicha violación es utilizado como parte del estimulo. Si bien

la verificación formal dinámica no es exhaustiva, puede ser aplicada a grandes diseños. Esta metodología complementa a la verificación realizada en la simulación y resulta ideal para encontrar casos extremos difíciles ocultos.

A partir de estos pilares surgen diversas metodologías y herramientas que resultan indispensables en la verificación funcional moderna. En la sección 2.3 se describe brevemente su historia y el papel que juegan las aserciones en la verificación funcional moderna.

2.3. Verificación basada en aserciones

2.3.1. Inicios

A principios de los 30, una rama diferente de la lógica se enfocaba en formalismos para describir cálculos, comenzando con las máquinas de Turing y continuando con el desarrollo de la teoría de autómatas finitos en la década del 50. En [55] se describe que uno de los inicios de la verificación formal de hardware tuvo lugar en 1957 en el trabajo de Alonzo Church [16]. En ese trabajo se proponía la lógica para especificar circuitos secuenciales planteando el problema que dado un programa de estados finitos, visto como una transición de estados finitos, cumplía o no con las especificaciones temporales dadas. Este problema conocido como *problema de desición de Church*. Este problema fue resuelto en 1962 [6] mostrando que la validez de un problema sobre una estructura finita es decidible. En terminología moderna, el problema de desición de Church es el problema de model-checking en el enfoque de tiempo lineal, reintroducido a principios de la década del 80.

La historia de la lógica temporal se remonta a tiempos recientes en [47] pero en el libro de Arthur Norman Prior [50] aparecen indicios de la lógica temporal moderna. En [34] se confirma la equivalencia expresiva entre la lógica temporal y la clásica. En el trabajo de Amir Pnueli [48] se presenta la aplicación de lógica temporal para verificar la correctitud de programas utilizando lo que se conocería como Lógica Temporal Lineal (LTL) sobre especificaciones de programas sin final. La LTL es una lógica temporal que incluye dos conectivos: *next* (siguiente) y *until* (hasta). Las fórmulas en LTL son construidas a partir de un conjunto de proposiciones atómicas

usando los típicos conectivos binarios así como el conector unario temporal *Next* y el binario *Until*. La conexión entre el model checking y el problema de desición de Church fue estudiada en [19] y en [51] en 1982. En 1976, Vaughan Ronald Pratt propuso utilizar lógica dinámica, una extensión de la lógica modal para especificar programas [49]. Esta lógica proponía un enfoque de saltos de tiempo para razonar sobre el comportamiento de los programas en contraste con el razonamiento de tiempo lineal de Pnueli. A principios de los 80, se aclara que la lógica temporal y la dinámica proveen dos diferentes perspectivas para especificar programas: La primera se basa en estados mientras que la segunda lo hace en acciones. Uno de los esfuerzos que combinan ambos enfoques es el lenguaje RCTL/Sugar. Es una lógica basada en la ramificación del tiempo sin noción de acciones. A diferencia de la lógica dinámica, que utiliza expresiones regulares en las instrucciones del programa, RCTL / Sugar utiliza expresiones regulares en los predicados de estado.

2.3.2. Introducción en la industria

Todo el desarrollo teórico fue puesto en practica en herramientas ([10] [54]). A fines 1990 y comienzo del 2000, el model checking comenzó a tener impacto en la industria. Esto llevo al desarrollo de herramientas basadas en LTL: ForSpec, desarrollado por Intel, y PSL, desarrollado por el consorcio Accelera. ForSpec fue publicado en 2000 y posee un lenguaje de tiempo lineal. Se agregaron dos modos útiles para los ingenieros: clocks y resets. ForSpec soporta construcciones orientadas a hardware así como una semántica uniforme para validación formal y dinámica. Una característica interesante es el poder de expresión logrado en las descripciones.

En el 2000, Accellera decidió que era necesario el desarrollo de un lenguaje de especificación estándar para que la verificación formal fuese aplicable en la industria. Dado que el objetivo era especificar propiedades y no diseños, se eligió el nombre de "property specification language" (PSL). Accellera escogió Sugar como lenguaje base para PSL en 2003 y además adoptó todas las características principales de ForSpec. En esencia, PSL es LTL, extendida con modos dinámicos (conocido como capa regular), clocks, y resets (conocidos como **aborts**). PSL además de incluir la sintaxis de Sugar, PSL incluye una extensión que permite saltos en el tiempo. En 2005, IEEE estandariza PSL [2]. SystemVerilog, también un estandar IEEE desde

2005, incluye SVA (SystemVerilog Assertions) [3], similar a PSL.

La verificación basada en aserciones es básicamente una metodología que utiliza a las aserciones para aplicaciones combinadas de simulación, verificación formal y semi-formal.

Inicialmente, las aserciones se incluían en los monitores para detectar comportamientos indeseados y que en general eran difíciles de capturar durante la simulación. Esto permitía aumentar en gran medida la observabilidad del diseño con una verificación más eficiente. En tiempos más recientes, las aserciones también se utilizan como entrada para realizar verificación formal.

2.3.3. Verificación dirigida por la cobertura

La verificación aleatoria con restricciones permite ejercitar casos que quizá no hayan sido previstos a la hora de armar el plan de verificación. Sin embargo, ya sea por el origen de la semilla generadora de números aleatorios o bien la repetición de valores generados, puede ocurrir que un testbench no produzca nunca determinados valores que ejerciten determinadas características de interés. Si el ingeniero de verificación pudiera conocer esta situación, debería detener las pruebas y pensar en alguna estrategia para lograr entradas que ejerciten las características restantes. Básicamente, la *Verificación dirigida por la cobertura* o *CDV* (del inglés Coverage Driven Verification) aborda esta problemática.

En el enfoque dirigido por la cobertura, la medida de la cobertura se utiliza para identificar casos que han sido ejecutados en vez de codificarlos explícitamente. Es por eso que resulta de vital importancia implementar el modelo de cobertura y evaluar ésta desde etapas tempranas del proyecto. Este mecanismo permite realizar ajustes sobre el generador de estímulos aleatorios y guiar al testbench a casos de prueba pendientes de generación.

Mediante el uso de aserciones es posible describir casos de interés los cuales deben generarse en algún momento de la simulación. Con el uso de aserciones, es posible definir desde casos combinacionales simples hasta secuencias complejas producidas a lo largo del tiempo. Si el generador aleatorio produce un caso que cumple con lo especificado en la aserción, se deja registro del suceso. Finalizada la simulación, el ingeniero de verificación evalúa que porcentaje de los casos de interés se han producido

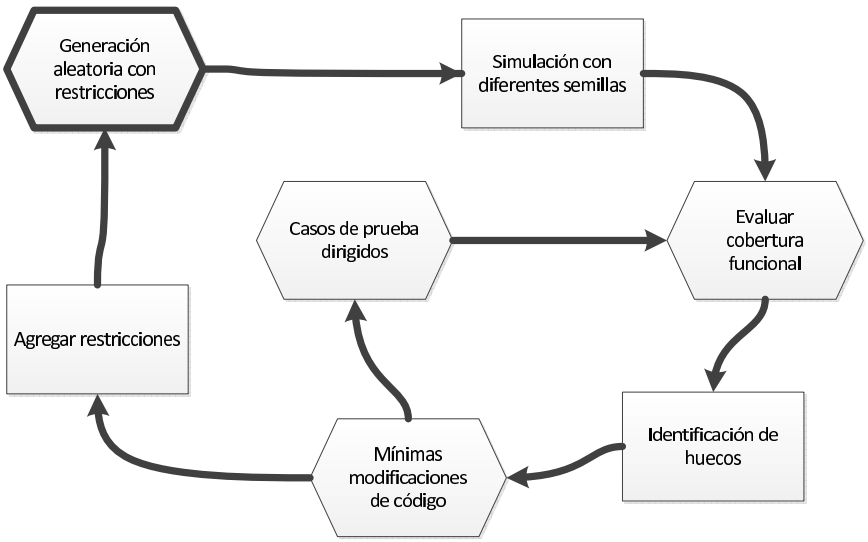


Figura 2.5: Fases en la verificación dirigida por la cobertura

según las aserciones activadas. De esta manera, se tiene conocimiento de qué restricciones del generador aleatorio ajustar para crear los casos restantes. La clave para lograr una CDV más eficiente es realizar el ajuste de manera automática. Para esto es necesario contar con el soporte de la herramienta de simulación y desarrollar estructuras computacionales altamente eficientes para registrar la ocurrencia de casos de interés. Además, el motor de valores aleatorios debe proveer la flexibilidad de cambiar las restricciones durante la simulación.

En el Capitulo 3 se presenta un caso de estudio donde se aplica CDV en el contexto de la verificación de unidades de punto flotante.

2.4. Conclusiones del capítulo

En este capítulo se presenta la necesidad de verificar funcionalmente un diseño. Un error dentro del diseño puede traer diferentes consecuencias, desde el desprestigio del fabricante hasta cuantiosas pérdidas económicas y físicas. Considerando que siempre existen probabilidades de error dentro de un diseño, la verificación funcional es la encargada de detectar dicho error.

Tradicionalmente las pruebas sobre los diseños eran realizadas por el mismo di-

señador considerando su interpretación y generalmente basándose en el mismo conjunto de casos tomados como referencia al momento de diseñar. Esta verificación, básicamente tiende a verificar que el diseño cumple con lo implementado y no con lo que indica la especificación funcional. La verificación funcional moderna tiene como objetivo proveer diferentes técnicas y enfoques que permitan una verificación profunda, ágil y flexible para diferentes proyectos. No existe una metodología universal que pueda aplicarse en todos y cada una de las partes del diseño. Por lo contrario, se proveen un conjunto de enfoques que pueden ser aplicados a diversas partes. Por un lado, la verificación formal, posee un sustento matemático y asegura que la porción de diseño verificado no contiene errores. La desventaja que presenta la verificación formal es que sólo es aplicable a casos combinacionales o de una cantidad finita de ciclos. Cuando el diseño es de mayor complejidad o la cantidad de ciclos no es conocida a priori, la carga computacional para la verificación es alta y lo vuelve inaplicable.

La verificación basada en simulación, por otro lado, sigue siendo el método preferido por los ingenieros. Uno de los motivos de esta permanencia es el desarrollo de nuevos enfoques la evolución de existentes. Por ejemplo, la verificación basada en transacciones de la verificación tradicional se ha visto enriquecida con el soporte de la generación aleatoria con restricciones y la programación orientada a objetos en los nuevos lenguajes de verificación. Este hecho, no solo provee herramientas para aumentar el nivel de abstracción en los entornos de verificación, sino que además, abre nuevos caminos para el desarrollo de frameworks estándar para verificación funcional. OVM y UVM son dos casos exitosos descriptos en este capítulo. Estos frameworks han sido reconocidos por las ventajas de reusabilidad y flexibilidad que conllevan. Por otro lado, el enfoque de la verificación basada en aserciones provee herramientas para análisis formal y de cobertura funcional basándose en descripciones declarativas homogéneas. Una consecuencia de las aserciones es la verificación dirigida por la cobertura. Esta permite dirigir la generación de estímulos a partes aun no ensayadas de un diseño.

Capítulo 3

Experiencias aplicando Frameworks para Verificación Funcional en unidades de punto flotante

3.1. Introducción

La mayoría de los sistemas digitales actuales incorporan cálculos lógicos y aritméticos que realizan desde simples decisiones combinacionales hasta la ejecución de complejos modelos físicos. Según las necesidades del sistema digital, éste puede involucrar arítmetica de punto fijo o punto flotante. La operaciones de punto fijo son más simples y por consiguiente su implementación en hardware también lo es. Sin embargo, la aplicación del punto fijo puede no resultar natural al diseñador. Por otro lado, las operaciones de punto flotante resultan más convenientes para cálculos complejos (por ejemplo en modelos físicos) pero a expensas de un hardware más complejo. La complejidad en el diseño de estos sistemas hace imprescindible la aplicación de la verificación funcional. En la literatura, es posible encontrar precedentes que aplican tanto verificación formal ([17], [31]) como aquellos que utilizan métodos basados en simulación ([29]).

La complejidad de estos sistemas también alcanza a la verificación funcional. Esto

se debe no sólo por el tamaño de cores que implementan la verificación de operaciones en punto flotante, sino también por los numerosos casos extremos o *corner cases* que implementan las operaciones. Esta característica hace a la verificación formal prácticamente inaplicable. De esta manera, los métodos basados en simulación siguen siendo los principales métodos de verificación de operaciones de punto flotante.

En este Capítulo se presentan experiencias en la verificación funcional de un Sumador/Restador decimal de punto flotante descripto en VHDL y que debe ajustarse al Estándar IEEE 754-2008 [4]. Estas experiencias incluyen el desarrollo de un plan de verificación y posteriormente la presentación de un entorno de simulación basados en dos frameworks de verificación mencionados en el Capítulo 2: Truss and OVM.

3.2. Trabajo relacionado

Dada la novedad de los frameworks de verificación de código abierto, poco trabajo relacionado se encuentra en conferencias y revistas. En [60] se presenta un caso de estudio de verificación funcional aplicando Teal and Truss [42] (T&T). En esa experiencia se presenta la verificación funcional de un arreglo sistólico para resolver problemas del camino mínimo.

Con el objetivo de demostrar las principales características del framework OVM, [12] presenta la verificación de un diseño de código abierto de un conversor 8b/10b RTL. En este trabajo queda demostrado que para alguien con escasa experiencia en programación orientada a objetos, la aplicación de la metodología puede ser intimidatoria. Sin embargo, se puede evaluar claramente la alta flexibilidad, portabilidad y reusabilidad del entorno de verificación una vez que ya se han dado los primeros pasos. Dentro del dominio de verificación de unidades de punto flotante, en [29] se presenta una herramienta para generar conjuntos de operandos que representan casos de prueba (FPgen). Los operandos son generados de acuerdo a restricciones establecidas en el plan de testeo. Los autores demuestran las capacidades de FPgen verificando una unidad de división de punto flotante.

3.3. El estandar IEEE754-2008

El estándar especifica métodos y formatos para arimética de punto flotante en sistemas digitales. Además recomienda formatos para intercambio de datos y condiciones y tratamiento de excepciones.

3.3.1. Formatos

Se caracterizan por su base, precisión y rango del exponente. Cada formato puede representar un único conjunto de datos de punto flotante. Todos los formatos pueden ser utilizados como formatos aritméticos, es decir, que pueden ser usados para representar tanto los operandos como los resultados de operaciones de punto flotante descriptas en el estándar. Se definen cinco formatos básicos:

a. Tres formatos binarios, codificados en ancho de 32, 64, y 128 bits.

b. Dos formatos decimales, codificados en ancho de 64 y 128 bits.

El estándar recomienda la generación de formatos adicionales mediante la extensión de los básicos.

3.3.2. Conjuntos de datos de punto flotante

El estándar especifica los posibles conjuntos de datos representables en punto flotante. El conjunto finito de números de punto flotante que pueden ser representados por un formato en particular es determinado por los siguientes parámetros enteros:

a. b = Base 2 o 10

b. p = Número de dígitos de la mantisa(precisión)

c. $emax$ = Exponente máximo e

d. $emin$ = Exponente mínimo e

$emin$ debe ser 1 - $emax$ para todos los formatos. Los valores de esos parámetros para cada formato básico se muestran en la Tabla 3.2. Cada formato es identificado por su base y el número de bits utilizado en su codificación.

Cuadro 3.1: Formatos de punto flotante IEEE754-2008.

Format	**Binary (b=2)**			**Decimal (b=10)**	
Parámetro	*binary32*	*binary64*	*binary128*	*decimal64*	*decimal128*
p, digitos	24	53	113	16	34
emax	+127	+1023	+16383	+384	+6144

3.3.3. Estrategias de redondeo

El estándar define que cinco modos de redondeo deben ser estar presentes en el diseño y deben poder ser configurables de manera estática (siempre la misma estrategia) y dimámica (la estrategia puede variar según las necesidades). Las estrategias propuestas son:

a. roundTiesToEven, el número debe ser el punto flotante más cercano al resultado infinitamente preciso; Si dos números de punto flotante, igualmente cercanos, pueden representar a un resultado infinitamente preciso que no es posible representar, debe elegirse aquel con el bit menos representativo par.

b. roundTiesToAway, el número debe ser el punto flotante más cercano al resultado infinitamente preciso; Si dos números igualmente cercanos cumplen esta condición, debe entregarse aquel con mayor magnitud.

c. roundTowardPositive, el resultado debe ser el número punto flotante más cercano del formato (posiblemente $+\infty$) y no menor que el resultado infinitamente preciso.

d. roundTowardNegative, El resultado debe ser el número punto flotante más cercano del formato (posiblemente $-\infty$) y no mayor al resultado infinitamente preciso.

e. roundTowardZero, el resutado debe ser el numero de punto flotante del formato mas cercano y no mayor en magnitud que el resultado infinitamente preciso.

3.3.4. Aritmética infinita

El comportamiento del infinito en artimética de punto flotante deriva de los limitados casos de aritmetica real con operandos arbitratiamente grandes en magnitud, cuando ese límite existe. Los infinitos deben ser interpretados en el sentido afín, esto es:
$-\infty < \{cualquier\ numero\ finito\} < +\infty$.
Las operaciones con operandos infinitos son usualmente exactas y por lo tanto no disparan excepciones.

3.3.5. Operaciones con NaNs

Un diseño que realiza cálculos de acuerdo al estandar debe soportar operandos del tipo Not-a-Number *(NaN)*. Existen dos sub clases: *sNaN* (signal not a number), y *qNAN* (quiet not a number). La primera engloba representaciones para variables no inicializadas y apliaciones de la aritmética (tales como infinitos complejos o de rango extremamente amplios). Por otro lado, los Quiet NaNs, deben ser dejados a criterios de la implementación mostrando información de diagnostico heredada de datos o resultados inválidos o no disponibles.

Bajo el tratamiento por defecto de una excepción, cualquier operación que indique una operación inválida en la que se espera un resultado de punto flotante, el resultado debe ser un Quiet NaN. La indicación de un NaNs debe ser reservada para operandos que, bajo tratamiento por defecto de la excepción, indican una excepción de operación invalida para cualquier operación de cálculo, sin considerar las conversiones.

3.4. Diseño de un Sumador/Restador IEEE754-2008

En este capítulo se presenta como caso de estudio la verificación de un Sumador / Restador de punto flotante con representación decimal diseñado según el estandar IEEE 754-2008 [4]. El diseño presentado está descripto en código VHDL RTL y soporta las siguientes características:

Cuadro 3.2: Formatos de punto flotante decimales

	signo	Combinación	bits exponente	bits mantisa
decimal32	31	30-26	25-20	19-0
decimal64	63	62-58	57-50	49-0
decimal128	127	126-122	121-110	109-0

a. Los tres formatos decimales de punto flotante definidos en el estándar: decimal32, decimal64 and decimal128 (ver Tabla 3.2).

b. Operandos pertenecientes a las siguientes clases: Ceros, números normales y no normalizados, infinitos, y N.a.N (not-a-number) de dos clases: sNaN (signal not a number), y qNAN (quiet not a number).

c. Flags para manejo de excepciones: underflow, overflow, inexact e invalid.

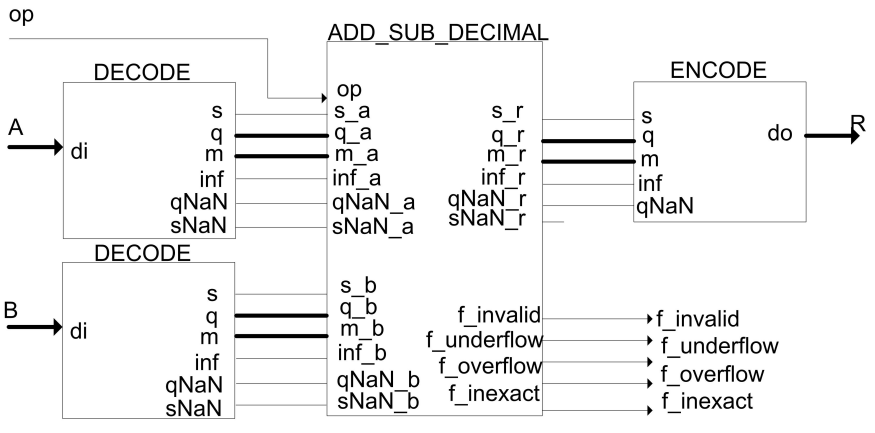
d. Todos los algorítmos de redondeo definidos en el estandar. Estos son: RoundTowardPositive, RoundTowardNegative, RoundTowardZero, RoundTiesToAway y RoundTiesEven.

La representación y valor del dato de punto flotante se infieren a partir de los campos en la Tabla 3.2. Por ejemplo, el valor del campo de combinación "11111" pertenece a la codificación de un NaN. Cuando el valor es "11110", entonces la codificación pertenece a $+\infty$ o $-\infty$ dependiendo del bit de signo. En cualquier otro caso, el campo de combinación contiene los bits principales tanto del exponente ajustado así como la mantisa del número finito. En [4] pueden encontrarse mayores precisiones sobre como inferir representaciones y valores de punto flotante.

La implementación del Sumador/Restador parametrizable se muestra en la Fig. 3.1. El módulo *DECODE* decodifica los operandos de entradas en señales de signo (s), exponente (q) y mantisa (m) así como en flags de infinito (inf), $sNaN$, y $qNaN$. El módulo *ADD_SUB_DECIMAL* realiza las operaciones aritméticas de suma o resta de punto flotante con los operandos decodificados. El módulo *ENCODE* codifica el valor de salida de *ADD_SUB_DECIMAL* según el formato correspondiente al estándar.

En el diseño de *DECODE* se ha considerado que, tal como indica el estándar, deben soportarse diferentes representaciones para el mismo número decimal: A diferencia del punto flotante binario, en formatos de punto flotante decimal un número puede admitir múltiples representaciones. El conjunto de representaciones en punto flotante de un mismo numero es llamado *cohorte* de ese número; los elementos del cohorte son distintas representaciones de un mismo número.

El sub-modulo *ADD_SUB_PROC* que se muestra en la Fig. 3.2, suma o resta los operandos según el código de operación presente en *op*. Para la suma o resta de números en punto flotante, es necesario forzar a que ambos operandos esten representados con el mismo exponente. Esta tarea la realiza el módulo *INPUT_MAKE* que realiza desplazamientos en los operandos ajustando la mantisa de manera acorde. El sub-modulo *ADD/SUB* realiza una operación especifica considerando de entrada el signo y la magnitud de ambos operandos con el mismo exponente. Con la mantisa extendida, el exponente y el signo, *OUTPUT_MAKE* genera mantisa y exponente de acuerdo al método de redondeo expecificado. Finalmente, *OUTPUT_MAKE*, determina si el resultado es inexacto así como si deben generarse flags de underflow u overflow.



[b]

Figura 3.1: Diseño del core IEEE_ADD_SUB

3.5. Plan de verificación

La construcción de un entorno de verificación va más allá de la instanciación de un framework. Además, es necesario conocer la especificación del diseño en términos de entradas, procesamiento y salidas. La generación de posibles entradas para un diseño suele ser una tarea difícil y, en algunos casos, imposibles (por ejemplo, diseños secuenciales donde una de sus entradas depende del resultado de uno o más ciclos anteriores). En esos casos, la completitud de la verificación depende del talento del ingeniero para reconocer al menos los casos más relevantes de entradas. En el contexto de la verificación del Sumador / Restador, los casos del (1) al (15) son definidos por restricciones matemáticas que configuran un generador de estímulos aleatorios.

Dado:

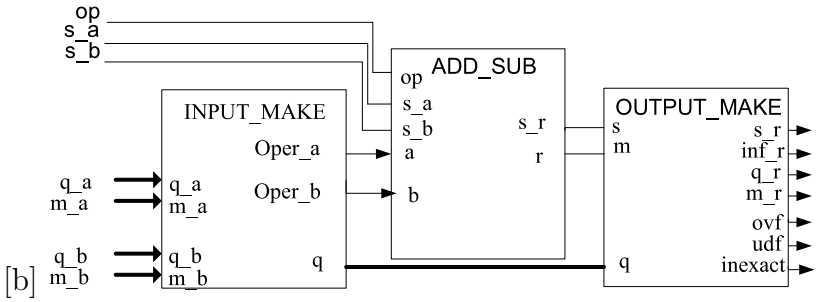
$$P = \begin{cases} 7 \; para \; decimal32 \\ 16 \; para \; decimal64 \\ 34 \; para \; decimal128 \end{cases}$$

donde P es la precisión de la mantisa para un formato dado,

$$Q_{max} = \begin{cases} 90 \; para \; decimal32 \\ 369 \; para \; decimal64 \\ 6111 \; para \; decimal128 \end{cases},$$

$$Q_{min} = \begin{cases} -101 \; para \; decimal32 \\ -398 \; para \; decimal64 \\ -6176 \; para \; decimal128 \end{cases},$$

donde Q_{max} and Q_{min} representan respectivamente los exponentes máximo y mínimos soportados para un formato dado:



[b]

Figura 3.2: Diseño del módulo IEEE_ADD_SUB_Proc

$\Xi = [Q_{min}..Q_{max}]$
$\Sigma = [-10^p - 1.,10^p - 1]$
y sea
$A = M_a * 10^{E_a}$
$B = M_b * 10^{E_b}$

Donde A y B son los operandos del DUV:

$$\{M_a, M_b, E_a, E_b / M_a, M_b \in \Sigma \wedge E_a, E_b = 0\} \tag{3.1}$$

$$\{M_a, M_b, E_a, Eb / M_a, M_b \in 10^{P-1}.,10^P - 1] \wedge E_a, E_b = 0\} \tag{3.2}$$

$$\{M_a, M_b, E_a, E_b / E_a, E_b \in \Xi \wedge M_a, M_b \in \Sigma\} \tag{3.3}$$

$$\{E_a, E_b, M_a, M_b / E_a = E_b \wedge (M_a + M_b) < 10^P - 1\} \tag{3.4}$$

$$\{M_a, M_b, E_a, E_b / \, |E_a - E_b| < 2 * P - 1 \wedge M_a, M_b \in \Sigma\} \tag{3.5}$$

$$\{E_a, Eb, M_a, Mb / (E_a - E_b) > Q_{max} - P\} \tag{3.6}$$

$$\{M_a, M_b, E_a, E_b / \, |E_a - E_b| < 2 * P - 1$$

$$\wedge M_a, M_b \in \Xi\} \tag{3.7}$$

$$\{A, B/A = 0 \wedge B \neq 0\} \tag{3.8}$$

$$\{A, B/A \neq 0 \wedge B = 0\} \tag{3.9}$$

$$\{A, B/A, B = \infty\} \tag{3.10}$$

$$\{A, B/A = -B\} \tag{3.11}$$

$$\{A, B/A = \infty \wedge B \in \Sigma\} \tag{3.12}$$

$$\{A, B/A \in \Sigma \wedge B = \infty\} \tag{3.13}$$

$$\{A, B/A, B = -\infty\} \tag{3.14}$$

$$\{A, B/A \in \Sigma \wedge B = sNaN\} \tag{3.15}$$

$$\{A, B/A = sNaN \wedge B \in \Sigma\} \tag{3.16}$$

Cada uno de estos casos de tests representan restricciones dentro de un entorno de verificación en SystemVerilog. La verificacción del DUV consiste en testear todas las arquitecturas de operandos IEEE754 (*decimal32*, *decimal64* and *decimal128*) con todos los casos presentados y utilizando todas las estrategias mencionadas en la

Sección 3.4.

3.6. Instanciación de los frameworks

Se construyen dos entornos de verificación para ALUs genéricos basandose en los principios de reusabilidad de ambos frameworks. Cada testbench admite la configuración en el formato de los operandos y representación para ser reutilizado en la verificación de cualquier ALU. En esta experiencia en particular, se implementan módulos específicos BCD para verificar un Sumador / Restador IEEE754. Como convención de nombres, el prefijo *ALU_xxx* representa módulos que pueden ser reutilizados para cualquier tipo de representación (ej. binario sin signo). Por otro lado, *ALU_BCD_xxx* son módulos programados específicamente para verificar diseños que trabajan con operandos IEEE754-2008. Tanto los entornos basados en T&T como aquellos basados en OVM de esta experiencia están codificados en SystemVerilog y son simulados bajo el entorno QuestaSim 6.5c de Mentor Graphics.

3.6.1. Truss

El framework Truss sugiere un diseño por capas donde el nivel superior interactúa a nivel de bit con el DUV mientras que el nivel inferior crea diferentes escenarios basados en los casos significativos descriptos en la Sección 3.5. La Fig. 3.3 muestra la arquitectura propuesta basándose en T&T.

En la capa *connection*, el módulo *BCD_Driver* traduce valores de transacciones a señales del Testbench mientras que *BCD_Monitor* realiza la tarea inversa, decodificando y analizando el resultado. La interconexión entre módulos es realizada mediante interfaces llamadas *Channels* en el contexto de Truss. El *ALU_Channel* posee un puerto de entrada donde los operandos junto con el operador matemático es enviado al DUV. Además, un puerto de salida captura y envía el resultado de la operación en conjunto con los flags al monitor. La capa *agent layer* atiende a los requerimientos de la capa de transacción. Dado que el diseño es completamente combinacional, no hay necesidad de sincronizar resultados, por lo tanto esta capa provee al driver de transacciones y asocia el resultado de la operación a la transacción actual.

En la capa *transaction layer*, tanto los operandos como el resultado son vistos

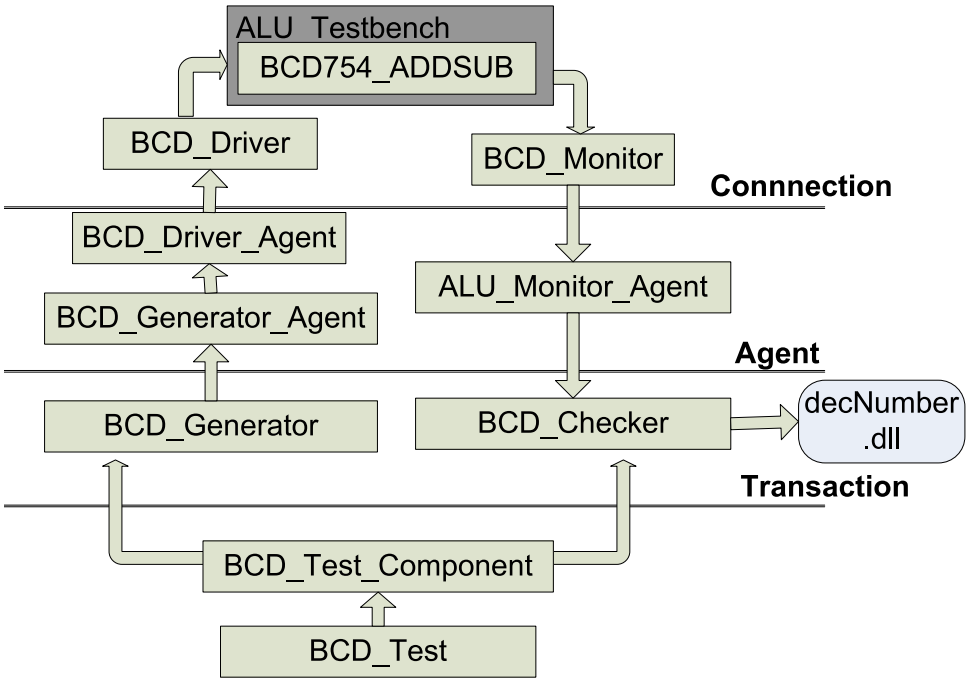


Figura 3.3: Instanciación de Teal & Truss para el Sumador / Restador

como dos conjuntos idénticos de mantisa y exponente. En esta experiencia, sólo las operaciones de suma y resta se han considerado. El componente *BCD_Generator* produce operadores y operandos de acuerdo a los tests definidos en la Secciíon 3.5 mediante la descripción de los mismos con restricciones (*constraints*) SystemVerilog. La clase *BCD_Abstract_Case* provee una plantila de métodos modelo para crear diferentes casos con restricciones con operaciones aritméticas aleatorias, es esta experiencia suma o resta. La verificación del resultado de la operación se encuentra en esta capa y el soporte lo implementa la clase *BCD_Checker*. Esta clase utiliza la biblioteca decNumber [22] como módelo de referencia para verificar que tanto el resultado como los flags del DUV sean correctos. En el nivel superior, el componente *BCD_Test_Component* configura el entorno y controla la generación de tests de una forma específica para verificar un comportamiento en particular del DUV. La Fig. 3.4 muestra las clases implementadas y su interrelación.

Como muestra de la reusabilidad del entorno propuesto, se puede mencionar que para verificar una ALU con diferente representación de operandos, tres clases son

totalmente reutilizables mientras que seis clases deben especializarse. Cada especialización infiere la codificación de treinta líneas de código.

3.6.2. OVM

La implementación con el framework OVM está basada en composición de objetos. En contraste con Truss, los tests no son parte del framework ya que describen una etápa específica en el plan de verificación. En el contexto de la verificación de la ALU un test *ALU_Ordinary_Tests* es invocado para generar transacciones cuya prueba no debería activar ningún flag a la salida del DUV. Por otro lado, *ALU_Corner_Tests* son transacciones que activan al menos un flag luego de la operación. Finalmente, *ALU_Regression_Tests* esta orientado para casos donde alguna prueba falla. Este test vuelve a ejecutar algunos de los casos ya probados y reanuda la ejecución de los restantes.

Una transacción en esta instancia de OVM representa una combinación específica de operandos y operación. En particular la clase *BCD_Transfer* restringe valores

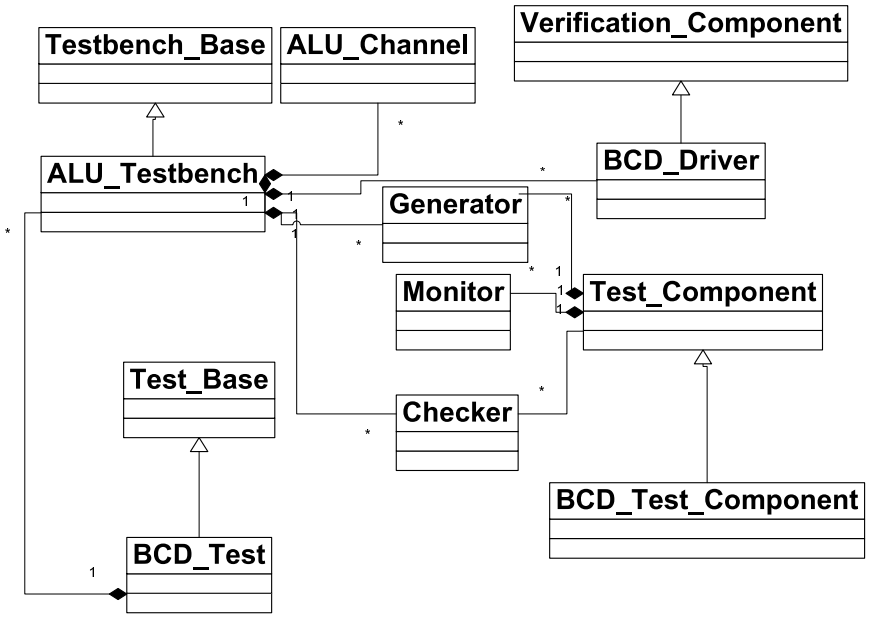


Figura 3.4: Diagrama de clases basado en T&T para la verificación del Sumador/-Restador.

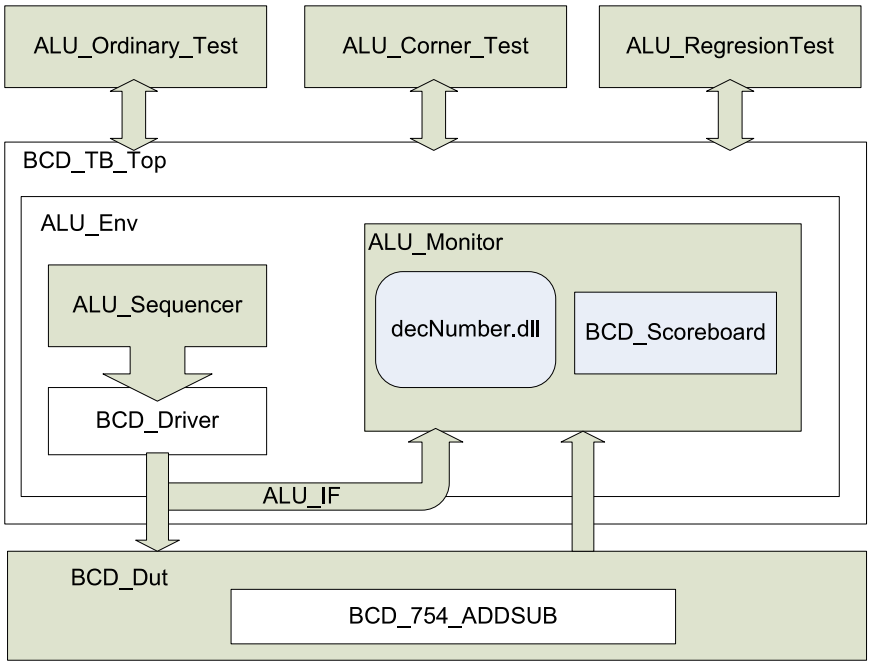


Figura 3.5: Instanciación de Ovm para la verificación del Sumador /Restador.

de mantisa y exponente a ser generados dentro un cierto rango. Cada clase que hereda de esta (*BCD_Transfer_Casex*) debe restringir sus operandos al subconjunto de números definidos en el estandar o bien redefinir el conjunto original para forzar un caso extremo. Una clase *BCD_Transfer_Casex* se implementa en SystemVerilog utilizando restricciones y siguiendo las definiciones mencionadas en 3.5. La clase *ALU_Base_Sequence* es un motor de transacciones básicas. Cada test define una secuencia de transacciones específica para luego enviarla al driver *BCD_Driver*. La clase *BCD_Driver* recibe transacciones a nivel de dato (en términos de mantisas y exponentes) y genera señales para el DUV en formato de operando IEEE754.

La clase *BCD_Monitor* decodifica el resultado desde la interfaz del DUV y lo compara con el del modelo de referencia basado en la biblioteca DecNumber de IBM. El resultado de esta comparación es luego enviado al scoreboard para su posterior análisis. Dado que OVM requiere que el DUV sea un módulo SystemVerilog, el DUV original escrito en HDL es encapsulado dentro de una clase SystemVerilog *ALU_Dut*. El diagrama de clases para el entorno implementado se muestra en la Fig. 3.6.

Para la implementación de una nueva representación numérica, el entorno OVM

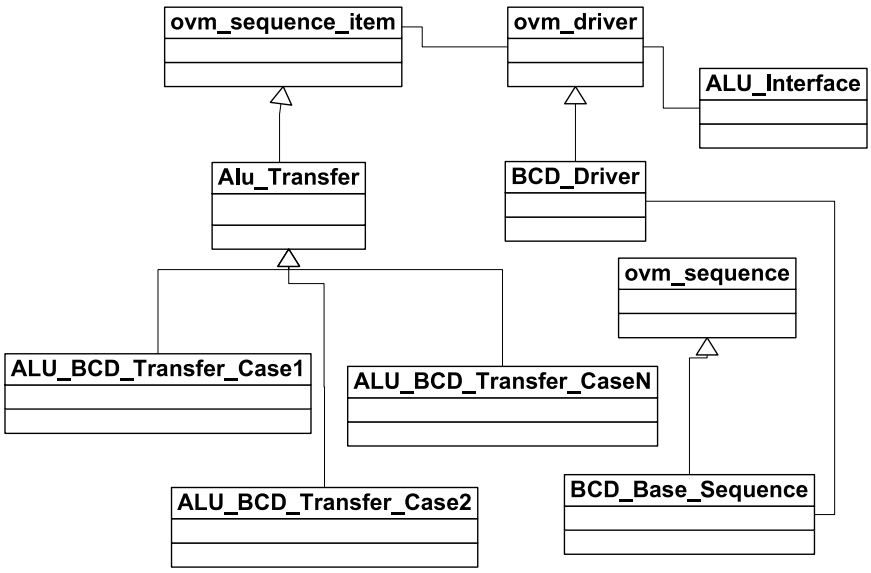


Figura 3.6: Diagrama de clases basado en OVM para la verificación del Sumador/-Restador.

requiere la especialización de solamente cuatro clases mientras que cuatro clases se mantienen intactas. Cada especialización requiere un promedio de veinte líneas de código.

3.6.3. Modelo de referencia

En esta experiencia se ha encapsulado a la biblioteca *decNumber* [22] para utilizarse como modelo de referencia que cumple con el estándar IEE754-2008. Esta soporta números enteros, de punto fijo y de punto flotante decimales incluyendo infinitos, NaN, y valores subnormales. Dado que esta biblioteca está completamente implementada en ANSI C, el módulo encapsulado utiliza la interfaz *Direct Program Interface* (DPI) de SystemVerilog para comunicar con funciones externas de C con testbenches de SystemVerilog. Tanto el encapsulado como la biblioteca son compilados para obtener una biblioteca de vínculos dinámicos (DLL) que es invocada desde el Testbench. Para poder verificar que el módulo BCD_ADD_SUB cumple con el estándar, el encapsulado de la biblioteca debe soportar la paremetrización en términos de cualquiera de los formatos soportados así como estrategias definidos en el

estándar.

3.7. Resultados

En esta experiencia, no sólo son considerados los resultados de la verificación, sino también se considera el esfuerzo humano invertido en aprender las tecnologías aplicadas asi como el tiempo invertido en la instanciación del framework. La siguiente subsección describe en primer lugar los errores detectados en el módulo IEEE-ADDSUB y luego se analizan los resultados de la aplicación de ambos frameworks.

3.7.1. Resultados de la verificación funcional

Estas nuevas metodologías son aplicadas luego de que el DUV ha sido probando utilizando métodos clásicos como tests dirigidos e inspección de código.

Ambos entornos evaluados llevan al DUV a producir los mismos errores. En particular, los errores evidencian diferencias en los signos de algunos resultados y errores al indicar flags en casos no extremos. Los mismos errores se producen independientemente del formato. A continuación se muestran algúnos de los errores detectados:

a. Caso de prueba 3: Caso de ejemplo: ∞ + 1.454E20 Error detectado: El diseño dispara el flag de *Inexacto*

b. Caso de prueba 8: Caso de ejemplo: 0E-71 + -3.920689E-26 Error detectado: El diseño dispara los flag de *Inexacto* e *Infinito*.

c. Caso de prueba 11: Caso de ejemplo:$\infty + (-\infty)$ Error detectado: El resutado del diseño ($sNaN$) difiere en signo con el modelo de referencia.

La verificación basada en frameworks propuesta dirige al DUV a los mismos errores que no fueron detectados previamente con los métodos tradicionales. Esta experiencia demuestra el potencial de las tecnologías aplicadas así como una forma más estructurada que los métodos tradicionales para evaluar diseños.

3.7.2. Evaluación de la experiencia

En esta experiencia, además del esfuerzo humano, se analiza algúnos criterios de calidad y características expuestas sobre los frameworks aplicados.

i. *Usabilidad:* T&T fue inicialmente escrito en C++ con el objetivo de proveer soporte para la POO. Luego, los autores decidieron reescribirlo usando SystemVerilog de modo tal de proveer una interfaz con un lenguaje orientado a la verificación funcional. Puede suceder que los ingenieros electrónicos posean mayor experiencia en la programación con C++ por lo tanto el tiempo de puesta en marcha de T&T sea menor al tiempo de instaciación de OVM. Por otro lado, OVM fue concebido como un framework en SystemVerilog puramente orientado a objetos, por lo cual puede resultar más accesible para ingenieros de la comunidad informática.

ii. *Portabilidad:* T&T provee una amplia lista de posibles simuladores donde ser ejecutado. La migración del código T&T a otro simulador implica simplemente en editar el script de compilación. A diferencia de T&T, la lista de simuladores soportados por OVM es más corta y sólo se provee documentación para el uso en simuladores de Cadence y de Mentor Graphics.

iii. *Reusabilidad:* Dado que varios de los componentes de OVM heredan de una superclase (*ovm_component*), los ingenieros pueden modificar o reusar varios componentes según su necesidad. Por otro lado, y a pesar de que T&T provee un entorno de verificación completo, resulta necesario implementar todas las capas del framework para verificar un diseño. Esta actividad puede insumir demasiado tiempo para un proyecto simple. Para este proyecto en particular, aplicar OVM insume el 50 % menos de líneas de código que T&T.

iv. *Disponibilidad:* T&T es una alternativa de código abierto propuesta y administrada por dos ingenieros. La información sobre bugs y documentación es publicada a traves de foros y listas de difusión. Por otro lado, si bien el código de OVM es abierto, es soportado y mantenido por dos companías líderes en herramientas para diseño electrónico (Mentor Graphics and Cadence). El soporte

técnico así como su documentación y versiones son actualizadas frecuentemente.

v. *Alternativa de código abierto:* En algunos proyectos, el costo de las herramientas de simulaciń profesionales pueden incrementar considerablemente el presupuesto. A pesar de que SystemVerilog es un estándar libre y de que OVM sea framework de verificación libre y de código abierto en SystemVerilog, no existen simuladores gratuitos que soporten este lenguaje. Por otro lado, la versión de C++ de T&T con un simulador de código abierto, tal como GHDL [24] o IcarusVerilog [59], es una combinación atractiva para empresas en desarrollo.

3.8. Conclusiones del capítulo

En este capítulo se describen experiencias al aplicar los Frameworks T&T y OVM en la verificación funcional del diseño un módulo Sumador / Restador. Dos entornos de verificación genéricos son construidos y luego se especializan las clases correspondientes al estandar IEEE754-2008. Aunque ambos entornos han descubierto los mismos errores del DUV, se han observado diferentes experiencias considerando aptitudes del equipo de verificación, aprendizaje de la tecnología, costo de la solución, etc. Con respecto al esfuerzo humano, se comprobó que para el entorno basado en T&T se necesitan mayor cantidad de líneas de código que para instaciar un entorno en OVM. Por otro lado, T&T resulta de más facil comprensión con menos conocimiento de OOP que OVM. A pesar de que SystemVerilog es un estandar de un lenguaje libre, no existen simuladores libres que lo soporten. En cambio es posible ejecutar la versión C++ de T&T en diversos simuladores libres y de código abierto (por ejemplo, IcarusVerilog). Finalmente puede destacarse mayor soporte técnico para OVM que para T&T teniendo en cuenta frecuentes actualizaciones tanto del código fuente como de su documentación.

Capítulo 4

Diseño metodológico de monitores de verificación para unidades de punto flotante

Gracias a la capacidad de reprogramación que proveen las FPGAs, un enfoque simple de verificación funcional puede ser realizado mediante la síntesis del diseño para un chip específico y observando el comportamiento de las salidas en una placa de prototipos. Sin embargo, realizar pequeñas modificaciones sobre el diseño para corregir errores pueden incurrir en largos tiempo de compilación y síntesis hasta poder probar el diseño nuevamente. Este enfoque además de ser lento, no garantiza cobertura funcional, lo que es particularmente serio en un sistema crítico donde la presencia de un error encubierto es inadmisible. La verificación funcional no suele ser una tarea trivial y en algunos casos resulta en un cuello de botella sobre proyectos de sistemas digitales. Por ejemplo, la verificación de unidades de punto flotante (FPU) ha sido reconocida como una tarea compleja [30]. En particular esa dificultad aparece por tres diferentes aspectos. Primero, el espacio de búsqueda crece exponencialmente cuando el dominio de entrada aumenta. Segundo e independientemente del ancho de los operandos, cada operación puede contener combinaciones de operandos que implican un análisis más complejo del resultado, por ejemplo, *not a number (NaN)*. Tercero, el estándar introduce la noción de cohorte para el formato de punto flotante decimal. Esto significa que, para un operando dado existen diversas formas de repre-

sentar el mismo valor numérico. Si el resultado de la operación es inexacto, la mayor cantidad de datos significativos es preservada cuando se selecciona aquel cohorte cuya mantisa sea el mayor número entero que pueda ser almacenado en conjunto con el exponente, por ejemplo: 100,0 es codificado como $1000 * 10^1$ mientras que 100,00 es codificado como $10000 * 10^2$

En trabajos tales como [30], [17], [7], [31], se han desarrollado tanto métodos formales como basados en simulación para tratar con el problema de la verificación de FPUs. A pesar de algúnos éxitos en la aplicación de métodos formales ([18],[31]), la simulación sigue siendo el enfoque principal para este problema, especialmente cuando se abordan los casos extremos. Estos casos requirían una importante cantidad de trabajo manual ([17] [9]) si se aplicaran métodos formales y en algunos casos resultaría imposible la descripción de ciertas reglas.

Los métodos basados en simulación involucran la generación de entradas para estimular al diseño así como también de métodos para comprobar los resultados de sus operaciones. Es así como los operandos de la FPU pueden ser creados mediante un generador aleatorio con restricciones mientras que un monitor de señales recolecta el resultado del DUV. Dado que el conjunto de restricciones definidas en el generador pueden no cubrir el espacio de entradas en su totalidad, suele ser necesario corregir el generador de estímulos de manera tal que pueda cubrir esos huecos en el espacio de cobertura que no han sido generados. En general, la información estadítica sobre las entradas previas puede ser utilizada para realizar ajustes en la generación de nuevos casos. Este enfoque recibe el nombre de Verificación Dirigida por la Cobertura o en inglés Coverage Driven Verification (CDV).

Este Capítulo completa el enfoque propuesto en el Capítulo anterior donde solo se describen los pasos para la construcción de un monitor para verificación. Las contribuciones explícitas de este capítulo son:

- La aplicación de una estrategia mixta donde se combinan test aleatorios restringidos o en inglés *constrained random tests* (CRT) y el enfoque de Verificación Dirigida por la Covertura (CVD).

- El desarrollo de dos nuevos componentes de Propiedad Intelectual para verificación (VIP) funcional de diseños de FPUs: Un generador de entradas y un monitor de chequeo y cobertura. El uso combinado de ambos componentes

aseguran la completa verificación funcional del DUV.

En este Capítulo, la generación automática de estímulos y la recolección de cobertura son implementados y aplicados de forma separada pero su utilización conjunta asegura la completa cobertura del diseño. La generación de entradas puede realizarse de forma aleatoria basándose en casos cuyas combinaciones no se hayan generado con el objetivo de acelerar el proceso de verificación. Por otro lado, se define un monitor que recolecta información desde la interfaz del DUV. El objetivo de este componente es, no solo chequear la correctitud del diseño y el chequeo del cumplimiento de algún estándar, sino también el de proveer un completo análisis de cobertura en términos del dominio de entradas y casos extremos.

Ambos módulos pueden ser utilizados tanto dentro de un testbench tradicional así como parte de un entorno de verificación avanzado basado en OVM [25] o UVM [1].

Los casos de estudio, se enfocan en la verificación de las operaciones de suma y resta según el estándar IEEE754-2008 [4]. La verificación se realiza utilizando los componentes propuestos sobre dos diseños de FPU con diferente representación de opeandos: Binaria y Decimal. El generador de entradas implementado y el monitor de salidas proveen flexibilidad y reusabilidad mediante la modificación de parámetros, algúnos estáticos y otros dinámicos.

4.1. Trabajo relacionado

En [30], se presenta una herramienta para generación de tests para operaciones de punto flotante (FP). Esa herramienta (FPgen) ha sido diseñada para verificación funcional basada en simulación y genera casos de prueba compuestos por operandos de punto flotante en la operación de división. En [14] se aborda la verificación de sumadores de punto flotante con especificaciones reusables. En ese trabajo, los autores usan la técnica de Verificación simbólica de modelos extendidos o del inglés *extended word-level Symbolic Model Verification* (SMV). En ese trabajo se han realizado mejoras en la verificación utilizando Diagramas de Decisión Multiplicativos de Poder Híbrido (*o del inglés Multiplicative Power Hybrid Decision Diagrams*) (*PHDDs) e incorporan simulación simbólica condicional así como técnicas de cor-

to circuito. Basándose en los casos de análisis, las especificaciones de sumadores de punto flotante son divididas en cientos de sub especificaciones independientes de la implementación.

4.2. El mecanismo de verificación

Independientemente de la complejidad del sistema aritmético, su entorno de verificación estará básicamente compuesto por un generador de entradas que estimulará el DUV y un monitor conectado a sus entradas y salidas mediante las cuales se lo evaluará y presentarán los resultados de la verificación. Los entornos de verificación basados en frameworks proveen una estructura reusable para crear escenarios complejos, adecuados para la verificación de este tipo de diseños

Los componentes Monitores recogen la información tanto de entrada como de salida del DUV, realizan el análisis de cobertura y envían en forma de transacción la información al componente de chequeo. Este último es el responsable de realizar la comprobación del resultado del diseño. En este Capítulo, la generación de casos de pruebas se trata de una versión optimizada a la presentada en el Capítulo anterior. Tanto el análisis de cobertura como la funcionalidad de chequeo se han integrado dentro del monitor con el objetivo de proveer un módulo de verificación auto contenido. Los pasos para la construcción del monitor de verificación son descriptos en 4.2.1 mientras que las técnicas para la generación de casos son presentadas en 4.2.2.

4.2.1. Monitor de Verificación

Basado en [21] para sistemas digitales en general, un monitor para la verificación de unidades de punto flotante debe proveer no solo la funcionalidad de chequeo de resultados y flags, sino también debe de reportar que porcentaje del vasto dominio de entrada ha sido verificado.

Por un lado, el análisis de cobertura es utilizado como métrica de confianza. En particular es utilizada para asegurarse que el plan de verificación haya sido completado y profundizado tanto como sea posible [8]. Por otro lado, el análisis de cobertura debe ser considerado como una herramienta para descubrir o localizar huecos en el dominio aún no verificado. Basándose en [28] los pasos para construir un monitor de

verificación para diseños de FPUs consisten en:

- División del Dominio: Dado el tamaño de un dominio de entrada (números en punto flotante), sería prácticamente inaplicable probar todas las posibles combinaciones de entradas, razón por la cual se realiza una división balanceada del mismo en intérvalos.
- Identificación de casos extremos: Tal como se menciona en [30] existen diversos casos numéricos que son esenciales para asegurar cobertura funcional. Esos casos pueden ser difíciles de alcanzar mediante el uso de valores aleatorios de entrada y deben de definirse por extensión.
- Selección de un modelo de referencia: La tarea principal del monitor es comprobar que el resultado del funcionamiento del DUV sea correcto. Un modelo de referencia permite al monitor realizar esa comprobación contrastando la salida de ambos.

Modelo de Cobertura

El objetivo principal del ingeniero de verificación es el de asegurarse que el diseño cumple con las especificaciones. Dado que resultaría impráctico y en algúnos casos imposibles realizar las pruebas con todas las combinaciones de entradas posibles, este debe asegurarse de que al menos los casos más significativos estén cubiertos. El primer paso de la metodología propuesta en este Capítulo se centra en realizar la división del dominio de entrada D en k subconjuntos o *bins*. Cada uno de estos contiene la descripción por comprensión del caso de prueba que representa. En este trabajo, el dominio D es dividido en espacios de idéntico tamaño por cuestiones de simplicidad. Es posible realizar divisiones que no consideren idéntico tamaño pero no será abordado en este trabajo. Un análisis más profundo sobre división de dominios puede encontrarse en [13]. Sean n_{min} and n_{max} valores mínimos y máximos respectivamente, entonces el ancho d del dominio D es:

$$d = |n_{max}..n_{min}| \tag{4.1}$$

y la capacidad de cada bin es determinada por $cap = \frac{d}{k}$, de modo tal que en el *i-esimo* bin le corresponden a aquellos elementos N tal que

$$(i * cap) + n_{min} < N \leq ((i+1) * cap) + n_{min} \tag{4.2}$$

Se dice entonces que el valor de k define la *granularidad* de la cobertura para los subconjuntos del dominio D. Cuanto mayor sea el valor de k, menor será la capacidad del bin. En términos de verificación, si el dominio de entrada y salida es dividido en k intérvalos, escoger un valor adecuado de k es una cuestión de compromiso entre completitud y tiempo en el proceso de verificación. Más específicamente en el contexto de la verificación de FPU , "El dominio de mantisa" S puede ser dividido en s bins y "El dominio de los exponentes" E puede ser dividido en e bins. Formalmente:

$$S_p = [S_0..S_s] \tag{4.3}$$

y

$$E_p = [E_0..E_e] \tag{4.4}$$

el conjunto de bins de mantisas y exponentes del operando p, p={A,B} respectivamente y

$$o \in O = \{+, -, *, /, ...\} \tag{4.5}$$

el conjunto de posibles operaciones, el dominio de posibles casos C es definido por:

$$C = S_A \times E_A \times S_B \times E_B \times o \tag{4.6}$$

Con el objetivo de que el análisis de cobertura contemple tanto entradas como salidas del DUV, el resultado puede lograrse mediante la división de la salida en mantisa y exponente. Formalmente el conjunto R se define como:

$$R = S_r \times E_r \tag{4.7}$$

Debe considerarse que el análisis de las salidas debe realizarse de manera independiente con el análisis de cobertura de valores de entrada. La razón principal es que existen combinaciones de operandos que nunca logarán generar determinados valores de salida (por ej. la suma de los mayores operandos posibles, nunca podría producir el resultado más pequeño a la salida). Sin embargo, el mecanismo hasta aquí descrito no considera el análisis de casos extremos que puedan aparecer durante la simula-

ción. Más aún, con un generador de valores aleatorios, puede suceder que nunca se produzcan operaciones que involucren Infinitos y NaNs. Para estos casos, un nuevo paso en la metodología propuesta implica la definición explícita de los casos que describen cada caso extremo. Esta tarea debe ser realizada a mano y debe adaptarse al estándar [4]. Cada propiedad que es definida explícitamente recibe el nombre de *Punto de Cobertura* o en inglés *Coverpoint*.

Se dice que la verificación de la funcionalidad del diseño está completamente cubierta si el monitor ha registrado al menos c instancias en cada bin C, al menos r instancias en cada bin de R y todos los puntos de cobertura han sido cubiertos al menos m veces. Los parámetros s, e y r, definen la granularidad de la cobertura mientras que c, y m definen el número de aciertos necesarios para considerar que el bin o el coverpoint haya sido cubierto.

También, se ha definido un parámetro adicional g que mide el grado de avance de la verificación. El objetivo de g es detener la simulación una vez que se haya logrado la cobertura adecuada, siguiendo la expresión de control:

$$\sum w_i * cpt_i + \sum w_j * cvg_j \geq g \tag{4.8}$$

donde w_i es el peso de cada coverpoint i, cpt_i es la cobertura del coverpoint i, w_j es el peso del bin j y cvg_j es la cobertura del bin j. La primer suma representa la porción de la cobertura total provista por los coverpoints mientras que la segunda está relacionada con los bins. La tarea de recolectar la información de cobertura y detener la simuación una vez que se haya superado el umbral g es realizada por un módulo especializado del entorno de verificación llamado *Coverage collector*.

Modelo de referencia

Alcanzar el 100 % de la cobertura funcional en la simulación no significa que el diseño cumpla con las especificaciones. La correctitud del diseño debe ser también verificada y se ha designado esa responsabilidad en este trabajo al monitor. En principio, la verificación del resultado es aún más importante que la cobertura en sí ya que esta última no debe de ser considerada si el diseño presenta comportamiento erróneo. Por esta razón, el modelo de referencia debe de proveer un mecanismo para detener la verificación ante resultados inesperados. El tercer paso propuesto en este

trabajo involucra la adopción de un modelo de referencia. Existen diversas formas de implementar un modelo de referencia. Cualquiera de las siguientes puede ser aplicada:

- Un diseño previamente verificado: Un diseño HDL el cual ya cumpla con el estándar es instanciado como parte del monitor. Cada caso generado alimenta tanto al DUV como al HDL de referencia. Finalmente, los resultados de ambos diseños son comparados.

- Definición tabular: Cada fila de una tabla consiste en un par de operandos, una operación y un resultado. Por más que resulte una tarea laboriosa, una aplicación de esta técnica se encuentra en [22].

- Biblioteca Software: Dado que la mayoría de los Lenguajes de Verificación de Alto Nivel (o en inglés High Level Verification Languages (HVL)) proveen una interfaz que permite interactuar con lenguajes de programación de alto nivel, es posible incluir una biblioteca software que provea algoritmos que imiten el comportamiento del DUV, para finalmente, poder comparar sus resultados.

Sea cual sea la opción adoptada, el modelo de referencia debe encontrarse en una unidad separada, usualmente conocida como *Checker*. En el caso más general, el Checker puede ser visto como una caja negra que recibe las entradas y salidas del DUV. Su función es reportar si el comportamiento del DUV es el adecuado y, en caso contrario, detener la simulación.

4.2.2. Generación de entradas

La generación de entradas puede ser tanto un proceso manual como autmático. El primero requiere un gran esfuerzo de diseño para describir cada posible entrada y su correspondiente salida. El segundo utiliza generación de estímulos aleatorios con restricciones para crear entradas que prueban una característica de interés del DUV. Un Test Aleatorio con Restricciones tiene diversas ventajas:

- Un CRT puede crear casos que no han sido planificados por el ingeniero de verificación.

- Reducen sesgos producidos por la intervención humana

- Aumenta la productividad

Las restricciones de un plan de verificación son priorizadas y evaluadas una por vez. Al mismo tiempo, se mide la cobertura del diseño.

Sin embargo, a pesar que todas las restricciones descriptas sean aplicadas, no significa completitud en la cobertura del diseño.

Idealmente, se espera que el incremento por caso de test (o en inglés Increment Per Test Case) sea:

$$IPTC = \frac{1}{|C|} \tag{4.9}$$

La Figura 4.1 muestra la performance CDV en su caso ideal vs. CRT. La caída del rendimiento del CRT se debe a que durante la simulación se generan casos de similares características que pertenecen al mismo bin. Para corregir este defecto, en este trabajo se agregan dos mejoras posibles al CRT:

- Cambiar la semilla del generador aleatorio cuando la performance comienza a disminuir bajo cierto valor *ct*. Repetir esta solución al menos *st* veces.

- Ajustar el generador aleatorio de manera tal que produzca entradas que no hayan sido cubiertas.

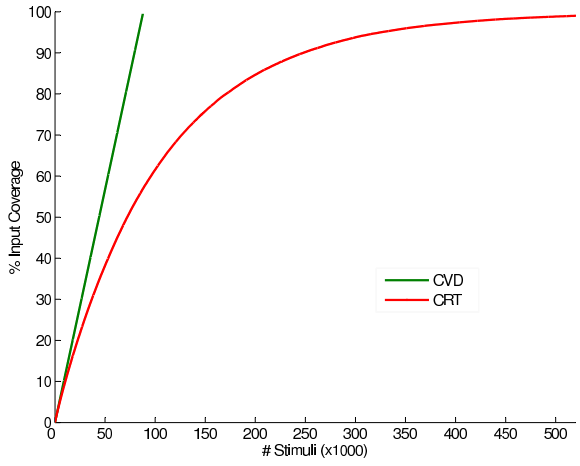


Figura 4.1: Comparación de coberturas - CVD vs CRT

4.3. Caso de estudio: Suma y Resta

El monitor es parametrizado en las siguientes características claves:

- Ancho de Operandos: 32, 64, 128 bits
- Representación (Binaria o Decimal)
- Estrategia de redondeo (como se menciona en 3.3.3)
- #Bins en los que se divide el Dominio (s, e, sr y er mencionados en 4.2.1)
- Codificación de operandos (0: Suma, 1:Resta)
- #Casos en cada bin (c mencionados in 4.2.1)
- #Casos en cada punto de cobertura (m mencionado in 4.2.1)
- Objetivo de cobertura (g mencionado en 4.2.1)
- ct (valor umbral donde comenzar el proceso dirigido por la cobertura)

Además, el monitor diseñado posee otros atributos configurables como creación de reportes, tamaños de buffers de entrada/salida (para diseños segmentados) y configuración de eventos de muestreo. Los coverpoints de utilidad para cada representación en particular son definidas en un archivo independiente utilizando sintaxis declarativa de SystemVerilog.

Para lograr una verificación dirigida por la cobertura, además se diseñó un generador ajustable. Este puede ser parametrizado según las siguientes características:

- Ancho del operando: 32, 64, 128 bits
- Representación (Binaria o Decimal)
- Codificación de las operaciones
- Casos extremos activos
- Restricciones de generación.

4.3.1. Estructura del generador

El generador está básicamente compuesto por varios sub componentes de entradas.

- Codificador aritmético: Puede ser implementado como un módulo o función que traduce un par de entrada $< Mantisa, Exponente >$ al formato de representación del DUV. Para aumentar la flexibilidad del generador, se deben incluir dos tipos de codificador (BCD y binario).

- Conjunto de restricciones: El conjunto de reglas para generar casos específicos que llevan al DUV a un estado de interés. En este contexto, las restricciones son expresadas en término de rangos numéricos de manera tal que el generador producirá valores aleatorios dentro de eses rangos.

- Interacción con el monitor: El generador debe proveer al monitor de una interfaz para comandar la CDV.

4.3.2. Definición de puntos de cobertura

Para el monitor propuesto, se han definido varios puntos de cobertura. Dado que existen cinco diferentes formatos posibles, algunos de los puntos de cobertura pueden ser útiles para evaluar cierto formato y pueden no tener sentido en otros.

A continuación se definen los puntos de cobertura específicos para las operaciones de suma y resta:

Para decimal64 y decimal128 sea Ξ el conjunto de valores válidos para exponentes y Σ los valores válidos de mantisa:

$$\Xi = [Q_{min}..Q_{max}] \tag{4.10}$$

$$\Sigma = [-10^p - 1.,10^p - 1] \tag{4.11}$$

y sean

$$A = M_a * 10^{E_a} \tag{4.12}$$

$$B = M_b * 10^{E_b} \tag{4.13}$$

Donde A y B son los operandos decimales del DUV. Se definen los siguientes puntos de cobertura:

$$\{M_a, M_b, E_a, E_b / M_a, M_b \in \Sigma \wedge E_a, E_b \in \Xi\} \tag{4.14}$$

$$\{M_a, M_b, E_a, E_b / M_a, M_b \in \Sigma \wedge E_a, E_b = 0\} \tag{4.15}$$

$$\{M_a, M_b, E_a, Eb / M_a, M_b \in [10^{P-1}.,10^P - 1] \wedge E_a, E_b = 0\} \tag{4.16}$$

$$\{M_a, M_b, E_a, E_b / (M_a + M_b) < 10^P - 1 \wedge E_a = E_b\} \tag{4.17}$$

$$\{M_a, M_b, E_a, E_b / M_a, M_b \in \Sigma \wedge |E_a - E_b| < 2 * P - 1\} \tag{4.18}$$

Los puntos de cobertura de 4.15 a 4.18 son definidos para asegurarse que el DUV realiza una correcta alineación del punto sin provocar overflow o underflow. Por otro lado, para aquellos diseños cuyos operandos tienen representación binaria:

$$\Xi = [0,0..,1,999., 98] \tag{4.19}$$

$$\Sigma = [-2^p - 1.,2^p - 1] \tag{4.20}$$

y sea

$$A = M_a * 2^{E_a} \tag{4.21}$$

$$B = M_b * 2^{E_b} \tag{4.22}$$

Donde A y B son operandos binarios del DUV:

$$\{M_a, M_b/(M_a + M_b) = 1\} \tag{4.23}$$

$$\{M_a, M_b/Ones(M_a), Ones(M_b) < i, \forall i : 0..P\} \tag{4.24}$$

$$\{M_a, M_b, E_a, E_b/Ones(M_a) = Ones(M_b)\} \tag{4.25}$$

Donde la función *Ones* representa la función de conteo de cantidad de unos en el argumento.

$$\{M_a, M_b, E_a, E_b/M_a - M_b < 2^{-P} \wedge E_a = E_b\} \tag{4.26}$$

Los casos de test 4.3.2 y 4.3.2 prueban la alineación de números flotantes binarios mientras que los casos 4.3.2 y 4.26 prueban diferentes secuencias de *Ones* en el operando representado.

Los siguientes puntos de cobertura son compartidos en el análisis de ambas representaciones:

$$\{M_a, M_b, E_a, E_b/(E_a - E_b) > Q_{max} - P\} \tag{4.27}$$

$$\{M_a, M_b, E_a, E_b/\,|E_a - E_b| < 2 * P - 1 \wedge M_a, M_b \in \Xi\} \tag{4.28}$$

$$\{A, B/A = 0 \wedge B \neq 0\} \tag{4.29}$$

$$\{A, B/A \neq 0 \wedge B = 0\} \tag{4.30}$$

$$\{A, B/A, B = \infty\} \tag{4.31}$$

$$\{A, B/A, B = \infty\} \tag{4.32}$$

$$\{A, B/A = -B\} \tag{4.33}$$

Los puntos de cobertura 4.27 a 4.33 aseguran el correcto comportamiento del DUV con el manejo del signo del resultado.

$$\{A, B/A = \infty \wedge B \in \Sigma\} \tag{4.34}$$

$$\{A, B/A \in \Sigma \wedge B = \infty\} \tag{4.35}$$

$$\{A, B/A, B = -\infty\} \tag{4.36}$$

Los puntos de cobertura definidos de 4.34 a 4.36 aseguran el correcto comportamiento del DUV al manejar infinitos en al menos uno de los operandos.

$$\{A, B/A \in \Sigma \wedge B = sNaN\} \tag{4.37}$$

$$\{A, B/A = sNaN \wedge B \in \Sigma\} \tag{4.38}$$

Los puntos de cobertura 4.37 y 4.38 comprueban el comportamiento del DUV al manejar números no normalizados.

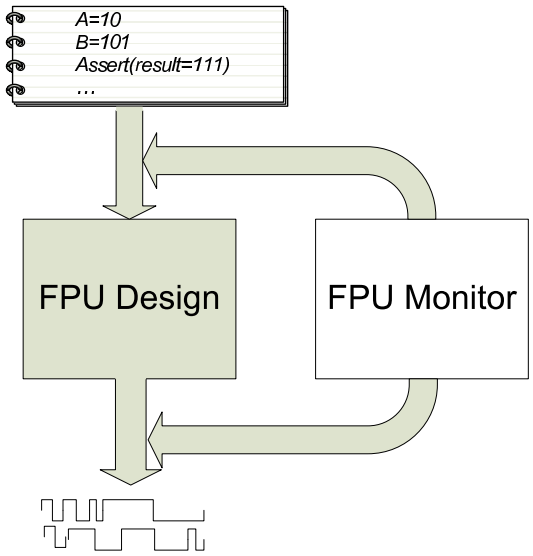


Figura 4.2: Monitor instanciado dentro de un testbench tradicional

4.4. Implementación

Siguiendo los pasos descriptos en la sección 4.3, se implementa un componente generador de entradas y un componente monitor para verificar diseños de FPUs que deben cumplir con el estándar IEEE754-2008. Ambos componentes pueden ser utilizados de manera independiente dentro de un entorno de verificación tradicional basado en testbenches (Fig. 4.2) así como dentro de complejos entornos de verificación basados en frameworks (Fig. 4.3) tal como se describe en el Capítulo anterior. Ambos componentes se han diseñado con el objetivo de proveer flexibilidad (soportar diferentes formatos), reusabilidad (ser utilizado en diferentes entornos de verificación) y fácil instanciación (estar listo para ser utilizado sólo con ajustar algunos parámetros). Se utiliza SystemVerilog como lenguaje de verificación de alto nivel, por lo tanto en el nivel superior, ambos componentes son vistos como módulos caja negra parametrizable.

4.4.1. Implementación del módulo generador

Este módulo restringe la generación de entradas según las propiedades descriptas en 4.3.2 utilizando sintaxis declarativa. La señal *csd* comanda el cambio en la semilla

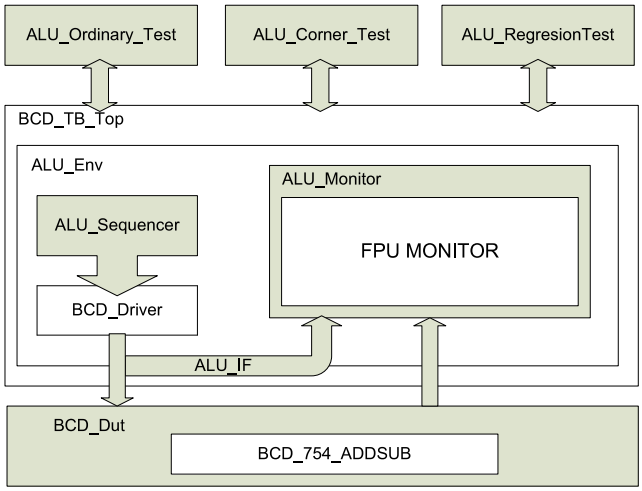


Figura 4.3: Monitor instanciado como parte de un entorno basado en OVM

de generación de valores aleatorios cuando la tasa de crecimiento de la cobertura cae bajo cierto umbral *ct*. Además, el módulo implementado debe generar entradas basándose en las restricciones del plan de verificación o bien aquellas reglas que son ajustadas por el monitor con el objetivo de ejecutar la CDV. La señal *cvd* selecciona el modo de generación. Tanto *cs* como *cvd* pueden ser controladas tanto desde el monitor como desde el entorno de verificación. La Figura 4.4 muestra la estructura de este componente. Debe notarse que la salida del módulo de verificación son señales HDL estándar.

4.4.2. Implementación del colector de cobertura

El *Colector de cobertura* es codificado como una unidad separada utilizando sintaxis declarativa SystemVerilog y utilizando construcciones nativas del lenguaje (bins, covergroups y coverpoints). Los parámetros del monitor definidos en 4.2.1 son codificados como parámetros de un módulo SystemVerilog y definen tamaños de estructuras y objetivos de cobertura.

En cada evento del muestreo, el colector marca el bin apropiado como un éxito y setea la señal *csd* si la performance de la cobertura ha bajado de cierto umbral *ct*. Si esto ocurre más de *sd* veces, la señal *cdv* es puesta en alto de modo tal que se inicie la generación de casos basados en la cobertura.

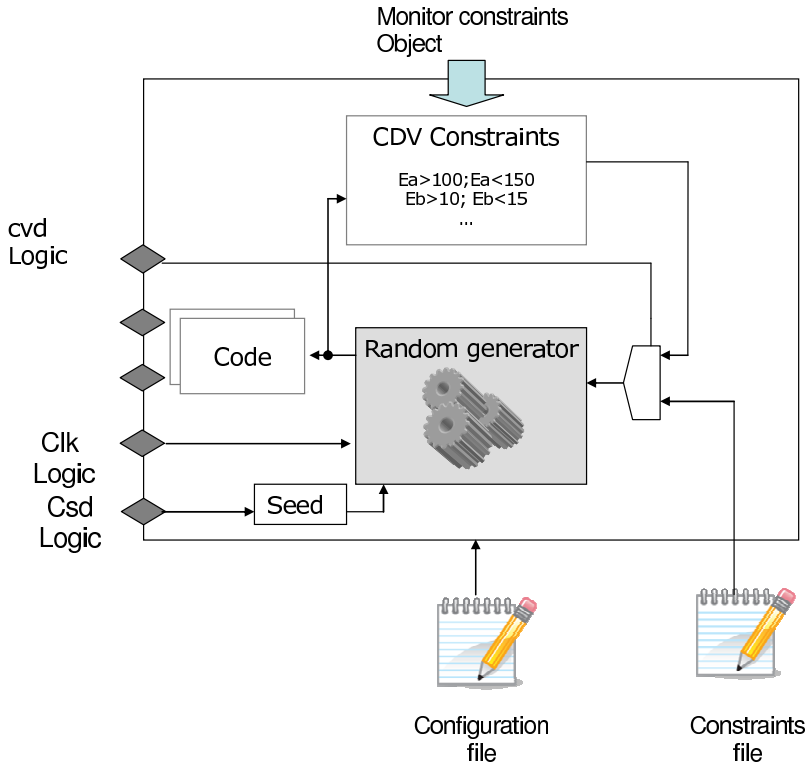


Figura 4.4: Estructura del generador de entradas

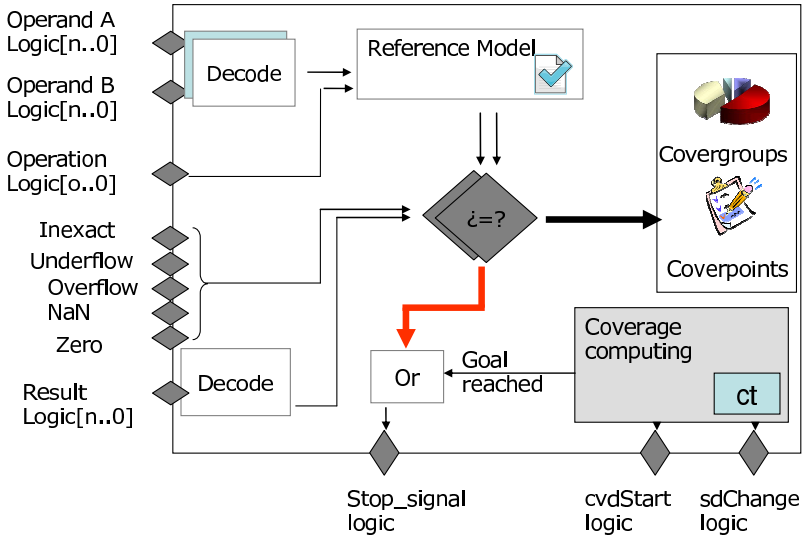
Dado que SystemVerilog no provee la funcionalidad de reporte en tiempo de simulación de la cobertura de un bin en particular, se implementa una estructura software mínima y de acceso rápido. Tal como se menciona en la Sección 4.2.1, el espacio de pruebas de entradas es construido mediante el producto cartesiano entre mantisa y exponente de ambos operandos. La estructura del bin es implementada utilizando una matriz de cinco dimensiones. Dado que al comienzo esta estructura será rala, se utiliza una estructura basada en diccionarios. Para encontrar algún bin aun no cubierto, se itera sobre la estructura hasta encontrar un bin vacío. La restricción que representa a ese bin, es creada y comunicada al generador de entradas.

4.4.3. Implementación del módulo Checker

Para la implementación de este módulo se utilizan dos bibliotecas software escritas en lenguaje C++. La biblioteca SoftFloat [32] se incluye para verificar diseños tienen representación binaria. Por otro lado, la biblioteca DecNumber de IBM [22]

se utiliza para proveer soporte a la verificación de diseños con aritmética decimal. Dado que ambas bibliotecas se encuentran codificadas en ANSI C++, se utiliza la interfaz de SystemVerilog DPI (Direct Programming Interface) de manera tal que el *Checker* realice llamada a funciones C++ de forma transparente desde una tarea SystemVerilog. Como el monitor es paramétrizable en representación tanto Binaria como Decimal, ambas bibliotecas son incluidas y luego se realiza el llamado de función apropiado,

Figura 4.5



4.4.4. Instantiación dentro de Frameworks de Verificación

Los módulos implementados pueden ser instanciados dentro de frameworks de verificación avanzados tales como Open Verification Methodology (OVM) o su sucesor Unified Verification Methodology (UVM). Esta subsección describe la inclusión del monitor y del generador como componentes OVM.

Un objeto *Sequencer* en OVM crea secuencias de entradas para estimular el DUV. En este contexto, el programador puede escribir un conjunto de restricciones para cada secuencia para probar un comportamiento particular del DUV. El generador descripto en este trabajo debe ser agregado como un nuevo tipo de secuencia. De esta forma, cada vez que se le solicita al Sequencer una nueva entrada, el generador

responderá con un nuevo caso a ser analizado. Dentro del contexto de OVM, este caso es llamado una *Transaction*.

Dado que la generación puede ser controlada por el generador de valores aleatorios con restricciones o dirigidos, el sequencer puede controlar la señal *cvd*.

Por otro lado, el monitor es más fácil de instanciar que el generador. Un componente monitor de OVM tiene el mismo comportamiento que el monitor implementado en este trabajo. El programador debe encapsular el código del monitor implementado dentro un componente monitor de OVM. De esta forma, cada señal de la interfaz SystemVerilog de OVM debe ser conectada a las entradas del monitor.

Los pasos para la contrucción del monitor descriptos en la Sección 4.2 incluye la funcionalidad de recopilación estadística. El componente de OVM *scoreboard* puede ser omitido o bien ser utilizado para realizar otro típo de métricas como por ejemplo cobertura de código.

4.5. Evaluación de los componentes de verificación

Para probar los módulos implementados se verifican dos diseños de FPU diferentes: uno con aritmética decimal [57] y otro con aritmética binaria [53]. Para esto, se planifica la selección de diversos parámetros con el objetivo de demostrar la flexibilidad y reusabilidad de los componentes desarrollados siguiendo los pasos definidos en 4.2. Además, se instancia el generador de entradas para demostrar las mejoras que se alcanzan al utilizar CDV.

Las pruebas se ejecutan bajo la siguiente plataforma:

- Hardware: AMD Turion ™X2, 4 GB RAM
- Software: Linux Debian 5, Mentor Graphics Questasim ™10.0a, OVM Framework 2.2

4.5.1. Configuración del Test 1

- Granularidad $s,\ e,\ r = b = 5,\ 10,\ 50,\ 100$

- Operaciones: *Suma y Resta*
- Codificación de Operaciones: 0 para suma y 1 para resta.
- Formato de operandos: *decimal64*, *decimal128*
- Estrategias de Redondeo: *Todas las estrategias*
- Coverpoints activos: 4.14 a 4.18 y de 4.27 a 4.38 descriptos en 4.3.2
- #hits c: 2
- #hits m: 2
- Valor umbral ct = 5 %, 10 %, 50 %
- Objetivo de la cobertura G : 100 %

No se detectan errores funcionales durante la ejecución.

4.5.2. Configuración del Test 2

- Granularidad s, e, $r = b$ = 5, 10, 50, 100
- Operaciones: *Suma y Resta*
- Codificación de Operaciones: 000 para Suma y 001 para Resta
- Formato de operandos: *binary64*, *binary32*
- Estrategias de Redondeo: *Todas las estrategias*
- Coverpoints activos: 4.18 a 4.38 descriptos en 4.3.2
- #hits c: 2
- #hits m: 2
- Valor umbral ct 5 %, 10 %, 50 %
- Objetivo de la cobertura G : 100 %

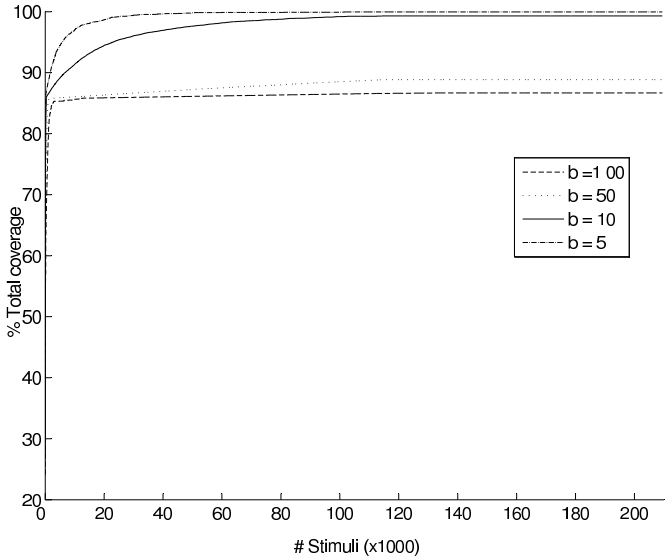


Figura 4.6: Cobertura Total para las diferentes capacidades de bins

No se detectan errores funcionales durante la ejecución.

A lo largo de ambas pruebas, existe un punto donde el crecimiento de la performance se detiene y se mantiene asintótica cercano al 80 %. La Fig. 4.6 muestra las tendencias de la cobertura para bins de diferentes tamaõs. Esto se debe a que un CRT produce cambios imperceptibles en la cobertura total dado que se comienzan a generar casos de iguales características a los antes ya generados. La cobertura permanece asintótica al 80 % para el resto de la simulación. En la Fig.4.7 se muestra el resultado de aplicar una estrategia mixta para diferentes valores de umbral. La mejora en la performance de la cobertura utilizando una estrategia mixta con un cierto valor umbral ct puede ser medida de la siguiente forma:

$$perf_{ct} = \frac{Time\ of\ CRT}{Time\ CDV_{ct}} \tag{4.39}$$

Para una estructura de cobertura con *b=100* (la más lenta en lograr el 100 % de cobertura) se aplica la estrategia mixta. La performance para los diferentes valores umbrales se resume en la Tabla 4.1

La cobertura total de la verificación para los diferentes valores umbrales se mues-

Cuadro 4.1: Performance del generador de entradas utilizando estrategia mixta

	ct =5 % de IPTC	**ct = 10 % de IPTC**	**ct = 50 % de IPTC**
Tiempo (ciclos)	221648	187210	100232
Aceleración	1.4x	1.9x	4.3x

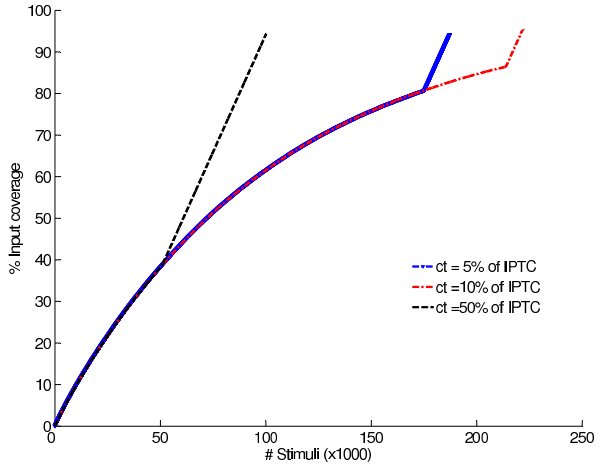


Figura 4.7: Cobertura de entrada utilizando estrategia mixta (b=100)

tra en la Fig.4.8

La flexibilidad del monitor se demuestra al poder ajustar diferentes parámetros para los distintos tests sin reescribir ni una sóla línea de código. La reusabilidad del monitor queda garantizada al tener que escribir sólo los casos extremos cuando se agrega una nueva operación a verificar. El procedimiento de comprobación el modelo de referencia y la infraestructura de cobertura están preparados para incorporar nuevas operaciones aritméticas del estándar tales como división y multiplicación. Finalmente, el modelo es fácilmente instanciable dentro de un módulo Verilog/SystemVerilog.

4.6. Conclusiones del capítulo

Este Capítulo presenta el desarrollo de dos componentes claves para la verificación de FPUs. Luego de describir los casos más significativos del estándar IEEE754-

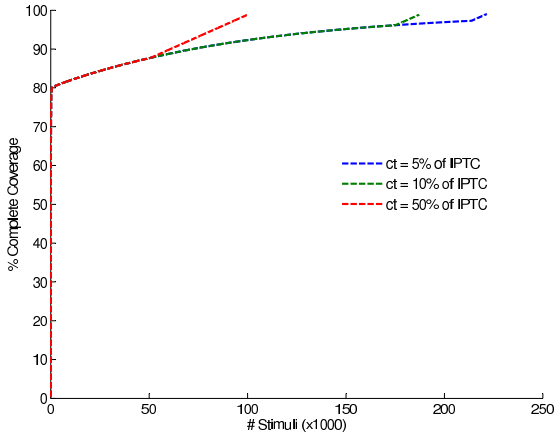


Figura 4.8: Cobertura total utilizando una estrategia mixta

2008 FPU, el Capítulo se orienta en la generación de casos de entrada basados en esas características. En particular, se presenta como caso de estudio la verificación de las operaciones de suma y resta. Dado que los casos de entrada generados no alcanzan a cubrir la verificación total del diseño, se inicia una etapa de generación de estímulos basándose en la cobertura. Con el objetivo de proveer un análisis completo de la cobertura, se establece una serie de pasos para definir el diseño de un monitor funcional. Primero, se dividen tanto el dominio de entrada como el de salida en bins de igual tamaño. Luego los casos extremos son definidos explícitamente

Ambos componentes se implementan en SystemVerilog y luego son evaluados como parte de un entorno de verificación de FPU basado el el framework OVM. Como DUVs se instancian dos diseños de FPU diferentes; uno que utiliza representación decimal y otro con representación binaria. Dado que no se han encontrado errores funcionales en ambos diseños durante la ejecución de los tests, puede concluirse que ambos cumplen con el estándar.

Por otro lado, las pruebas muestran que un CRT crea casos solapados que provocan el congelamiento en la cobertura total del diseño. Para mejorar este inconveniente, la semilla de generación de números aleatoria es cambiada. Si bien la semilla fue reemplazada dos veces, no se encontraron cambios significativos en la performance. La verificación continua con una fase de CVD. Se observa que este enfoque no sólo acelera el proceso de verificación hasta 4.5x sino que también es capaz de encontrar

casos no cubiertos.

La construcción estructurada y genérica del monitor muestra que es posible lograr componentes de verificación de gran flexibilidad y reusabilidad.

Capítulo 5

Verificación automatizada

5.1. Introducción

En el capitulo 2 se mencionó la importancia de que la fase de verificación funcional sea independiente de la fase de diseño. Es frecuente que tanto el plan de verificación como un entorno para la verificación preliminar de un diseño esté disponible antes de la primer versión de éste. Es por eso que tanto en el presente capítulo como en el siguiente, se propone un enfoque de la verificación funcional como herramienta para asistencia al equipo de diseño. La asistencia que aquí se propone apunta a, además de indicarle al diseñador sí lo implementado responde a las especificaciones funcionales, proveer de una herramienta de análisis y de depuración. En particular, y siguiendo con el eje central de la tesis, se presentan aportes destinados al análisis de diseños que involucran operaciones de punto fijo y punto flotante aplicando verificación basada en simulación.

En este Capítulo se presentan enfoques para la automatización de pruebas dentro de un entorno simulado así como estrategias para el análisis de resultados de las mismas. A diferencia de los Capítulos 3 y 4, la automatización que aquí se presenta vá más allá de la generación automática de entradas. Primero, se aborda la automatización para la creación de escenarios reales con gran precisión y mayor nivel de complejidad. Segundo, se propone el análisis de resultados orientado a proveer información sobre errores, desviaciones o cobertura de código. Finalmente, la automatización también involucra la presentación de los resultados de manera tal que

sean de utilidad para el diseñador. Todas las estrategias son coordinadas por una descripción de instrucciones recopiladas en un *Script*. En la Sección 5.2, se presentan las herramientas que son utilizadas para la automatización de pruebas mientras que en la sección 5.3, se proponen herramientas para el análisis de resultados. Finalmente, en la sección 5.6, se presenta un caso de estudio donde se aplican las estrategias desarrolladas.

5.2. Generación automatizada de casos de prueba

En el capítulo 3 se presenta la generación de estímulos aleatorios para la verificación basada en simulación de un diseño combinacional. Si bien el espacio de posibles valores de entrada es extenso e intratable, su tamaño es acotado. En los diseños secuenciales el tamaño del espacio de pruebas crece exponencialmente con cada ciclo de reloj ya que los valores de las entradas dependen de su estado anterior. Este hecho causa que la generación de casos de prueba para circuitos secuenciales sea más compleja y necesite un tratamiento diferente. Es por eso que resulta indispensable un análisis del espacio de posibles entradas y su comportamiento a lo largo de la prueba. La automatización puede generarse de diversas formas y en diferentes niveles de abstracción. En este trabajo se propone la utilización de Scripts para generación de escenarios y restricciones para nivel de secuencias.

5.3. Automatización mediante Scripts

Un *Script* (del latín *scriptum*, escrito) es un archivo de órdenes o archivo de procesamiento por lotes, que por lo regular se almacena en un archivo de texto plano. Los Script son generalmente interpretados. En el contexto de la verificación funcional, el uso habitual de los Script es el de realizar diversas tareas como generar valores del escenario, configurar rutas de almacenamiento de reportes e iniciar, controlar y detener la simulación. En algunos casos, la automatización puede llegar a ser extrema e involucrar:

- Compilación del entorno y el DUV

- Control de la Simulación
- Confección de gráficas detalladas.
- Reporte de Resultados/Errores al equipo de Verificación y Diseño mediante correo electrónico.

Un Script genérico para automatización de pruebas consta de los siguientes comandos:

Script 1 Script genérico para automatización de pruebas

```
1: Definición de escenarios ϵ = {ε_1, ..., ε_n}
2: Definición de conjunto de parámetros del diseño π = {π_1, ..., π_n}
3: para todo ε_i in ϵ hacer
4:   para todo π_j in π hacer
5:     Iniciar Simulación
6:     mientras no se produzcan errores hacer
7:       Simular DUV con parámetros π dentro de un escenario ε_i
8:     fin mientras
9:     si No ocurrieron errores entonces
10:      Registrar Cobertura
11:      Registrar Resultados de la simulación
12:    si no
13:      Registrar Situación del error
14:      Detener Script
15:    fin si
16:  fin para
17: fin para
```

El Script 1 define una estructura genérica para la automatización de pruebas. La línea 1 define por extensión todos los escenarios de prueba posibles que se han definido en el plan de verificación. En la línea 2 se definen las combinaciones o tuplas de parámetros posibles con los que se puede configurar el DUV. En el caso de ser más de un parámetro con los que se configure el DUV, pueden definirse por extensión todos los valores posibles que puede adoptar ese parámetro. En tal caso se debe agregar un nivel de iteración al bucle principal por cada parámetro extra. Las líneas

de 3 a 17 constituyen el bucle principal de la simulación. En la línea 7 la simulación se está ejecutando y finalizará cuando se hayan completado todas las secuencias de los tests. En el caso de producirse errores, no solo debe detenerse la simulación del escenario actual, sino que también debe cancelarse la ejecución de todo el bucle principal de verificación. Desde el punto de vista de la verificación funcional, carece de sentido recolectar cobertura funcional o continuar ejecutando pruebas sobre un diseño con errores ya que hay altas probabilidades que el error vuelva a repetirse con todas las secuencias. Sin embargo, desde el enfoque propuesto en este trabajo, se debe proveer de la mayor cantidad de información posible al diseñador y los casos con errores deben registrarse, y por el contrario, con el máximo nivel de detalle. La cobertura funcional en diseños con errores no aporta información significativa adicional, por lo tanto tampoco tiene sentido recolectarla.

Script 2 Enfoque propuesto

```
1: si No ocurrieron errores entonces
2:     Registrar Cobertura Funcional
3:     Registrar Resultados de la simulación
4: si no
5:     Registrar últimas secuencias sin error
6:     Registrar Situación del error
7:     Detener Script
8: fin si
```

Las líneas 9 a 15 deben reemplazarse por las propuestas en el algorítmo 2. Debe tenerse en cuenta que deben proveerse de mecanismos para registrar con el mayor nivel de detalle la situación del error desde el entorno de verificación. Estos mecanísmos se describen en la sección 5.3.1. El lenguaje para la codificación del Script en verificación funcional es generalmente TCL ya que la mayoría de los simuladores lo soportan. Otra opción posible es la utilización de algún lenguaje de Scripting soportado por el sistema operativo, por ejemplo, Bash de Unix. Esta última opción podría crear diferentes procesos que involucren un escenario específico, y ser ejecutados en paralelo si el simulador y la plataforma donde se ejecute el mismo lo soportan.

5.3.1. Automatización mediante componentes de verificación

El enfoque anterior implica iniciar una nueva simulación por cada combinación de parámetros y escenario. Esto se debe a que es necesario reconfigurar el entorno de verificación y el DUV seguún las necesidades de la prueba. Sin embargo, si la configuración del DUV es parametrizable, la automatización puede producirse desde los componentes de verificación en diferentes niveles de abstracción. En el más bajo nivel, las *transacciones* son convertidas a señales interpretadas por el DUV. Este nivel tiene una tarea establecida, y su comportamiento es siempre el mismo. Por esta razón, la automatización no involucra este nivel.

El nivel de *Secuencia* genera transacciones que contienen diferentes atributos, como por ejemplo, comandos y direcciones. En este caso es posible asignar diferentes valores a uno o más atributos de manera automática. Los valores pueden ser generados en forma aleatoria con restricciones o bien siguiendo un criterio de generación asignado por el nivel superior. Para la generación aleatoria se define $\overline{\alpha} = \{\alpha_a, \alpha_2, \ldots, \alpha_n\}$ al vector de atributos de la transacción y $R = \{R_1, R_2, \ldots, R_n\}$ al conjunto de restricciones. Es posible aplicar un subconjunto $R'(\overline{\alpha})$ de R con el objetivo de obtener automáticamente una nueva transacción. La selección de R' depende de que característica del DUV se desea analizar y debe responder a lo descripto en el plan de verificación.

En caso de que la prueba involucre atributos con valores constantes o conocidos a priori, la generación automática puede realizarse mediante una función matemática, (por ejemplo $seno(\alpha_i)$), constantes o bien valores previamente definidos en un archivo o tabla. Esta última estrategia es muy usada ya que permite definir valores de atributos sin necesidad de modificar el componente generador de secuencias (*Sequencer*) ni conocer detalles de la implementación del entorno. Otra ventaja interesante es que estos valores pueden proceder de ejecuciones previas, siendo de gran utilidad para el desarrollador en la tarea de depuración.

A nivel de *Test*, es posible automatizar la selección de diferentes secuencias. Si el nivel de Secuencia posee asignación directa de ciertos atributos, el Test debe indicar el origen de esos valores, ya sea una función matemática, la ruta del archivo donde se encuentran los valores o bien un valor fijo. En el caso de que el nivel de Secuencia genere valores de manera aleatoria, es responsabilidad del Test administrar

el subconjunto R'. La administración consiste en modificar R' de manera tal de generar secuencias válidas que conduzcan al DUV a un estado de interés. En ciertos casos, R' es generado de manera inválida con el objetivo de evaluar el DUV bajo condiciones de error. En tales casos, los Tests son específicos y su cobertura no debe ser evaluada. Es por eso, que también es responsabilidad del Test activar o desactivar el análisis de resultados y la confección de reportes.

5.4. Análisis automatizado de resultados

Al igual que en la sección 5.2, el análisis del comportamiento del DUV puede ser automatizado tanto a nivel de escenario mediante scripts o bien desde el entorno de verificación mediante componentes. Desde el enfoque tradicional, el entorno de verificación debe reportar los errores que se han encontrado durante la simulación indicando el estado de entradas, salidas y, si es posible, el estado interno del DUV.

5.4.1. Análsis mediante Scripts

En el enfoque propuesto por este trabajo, el análisis de los resultados debe presentar el mayor nivel de detalle de manera tal de asistir al diseñador. Dado que el formato de representación de los mismos puede contener diferente nivel de detalle, se propone el uso de diversas herramientas. Si bien los simuladores actuales proveen gran cantidad de herramientas para esta tarea, suele ser necesario el uso de herramientas externas. En el caso de que el DUV no haya presentado errores durante la simulación, se inicia una cadena de análisis como se muestra en la Fig. 5.1. Cada una de las etapas cumple una función específica que es de utilidad para el diseñador.

- Análisis de cobertura de código: En general, esta herramienta permite detallar que partes del diseño HDL se han activado. La mayoría de los simuladores soportan esta característica por lo cual no es necesario iniciar una herramienta externa.

- Análisis de cobertura funcional: Tanto el ingeniero de verificación como el diseñador utilizan esta información para estimar que proporción del plan de

cobertura se ha evaluado. La mayoría de los simuladores poseen la capacidad de analizar esta información.

- Análisis de área: Este tipo de análisis es dependiente de la tecnología a la que esté orientado el diseño. En particular, es necesario conocer los posibles dispositivos donde este será sintetizado. Cada fabricante provee su propia herramienta de síntesis o análisis que será invocada desde el Script.

- Análisis de frecuencia máxima: El análisis estático de frecuencia o STA (del inglés Static timing analysis) es un método para el cálculo de tiempos esperados en un sistema digital sin la necesidad de simulación. En particular, en el enfoque propuesto se enfoca en el análisis del camino crítico (en inglés *critical path*) definido como el mayor tiempo de procesamiento entre la entrada de la señal y su resultante.

- Análisis de Consumo: Estas herramientas realizan estimaciones de potencia consumida. Para esto, las mismas deben contar con la descripción del diseño. Más precisamente, utilizan información sobre el conexionado y las características de los recursos utilizados en determinada tecnología.

- Otros análisis: En el enfoque propuesto, se propone realizar análisis adicionales como por ejemplo, diferencias númericas o desviaciones. Las herramientas de simulación no proveen esta característica. Para el caso de análisis y reporte es posible utilizar herramientas como MatLab u Octave ya que proveen la capacidad de ejecución desde el mismo script.

5.4.2. Análsis mediante componentes de verificación

Dentro de la cadena de análisis presentada anteriormente existen etapas que necesitan algún tipo de soporte por parte del entorno de verificación. Por ejemplo, para indicar que se han producido errores, es necesario contar con un modelo de referencia que indique cual es el resultado esperado. Además debe de proveerse una estructura que permita ir registrando las secuencias de entradas que se han producido. Fuera del ámbito de los frameworks, ambos componentes pueden ser vistos como una única unidad conocida como *monitor de verificación*. Desde un entorno de verificación

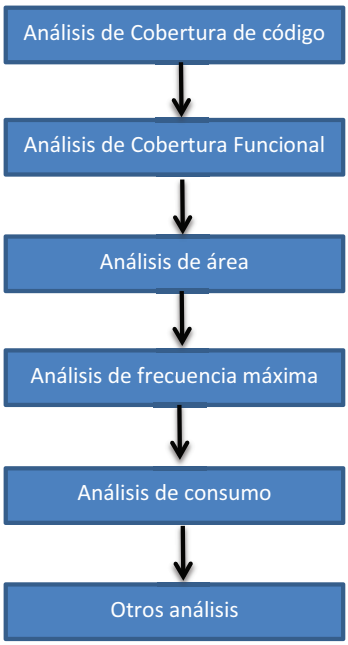


Figura 5.1: Etapas propuestas para el análisis

basado en framework hay dos componentes involucrados en esta tarea, el monitor de señales y el Scoreboard. Dado que este trabajo está fuertemente sustentado con el uso de frameworks, el tratamiento se realizará por separado.

Monitor de señales

Este componente interactúa directamente con la interfaz del DUV. Su función es generalmente conocida como Anti - Driver, ya que realiza la tarea inversa a este. En general, el monitor realiza las siguientes tareas:

- Recolección de señales del DUV: El Monitor decodifica las señales provenientes del DUV y crea una transacción de salida de mayor nivel de abstracción. Por ejemeplo, a partir de una secuencia de bits, se interpretan, teniendo en cuenta cierto protocolo, y se presenta en formato (Comando, Operación, Dato). Esta transacción es enviada posteriormente al Scoreboard.

- Verificación de protocolos: Dado que el monitor debe conocer estrictamente el protocolo de comunicaciones que posee el DUV, las violaciones de protocolo y su cobertura funcional son analizadas dentro del monitor.

- Logging de señales: Las señales deben ser almacenadas para su posterior análisis y visualización. En general, los simuladores actuales proveen rutinas para manejo eficiente de archivos.

Scoreboard

Este componente trabaja con transacciones en alto nivel que recibe desde el Monitor. La principal razón por la cual el Scoreboard trabaja a este nivel es la independencia de detalles de implementación del DUV y del modelo de referencia. Esto permite al equipo de verificación utilizar la misma estructura para verificar diferentes implementaciones, reutilizando completamente el entorno o bien implementando una acotada y pequeña porción de código del monitor. Las tareas del monitor son las siguientes:

- Analizar cobertura funcional: Utilizando un modelo de cobertura acorde al plan de verificación, se registra la ocurrencia de los atributos de la transacción de salida.

- Contrastar resultados: La comparación de las señales de salida se realiza utilizando un modelo de referencia. El resultado de esta comparación no es solamente binaria. Este trabajo propone que los resultados de la comparacón puedan ser valores numéricos, como por ejemplo, porcentaje de desviación.

- Confección de reportes: El scoreboard debe llevar estadísticas de fallas y aciertos. El formato de reporte puede ser un simple archivo de texto separado por comas.

- Recopilación de resultados para herramientas externas: Dado que en algunos casos puede ser necesario realizar análisis con herramientas externas, debe proveerse de métodos de comunicación con estas, por ejemplo, archivos de texto o mediante libreías.

Modelos de referencia Software

Resulta valioso poseer de antemano un modelo del diseño escrito en algún lenguaje de programación de alto nivel. Este tipo de modelos es particularmente útil cuando

el diseño está orientado a acelerar alguna una rutina software. La mayoría de los entornos de verificación soportan algún tipo de integración de rutinas en lenguaje de alto nivel. Por ejemplo, la interfaz DPI (Direct Program Interface) permite incluir dentro de un entorno SystemVerilog rutinas escritas en C++ de forma transparente e invocarlas como si se tratase de una simple función en SystemVerilog.

Modelos de referencia Hardware

En muchas ocaciones, un proyecto de diseño de hardware consiste en la optimización de alguna característica de un diseñó existente, por ejemplo, consumo o área. En tales casos, el diseño existente puede ser utilizado como modelo de referencia ya que este debe ser funcionalmente equivalente al DUV. Desde el Scoreboard, los datos considerados como correctos deben ser vistos a idéntico nivel de abstracción.

Modelos de referencia externos

En algunos casos, el DUV realiza cierto tipo de tarea que resultaría costosa computacionalmente la simulación de un modelo de referencia. En otros casos, ya existen herramientas comerciales que realizan esa tarea o bien, ya existe hardware que emula ese comportamiento. En este trabajo, se propone la comunicación desde el entorno simulado (en particular, desde el scoreborard) hacia el exterior utilizando puertos de comunicación. Un ejemplo típico es la verificación de diseños de filtros en imágenes. En ese caso, puede utilizarse la herramienta MatLab para realizar el filtrado en paralelo e independiente con la simulación del DUV. La conexión entre ambas herramientas puede realizarse utilizando sockets TCP o bien mediante librerías compartidas (dll).

5.4.3. Nivel Transacción

El desarrollo de componentes de verificación para este nivel implica un análisis de las interfaz del DUV. En este nivel se escribe de manera atómica una operación sobre la interfaz del DUV. Es por eso que resulta necesario conocer los protocolos de comunicación con los que funciona el diseño. Por ejemplo, una transacción puede ser un paquete de datos que se escribirán en formato serie asincrónico en el DUV.

5.4.4. Nivel Secuencia

El nivel de secuencia se describe como una sucesión coherente de transacciones que provocan un cambio de estado en el DUV. El contenido de las transacciones puede ser aleatoria o fija siempre que sean valores válidos. El ejemplo más claro es la inicialización de un DUV donde se deben escribir determinada serie de comandos en forma de paquetes para su configuración y puesta en funcionamiento.

5.4.5. Nivel Test

Un test define una serie de comandos o secuencias que conducen al DUV a un estado que interesa verificar. En este nivel deben establecerse que secuencias se utilizarán. Cuando las secuencias se generan a partir de valores aleatorios, los tests se conocen como Aleatorios Restringidos o CRT (del ingés Contrained Random Test). En caso de que las secuencias se generen utilizando valores específicos, se los clasifica como Tests Dirigidos (Directed Test). También puede suceder que sobre un atributo de la transacción se generen todos los casos posibles. Este test recibe el nombre de Test Exhaustivo (o Exhaustive Test). Este tipo de test es aplicable cuando el atributo de la transacción tiene una cantidad finita y manejable de combinaciones. Los tests de regresión (en inglés Regression Test) son útiles para la depuración del diseño luego que este ha sido modificado. Su objetivo es verificar que el diseño siga cumpliendo con las especificaciones luego de la reestructuración.

5.4.6. Nivel Escenario

El análisis del nivel de escenario implica conocer las características del DUV y el entorno con el que interactuará. En general, un escenario describe parámetros como:

- Configuración del entorno: Se describen retardos, cantidad de tiempo que se ejecutará la prueba, configuración de reportes, etc.

- Configuración del DUV: Se trata de la configuración de parámetros específicos del diseño, como por ejemplo, resolución de las señales o arquitectura a utilizar (*Parameters* en SystemVerilog y *Generic* en VHDL).

- Pruebas que se realizarán: De una biblioteca de pruebas de nivel Test, se eligen cuales se ejecutarán y si es necesario, se indica en que orden.

5.5. Caso de estudio: Verificación de modelos HIL para convertidores de potencia

Esta sección describe la verificación completa del modelo de un convertidor de potencia para PFC (Power Factor Correction) como ejemplo de aplicación. Esta sección presenta las ventajas de la verificación con HIL (Hardware In the Loop) y la necesidad de automatizar su proceso de verificación. Inicialmente se presentan algunos detalles del modelo que será verificado y posteriormente se presentan cada uno de los componentes de verificación.

5.5.1. Motivación

En el desarrollo de fuentes conmutadas, es indispensable la verificación del controlador de manera eficiente. Esta verificación apunta a evitar irregularidades en su funcionamiento antes de su fabricación. Con el objetivo de realizar una verificación completa, la simulación se realiza en conjunto con el convertidor, tal como funcionaría en su versión final. De esta forma, se pueden detectar los errores de implementación así como analizar no idealidades — como retardos en la conversión y segmentación. De esta manera, esta etapa de simulación puede probar al controlador en su versión final ya que este estará siendo usado con el modelo del convertidor de potencia.

Dado que el sistema final contiene partes analógicas y digitales, la simulación puede ser realizada utilizando alguna herramienta comercial de simulación mixta que simule ambas partes. Sin embargo, el proceso de simulación mixta puede ser lento, tornando esta etapa en extensa. Para acelerar el proceso de debbuging, en vez de simularlos, los modelos pueden ser emulados en una FPGA (Field Programmable Gate Array) si es que la especificación utilizada es sintetizable. Este enfoque es conocido como HIL (Hardware in the Loop), y puede ser usado en cualquier dispositivo Hardware incluyendo computadoras y FPGAs, para emular el convertidor de potencia. En [52], se muestra como la emulación de sistemas HIL pueden reducir

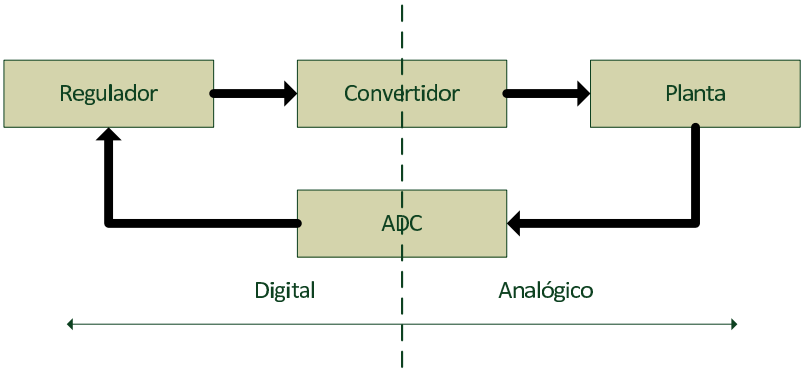


Figura 5.2: Diagrama conceptual de una fuente conmutada

de manera abrupta la duración de la verificación. En ese trabajo, las pruebas han logrado una aceleración de hasta 29,000 veces en comparación con herramientas de simulación mixta. Esto despeja cualquier duda de la utilidad del enfoque HIL. Esta estrategia motiva a contar con un modelo de convertidor previamente verificado para usarlo en HIL. La tarea de verificación de un sistema HIL debe contemplar diversos escenarios de tensión y corriente así como distintas configuraciones en el convertidor. Precisamente, el caso de estudio que se presenta en este Capítulo se orienta a automatizar tal proceso de manera de poder contar con herramientas que aceleren aun más la tarea global de verificación de la fuente.

La mayoría de los trabajos previos han usado computadoras para implementar HIL cuando el paso de integración se encuentra en el orden de decenas o centenas de microsegundos [37]. Esto puede parecer suficiente para bajas frecuencias de conmutación, en el orden de kilohertz o decenas de kilohertz. Si el dispositivo objetivo funciona a mayor frecuencia, el paso de integración debe ser más pequeño para obtener un modelo más preciso. Con el objetivo de lograr decenas o centenas de nanosegundos, se han introducido FPGAs en la emulación HIL [5, 15, 36, 40, 43, 46].

5.6. Breve reseña de Modelos HIL

En este Capítulo se utiliza como ejemplo de aplicación un convertidor de potencia para PFC, pero conclusiones similares pueden ser extraídas para otras topologías o aplicaciones.

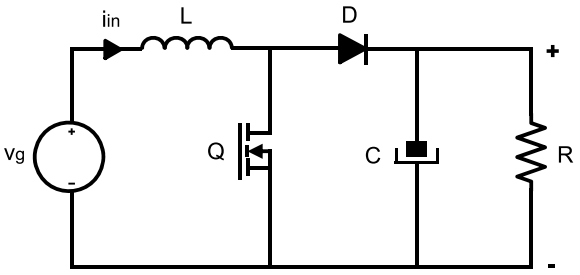


Figura 5.3: Diagrama esquemático de un Boost converter

La figura 5.2 presenta, de manera resumida, las partes que componen una fuente conmutada en configuración de lazo cerrado. El regulador, en este caso digital, realiza el cálculo de PWM (del ingés *Pulse width modulation*) adecuado para que un conmutador genere una inducción correcta en el convertidor. El convertidor producirá determinadas tensiones que se verán reflejada en la planta. Dado que la planta representa un fenómeno físico de naturaleza analógica, un ADC (del inglés *Analog to Digital Converter*) ecalúa la acción del convertidor. Esta información, es suministrada al regulador de manera digital para ajustar el ciclo PWM.

Sin embargo, en este trabajo en particular, se aborda la verificación de un sistema a lazo abierto, es decir, sin el regulador. La razón fundamental es que el funcionamiento del regulador introduce correcciones que podrían encubrir errores propios del diseño del convertidor.

Un modelo de convertidor necesita calcular sus variables de estado, por ejemplo: Corriente de entrada, I_{in} y tensión de salida (V_{out}), en cada paso de simulación. En esta experiencia, la frecuencia de conmutación ha sido fijada en 100 kHz, y cada período de conmutación ha sido dividido en 1000 intérvalos de tiempo. De esta manera, el intervalo de tiempo es $\Delta t = 10\ ns$. Además debe considerarse que 10 ns es también la resolución de pulso PWM (Pulse Width Modulation).

En este ejemplo de aplicación, cada componente del convertidor de potencia se modela con sus características ideales. Sin embargo, es posible agregarle parámetros característicos (por ejemplo, tiempos de conmutación) ligados a componentes comerciales.

El modelo debe considerar cual es el estado del conmutador Q de la Fig. 5.3 ya sea abierto o cerrado. Además, el estado del convertidor depende del valor de

la corriente de entrada dado que puede ser positiva entonces los diodos conducen (CCM o Continuous Current Mode), o pueden ser cero y los diodos no conducen (DCM o Discontinuous Current Mode)[23]. De esta forma, surgen tres posibilidades (conmutador cerrado, conmutador abierto en CCM y conmutador abierto en DCM). Estos están definidos respectivamente en las ecuaciones (5.1), (5.2) y (5.3):

$$\begin{aligned} i_{in}(k) &= i_{in}(k-1) + \frac{\Delta t}{L} \cdot v_g \\ v_{out}(k) &= v_{out}(k-1) - \frac{\Delta t}{C} \cdot i_R \end{aligned} \tag{5.1}$$

$$\begin{aligned} i_{in}(k) &= i_{in}(k-1) + \frac{\Delta t}{L} \cdot (v_g - v_{out}) \\ v_{out}(k) &= v_{out}(k-1) + \frac{\Delta t}{C} \cdot (i_{in} - i_R) \end{aligned} \tag{5.2}$$

$$\begin{aligned} i_{in}(k) &= 0 \\ v_{out}(k) &= v_{out}(k-1) - \frac{\Delta t}{C} \cdot i_R \end{aligned} \tag{5.3}$$

El modelo del convertidor de potencia debe implementar esas tres ecuaciones para poder simular su comportamiento. Pueden agregársele otros elementos como, corrientes parásitas, pérdidas y otras no linearidades que no han sido incluidas en este trabajo por cuestiones de simplicidad.

5.7. Verificación del BoostConverter QXY

En este Capítulo se ha tomado la implementación de un modelo de convertidor de potencia. En general, se trata de un modelo implementado con aritmética en punto fijo (QXY) mientras se toma como referencia un modelo de convertidor con representación en punto flotante previamente verificado. El objetivo de esta sección es verificar el funcionamiento del mismo así como evaluar el posible impacto que

produce el uso de aritmética en punto fijo. La evaluación se realiza considerando diferentes condiciones de carga a la salida del convertidor mientras que es estimulado con diferentes tensiones de entrada.

Para la verificación del diseño se ha propuesto la implementación de un entorno basado en el framework OVM. A continuación se describen los diferentes componentes del entorno de verificación y se analizan las opciones para su implementación.

5.7.1. Breve reseña del formato QXY

El formato numérico Q es una representación de punto fijo donde es posible definir de manera independiente los bits pertenecientes a la parte entera y a la parte fraccionaria. Un número en formato $Qm.n$ utiliza $n + m + 1$ bits, donde:

- Q indica que el número se encuentra en notación Q inicialmente utilizada por Texas Instruments para números en punto fijo con signo (El nombre Q es una reminiscencia del símbolo estándar para representar números racionales).
- m (opcional, inicialmente asumido como uno o cero) indica la cantidad de bits destinados al almacenamiento de la parte entera.
- n es el número de bits utilizado para almacenar la parte fraccionaria. El caso especial cuando $n = 0$ hace referencia a los números enteros.

Su rango es $[-(2^m), 2^m - 2^{-n}]$ mientras que su resolución es -2^{-n} Por ejemplo, un número en formato Q14.1:

- requiere $14 + 1 + 1 = 16bits$
- su rango es $[-2^{14}, 2^{14} - 2^{-1}] = [-16384{,}0, +16383{,}5]$
- su representación $[0x8000, 0x80010xFFFF, 0x0000, 0x0001 \cdots 0x7FFE, 0x7FFF]$

5.7.2. Interface

La interconexión del entorno OVM con el DUV se realiza mediante interfaces de SystemVerilog. Estas contienen la información de diferentes señales de distinto tipo

que son enviadas y recolectadas del convertidor. Sin embargo, una interfaz estándar no puede contener señales del tipo real ya que el estándard de SystemVerilog sólo admite señales digitales en sus interfaces [1] En el caso del modelo QXY la interfaz contiene las siguientes señales (en sintaxis SystemVerilog):

- **logic Clk**: Transporta la señal del clock común a todo el entorno de verificación.
- **logic Reset**: Señal para inicialización de todos los módulos HDL y sincronización con demás componentes de OVM.
- **logic [12:0] Ir** Corriente de carga. Es la señal en formato QXY que se obtiene del BoostConverter
- **logic [12:0] Vin**: Es el valor en formato QXY de la tensión de entrada al BoostConverter.
- **logic [9:0] DutyCycleIn**: Valor del ciclo útil para el generador PWM.
- **logic [11:0] Vout**: Es el valor en formato QXY de la tensión de salida del BoostConverter
- **logic [11:0] Iin**: Corriente de entrada en formato QXY al convertidor.

A continuación se detallan los componentes de OVM involucrados en la verificación del modelo QXY.

5.7.3. Sequencer

Este componente genera transacciones a partir de diferentes escenarios de interés para la verificación BoostConverterQXY. Este contiene:

- Escenarios donde varía la tensión de entrada (V_{in}) y otros donde se mantiene fija.

[1] En módulos donde es primordial el uso de valores reales, este tipo de señales se declararán dentro del módulo e invocados utilizando el nombre jerárquico de la señal (ejemplo $\$root.<nombreModulo>.<NombreSenalReal>$).

- Escenarios donde varía la corriente de carga (I_r) y otros donde se mantiene fija.
- Combinación de las anteriores.

La descripción de los escenarios se realiza utilizando sintaxis declarativa de SystemVerilog describiendo políticas de generación dentro del sequencer. El sequencer genera transacciones que simulan tensiones de corriente alterna (CA) o bien de corriente continua. En el caso de una tensión de entrada de corriente alterna y rectificada en media onda, se pre-calcula una tabla que corresponde a tensiones provenientes de la red eléctrica (240V-60Hz). Por otro lado, si el escenario involucra tensiones en corriente continua, se fija una tensión constante y a partir de allí se le aplican diferentes modificaciones como se describe más adelante.

5.7.4. Driver

El Driver toma de la transacción generada por el Sequencer los valores de entrada (Vin e Ir) en representación en punto flotante y los traduce a señales con formato QXY para inyectarlos en la interfaz. Además el Driver debe calcular el ancho de pulso correspondiente para la tensión de entrada provista en la transacción. El cálculo del PWM se realiza siguiendo la ecuación:

$$PWM_{Duty} = 100 - (V_{in}/V_{out} * 1000{,}0) \tag{5.4}$$

5.7.5. DUV

El DUV es simplemente un módulo SystemVerilog que encapsula a la unidad en testeo BoostConverterQXY.

5.7.6. Monitor

El monitor captura las señales de salida del DUV con representación QXY y las traduce a valores con representación en punto flotante. Con estos valores, se conforma una transacción de salida en cada paso de simulación. La transacción de salida es posteriormente enviada al Scoreboard para su análisis. Si bien el monitor decodifica

ciclo a ciclo las señales provenientes del DUV, por cuestiones de eficiencia y posterior manipulación de los resultados generados, es suficiente registrar transacciones con menos frecuencia. Es por esto que el monitor ha sido parametrizado con las siguientes configuraciones:

- Sample_freq: Establece una división de frecuencia sobre el reloj del sistema para indicar cuando debe registrarse la transacción de salida.
- Sample_avg: Indica si la muestra que se registrará proviene de los promedios de ciclos anteriores o bien de un valor puntual.

5.7.7. Scoreboard

El Scoreboard realiza varias tareas. Primero, recibe transacciones de salida desde el monitor y transacciones de entrada desde el sequencer con el objetivo de alimentar el modelo de referencia. Luego, es posible comparar ambos resultados, realizar cálculos estadísticos y confeccionar reportes. En este caso, la transacción de salida proveniente del monitor se compone por dos valores de punto flotante: V_{out} e I_{in}. Ambos valores pueden ser comparados con los del modelo de referencia y de esta manera calcular errores, desviaciones o bien generar reportes.

El modelo de referencia, puede formar parte del Scoreboard si se trata de un algoritmo o una simple función de transferencia. En este caso, el modelo de referencia es un módulo BoostConverter con representación en punto flotante. Dado que este módulo contiene código concurrente, no puede ser directamente instanciado dentro del Scoreboard, pero sí puede hacerse referencia a sus señales utilizando identificación jerárquica y es por eso que se ha incluido como parte del Scoreboard. En este caso, el modelo de referencia es instanciado en el TB de alto nivel y desde el Scoreboard se hace referencia jerárquica mediante $\$root.converter\_top.REF.<NombreSenal>$.

5.7.8. Transacciones

Una transacción para el BoostConverter es una instantánea de los valores de tensión y corriente que ingresan o egresan del convertidor. Se han dividido las transacciones de las pruebas en transacciones de entrada (o requerimientos) y transacciones de salida (o resultantes) y contienen los siguientes datos:

- Entrada

  V_{in}: (punto flotante) Valor que define la tensión de entrada para el instante actual.

  I_r: (punto flotante) Valor que define la corriente de salida del circuito.

- Salida

  V_{out}: (punto flotante) Tensión de salida del DUV

  I_{in}: (punto flotante) Corriente en el lazo de entrada del DUV

Según el tipo de test que se realice, puede ser necesario aleatorizar algunos componentes de la trasacción. Con este objetivo, la transacción posee soporte para aleatorizar los siguientes valores:

- R_Load: (punto flotante) Indica la resistencia de carga conectada a la salida del circuito.
- P_Load: (punto flotante) Indica la potencia que debe ser entregada por el circuito
- desvioVinMax: (entero) Indica el máximo valor que puede producirse en Vin.
- desvioVinMin: (entero) Indica el mínimo valor que puede producirse en Vin.
- desvioVin: (punto flotante) Es un valor entre [desvioVinMin..desvioVinMax], dependiente del tiempo, que se produce en torno a una tensión fija causando oscilaciones a los valores de Vin. Las oscilaciones responden a una función sinusoidal.
- desvioR_LoadMax: (entero) Indica un valor mínimo en la resistencia de carga.
- desvioR_LoadMin: (entero) Indica un valor máximo en la resistencia de carga.
- desvioR_Load: (punto flotante) Es la variación entre [desvioR_LoadMin..desvioR_LoadMax]de la resistencia de carga en función del tiempo. Los valores de este parámetro tienen un comportamiento ascendente (hasta R_LoadMax) y luego descendente (hasta R_LoadMin).

- desvioP_LoadMax: (entero) Indica un valor mínimo en la potencia a la salida.
- desvioP_LoadMin: (entero) Indica un valor máximo en la potencia de salida.
- desvioP_Load: (punto flotante) Es la variación entre [desvioP_LoadMin..desvioP_LoadMax] de la resistencia de carga en función del tiempo. Los valores de este parámetro tienen un comportamiento ascendente (hasta P_LoadMax) y luego descendente (hasta P_LoadMin).

Los valores aleatorizados son utilizados en diferentes restricciones las cuales no necesariamente están activas todas a la vez. El Sequencer es el encargado de activar y desactivar dichas restricciones según el test que se desee ejecutar.

5.8. Verificación con OVM del convertidor a lazo abierto

Se ha diseñado un entorno basado en OVM de manera tal que pueda ser reutilizado en las diferentes pruebas. Para cada una de estas se describe que configuración o modificación es necesaria para su ejecución. Los módulos Sequencer, Driver, Monitor y Scoreboard son comunes a todas las pruebas y configurados según la prueba lo requiera.

5.8.1. Scoreboard

Con el objetivo de proveer un profundo análisis del comportamiento del Boost-Converter, el Scoreboard se codifica de manera tal de poder obtener y almacenar los valores de:

- $I_{in}\_Referencia$
- $V_{out}\_Referencia$
- V_{out}_DUV
- I_{in}_DUV

- $AbsVout = V_{out}\_Referencia - V_{out}\_DUV$
- $AbsI_{in} = I_{in}\_Referencia - I_{in}\_DUV$

Cada uno de estos ítems se almacena en archivos de texto por separado para su posterior tratamiento con alguna herramienta de análisis (Por ejemplo, Matlab). La razón de elegir este formato de archivo es que, a pesar de ocupar mayor espacio, provee flexibildad para ser utilizado en diferentes herramientas de análisis y posibilita la rápida interpretación humana.

5.8.2. Driver

Posee un bucle infinito que, ante un evento (configurable), traduce los valores de la transacción a señales de la interfaz. En este caso, el evento se establece con el cambio en la señal de la tensión de entrada V_{in}. La interpretación de las señales se realiza según la siguiente fórmula:

$$V_{in} = \left\lfloor I_r/1024/10^{-9} * 5^{-3} \right\rfloor \tag{5.5}$$

o directamente en sintaxis SystemVerilog:

c_if.Ir = $floor(req.Ir/1024.0/$pow(10.0,-9) *$pow(5.0,-3))

En los casos donde se utilicen las tensiones y valores de ciclos de trabajo precalculados se utilizará además el módulo InputGenerator de tal manera que:

- c_if.DutyCycleIn = $root.converter_top.gen.PwmDutyCycle
- c_if.Vin = $floor($root.converter_top.gen.VgGen*8.0)

Mientras que en los demás casos donde las transacciones provengan del Sequencer de OVM se utiliza:

- c_if.Vin = $floor(req.Vin*8.0)
- c_if.DutyCycleIn=1000-((req.VinDuv/400)*1000.0);

5.8.3. Monitor

Colecta datos en cada conmutación del PWM. El monitor es configurable mediante los parámetros Sample_freq y Sample_avg. El primer parámetro define con que frecuencia se realiza el muestreo de los datos. Si las muestras se toman en cada uno de los ciclos, la simulación podría resultar más lenta. El segundo parámetro indica si deben considerarse valores puntuales o bien valores promediados en los últimos Sample_freq ciclos.

5.8.4. Definición de escenarios

Carga tipo corriente (IR)

Las pruebas deben configurarse con los siguientes parámetros:

- Vin: Dado por tabla (o memoria)
- Ciclo de Trabajo: Dado por tabla (o memoria)
- IR: Fija

Los valores de cada transacción son provistos por la memoria de ciclos de trabajo y tensiones generadas por el Driver. La corriente se ha fijado en 0.75(A). El monitor colectará datos en cada conmutación del PWM:

- Sample_freq = 1000
- Sample_avg = TRUE

Carga tipo resistiva (R)

El escenario de esta prueba está descrito por el BoostConverter con una resistencia fija como carga a la salida del diseño de referencia y el DUV. El valor de esa resistencia forma parte de la transacción.

- Vin: Dado por tabla (o memoria)
- Ciclo de Trabajo: Dado por tabla (o memoria)

- R: Fija, donde $I_r = V_{out}/R$

A diferencia de la carga tipo corriente, la transacción debe configurarse de manera tal que la corriente I_r tenga la forma:

- $I_r(t) = V_{out}(t-1)/R_{load}$
- $R_{load} = 1000\Omega$

Además, por cuestiones de flexibilidad, se introduce la propiedad R_{Load} a la transacción de manera tal que permita definir la resistencia de carga conectada a la salida. Para las pruebas se ha utilizado una resistencia de = 1000Ω Por otra parte, el monitor colectará datos en cada conmutación del PWM, por lo tanto:

- Sample_freq = 1000
- Sample_avg = TRUE

Carga tipo potencia (P)

El escenario de esta prueba describe una situación donde la carga conectada al BoostConverter es una potencia fija. Esta carga está determinada por una constante del entorno.

- V_{in}: Dado por tabla (o memoria)
- Ciclo de Trabajo: Dado por tabla (o memoria)
- P: Fija, donde $I_r = P/V_{out}$

En las Transacciones se ha tomado un valor fijo de 300 watts. Al igual que en el caso de Carga tipo Resistiva, debe considerarse que la corriente varíe, por lo cual una nueva transacción es creada:

- $P_{Load} = 300(watt)$
- $I_r(t) = P/V_{out}(t-1)$

El monitor colecta datos en cada conmutación del PWM, por lo tanto:

- Sample_freq = 1000
- Sample_avg = TRUE

Carga tipo corriente y tensión de entrada fija

La prueba se realiza en un escenario similar al de tipo corriente mencionado pero en con una tensión de entrada fija. Para las transacciones se define la tensión de entrada fija y continua con valores correspondientes a la red eléctrica doméstica.

- Vin = 230 (volts)
- Ir = 0.75 (ampere)

Dado que se trata de una prueba de una duración considerable, el monitor se configura:

- Sample_freq = 5000
- Sample_avg = TRUE

Carga tipo corriente y tensión de entrada con oscilaciones

El escenario de esta prueba describe una situación donde la carga es del tipo corriente y la tensión de entrada se mantiene en torno a una tensión fija, pero con un ruido periódico de comportamiento conocido. El ruido responde a una onda sinusoidal y podría emular la influencia algún artefacto lindero. Para la generación de transacciones, la tensión de entrada responde a la función:

- VinDuv=210 + 30* sine(t);
- VinRef=210 + 30* sine(t);

Produciendo de esta manera un ruido periódico sobre la tensión de entada. Dicho ruido posee una frecuencia de 1600 hz.

Para la construcción del Monitor debe considerarse que las fluctuaciones de V_{in} a la entrada del Converter pueden suceder ciclo tras ciclo y es necesario un análisis minucioso de la salida. Es por eso que en este caso se configura el Monitor para enviar muestras al Scoreboard en cada ciclo de simulación. Por lo tanto se configura el monitor:

- Sample_freq = 1
- Sample_avg = FALSE

Carga tipo resistencia variable y tensión de entrada fija

En esta prueba, la resistencia de carga inicial comienza a incrementar su valor hasta desvioR_LoadMax y luego lo decrementa hasta desvioR_LoadMin.

Para las transacciones de esta prueba se definen:

- R_Load (carga inicial): 1000 (ohms)
- desvioR_LoadMax: 1500 (ohms)
- desvioR_LoadMin: 1000 (ohms)

De esta manera la carga asciende hasta la mitad de la prueba y luego desciende hasta el valor inicial. Se utilizan incrementos de manera tal que el máximo se produzca a la mitad de la prueba y el mínimo al final.

Dado que se trata de una prueba con variaciones en la carga, el monitor debe capturar las salidas ciclo a ciclo:

- Sample_freq = 1
- Sample_avg = FALSE

Carga tipo potencia variable y tensión de entrada fija

En esta prueba, la potencia conectada a la salida se incrementa hasta el valor devioP_LoadMax y luego desciende hasta desvioP_LoadMin. Para una transacción en esta prueba se definen:

- P_Load (carga inicial): 300 (watt)
- desvioP_LoadMax: 400 (watt)
- desvioP_LoadMin: 100 (watt)

Dado que el valor con el que finaliza la prueba (desvioP_LoadMin) es distinto al valor inicial, se debe tener en cuenta que el valor máximo de potencia no se encontrará en la mitad de la prueba. Para los parámetros dados, esto sucede transcurrido un cuarto de la prueba.

Al igual que en otras pruebas donde hay variaciones en la carga, el monitor necesita recolectar las salidas ciclo a ciclo:

- Sample_freq = 1
- Sample_avg = FALSE

5.8.5. Scripts

El Script para la instanciación de la prueba se resume en tres grandes etapas. La primera realiza la definición por extensión de todos los tests que serán evaluados así como las configuraciones del diseño. Luego, dentro de un ciclo iterativo se encuentran las etapas restantes. La segunda realiza la preparación de carpetas contenedoras con nombres significativos e interpretables tanto para el acceso por parte del verificador (inspección visual de los resultados) como por parte de algún script para su posterior análisis (escaneo en profundidad del árbol de carpetas y confección de resultados finales). Por último se inicia la simulación.

Script 5.1: Script para verificación del convertidor

```
1  set tests_list {sin_noisy_test ac_current_test
       ac_resistance_test ... }
2  set bitsIin {16..47}
3  set bitsVout{16..47}
4  foreach testname $tests_list {
5          foreach valorIin $bitsIin {
6                  foreach valorVout $bitsVout {
7                          set nArchivo ""
8                          append nArchivo $testname/
                              bitsIin$valorIin/bitsVout$valorVout
9                          mkdir $nArchivo
10                         eval vsim -c -novopt  converter_top  +
                              OVM_TESTNAME=$testname -L ../
                              ModeloQXYLazoAbierto/work -sv_lib
                              lib/mathLib +CFG=monitor.txt
                              -GNv=$valorVout -GNi=$valorIin +
                              OUT_DIR=$nArchivo
11                         run 20ms
12         }
```

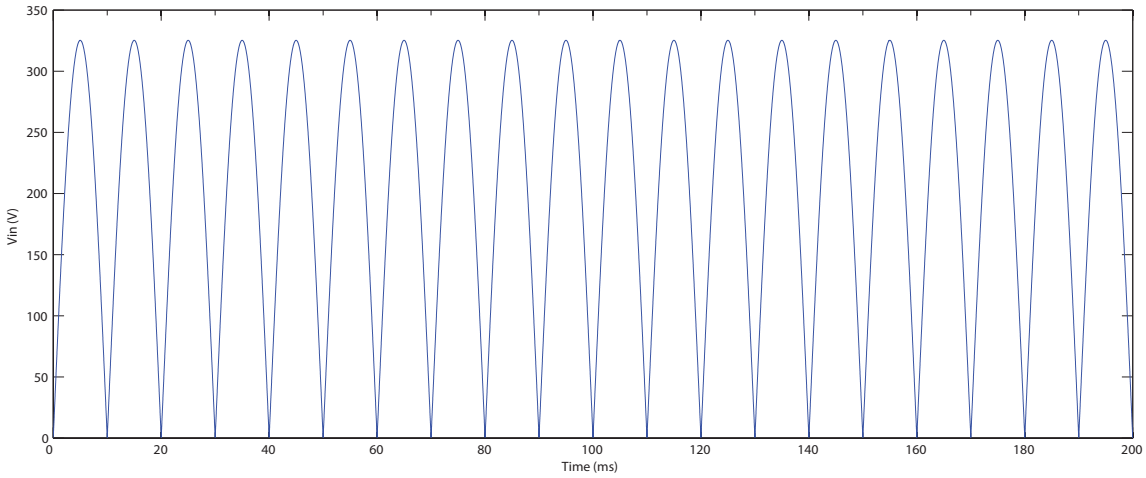


Figura 5.4: Tensión de entrada alternada

```
13          }
14      }
```

5.9. Resultados

Las pruebas definidas anteriormente se implementan e instancian dentro de un entorno de verificación basado en OVM implementando cada una de las transacciones y ejecutándolas de una a la vez.

En esta sección se presentan los resultados más relevantes de las pruebas haciendo hincapié en aquellos relacionados con la verificación funcional. Los resultados completos de las pruebas pueden encontrarse en el Apendice A. Todos los casos son representados en gráficas utilizando Scripts para análisis escritos para la herramienta MatLab.

Las pruebas que involucran tensiones de entradas correspondientes a corriente alterna, son extraídas de tabla y pertenecen a varios semiciclos de tensión alterna procedentes, en este caso, de la red eléctrica (A.1).

En casos donde la tensión de entrada contiene fluctuaciones, se muestra una gráfica para cada caso en particular. Por último, los casos que poseen tensión fija a la entrada no se representan.

El tiempo de ejecución de cada prueba así como la frecuencia de muestreo de las salidas dependen del comportamiento de las entradas. Si estas tienen un comportamiento periódico o monótono, se realizan pruebas largas y se promedia en gran cantidad de ciclos. En cambio, si los valores son dinámicos, ya sean aleatorios o con algún comportamiento conocido, las pruebas son cortas y con muestreos de alta resolución.

Carga tipo corriente y tensión de entrada alternada

El convertidor se comportó de manera muy similar al modelo de referencia. Se ha detectado que el error en ambos modelos varía siguiendo el patrón de los ciclos de tensión alterna siendo el error negativo cuando el semiciclo parte desde o vuelve a cero. Se detectó que cuando la tensión de entrada es cercana a cero, el modelo QXY provee menos tensión que el modelo de referencia. La situación opuesta ocurre cuando la tensión de entrada llega a su valor máximo. Esto se le atribuye a un error en la resolución. El análisis de resolución es presentado en profundidad en el Capitulo 5.

El comportamiento de la corriente I_{in} descrito es similar al de la tensión V_{out}.

Carga tipo resistencia y tensión de entrada alternada

Para esta prueba, el Convertidor no presenta diferencias significativas con el modelo de referencia. Sin embargo, es posible ver que la desviación máxima por encima del valor ideal en ambos modelos no se produce ni en el máximo del semiciclo ni en cero.

De las diferencias entre ambos modelos se puede decir que también se sigue un patrón solidario a los ciclos de la tensión alterna de entrada. Sin embargo, no se encuentra una relación directa del error con el valor del semiciclo.

Carga tipo potencia y tensión de entrada alternada

No se han detectado errores funcionales en esta prueba. Las diferencias entre ambos modelos es menor que para otras pruebas. Se detectó que ambos modelos se aproximan al valor ideal (400 volt) cuando la tensión de entrada es cercana a cero y cuando alcanza su valor máximo.

Carga tipo corriente y tensión de entrada fluctuante

En esta prueba, no se han detectado errores funcionales. De la comparación con el modelo de referencia puede encontrarse un defasaje en los valores de V_{out}. Este defasaje presenta un retardo de 2ms.

5.9.1. Cobertura de código

El módulo BoostConverter con representación QXY posee una cobertura de código del 100 %. Esto implica que cada condición, cada señal y cada módulo ha estado activo al menos una vez. Además, esto indica que no existe dentro del diseño alguna especificación que por la cual se infiera hardware sin funcionalidad.

5.9.2. Profiling

Se realizó un Profiling de performance que indica tiempo de simulación consumido por cada uno de los módulos involucrados en el entorno. Los reportes arrojados por la herramienta Questasim, indican que los módulos que mayor tiempo insumen son:

- Modelo de referencia: 25 %
- Generador V_{in}: 8 %
- Scoreboard: 3 %
- Driver: 2.8 %

El reporte de profiling indica que la simulación del modelo de referencia y el generador de tensiones son los módulos que consumen mayor tiempo de simulación. Por otro lado, la simulación del DUV utiliza el 0.1 % del tiempo de simulación. Estos resultados permitirían al equipo de verificación centrar esfuerzos en la optimización del modelo de referencia. Cabe señalar que una vez verificado el ConverterQXY, no será necesaria la utilización del modelo de punto flotante, acelerando de esta forma la verificación.

5.10. Conclusiones del capítulo

En este Capítulo se presenta la automatización de pruebas para un entorno de verificación. A lo largo del Capítulo se muestran las diversas alternativas para la automatización según el nivel de abstracción. Si bien se plantea un caso de estudio industrial real, el seguimiento de las pautas descriptas para afrontar una verificación automatizada son aplicables a cualquier proyecto de hardware digital que utilicen un modelo físico.

En el caso de estudio analizado, no se encontraron errores funcionales para las pruebas definidas. Sin embargo, la verificación funcional muestra que es posible aprovechar los resultados de las pruebas para aclarar diferentes interpretaciones y conceptos que pueden diferir entre el equipo de diseño y el de verificación. La precisa presentación de resultados permiten al diseñador extraer diversas conclusiones de interés. Por ejemplo, es posible detectar algunas relaciones entre las desviaciones y los semiciclos de entradas. De esta forma, pueden inferirse reparaciones o modificaciones necesarias para la optimización del diseño. Partiendo del análisis de profiling puede deducirse que la mayor cantidad del tiempo de verificación es insumido por la ejecución del modelo de referencia, mientra que el tiempo de la simulación del DUV es insignificante. Esto indica que cuando se utilice el modelo final del convertidor, por ejemplo, emulado en una FPGA, la velocidad será mucho mayor.

Por último, las diferencias numéricas en los valores entre el modelo de referencia y el DUV encontradas en algunos casos se atribuyen a la falta de resolución de los registros que almacenan los estados internos. En el próximo Capítulo se utiliza la verificación funcional como herramienta para corroborar y corregir este tipo de situaciones.

Capítulo 6

Análisis de resolución asistido por la Verificación funcional

En el Capítulo anterior, se propone a la verificación funcional básicamente como una herramienta diseño. En ese Capítulo se propone que los resultados de la verificación sirvan para evaluar el desempeño del DUV (Design Under Verification) en la mayor cantidad de escenarios posibles y de esa manera poder tomar algunas decisiones en él. En este Capítulo se propone el uso de la verificación funcional para un problema concreto que es el análisis de resolución. En particular, y continuando con el caso de estudio ya presentado, se propone el análisis de resolución para convertidores de potencia orientados a HIL que utilizan aritmética de punto fijo. El presente Capítulo se organiza de la siguiente manera: En la sección 6.1 se presentan los sistemas HIL así como trabajos relacionados. En la sección 6.1.1 se presenta la incidencia de la resulución en HIL y se exponen los detalles a considerar en un caso de aplicación concreto. En las secciones 6.2 y 6.3 se presentan dos métodos para el análisis de resolución, uno analítico y otro basado en simulación. En la sección 6.4 se presentan los resultados de los experimentos y se muestra el análisis de área y frecuencia y los resultados del análisis de resolución utilizando ambos métodos. Finalmente, en la Sección 6.5, se presentan las conclusiones del Capítulo.

6.1. Introducción

La aritmética de punto flotante es probablemente la representación más fácil y natural para diseñar un modelo de convertidor de potencia. En [38], se muestra un sistema HIL que utiliza el paquete *float_pkg* de VHDL2008 mientras que en [44] se hace lo propio utilizando una HLST (High Level Synthesis tool) donde se evalúa la precisión de las operaciones de punto flotante. Sin embargo, el problema de esta aritmética, es que no fue soportada de manera nativa por las herramientas de sítensis hasta tiempos recientes. Los sistemas HIL, al igual que otros diseños que utilizan aritmética de punto flotante, se caracterizan por una alta ocupación de área y una baja frecuencia de reloj en el dispositivo objetivo.

En el contexto de HIL, [52] muestra una comparación de varias técnicas de simulación tales como simuladores de señales mixtas, VHDL sintetizable y modelos no sintetizables, modelos de punto fijo y punto flotante y modelos traducidos automáticamente a VHDL. Esta comparación prueba que la artitmética de punto fijo obtiene los mejores resultados de síntesis (área y velocidad) pero incrementa el tiempo utilizado en la etapa de diseño. No se conocen trabajos previos que hayan estudiado la resolución de variables de estado de un modelo de convertidor. Cuando un convertidor de potencia es diseñado, sus variables de estado deben ser calculadas y almacenadas (en registros en el caso de una implementación HDL) con la correcta resolución.

En cada paso del tiempo, estas variables deben ser actualizadas agregando pequeños valores. El ancho de los registros definen la resolución de una variable. El Vocabulario Internacional de Metrología [33] define la resolución como “el cambio más pequeño en una cantidad medible que causa un cambio perceptible en la indicación correspondiente.” Además define precisión como “El grado de concordancia entre el valor de la magnitud medida y un valor de la cantidad real de la misma”.

En [20], los autores explican que la resolución de las variables internas está directamente relacionada con la precisión de su cómputo. Más aún, si estas variables se modifican dentro de un algoritmo iterativo, esta relación suele ser mucho más compleja.

Dado un ancho de señal constante, cuanto más pequeño es el intervalo de tiempo (el que requiere mayor precisión), más pequeña es la resolución de los cálculos. De esta manera, cuando el sistema utiliza altas frecuencias de conmutación e integración, el

ancho de la señal debe ser incrementado. Es aquí donde aparece un compromiso entre resolución de los cálculos y velocidad de simulación. Además, en sistemas emulados, deben analizarse los recursos de un sistema HIL (por ejemplo: área ocupada en la FPGA). Cuando se utiliza punto fijo, el ancho de las variables es crítico pero los detalles de resolución también deben ser considerados cuando se trabaja con punto flotante. En el caso de un HIL para un convertidor de potencia, la resolución necesaria para las señales aritméticas depende de los parámetros de diseño del modelo pero también las tensiones y corrientes de entradas y salidas que cambian a través del tiempo. De esta manera, el cálulo de resolución no es simple de calcular a priori.

Un error casi imperceptible producido y acumulado en cada ciclo causado por resolución insuficiente puede convertirse en inaceptable a lo largo del funcionamiento. La solución inmediata a este problema involucraría almacenar a las variables en registros de mayor ancho. Sin embargo, esta solución empírica aumenta la ocupación de área y disminuye la frecuencia de cálculo en el dispositivo programable. Independientemente de la tecnología utilizada, la optimización de la frecuencia de reloj y de la utilización de recursos es una actividad importante en el diseño de un sistema hardware. Por un lado, la optimización de recursos permite incluir mayor cantidad de módulos sintetizados en una sóla pastilla, o bien reducir el consumo de potencia. Por otro lado, y dentro del contexto de sistemas HIL, una mayor frecuencia de reloj provee emulaciones más rápidas. Estos hechos, motivan la necesidad de realizar un análisis de resolución para utilizar el tamaño correcto de registros para cada aplicación.

Algunos trabajos, como [46], utilizan punto fijo para la representación del modelo de convertidores de potencia conmutados. Sin embargo, sus autores han utilizado registros de tamaño fijo y específico para almacenar los resultados intermedios. Esos anchos han sido fijados por aspectos de contrucción que van más allá del modelo matemático del convertidor tales como el ancho de los multiplicadores de la FPGA (por ejemplo, 18 bits en Xilinx Virtex 5) o la resolución del ADC (Analog to Digital Converter) utilizado. En [5] se presenta un método robusto para minimizar el error en lineas de corriente en motores propulsión reconstruidos desde una simple medición del link-dc. En [39] se presenta un núcleo de procesador digital de ultra baja latencia para sistemas HIL. Este núcleo tiene como objetivo ser utilizado en la validación

de varios diseños industriales con un amplio rango de tensiones y corrientes. Sin embargo, la resolución de los registros utilizada para almacenar las variables es fijo y elegido para soportar el peor escenario. Si se utilizara la resolución correcta para cada escenario se optimizaría la latencia, la frecuencia y el área ocupada en el dispositivo.

En este trabajo, se proponen dos métodos para encontrar el ancho óptimo para las variables de estado en sistemas HIL. El primer método está basado en simulación y provee la resolución óptima basándose en la performance de cada modelo real. El segundo método es analítico y provee como resultado un valor conservador de resolución para tensión y corriente. Aunque el primer método requiere mayor cantidad de recursos de cálculo, provee los resultados más precisos de mínima resolución que permite una determinada exactitud. Más aún, este método garantiza que le convertidor se comportará tal como fue especificado y que no se provocaran valores inesperados de tensión y corriente. El segundo método puede ser fácilmente aplicable por el diseñador cuando no existen fuertes restricciones de área y frecuencia. La validez de los resultados de este último queda demostrada con los resultados del primero. Finalmente, se realiza un análisis de área y frecuencia para mostrar la importancia de seleccionar una correcta resolución.

6.1.1. El problema de la resolución

Un trabajo de referencia sobre análisis de resolución es [11]. En ese trabajo se abordan diferentes técnicas para obtener el valor óptimo de resolución considerando el compromiso entre performance y presición. En su desarrollo, se presentan métodos basados en aritmética afín, aproximaciones utilizando LTI (Line time-invariant) y aritmética no lineal. Sin embargo, cada uno de los métodos propuestos resultan en la sobreestimación de la resolución. Los autores concluyen que, conceptualmente, la forma más simple de comprobar que, una representación de presición finita cumple con las especificaciones, es probándola. Típicamente, las simulaciones se realizan dos veces: Una con la máxima resolución posible y otra con la resolución propuesta. Este enfoque es el utilizado con más frecuencia en la industria actual.

La resolución de sistemas HIL no ha sido abordado en la literatura. Los primeros trabajos sobre sistemas HIL utilizaban frecuencias de conmutación muy bajas, entonces los problemas de resolución no aparecían debido a que existe una relación entre

resolución y frecuencia de conmutación como se mostrará más tarde. Sin embargo, las mejoras en la electrónica digital han hecho posible incrementar la frecuencia de conmutación y reducir el paso de integración de las variables de estado.

El ancho de una señal debe ser elegido de manera tal de cumplir con el compromiso entre presición y velocidad de la simulación. El ancho necesario queda determinado tanto por la máxima magnitud que puede almacenarse y su resolución. En el caso de una variable de estado, la resolución debe ser considerada teniendo en cuenta los valores incrementales que esta puede tomar en cada intervalo de tiempo.

El ancho se incrementa considerando la diferencia entre órdenes de magnitud entre el valor de la señal y sus respectivos incrementos. De esta manera, la señal debe contar con el ancho correcto para poder almacenar ambos valores con suficiente resolución.

El ancho de la señal x puede ser determinada por la Eq. (6.1):

$$width_x = \lceil \log_2 \frac{x}{\Delta x} \rceil + n \tag{6.1}$$

donde x es el valor de la variable, Δx es el valor incremental y n es el número de bits necesarios para almacenar el valor incremental. Cundo $n = 1$, esta ecuación otorga el número mínimo de bits necesarios para almacenar x y Δx simultaneamente. De esta forma, Δx se representa con '0' o '1', por lo tanto sólo múltiplos de Δx pueden ser agregados, lo cual resulta en valores de baja resolución. Entonces, n debe ser mayor que 1, y de esa manera, pueden almacenarse pequeños valores incrementales cercanos a Δx. A continuación se presentan dos métodos para obtener resoluciones válidas: El primero es basado en simulación y con una estrecha relación con la verificación funcional. El segundo es analítico y provee resultados de una manera simple, aunque sobre dimensionados.

6.2. Método 1 - Enfoque basado en simulación

Este método está basado en la simulación y partiendo del modelo de convertidor real. De hecho, un conjunto de modelos de convertidores con diferentes configuraciones son simulados para encontrar el escenario que describe el peor caso. Dado que los modelos de convertidores están orientados a hardware sintetizable, puede acele-

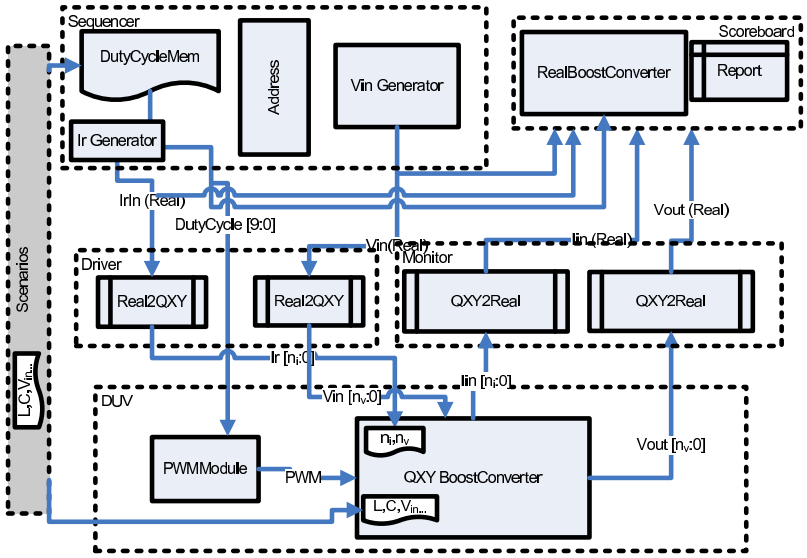


Figura 6.1: Entorno de simulación basado en OVM

rarse el proceso de simulación mediante la emulación del mismo utilizando hardware reconfigurable (por ejemplo, FPGA). Con el objetivo de lograr flexibilidad en la prueba de cada convertidor, se ha diseñado un entorno de verificación. Este entorno es útil, no solo para obtener la optima resolución con mínima ocupación de recursos y máxima frecuencia de funcionamiento, sino también para garantizar el correcto comportamiento del convertidor final. Los convertidores se evalúan instanciándolos con diferentes configuraciones y comparando sus variables y salidas con un modelo de referencia. La prueba consiste en generar distintos anchos para los registros de tensión (n_v) y corriente (n_i). Dentro del contexto de verificación funcional, el convertidor toma el rol de DUV. El DUV utilizado en este trabajo acepta valores de parámetros (n_i, n_v) de 16 a 47 bits. En ensayos preliminares se ha encontrado que utilizar resoluciones menores a 16 bits provoca que el diseño produzca errores inaceptables en I_{in} y V_{out}. Un error es considerado inaceptable cuando la diferencia entre las salidas del modelo de convertidor y las del modelo de referencia tienen un error mayor al 20 %. El modelo utilizado como referencia en este trabajo es esencialmente la implementación del convertidor utilizando variables de punto flotante de 64 bits.

El entorno de verificación está basado en el framework OVM (Open Verification Methodology) [25] dado que provee una estructura flexible y reutilizable necesaria

para verificar y analizar a los modelos de convertidores en los escenarios y pruebas propuestas. El entorno sigue un enfoque por capas. Las capa inferior (DUV) interactúa con el convertidor a nivel de señales digitales. En la capa media, el *Driver* convierte valores de punto flotante de tensión y corriente a registros de punto fijo. Además, al mismo nivel de abstracción, el *Monitor* decodifica las salidas del convertidor en punto fijo a valores de punto flotante. En el nivel superior, el *Sequencer* provee mecanismos para crear secuencias de valores de tensión provenientes de la red eléctrica. Mediante el *Scoreboard* realiza el cálculo de error de I_{in} y V_{out} y almacena dichos errores para su posterior análisis. Además, se incluyen métodos basados en aserciones que detienen la simulación cuando el error de I_{in} o V_{out} excede cierto valor umbral de tolerancia. Este mecanismo desecha las combinaciones inválidas de (n_i, n_v) acelerando el proceso de simulación.

6.3. Metodo 2 - Enfoque analítico

El problema principal del Método 1 es que son necesarias muchas simulaciones para obtener la mínima resolución que se requiere para una precisión fijada. Dado que este proceso puede ser prolongado, es deseable un método más rápido. En esta sección se propone un método analítico que utiliza una fórmula simple, similar a la Eq. (6.1). Sin embargo, este método concluye en una cantidad de bits superior a la propuesta por el Método 1, por lo cual no garantiza una resolución mínima para cierta precisión. Con el objetivo de proveer valores de resolución conservador, la Eq. (6.1) no considera valores típicos o medios sino que considera el peor caso, por lo tanto, x sería el valor máximo de la variable de estado que será permitido en la simulación, mientras que Δx debería ser el mínimo valor incremental razonable de la variable de estado. El valor máximo, o al menos su valor aproximado es fácil de calcular ya que solo depende de los transitorios que serán calculados. Sin embargo, hallar el mínimo incremento no es una tarea trivial. Δx puede ser 0, entonces no se necesitarían bits extras para almacenar ese incremento. Por lo tanto, debe encontrarse un valor para Δx excluyendo el 0. El problema aparece cuando Δx es cercano a 0, ya que puede ser un valor extremadamente pequeño, necesitándose muchos bits para almacenar esa variable.

Sin embargo, los valores de incrementos muy pequeños tienen un bajo impacto sobre la simulación, especialmente si el error ocurre durante periodos muy cortos.

Para encontrar un grado de compromiso entre precisión y tiempo de simulación, o recursos hardware si se trata de emulación, el segundo método determina resoluciones de variables de estado lo suficientemente grandes, la mayoría del tiempo, pero no todo el tiempo.

En este caso en particular, se propone escoger un tamaño de registro muy preciso el 95 % del tiempo. La precisión es menor durante ese 5 % ya que, durante ese tiempo, los valores incrementales son extremadamente cercanos a 0.

El convertidor de potencia presentado en la sección 5.6 se utiliza como ejemplo de aplicación. Pueden realizarse cálculos similares para otras aplicaciones o topologías tomando en cuenta las ecuaciones de sus variables de estado así como el rango de sus posibles valores. Los valores de estado de un convertidor con corrección de potencia se describe en (5.1), (5.2), y (5.3).

En (5.1), el valor incremental de la tensión de salida no es normalmente cercana a $0V$, dado que la carga normalmente no es cercana a $0A$ en esta aplicación. Sin embargo, el valor incremental de la corriente de entrada puede ser cercano a $0A$ ya que depende de la tensión de entrada *ac*. Fig. 6.2 muestra la tensión de entrada rectificada a través de un semi ciclo. Se descarta el 5 % del tiempo donde aparecen los valores más pequeños, entonces el valor mínimo de tensión que será tomado en la ecuación Eq. (5.1) es 25,01 V. De esta manera, el valor incremental de la corriente de entrada de la ecuación (5.1) es $\frac{\Delta t}{L} \cdot v_g = 5{,}002 \cdot 10^{-5}\ A$, considerando $L = 5\ mH$. Tomando este valor de incremento como el peor caso, el ancho de la variable de corriente debería ser:

$$\begin{aligned} width_{i_{in}} &= \lceil \log_2 \frac{i_{in}}{\Delta i_{in}} \rceil + n \\ width_{i_{in}} &= \lceil \log_2 \frac{8}{5{,}002 \cdot 10^{-5}} \rceil + n \\ width_{i_{in}} &= 18 + n \end{aligned} \tag{6.2}$$

El ancho para el registro de la variable de corriente de entrada debería ser $18+n$, donde n es el número de bits utilizados para para almacenar el valor incremental, tal como se explicó anteriomente. En la ecuación previa, el valor máximo de corriente

de entrada se ha fijado en $8A$, pero este parámetro, ajustado por el diseñador, es una limitación para los transitorios que pueden ser simulados por el modelo.

El mismo análisis debe ser realizado con las demás ecuaciones del modelo del convertidor. La corriente de entrada en la Ec. (5.2) no presenta problemas de resolución, dado que la diferencia entre las tensiones de entrada y salida no es cercana a $0V$. Sin embargo, la diferencia entre corrientes, necesarias para calcular la tensión de salida, periódicamente es cercana a $0V$. La Fig. 6.3 muestra ambas corrientes. La misma metodología se aplica descartando el 5 % del tiempo cuando la diferencia es más cercana a $0V$. Como puede observarse, la diferencia mínima, una vez que el porcentaje se descarta, es de 0,0656 A. Con esta diferencia, el valor incremental de la tensión de salida en la Ec. (5.2) es $\frac{\Delta t}{C} \cdot (i_{in} - i_R) = 6{,}56 \cdot 10^{-6}\ V$, si $C = 100\ \mu F$. Considerando este valor de incremento mínimo, el ancho de la variable de la tensión de salida es:

$$\begin{aligned} width_{v_{out}} &= \lceil \log_2 \frac{v_{out}}{\Delta v_{out}} \rceil + n \\ width_{v_{out}} &= \lceil \log_2 \frac{1{,}000}{6,56 \cdot 10^{-6}} \rceil + n \\ width_{v_{out}} &= 27 + n \end{aligned} \tag{6.3}$$

De esta manera, el ancho para la variable de tensión de salida debe ser $28 + n$ bits. En particular, la tensión máxima de salida se ha fijado en $1{,}000V$, y esto es nuevamente, un parámetro seleccionado por el diseñador para esta aplicación.

Finalmente, el mismo análisis debe ser relizado para la Ec. (5.3). En este caso, la corriente de entrada no afronta ningún problema de resolución ya que la corriente de entrada es de $0A$ en DCM, sin valores incrementales. Además, la tensión de salida cambia con la corriente de salida, pero la carga no es normalmente cercana a $0A$, por lo tanto, no deben realizarse mayores consideraciones en este caso.

En resumen, la señal de corriente de entrada debe tener un ancho de $18 + n$ bits, mientras que la tensión de salida debe tener un ancho de $28 + n$ bits. El parámetro n describe el número de bits del valor incremental que es al menos 1 bit. Como se explicara anteriormente, la Ec. (6.1) representa el número de bits necesarios para almacenar a la variable y a su valor incremental. Δx representa los valores típicos de

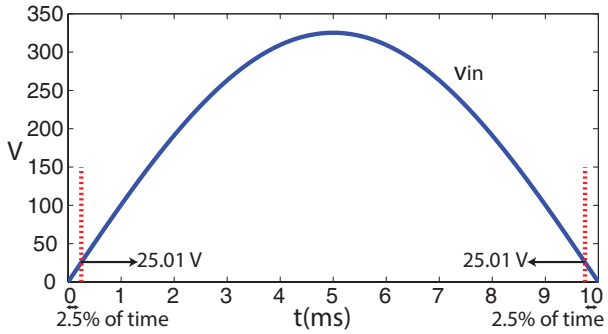


Figura 6.2: Valores de tensión de entrada mínimos considerados para el análisis de resolución

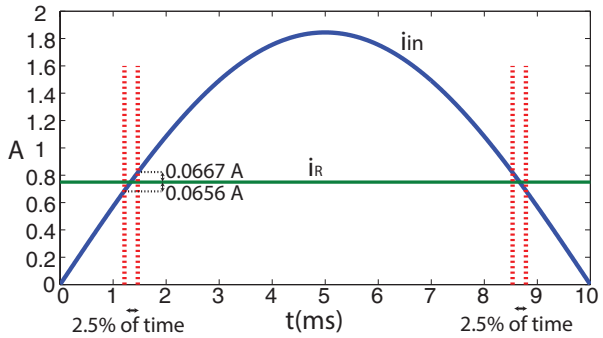


Figura 6.3: Diferencias mínimas de corriente consideradas para el análisis de resolución

incrementos, pero el valor del incremento real no es exactamente igual a Δx, por lo cual más de un bit ($n > 1$) es necesario para almacenar ese incremento con presición. Considerando los resultados del Método 1, este trabajo propone que el valor de n es aproximadamente 8 para poseer suficiente resolución. Por lo tanto, la resolución para variables de corriente de entrada debe ser de 26 bits, mientras que la resolución para las de tensión de salida debe ser de 36 bits, considerando este caso específico donde L = 5 mH y C = 100 μF.

En este trabajo, el análisis ha sido desarrollado para señales de punto fijo. Cuando se utilizan señales de punto flotante, puede pensarse que no es necesario determinar el ancho de la señal ya que las variables de punto flotante permiten almacenar valores pequeños o grandes sin modificar su ancho. Sin embargo, tal como se mencionó anteriormente, la variable debe almacenar tanto su valor típico así como su

valor incremental. Una variable de 32-bit IEEE-754 utiliza 24 bits para almacenar su mantisa: un 1 (uno) fijo y 23 adicionales. Si V_{out} es de aproximadamente $1,000\ V$, el MSB en punto flotante es 2^9 y el LSB es 2^{-14}, que es $6{,}103 \cdot 10^{-5}\ V$. Según la Ec. (6.3), si se considera el peor caso, el valor de incremento para la tensión de salida es $6{,}56 \cdot 10^{-6}$, entonces una señal de punto flotante 32-bit IEEE 754 no posee suficiente resolución para simular esta aplicación.

Cuando se compara una señal de 32-bit IEEE-754 con el análisis previo, la señal de punto flotante tiene 24 bits de mantisa, mientras que la corriente de entrada debería ser de 26 bits, y la tensión de salida debe ser, por lo tanto, de 36 bits. Una posible solución sería utilizar variables de 64-bit IEEE-754 pero el área del diseño para la implementación hardware sería mayor y la velocidad de la simulación se encontraría seriamente afectada.

Los anchos propuestos para corriente de entrada y tensión de salida han sido elegidos considerando el peor caso razonable. Los resultados de los experimentos llevados a cabo con el Método 1 sirven de sustento para verificar los resultados del Método 2. Por un lado, los resultados comprobarán que la desición de descartar el 5 % del tiempo puede ser elegido utilizando el peor caso. Esto significa que agregar más bits no mejorarán significativamente la precisión de la simulación. Por otro lado, los resultados demostrarán que puede utilizarse el peor caso para calcular los anchos de la variables, o bien puede utilizarse un ancho optimizado para acotar la simulación. Los resultados experimentales en la Sección 6.4 han sido extraídos de modelos de punto fijo, pero también podrían aplicarse para corroborar si la mantisa de un modelo IEEE-754 es lo suficientemente precisa.

6.4. Resultados

6.4.1. Escenarios de prueba

La evaluación de la resolución es realizada mediante un modelo de convertidor con diferentes configuraciones de V_{in}[1], L, C, V_{out} y P_{out}, de a una combinación a la vez. Este espectro de posibles configuraciones (Tabla 6.1) describe algunos posibles

[1] RMS rectified input voltage

escenarios reales. Los valores de L y C permanecen fijos para la evaluación de cada escenario.

Cuadro 6.1: Escenarios de evaluación del convertidor

	L	C	V_{in}	V_{out}	P_{out}
Escenario 1	5 mH	100 μF	230 V	400 V	300 W
Escenario 2	1 mH	100 μF	230 V	400 V	300 W
Escenario 3	1 mH	100 μF	110 V	300 V	150 W
Escenario 4	1 mH	470 μF	230 V	400 V	300 W

Para realizar un análisis más amplio, cada DUV con su configuración específica es simulada utilizando diferentes cargas en cada escenario:

a. Test de corriente: La carga es modelada como un drenador de corriente.

b. Test de potencia: La carga es modelada como un consumidor de potencia.

c. Test resistivo: La carga es modelada como una resistencia.

El Método 1 se evalúa simulando cada convertidor dentro de un entorno de verificación. La configuración del DUV y el tipo de carga se mantienen fijo durante la simulación. Los tests se ejecutan en configuración de lazo abierto siguiendo la misma secuencia de valores de tensión de entrada. Al utilizar lazo abierto, se permite que los valores de tensión y corriente evolucionen libremente. Tal como se mencionara en el Capítulo anterior, el regulador realizaría correcciones sobre el controlador llevando al convertidor a ocultar errores causados por insuficiente resolución. En ese caso, se llevaría al diseñador a una elección incorrecta de parámetros.

La performance de cada modelo de convertidor es evaluada considerando el valor medio absoluto en los valores de I_{in} y V_{out} durante la simulación. Debe mencionarse que el error relativo no puede evaluarse, en particular en análisis de corriente, ya que hay situaciones donde sus valores se hacen cero y el error es indeterminado. El tiempo de simulación se ha fijado en 140 ms. Esta duración se ha seleccionado para permitir que las señales de I_{in} y V_{out} se estabilicen luego del período de transitorios,

asegurándose que el modelo del convertidor es evaluado tanto en estado transitorio como en estado estacionario. La ejecución de los tests está automatizada mediante un script *TCL* (Tool Command Language), utilizando la metodología mencionada en el Capítulo anterior. Este script fija los parámetros de diseño n_i y n_v, así como el escenario y las condiciones de la prueba. Finalizada la preparación del escenario, se inicia la simulación. Luego, cuando ha finalizado la prueba sobre el escenario específico, se reportan los errores. Los errores funcionales son listados mientras que las desviaciones son organizados por grupos con determinado error umbral. De esta manera se provee al diseñador con un conjunto de combinaciones (n_i, n_v) que conducen al convertidor a producir en el peor caso un determinado error umbral.

Luego, el Método 2 se evalúa aplicando la Ec. (6.1) para cada escenario y variable de estado, tal como se describió anteriormente en 6.3. Los resultados de su aplicación se presentan en 6.4.4.

6.4.2. Análisis de área y frecuencia

La aplicación del Método 1 resulta en un conjunto de soluciones (n_i, n_v) para construir un modelo de convertidor de punto fijo que cumple con restricciones de error absoluto de I_{in} y V_{out} tolerable. Además, el Método 2 puede ser utilizado para hallar una solución conservadora, aunque no considera algún tipo de especificación del error.

Todas las combinaciones brindadas por el Método 1 cumplen con las restricciones pero, existe un subconjunto de esas combinaciones, quizá una, que minimiza el área del diseño en el dispositivo, optimiza la velocidad del reloj o ambas a la vez. Con el propósito de analizar el impacto en área y frecuencia, todos los convertidores son sintetizados utilizando cada una de las diferentes combinaciones de parámetros de diseño (n_i, n_v). El área del modelo en cierto dispositivo es evaluado en términos del número inferido de multiplicadores, LUTs (Look-Up Tables) y slices. Por otro lado, se evalúa la velocidad máxima de cada convertidor. Un script *TCL* se utiliza para automatizar el proceso. Primero, la síntesis es realizada con diferentes combinaciones de parámetros de diseño (n_i, n_v) y luego se realiza un análisis automático de sus reportes de síntesis para obtener la información de área y frecuencia máxima.

En este trabajo, la síntesis está destinada a la FPGA *Virtex V xc5vlx20t* uti-

lizando la herramienta XST de Xilinx con parámetros por defecto y restricciones automáticas. El convertidor se ha sintetizado 256 veces para diferentes combinaciones: 16 valores para n_i (de 32 a 47 bits) x 16 valores de n_v (en el mismo intérvalo).

Para todas las combinaciones de (n_i, n_v), la herramienta de síntesis ha inferido dos multiplicadores de 18x18-bit, con lo cual se puede deducir que es necesario analizar el área considerando las LUTs solamente. La Fig. 6.5 contempla un subconjunto de parámetros de diseño (n_i, n_v). Los convertidores que poseen esas combinaciones utilizan al menos esa cantidad de slices. Puede observarse que el número de slices usado crece linealmente cuando se agrega un nuevo bit en registros, tanto en tensión como en corriente.

A continuación se considera la información sobre la máxima frecuencia para un convertidor con determinada configuración. Las líneas horizontales en la Fig. 6.6 muestran que para cualquier ancho de registro de corriente evaluado, la máxima frecuencia se mantiene para un determinado ancho de registro de tensión. Esto significa que la máxima frecuencia del circuito depende de la cantidad de bits utilizados en los registros de V_{out} en esta aplicación. En particular, esta dependencia se debe a un camino crítico en el lazo de cálculo de V_{out}. Este cálculo no puede ser segmentado ya que el cómputo de cada nuevo estado, k, necesita conocer el estado inmediatamente anterior, k-1, tal como se enuncia en las Ec. (5.1), (5.2) y (5.3).

6.4.3. Resultados del Método 1

Los experimentos son llevados a cabo utilizando las siguiente infraestructura software/hardware:

- *Simulador*: Mentor Graphics Questasim-64bit® 10.0c.
- *Synthesizer*: Xilinx XST®0.4D.
- *Hardware*: Intel i5® - 4Gb RAM utilizando Microsoft Windows 7 ®.

Luego de ejecutar todos los tests en cada uno de los escenarios, se analizan los errores para tensión y corriente. La performance de los convertidores se resume gráficamente a través de proyecciones. En la Fig. 6.4, el eje x representa los valores que se le asignan a n_i, y el eje y representa los valore asignados a n_v. El eje z representa

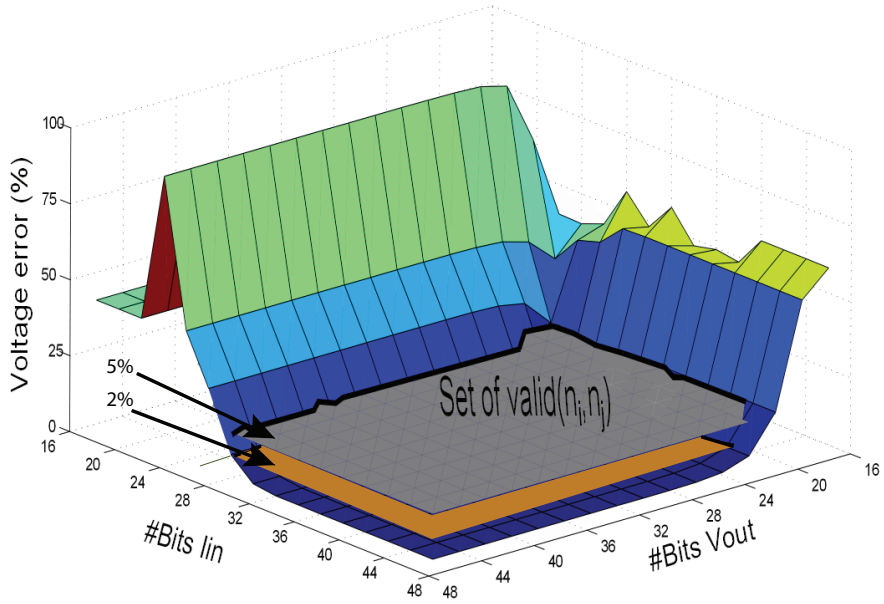


Figura 6.4: Planos proyectados para parámetros de diseño de convertidores con errores de tensión $error \leq 5\,\%$ y $error \leq 2\,\%$

el porcentaje de error. De esta manera, la información de error en V_{out} e I_{in} describe una superficie tridimensional. A modo de ejemplo, la Fig. 6.4 muestra un plano para esas combinaciones de parámetros con un error en V_{out} del 5 % y un plano para aquellas con un error del 2 %. Las curvas de nivel de las Figs. 6.7 a 6.10 son proyecciones de la intersección entre la superficie de error y los planos paralelos a el plano (x, y) en diferentes porcentajes de error. De esta forma, cada curva delimita un conjunto de parámetros de diseño con cierto error umbral. Las curvas presentadas con linea punteada representan proyecciones en error de corriente, mientras que las curvas con línea continua representan errores en tensión.

Puede observarse en las Figs. 6.7 a 6.10 que ninguna curva etiquetada de tensión intersecta a una de corriente con las misma etiqueta. Este fenómeno se repite para todos los tests y en todos los escenarios y demuestra que el error en corriente acota la selección de parámetros de diseño dentro de las combinaciones válidas. Además,

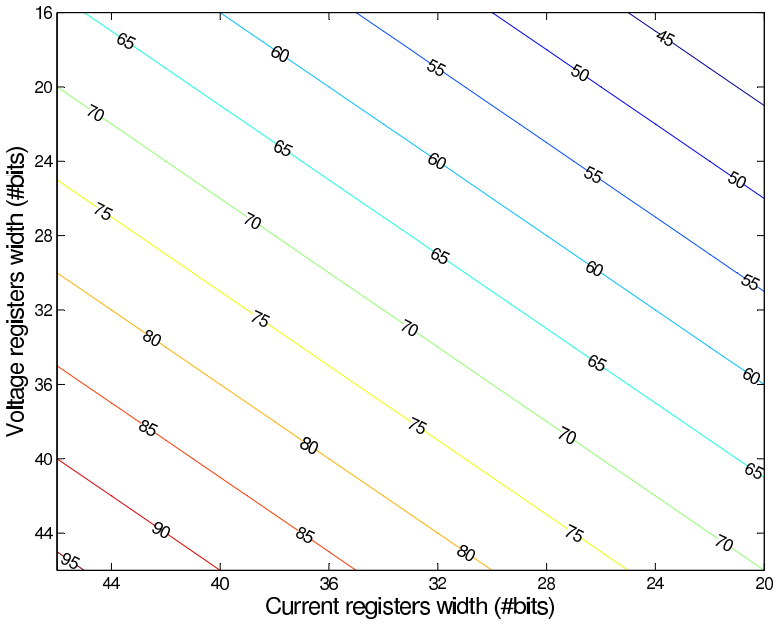


Figura 6.5: Utilización de Slices para el convertidor de punto fijo

se muestra que el porcentaje de error en tensión es siempre menor que el porcentaje de error en corriente para cualquier combinación analizada (n_i, n_v).

Las Figs. 6.7 a 6.10 resumen el error obtenido en los modelos de convertidor. Puede observarse que el error en corriente puede variar entre 50 % y 0 % en sólo cinco bits de resolución (por ejemplo, 17 a 22 bits en corriente). Este hecho, refuerza la hióptesis de que una selección incorrecta de parámetros de diseño pueden llevar a valores de tensión y corriente indeseables. Cuando n_i es mayor que 23 y n_v es mayor que 30, el error medio absoluto en corriente y tensión es cero o tiende a cero.

Cuadro 6.2: Parámetros de diseño n_i, n_v para errores del 2 %

	Corriente	Potencia	Resistiva
Escenario 1	(23, 30)	(24, 30)	(22, 30)
Escenario 2	(24, 31)	(24, 32)	(22, 32)
Escenario 3	(23, 31)	(23, 32)	(23, 32)
Escenario 4	(22, 31)	(23, 31)	(24, 31)

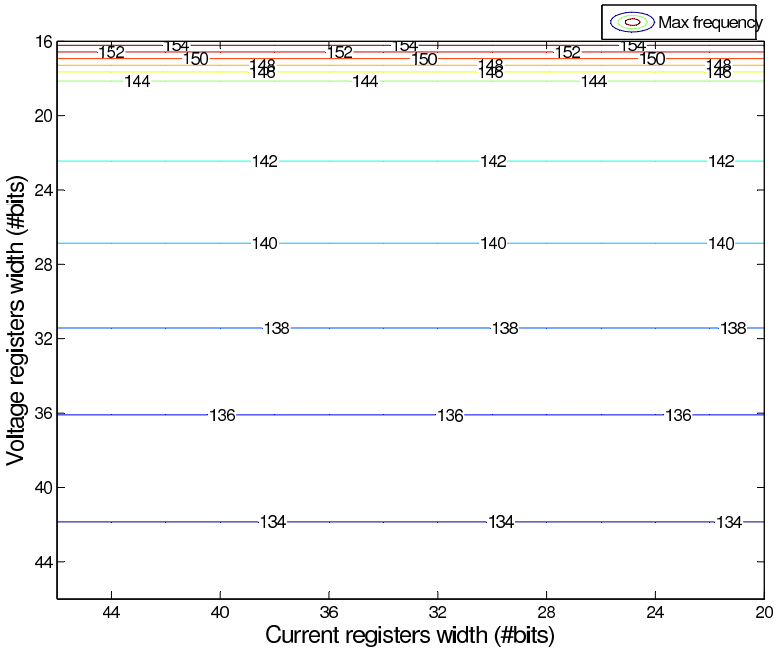


Figura 6.6: Frecuencia de reloj máxima

Las Tablas 6.2 y 6.3 resumen las mejores soluciones, considerando los resultados de la síntesis, que cumplen las restricciones para construir un convertidor con un error absoluto medio del 2 % y del 5 % respectivamente

En las Figs. 6.5 y 6.6, puede apreciarse que hay una diferencia de aproximadamente 5 slices y 2 MHz entre la curva con el menor error en corriente y en tensión y aquellas con el mayor error. Es por eso que para los escenarios propuestos, la selección correcta de parámetros de diseño puede ser realizada utilizando los máximos valores de la Tabla 6.2, n_i=24, n_v=32.

Cuadro 6.3: Parámetros de diseño n_i, n_v para errores del 5 %

	Corriente	Potencia	Resistencia
Escenario 1	(21, 26)	(20,25)	(20, 26)
Escenario 2	(21, 29)	(21, 29)	(20, 30)
Escenario 3	(22, 30)	(22, 30)	(22, 31)
Escenario 4	(21, 29)	(21,30)	(22, 29)

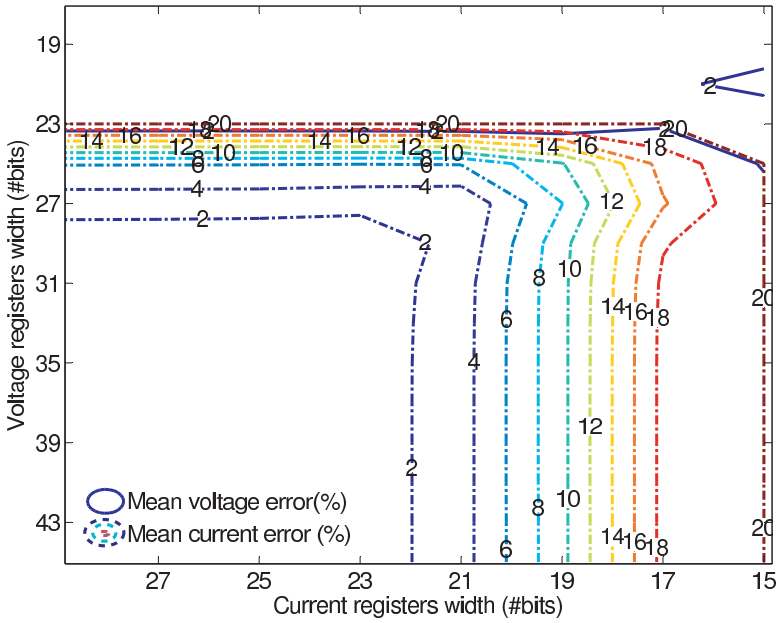


Figura 6.7: Errores de tensión y corriente en Escenario 1

6.4.4. Resultados del Método 2

A continuación se aplica el método analítico para obtener combinaciones conservadoras de parámetros de diseño (n_i,n_v). Dado que este se basa solamente en el modelo del convertidor, su aplicación no considera ni el tipo de carga ni un error tolerable. Sin embargo, como la configuración del escenario sí tiene impacto sobre el diseño, el método se aplica para cada configuración. La Tabla 6.4 resume los resultados de la aplicación del Método 2 para cada escenario. En la tabla puede observarse que la resolución para los registros de tensión varía entre 34 y 38 bits, mientras que la resolución de los registros de corriente varía entre 23 y 26 bits. Si se consideran las resoluciones más altas del Método l (considerando los diferentes escenarios y tipo de carga), es posible notar que el Método 2 es conservador, tal como se preveía, y que el valor n = 8 se adecua para esta aplicación específica. Sin embargo, el parámetro n puede ser difícil de conocer a priori si el Método 1 uno no fue aplicado con anterioridad.

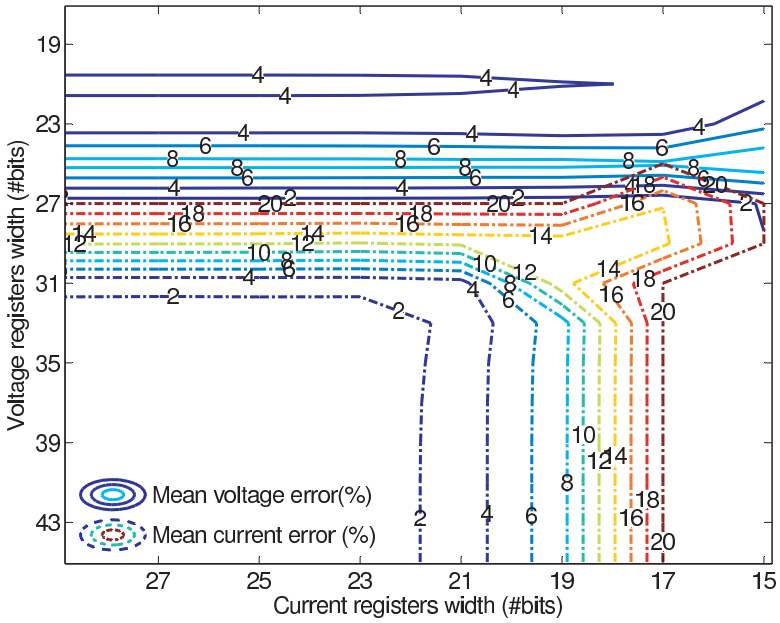


Figura 6.8: Errores de tensión y corriente en Escenario 2

Cuadro 6.4: Parámetros de diseño n_i, n_v utilizando el método analítico cuando $n = 8$

	$(n_i,\ n_v)$
Escenario 1	(26, 36)
Escenario 2	(23, 36)
Escenario 3	(25, 34)
Escenario 4	(23, 38)

Dado que el Método 2 no considera el error máximo tolerable en la Ec. (6.1), existe un valor umbral de error para el Método 1, donde las resolución de las variables son mayores que para el mismo caso del Método 2. Esta situación puede observarse comparando el Escenario 4 en las Tablas 6.2 (Carga resistiva) y 6.4. En ese caso, las resolución para n_i resultante del Método 1, es un bit más grande. De todos modos, puede observarse que en la Tabla 6.3 esta situación no ocurre para errores tolerables mayores al 5 %.

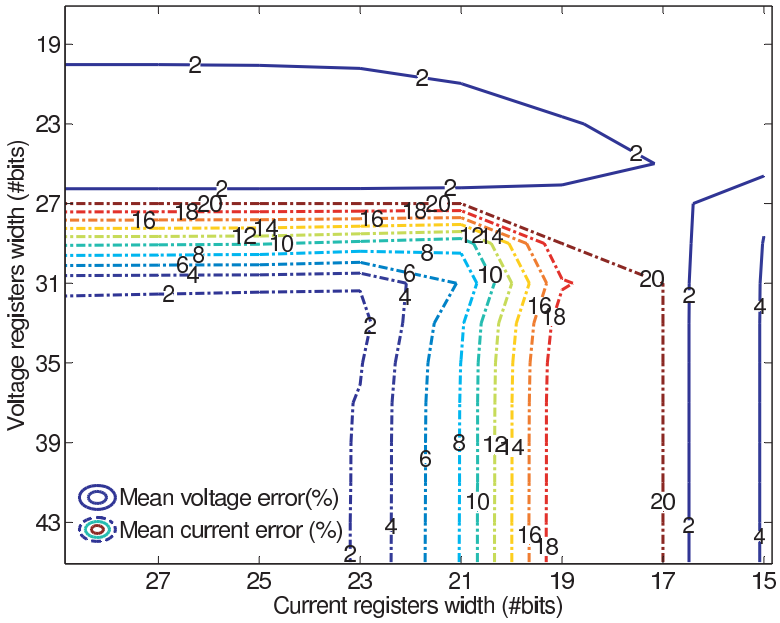


Figura 6.9: Errores de tensión y corriente en Escenario 3

6.5. Conclusiones del capítulo

Este Capítulo se enfoca en el estudio de resolución de variables de estado de punto fijo en modelos de convertidores de potencia. Para esto se proponen dos métodos diferentes. El primer método está basado en simulación. El análisis de resolución se realiza midiendo el error absoluto medio del modelo de convertidor considerando varias combinaciones de resoluciones en tensión y corriente mientras se verifica su funcionalidad. Aunque la desventaja del Método 1 puede ser el tiempo de ejecución, éste provee la resolución óptima de registros y garantiza el correcto comportamiento del modelo.

El segundo método es analítico y sobreestima la resolución de las variables de estado. Esta sobreestimación puede incrementar la utilización de recursos en el dispositivo y puede acotar la frecuencia de reloj, pero más importante es el hecho de que el error de presición provocado por el modelo es desconocido. Sin embargo, su aplicación es rápida y simple.

Luego de la simulación de un conjunto completo de configuraciones, el primer

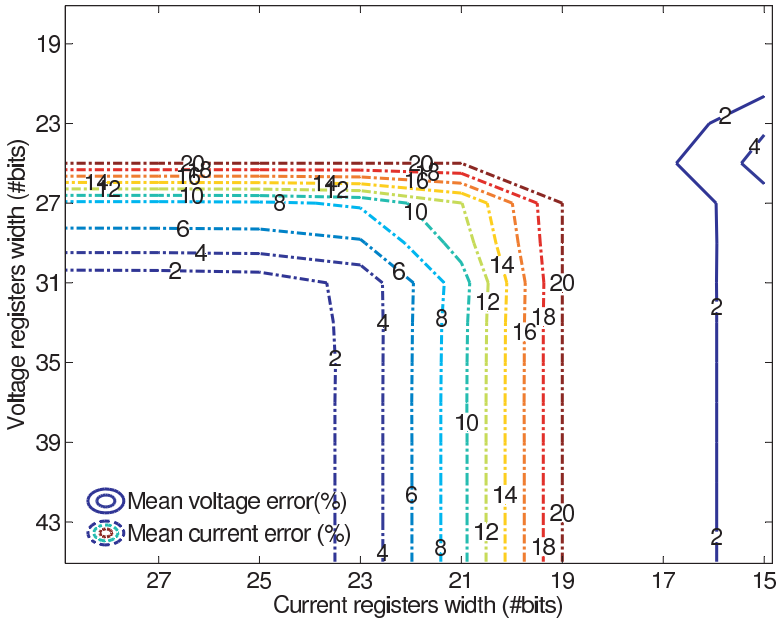


Figura 6.10: Errores de tensión y corriente en Escenario 4

método demuestra que el error absoluto medio en corriente es siempre mayor que el error en tensión para cualquier resolución en el ejemplo de aplicación propuesto.

Dado que varias configuraciones cumplen la misma restricción en error de tensión y corriente, se realiza un análisis de la síntesis de manera de poder lograr la mayor resolución con velocidad de reloj y área del diseño optimizados. Los resultados de la síntesis indican que el uso de slices en el dispositivo destino se incrementa linealmente cuando un nuevo bit es agregado tanto en registros tensión como en los de corriente. La síntesis, además, ha demostrado que la máxima frecuencia de reloj es gobernada por la resolución de los registros de tensión.

El Método 1 demuestra que la resolución calculada por el Método 2 es correcta pero conservadora. Además, el análisis de área y frecuencia muestra las diferencias en el área y frecuencia máxima entre el enfoque conservador y los valores óptimos del método basado en simulación. Para el ejemplo propuesto, la resolución calculada por el Método 1 mejora el área del convertidor en un 10 %, mientras que la mejora en la frecuencia de reloj es bastante pequeña (aproximadamente el 2 %).

Los experimentos han demostrado que los errores crecen abrúptamente de 0 %

a 50 % en sólo cinco bits bajo la resolución óptima y para el ejemplo de aplicación propuesto. Si la resolución está por debajo ese intervalo, los errores provocados se tornan inaceptables.

En resumen, la aplicación de los métodos propuestos demuestra que es posible implementar convertidores de potencia conmutados utilizando representación de punto fijo sin pérdida de precisión mientras se optimiza el área y frecuencia en el dispositivo.

Capítulo 7

Conclusiones del trabajo

El trabajo presenta una serie de aportes orientados a mejorar la verificación funcional moderna. A lo largo de los diferentes Capítulos se presentan problemáticas reales presentes en proyectos de desarrollo de sistemas digitales y es el objetivo de este trabajo abordarlas. En los primeros Capítulos se reconoce la importancia de la verificación funcional dentro del ciclo de desarrollo. Allí se mencionan las diversas herramientas que presenta tanto la verificación funcional tradicional como la moderna. Las fortalezas y debilidades de cada herramienta se ven expuestas en cada Capítulo al abordarse diferentes casos de estudios. Se exponen las fortalezas destacadas en la verificación de cierto diseño mientras que las debilidades son analizadas, mejoradas y generalizadas a otros tipos de problemas. Si bien al final de cada Capítulo se presentan conclusiones específicas para el mismo, en este Capítulo se presentan brevemente algunas conclusiones globales del trabajo y posible trabajo futuro sobre la misma línea de investigación.

7.1. La interminable tarea de verificación

En el Capítulo 2, se expone un caso de estudio de un módulo de funcionalidad simple pero no menos importante. Se trata de un Sumador /Restador, cuya funcionalidad puede resultar trivial. Sin embargo, la verificación de unidades de punto flotante involucra un anális exhaustivo no sólo del dominio de posibles valores de entrada, sino también del comportamiento de las operaciones dentro de ese dominio.

En el estudio se evidencia el enorme espacio de búsqueda que debe ser analizado para una verificación funcional completa. El tratamiento de diseños de mayor complejidad involucra espacios de búsqueda aun mayores por lo cual se propone el estudio de casos tanto típicos como extremos. En ese caso puntual, el análisis del dominio de posibles entradas es traducido a un conjunto de restricciones para generar conjuntos de operandos. Los resultados muestran que la generación aleatoria con esas restricciones conducen al diseño a casos no previstos o bien no interpretados por el diseñador. Si bien las operaciones presentadas en este Capítulo se limitan a la suma y resta, se describe como extender esta estrategia a otras operaciones aritméticas. El Capítulo se sustenta en la verificación funcional utilizando entornos basados en dos frameworks.

El objetivo del Capítulo 3 es alcanzar una completa verificación funcional mediante la generación dirigida por la cobertura de casos de prueba. Los resultados indican la cobertura funcional crece a medida que el generador aleatorio produce casos diferentes. En un punto este comportamiento permanece asintótico. Para completar la cobertura, se adoptan dos estrategias. La primera involucra el cambio de la semilla de generación, pero la mejora es casi nula. La segunda estrategia analiza los casos aún no ejecutados y reconfigura el generador para que sean probados. En este caso la mejora se comporta linealmente y logra una completa cobertura. Debe aclararse que esta estrategia es aplicable ya que, en el instante donde la cobertura es asintótica, el espacio de búsqueda ha sido disminuido varias magnitudes.

7.2. Un nuevo enfoque de la verificación

En la segunda parte del trabajo se aborda la verificación funcional de un modelo de convertidor de potencia. En particular, el modelo está orientado a la simulación en HIL y por lo tanto el diseñador puede sintetizar el modelo del convertidor junto con el controlador dentro del hardware final para la evaluación del conjunto. El enfoque propuesto pretende enriquecer los resultados de la verificación de manera tal de mejorar el globalmente el desarrollo del proyecto. Dado que la verificación funcional, principalmente aquella basada en simulación, debe estimular y evaluar al diseño o una parte de él, se pretende recolectar y utilizar la información de las pruebas exito-

sas en la verificación de manera tal que puedan ser utilizadas como realimentación para otros procesos, como por ejemplo, diseño. En el caso de estudio presentado, a la vez que se verifica funcionalmente el modelo, se extraen características de su funcionamiento. En este caso, las estadísticas relevadas ayudaron a ajustar parámetros que modelaban componentes analógicos. Luego, se realiza un análisis de resolución sobre registros que almacenan dos variables de estado (I_{in} y V_{out}). Como la verificación funcional debe considerar el análisis de todas las posibles configuraciones, sus resultados son utilizados para la elección de los parámetros de diseño de resolución. Tal como se menciona y se demuestra en el trabajo, la elección de estos parámetros es una tarea difícil por métodos analíticos. Como resultados de este análisis se obtiene que es posible obtener una resolución aceptable y óptima en relación a máxima frecuencia de funcionamiento y mínimo consumo de recursos.

7.3. Producción surgida del presente trabajo de tesis

A lo largo del trabajo de tesis, se han realizado tres publicaciones en congresos especializados (*Experiences applying framework-based functional verification to a design for programmable logic* [45], *Generic construction of monitors for Floating-Point Unit designs* [28], *Components for Coverage-Driven Verification of floating-point units*[27]) con referato internacional. Esos trabajos surgieron para verificar los sistemas digitales propuestos como en [29]. Con el objetivo de demostrar lo presentado en [52], se ha desarrollado en colaboración con un conjunto de investigadores de la UAM (Universidad Autónoma de Madrid) un entorno para la verificación funcional de modelos de convertidores de potencia para sistemas HIL. De esta interacción surge la motivación del enfoque colaborativo de la verificación funcional en el proceso de desarrollo presentado en este trabajo de tesis. Los resultados del trabajo en conjunto han sido publicados en el Journal *Transactions on industrial informatics* de IEEE (*Resolution Analysis of Switching Converter Models for Hardware-In-the-Loop* [26]).

Con respecto a la transferencia de tecnología al sector productivo, se han llevado a cabo tres proyectos. El primer proyecto consistió en el desarrollo de un entorno basado en OVM [25] orientado a verificar un Bridge de protocolo VME a AMBA

para la empresa *INVAP S.E.*. El trabajo titulado *Verificación Funcional para un Bridge VME/AMBA* se encuentra en el marco del Convenio específico entre INVAP SE y la Facultad de Ciencias Exactas, UNCPBA (Expediente No 1-41091/11). El segundo proyecto realizado para la empresa *Redimec S.R.L.* es titulado *Desarrollo de un módulo conversor de protocolo ARINC429* se encuentra en el marco del Convenio específico Redimec SRL y la Facultad de Ciencias Exactas, UNCPBA (Resolución N2282, Expediente 1.-18562/2003) cuya aplicación está orientada al sector aeronáutico militar. Por último, se desarrolló un entorno basado en UVM para la verificación de un controlador de radar meteorológico en el proyecto titulado *Verificación de diseños de sistemas digitales* en el marco del Convenio específico entre EMTECH S.A. y la Facultad de Ciencias Exactas, UNCPBA (RR 1904/13).

7.4. Trabajo futuro

El trabajo aborda técnicas e infraestructuras para mejorar la calidad del proceso de verificación funcional. Sin embargo, poco se ha tratado sobre la aceleración de este proceso. Como trabajo futuro se propone desarrollar técnicas para acelerar la etapa de verificación funcional de un sistema digital. Entre los objetivos específicos se planea:

- Partiendo de técnicas basadas en simulación se pretende acelerar la verificación mediante:
    - La emulación tanto del DUV (Device Under Verification) así como parte del testbench en hardware reprogramable.
    - Sintetizar aserciones sobre hardware reprogramable, permitiendo así el monitoreo y chequeo de señales sobre la implementación del dispositivo final.
    - Especializar técnicas HIL (Hardware-In-the- Loop) para emulación de planta. Esto resultaría particularmente útil en la verificación de diseños industriales donde no es posible o resulta riesgosa la interacción con un dispositivo defectuoso.

- Desarrollar técnicas para agilizar el proceso de verificación dirigido por la cobertura. En particular, se pretende optimizar el conjunto casos de pruebas de manera tal de lograr máxima cobertura con la mínima cantidad de estímulos.

Apéndice A

Verificación del modelo de convertidor de potencia: Resultados

En esta sección se presentan los resultados completos recolectados durante la verificación funcional. Se considera oportuno presentar los resultados aquí ya que un resumen de los mismos es suficiente para corroborar la hipótesis presentada en los respectivos capítulos.

Los resultados se presentan en forma de gráficas. Las gráficas fueron representadas utilizando la herramienta MatLab y dependiendo del detalle requerido, se utiliza una ampliación de una instantánea acotada para obtener una vista más detallada. Las pruebas que involucran tensiones de entradas correspondientes a corriente alterna, son extraídas de tabla y pertenecen a varios semicíclos de tensión alterna procedentes, en este caso, de la red eléctrica (A.1).

En casos donde la tensión de entrada contiene fluctuaciones, se muestra una gráfica para cada caso en particular. Por último, los casos que poseen tensión fija a la entrada no se representan.

El tiempo de ejecución de cada prueba así como la frecuencia de muestreo de las salidas depende el comportamiento de las entradas. Si estas tienen un comportamiento periódico o monótono, se realizan pruebas largas y se promedia en gran cantidad de ciclos. En cambio, si los valores son dinámicos, ya sean aleatorios o con algún

comportamiento conocido, las pruebas son cortas y con muestreos de alta resolución.

A.1. Carga tipo corriente y tensión de entrada alternada

La ejecución de las pruebas se realizó durante 200ms. Los resultados muestran valores de tensión muy similares entre el modelo de referencia y el modelo QXY. Ambos modelos muestran tensiones de salida que siguen el patrón de los semicíclos de la tensión.

Si bien en la Figura A.3 es difícil encontrar las diferencias entre ambos modelos, la Figura A.3 muestra que el error entre ambos modelos varía siguiendo el patrón de los ciclos de tensión alterna siendo el error negativo cuando el semicíclo parte desde o vuelve a cero.

En los primeros 60 ms el patrón del error es el mismo sólo que su amplitud va creciendo (transitorio). Transcurrido ese tiempo, el régimen del error, es permanente. La Figura A.4 muestra una vista ampliada de uno de los ciclos de variación, en particular el que se encuentra entre 80 ms y 90 ms. De la diferencia puede deducirse que cuando la tensión de entrada es cercana a cero, el modelo QXY provee menos tensión que el modelo de referencia. La situación opuesta ocurre cuando la tensión

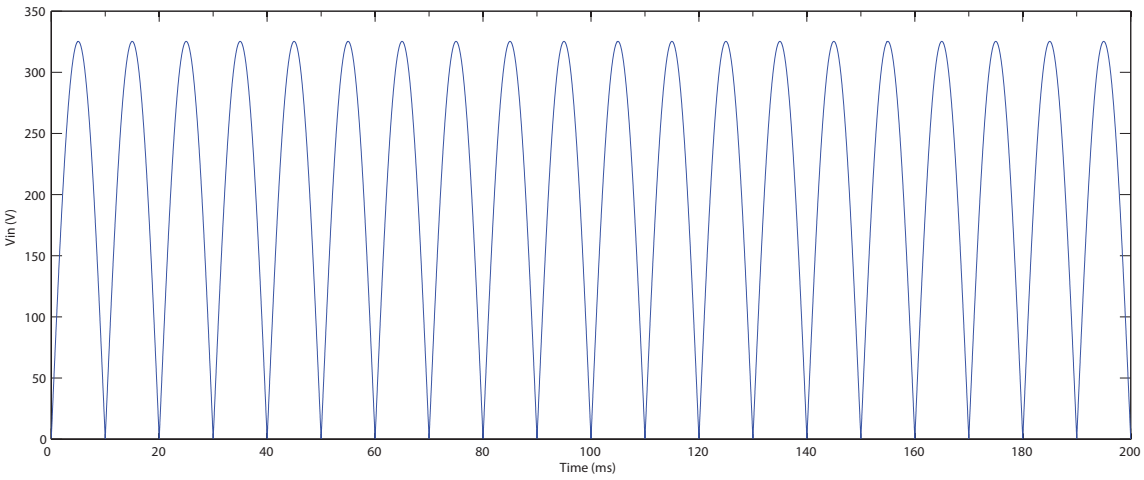


Figura A.1: Tensión de entrada de corriente alternada $V_{in} = VAC$

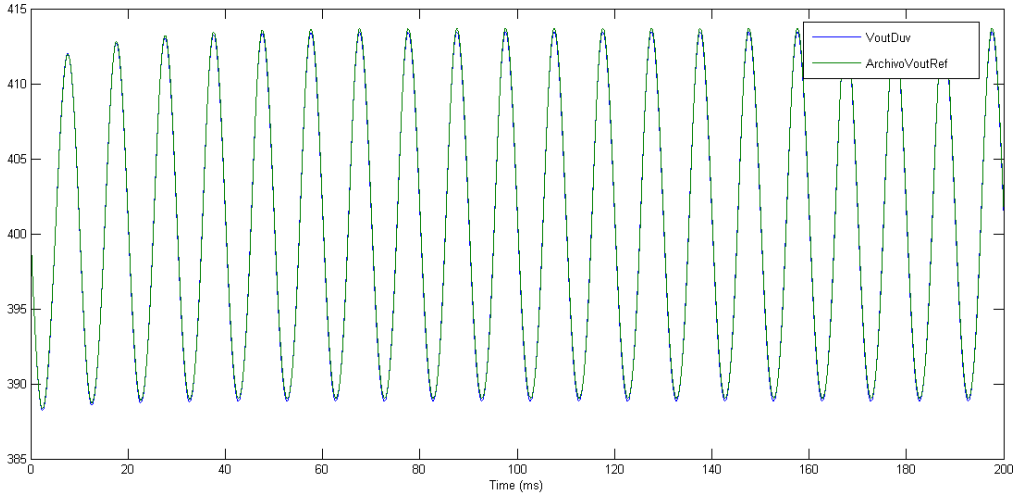


Figura A.2: Tensión de salida - Carga tipo corriente y $V_{in} = VAC$

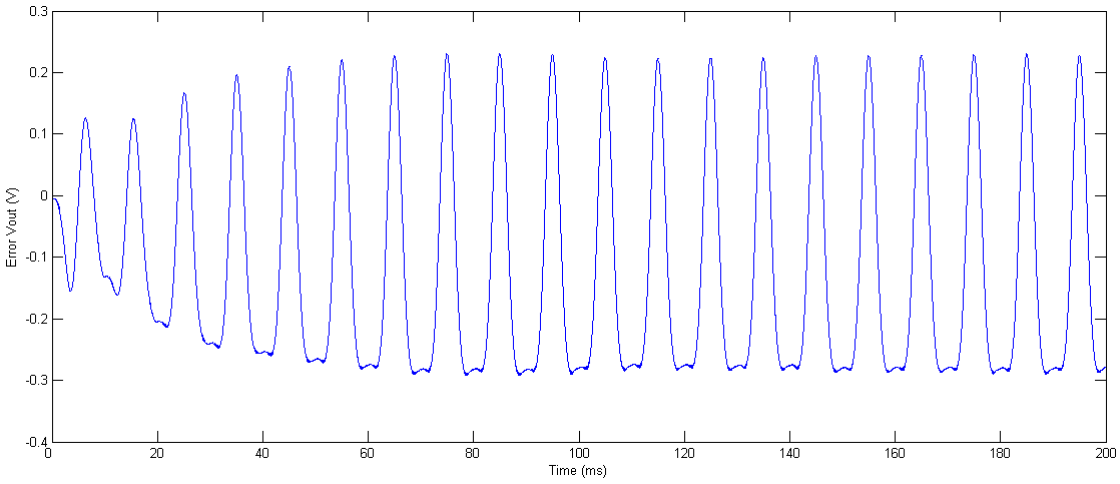


Figura A.3: Error en tensión de salida - Carga tipo corriente y $V_{in} = VAC$

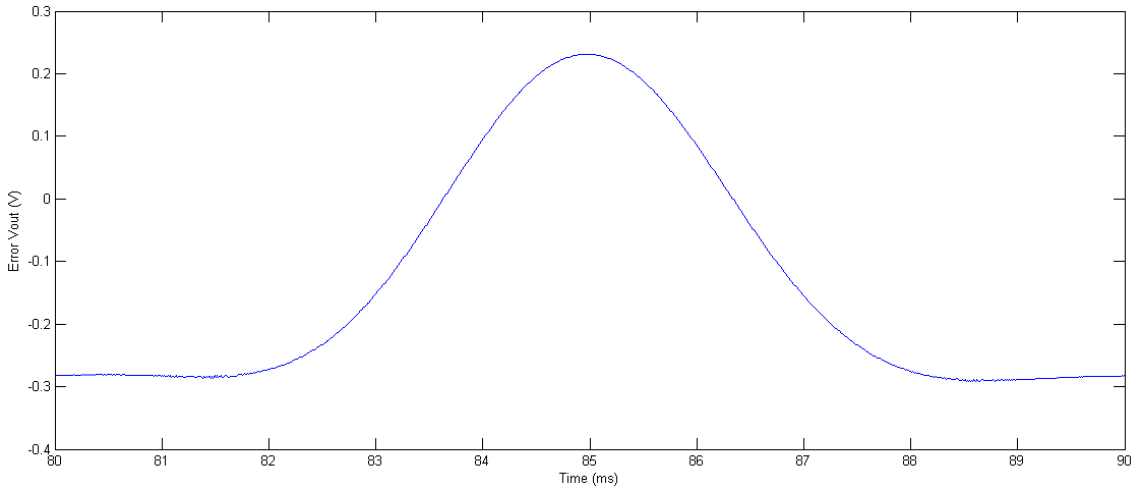


Figura A.4: Error en tensión de salida detallado entre 80ms y 90ms - Carga tipo corriente y $V_{in} = VAC$

de entrada llega a su valor máximo.

El comportamiento de la corriente de entrada descrito es similar al de la tensión de salida. Puede observarse que transcurrido los mismos 60 ms, el comportamiento de la tensión es periódico y sigue el patrón de los semicíclos de la tensión de entrada alterna.

Sin embargo, puede observarse de la A.5 que el comportamiento del error en la corriente de entrada es diferente pero aun así, sigue siendo periódico.

Puede observarse en las Figura A.7 que el comportamiento del error tiene comportamiento periódico. En particular puede observarse que el error es cercano a cero cuando el valor de la tensión de entrada es cercano a cero o bien tiene su valor máximo. Por otra parte los valores de tensión de salida del modelo QXY son mayores a los del modelo de referencia en la parte creciente del semicíclo de tensión alterna y menores en la decreciente, obteniendo errores positivos y negativos respectivamente.

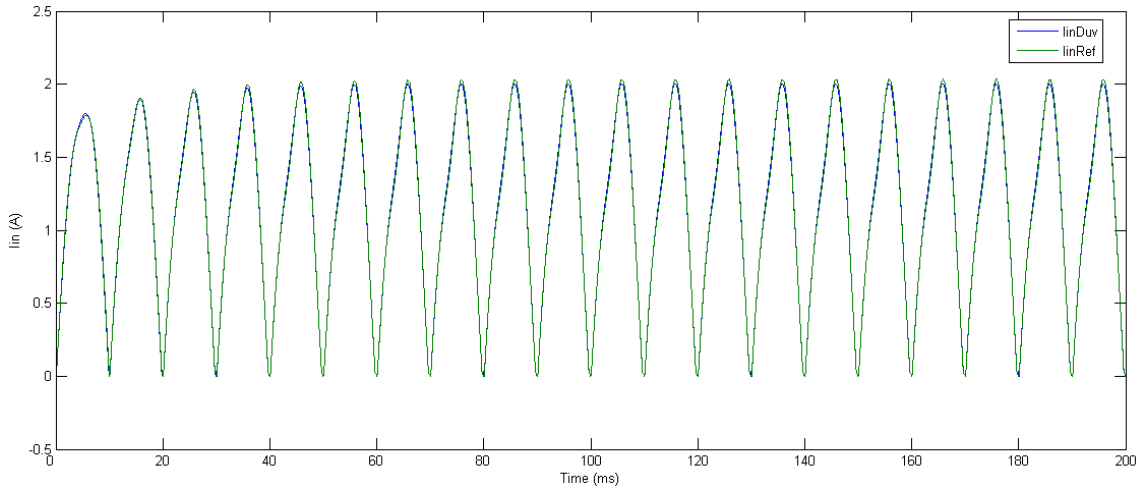


Figura A.5: Corriente de entrada - Carga tipo corriente y $V_{in} = VAC$

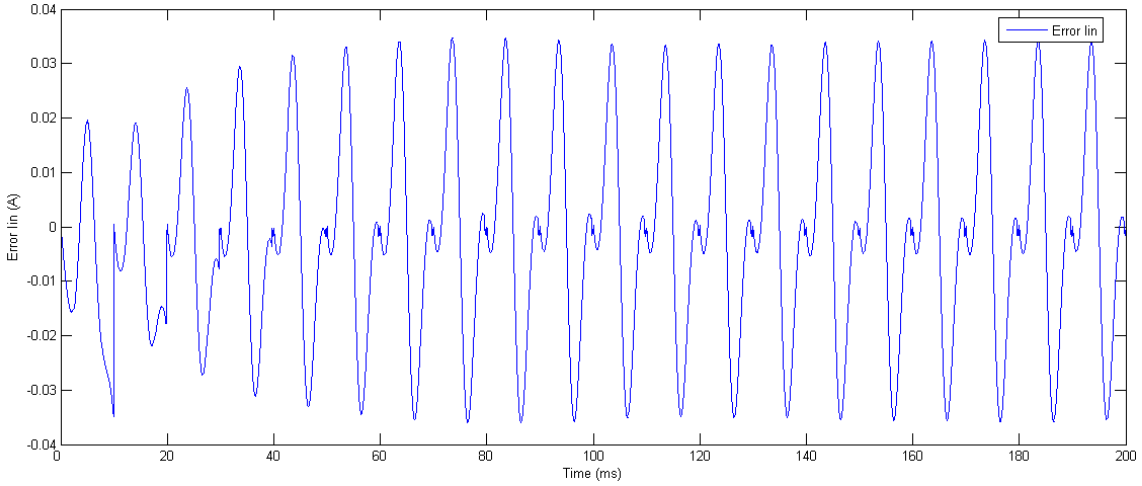


Figura A.6: Error en corriente de entrada - Carga tipo corriente y $V_{in} = VAC$

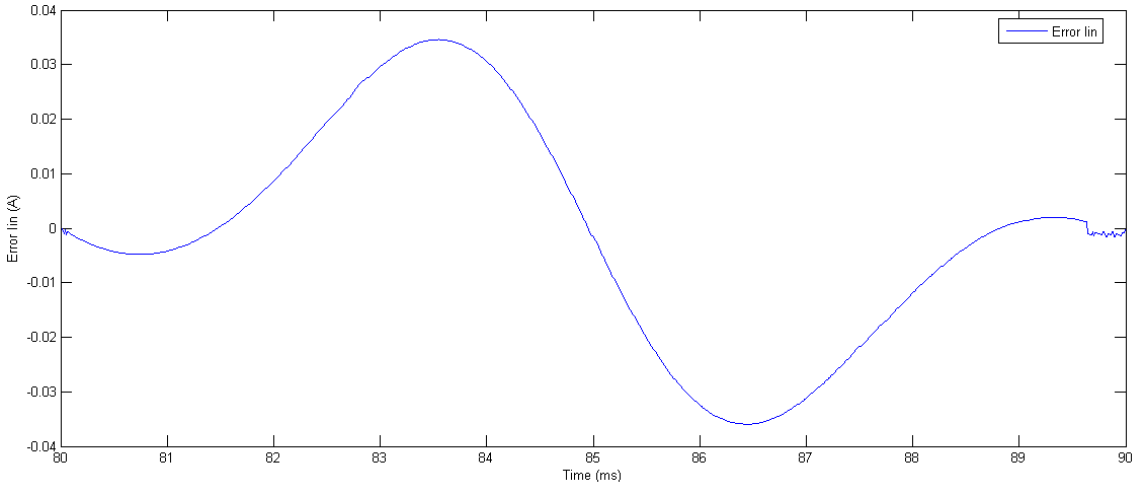


Figura A.7: Error en corriente de entrada detallado entre 80ms y 90ms - Carga tipo corriente y $V_{in} = VAC$

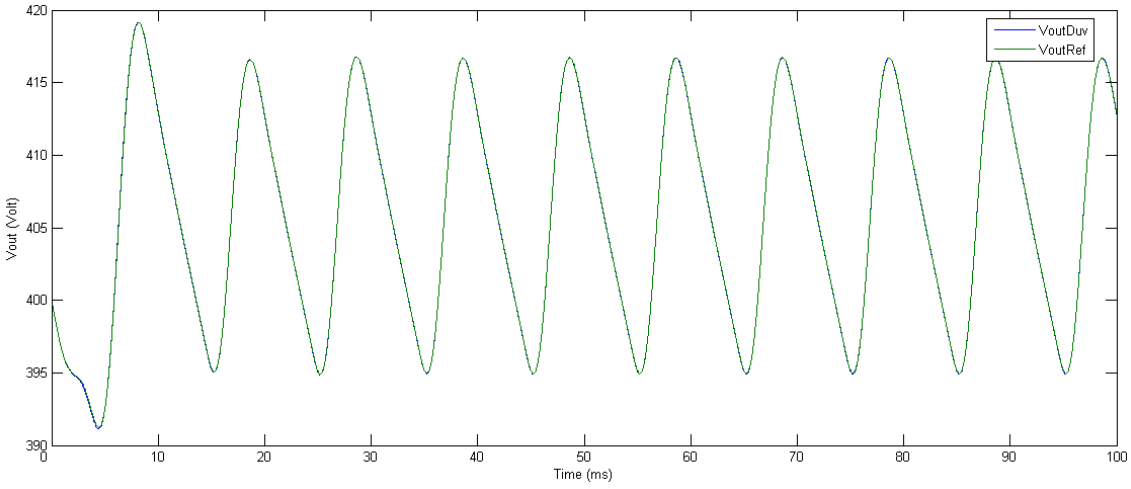


Figura A.8: Tensión de salida - Carga tipo resistencia y $V_{in} = VAC$

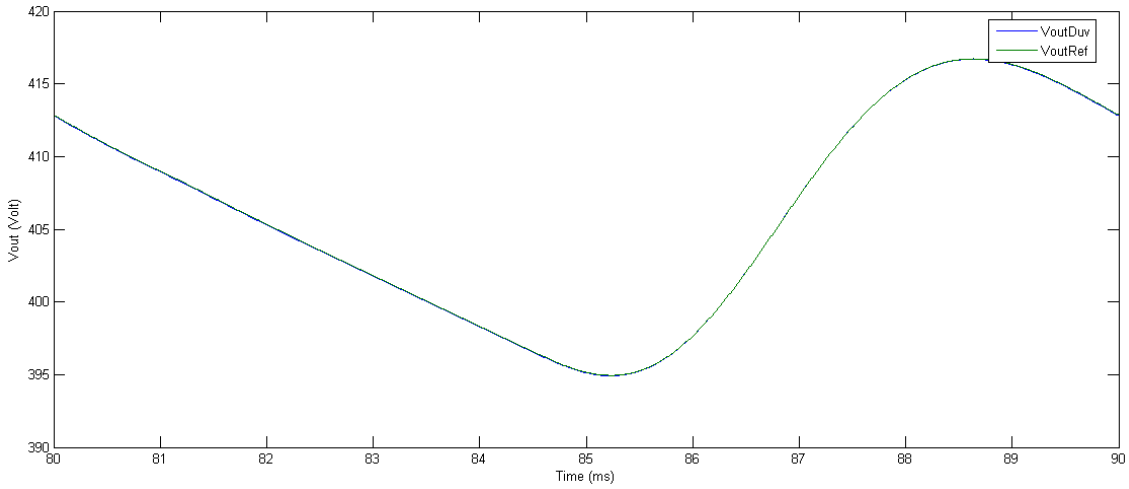


Figura A.9: Tensión de salida detallada entre 80ms y 90ms - Carga tipo resistencia y $V_{in} = VAC$

A.2. Carga tipo resistencia y tensión de entrada alternada

La Figura A.8 muestra que el comportamiento de ambos modelos de convertidor es periódico. Sin embargo, es posible ver que la desviación máxima por encima del valor ideal en ambos modelos no se produce ni en el máximo del semicíclo ni en cero.

El valor ideal es aproximadamente 1.5ms antes y después del valor máximo del semicćlo de tensión de corriente alterna. Esta característica se puede apreciar en la Figura A.9.

De las diferencias entre ambos modelos se puede decir que también sigue un patrón monótono luego de 10ms (primer semicíclo). En general el valor de tensión de salida del modelo QXY es menor que el del modelo de referencia. Este patrón tiene la misma frecuencia que los ciclos de la tensión alterna de entrada. Sin embargo, no se encuentra una relación directa del error con el valor del semicíclo.

Al igual que en análisis anteriores, el patrón de las corriente de entrada es monótono y a igual frecuencia que los semicíclos de la tensión de entrada alterna. Puede observarse además que la corriente de entrada es nula en los valores cero de los semicíclos de la tensión alterna.

Por otro lado el análisis de errores en corriente de entrada no es trivial. La Figura

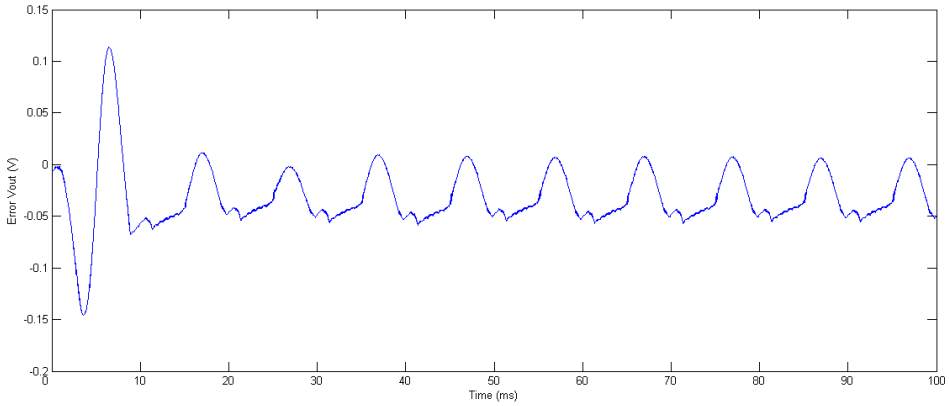


Figura A.10: Error de tensión de salida - Carga tipo resistencia y $V_{in} = VAC$

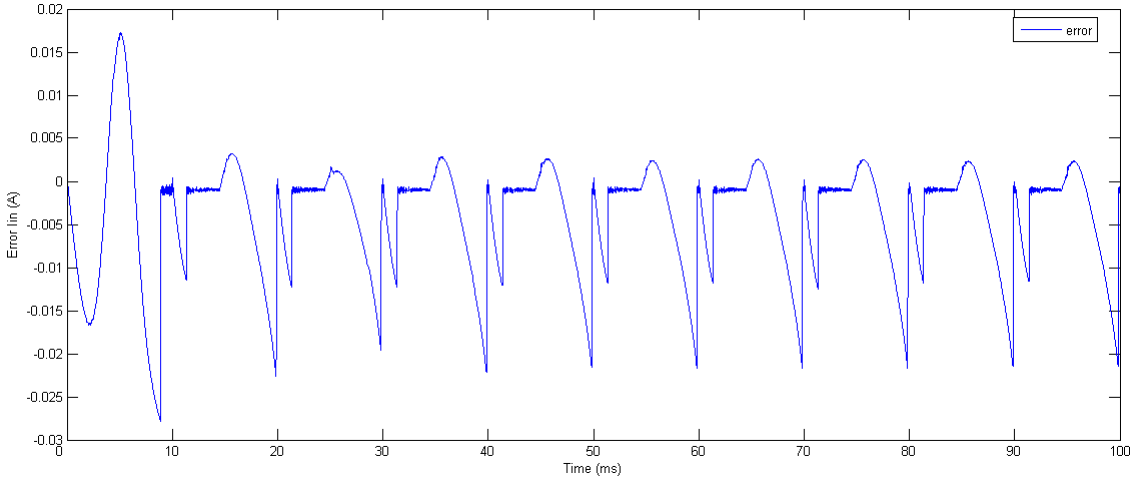


Figura A.11: Error en corriente de entrada - Carga tipo resistencia y $V_{in} = VAC$

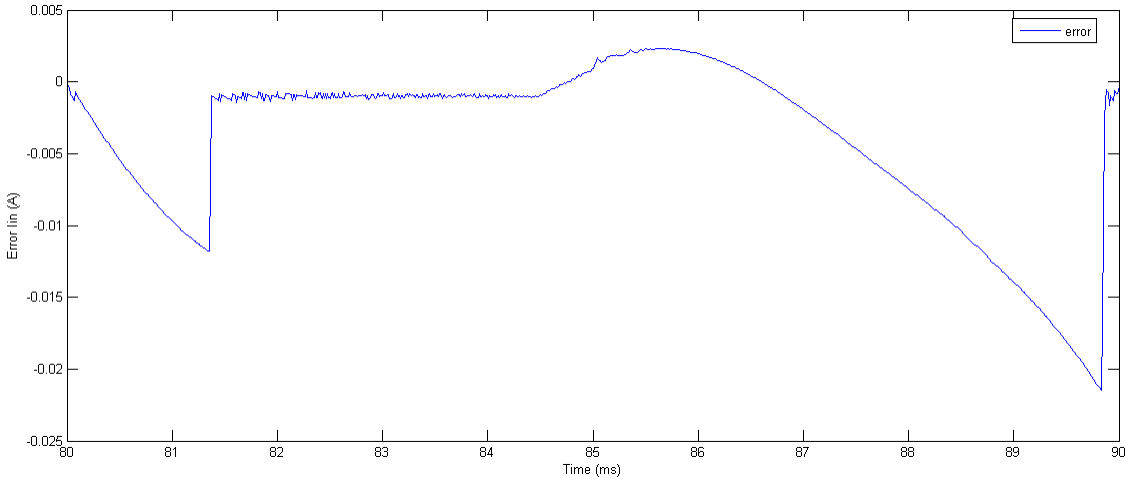


Figura A.12: Error en corriente de entrada detallado entre 80ms y 90 ms - Carga tipo resistencia y $V_{in} = VAC$

A.12 muestra la región ampliada de la Figura A.11 entre 80 ms y 90 ms.

A.3. Carga tipo potencia y tensión de entrada alternada

La Figura A.13 muestra la tensión para carga tipo potencia de ambos modelos. Es posible ver a través de la vista ampliada presentada en Figura A.14 que existen muy poca diferencia entre ambos modelos y que la proximidad a el valor ideal (400 volt) se corresponde a los instantes que la tensión de entrada es cercana a cero y cuando tiene su valor máximo. En la parte creciente del semicíclo, los valores de tensión de entrada en ambos modelos se encuentran por debajo del ideal y en la decreciente por encima del mismo.

Al igual que en carga tipo corriente, puede decirse que el comportamiento del error es monótono luego de los primeros 60 ms. Puede deducirse además que el error es nulo 1.5ms antes y después del máximo del semicíclo de tensión alterna. Los errores son máximos positivos en el máximo valor del semicíclo y máximo negativo en las cercanías del cero.

Con respecto a las corrientes de entrada, puede deducirse a partir de la Figura

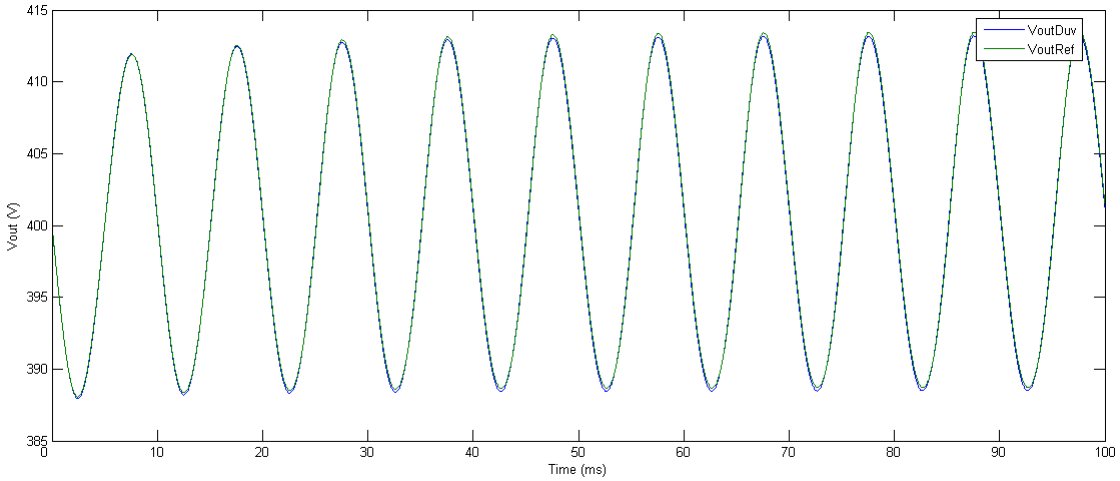


Figura A.13: Tensión de salida - Carga tipo potencia y $V_{in} = VAC$

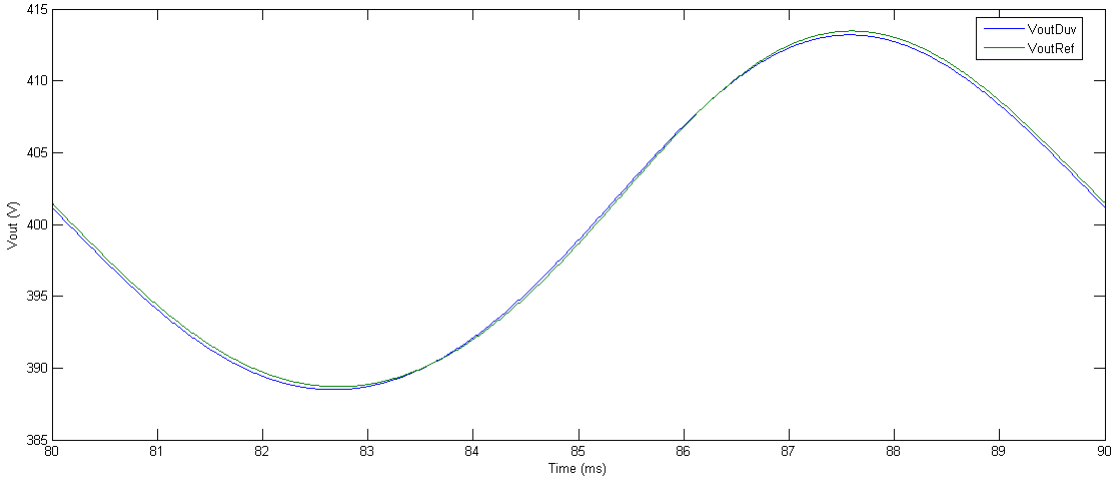


Figura A.14: Tensión de salida detallada entre 80ms y 90ms - Carga tipo potencia y $V_{in} = VAC$

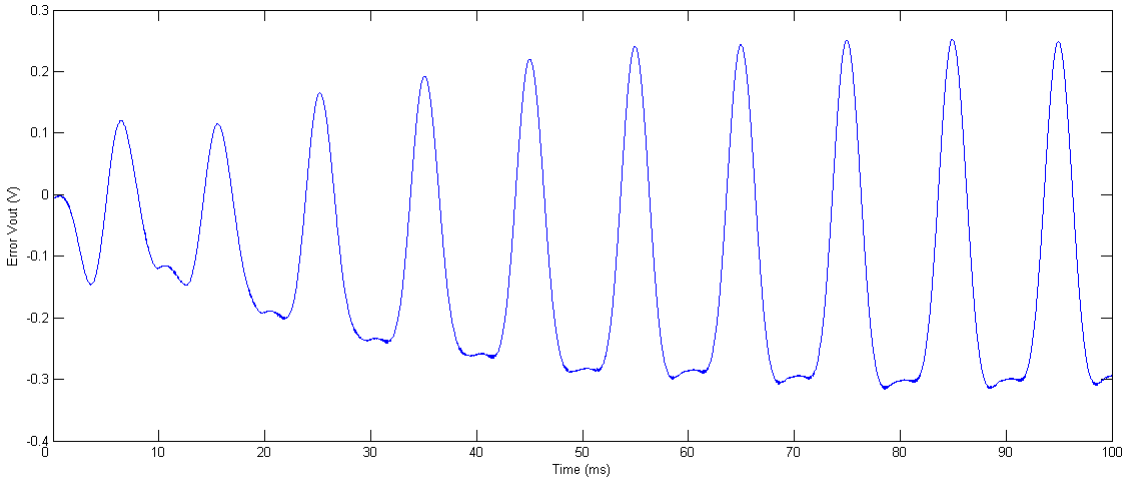


Figura A.15: Error de tensión de salida - Carga tipo potencia y $V_{in} = VAC$

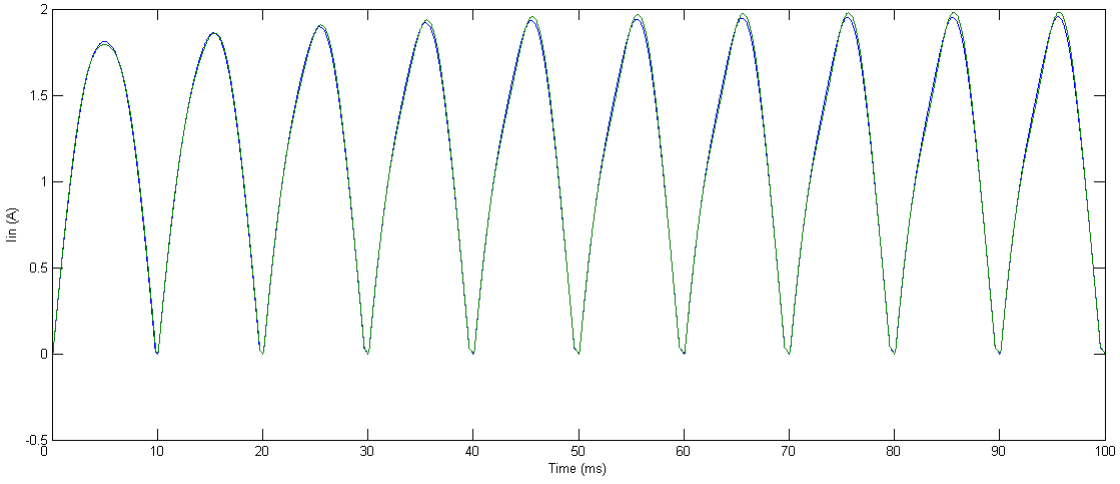


Figura A.16: Corriente de entrada - Carga tipo potencia y $V_{in} = VAC$

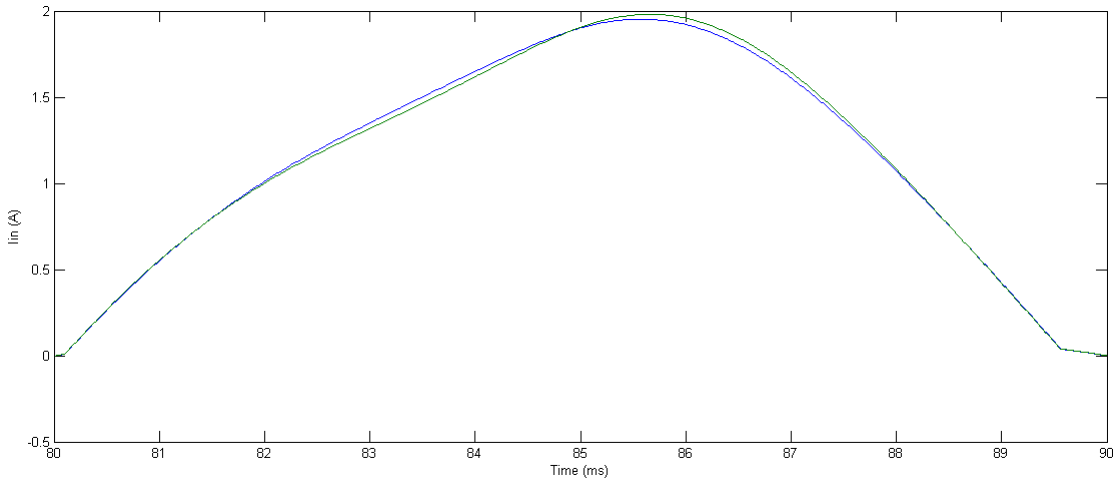


Figura A.17: Error en corriente de entrada detallado entre 80ms y 90ms - Carga tipo potencia y $V_{in} = VAC$

A.16 que siguen el patrón de los semicíclos de tensión alterna, pero el valor máximo lo alcanzan instantes después de la máxima tensión de entrada.

El error en corriente de entrada mostrado en la Figura A.18 no tiene un análisis trivial, pero es posible deducir que la mínima diferencia se produce al mismo tiempo que el máximo y mínimo del semicíclo de tensión alterna.

A.4. Carga tipo corriente y tensión de entrada fluctuante

En la Figura A.20 se han representado los valores de tensión de entrada que se han usado en la prueba. Es posible ver que se trata de una tensión sinusoidal entorno a una tensión de 210 volts.

Es posible ver que, si bien la frecuencia con la que fluctúa, la tensión de entrada es mayor a la producida por inducción, se pueden notar que en la Figura A.21 los puntos de inflexión sobre el comportamiento de la tensión de entrada es provocado por las fluctuaciones.

Puede apreciarse que los errores de tensión de salida responden al mismo patrón de comportamiento que la tenión de entrada, con la diferencia de un defasaje de 2ms.

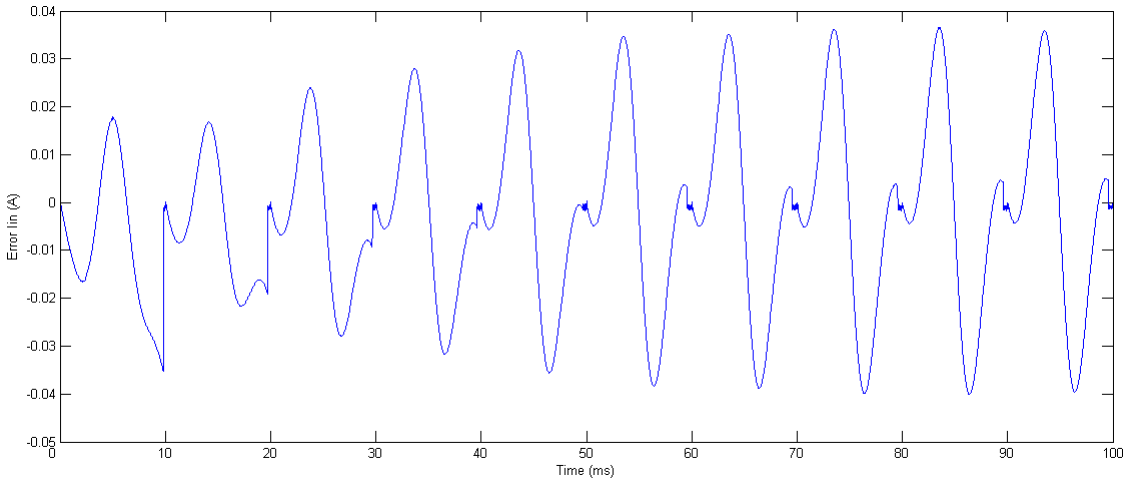


Figura A.18: Error en corriente de entrada - Carga tipo potencia y $V_{in} = VAC$

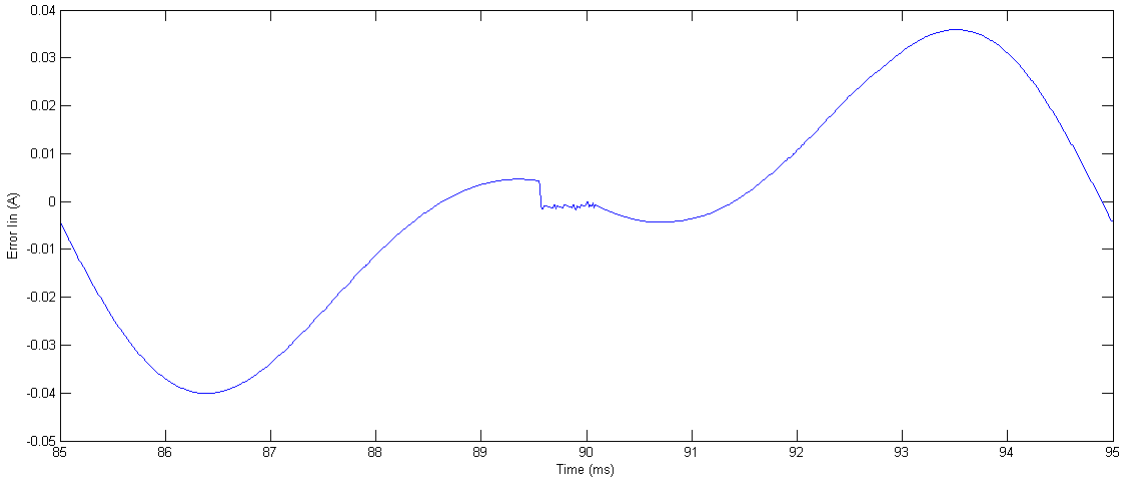


Figura A.19: Error en corriente de entrada detallado entre 85ms y 95ms - Carga tipo potencia y $V_{in} = VAC$

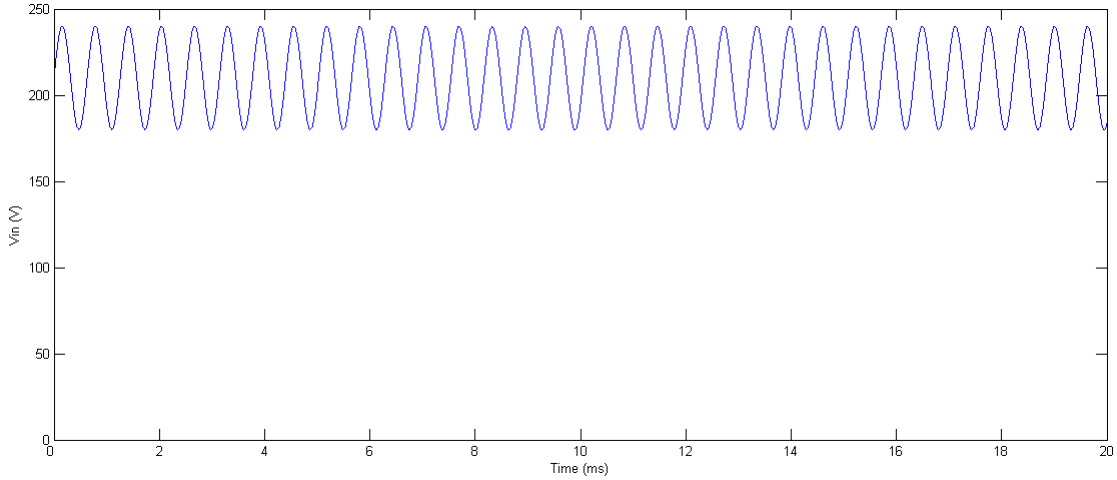


Figura A.20: Patrón de tensión de entrada fluctuante $V_{in} = VF$

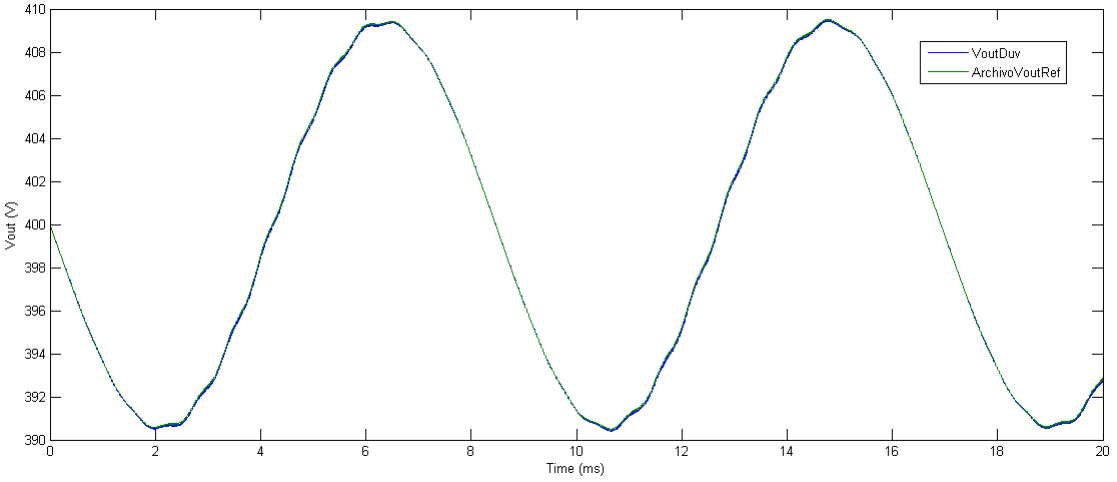


Figura A.21: Tensión de salida - Carga tipo corriente y $V_{in} = VF$

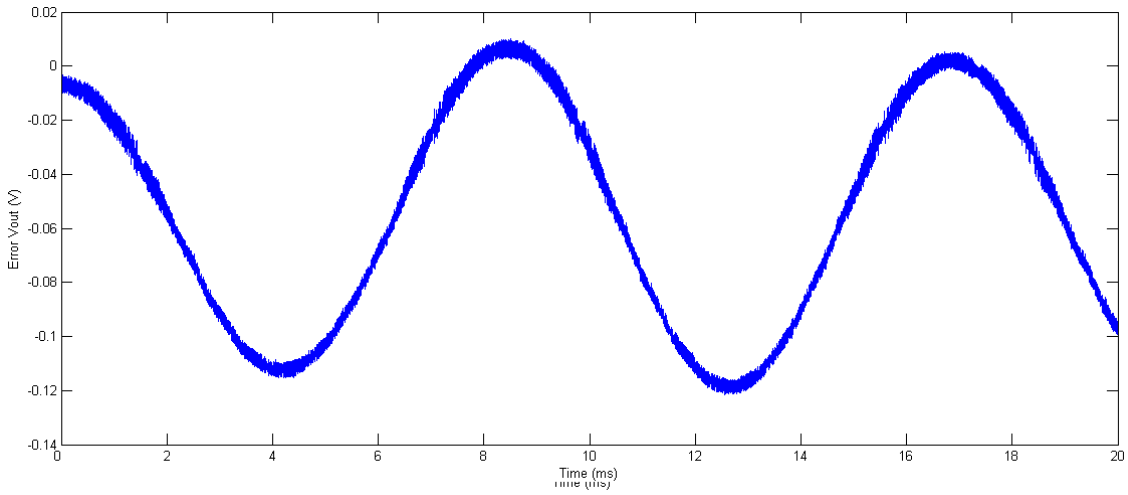


Figura A.22: Error en tensión de salida - Carga tipo corriente y $V_{in} = VF$

En la Figura A.24 es imposible visualizar el comportamiento de la corriente del modelo QXY ya que se encuentra justo detrás de los valores del modelo de referencia. Además puede apreciarse que de ciclo a ciclo se producen fluctuaciones en ambos modelos. Las pequeñas diferencias de corriente de entrada para ambos modelos son mejor apreciadas en la Figura A.25 donde en el peor de los casos no supera los 0.01. Contrastando las Figura A.22 y Figura A.25, puede concluirse que el máximo valor de corriente ocurre cuando el error de tensión es máximo (negativo) y el error de corriente es cero.

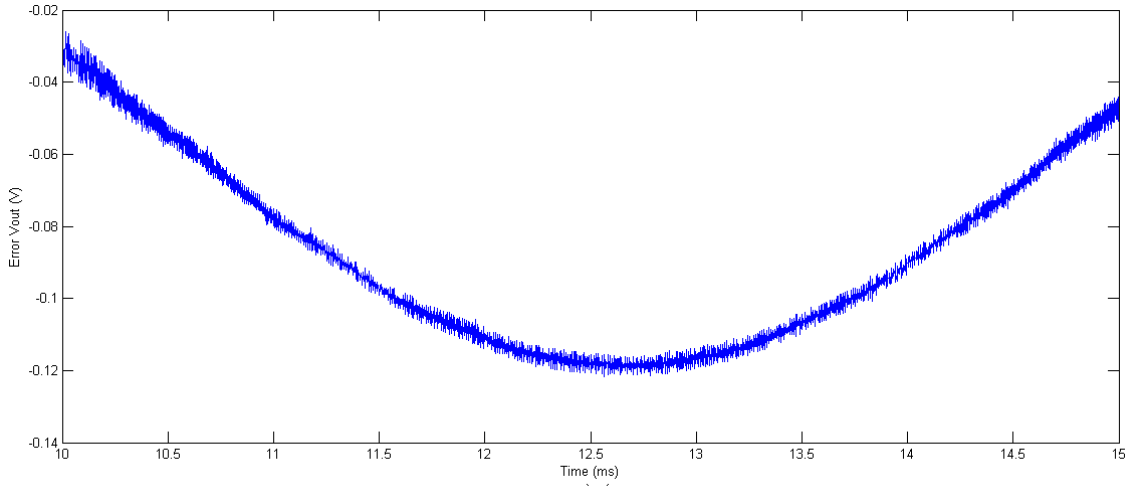


Figura A.23: Error en tensión de salida detallado entre 10 y 15 ms - Carga tipo corriente y $V_{in} = VF$

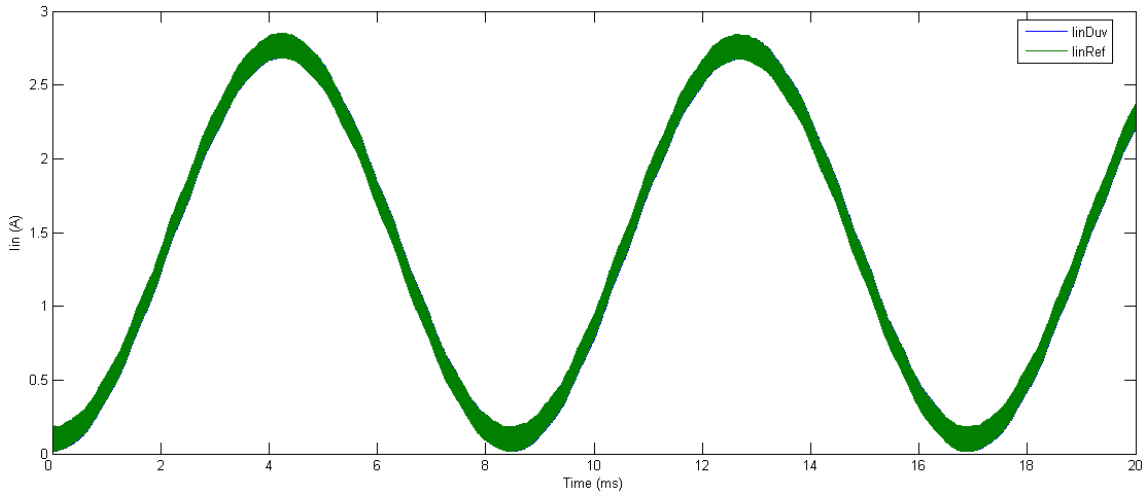


Figura A.24: Corriente de entrada - Carga tipo corriente y $V_{in} = VF$

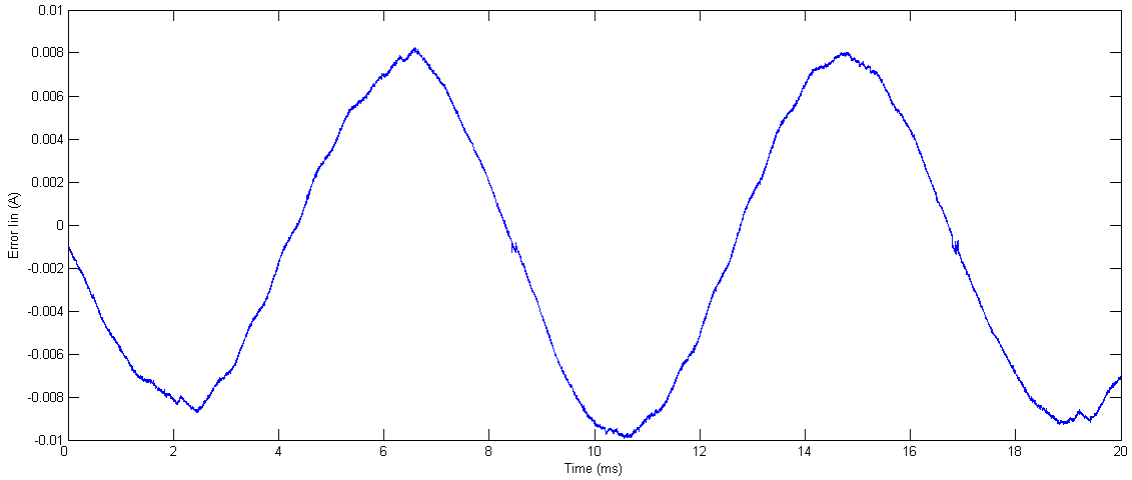


Figura A.25: Error en corriente de entrada - Carga tipo corriente y $V_{in} = VF$

Bibliografía

[1] Uvm (standard universal verification methodology). URL http://www.accellera.org/downloads/standards/uvm.

[2] Standard for property specification language (psl). 2007. doi:10.1109/IEEESTD.2007.4408637. URL http://ieeexplore.ieee.org/servlet/opac?punumber=4408635.

[3] Standard for systemverilog - unified hardware design, specification, and verification language. 2007.

[4] Ieee standard for floating-point arithmetic. 2008.

[5] E.M. Adžić, M.S. Adžić, V.A. Katić, D.P. Marčetić, y N.L. Ćelanovič. Development of high-reliability ev and hev im propulsion drive with ultra-low latency HIL environment. *Industrial Informatics, IEEE Transactions on*, 9(2):630 – 639, 2013. ISSN 1551-3203. doi:10.1109/TII.2012.2222649.

[6] J.R. Bchi. On a decision method in restricted second order arithmetic. En *Int. Congress on Logic, Method, and Philosophy of Science*. Stanford University Press, 1962.

[7] N.H.F Beebe. IEEE 754 floating - point test software. 2010. URL www.math.utah.edu/beebe/software/ieee.

[8] Janick Bergeron, Eduard Cerny, Alan Hunter, y Andy Nightingale. *Verification Methodology Manual for SystemVerilog*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2005. ISBN 0387255389.

[9] Sven Beyer, Christian Jacobi, Daniel Krning, Dirk Leinenbach, y Wolfgang Paul. Putting it all together formal verification of the vamp. *International Journal on Software Tools for Technology Transfer (STTT)*, 8:411–430, 2006. ISSN 1433-2779.

[10] Jayanta Bhadra, Magdy S. Abadir, Sandip Ray, y Li-C. Wang. A survey of hybrid techniques for functional verification. *IEEE Design & Test of Computers*, 24(2):112–122, 2007. ISSN 0740-7475.

[11] D. Boland y G.A. Constantinides. A scalable precision analysis framework. *Multimedia, IEEE Transactions on*, 15(2):242–256, 2013. ISSN 1520-9210. doi: 10.1109/TMM.2012.2231666.

[12] Oswaldo Cadenas y Elas Todorovich. Experiences applying ovm 2.0 to an 8b/10b rtl design. En *5th Southern Conference on Programmable Logic*, págs. 1–8. 2009.

[13] C.I. Castro, M. Strum, y Wang Jiang Chau. Automatic generation of a parameter-domain-based functional input coverage model. En *Test Workshop (LATW), 2010 11th Latin American*, págs. 1 –6. 2010. doi:10.1109/LATW.2010.5550344.

[14] Yirng-An Chen y Randal Bryant. Verification of floating-point adders. En Alan Hu y Moshe Vardi, eds., *Computer Aided Verification*, tomo 1427 de *Lecture Notes in Computer Science*, págs. 488–499. Springer Berlin / Heidelberg, 1998. ISBN 978-3-540-64608-2.

[15] Yuan Chen y V. Dinavahi. Digital hardware emulation of universal machine and universal line models for real-time electromagnetic transient simulation. *Industrial Electronics, IEEE Transactions on*, 59(2):1300 – 1309, 2012. ISSN 0278-0046. doi:10.1109/TIE.2011.2157296.

[16] Alonzo Church. Applicaton of recursive arithmetics to the problem of circuit synthesis. En *Summaries of Talks Presented at The Summer Institute for Symbolic Logic. Communications Research Division, Institute for Defense Analysis.*, pág. 350. 1957.

[17] E. Clarke, S. German, y X. Zhao. Verifying the srt division algorithm using theorem proving techniques. En Rajeev Alur y Thomas Henzinger, eds., *Computer Aided Verification*, tomo 1102 de *Lecture Notes in Computer Science*, págs. 111–122. Springer Berlin / Heidelberg, 1996.

[18] Edmund Clarke. The birth of model checking. En Orna Grumberg y Helmut Veith, eds., *25 Years of Model Checking*, tomo 5000 de *Lecture Notes in Computer Science*, págs. 1–26. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-69849-4.

[19] L.A. Clarke, J. Hassell, y D.J. Richardson. A close look at domain testing. *Software Engineering, IEEE Transactions on*, SE-8(4):380–390, 1982. ISSN 0098-5589. doi:10.1109/TSE.1982.235572.

[20] G.A. Constantinides, N. Nicolici, y A.B. Kinsman. Numerical data representations for FPGA-based scientific computing. *Design Test of Computers, IEEE*, 28(4):8 – 17, 2011. ISSN 0740-7475. doi:10.1109/MDT.2011.48.

[21] Kypros Constantinides, Onur Mutlu, Todd Austin, y Valeria Bertacco. Software-based online detection of hardware defects: Mechanisms, architectural support, and evaluation. págs. 97–108. 2007. ISBN 978-0-7695-3047-5. ISSN 1072-4451. 40th Annual IEEE/AMC International Symposium on Microarchitecture, Chicago, IL, DEC 01-05, 2007.

[22] Mike Cowlishaw. *The decNumber C library*. IBM Corporation, 3.91 ed$^{\underline{\text{ón}}}$., 2008. URL ftp://ftp.math.utah.edu/pub/mathcw/decNumber/decNumber-icu-368.zip.

[23] Robert W. Erikson y Dragan Maksimovic. *Fundamentals of Power Electronics*. Kluwer Academic Publishers, 2004.

[24] Tristan Gingold. Ghdl simulator. URL https://gna.org/projects/ghdl/.

[25] Mark Glasser. *Open Verification Methodology Cookbook*. Springer Dordrecht Heidelberg London New York, 2009.

[26] O. Goñi, A. Sanchez, E. Todorovich, y A. de Castro. Resolution analysis of switching converter modelsfor hardware-in-the-loop. 2013. doi:10.1109/TII.2013.2294327.

[27] O. Goñi y E. Todorovich. Components for coverage-driven verification of floating-point units. En *Programmable Logic (SPL), 2014 IX Southern Conference on*, págs. 1–7. 2014. doi:10.1109/SPL.2014.7002208.

[28] O. Goñi, E. Todorovich, y O. Cadenas. Generic construction of monitors for floating point unit designs. En *Programmable Logic (SPL), 2012 VIII Southern Conference on*, págs. 1–8. 2012. doi:10.1109/SPL.2012.6211776.

[29] Elena Guralnik, Merav Aharoni, Ariel Birnbaum, y Anatoly Koyfman. Simulation based verification of floating point division. *IEEE Transactions on Computers*, 2010.

[30] Elena Guralnik, Merav Aharoni, Ariel J. Birnbaum, y Anatoly Koyfman. Simulation-based verification of floating-point division. *IEEE Trans. Comput.*, 60:176–188, 2011. ISSN 0018-9340.

[31] John Harrison. Formal verification of ia-64 division algorithms. En Mark Aagaard y John Harrison, eds., *Theorem Proving in Higher Order Logics*, tomo 1869 de *Lecture Notes in Computer Science*, págs. 233–251. Springer Berlin / Heidelberg, 2000.

[32] John Hauser. Softfloat. 2010. Technical Report - International Computer Science Institute.

[33] JCGM. *International vocabulary of metrology - Basic and general concepts and associated terms (VIM) Vocabulaire international de metrologie - Concepts fondamentaux et generaux et termes associes (VIM)*. International Organization for Standarization, 3rd ed$^{\underline{\text{ón}}}$., 2008.

[34] J.A.W. Kamp. *Tense Logic and the Theory of Order*. Tesis Doctoral, UCLA, 1968.

[35] M. Kantrowitz y L.M. Noack. I'm done simulating; now what? verification coverage analysis and correctness checking of the decchip 21164 alpha microprocessor. En *Design Automation Conference Proceedings 1996, 33rd*, págs. 325–330. 1996. ISSN 0738-100X. doi:10.1109/DAC.1996.545595.

[36] S. Karimi, P. Poure, y S. Saadate. An HIL-based reconfigurable platform for design, implementation, and verification of electrical system digital controllers. *Industrial Electronics, IEEE Transactions on*, 57(4):1226 –1236, 2010. ISSN 0278-0046. doi:10.1109/TIE.2009.2036644.

[37] Bin Lu, Xin Wu, H. Figueroa, y A. Monti. A low-cost real-time hardware-in-the-loop testing approach of power electronics controls. *Industrial Electronics, IEEE Transactions on*, 54(2):919 – 931, 2007. ISSN 0278-0046. doi:10.1109/TIE.2007.892253.

[38] Ó. Lucía, I. Urriza, L.A. Barragán, D. Navarro, Ó. Jiménez, y J.M. Burdío. Real-time FPGA-based hardware-in-the-loop simulation test bench applied to multiple-output power converters. *Industry Applications, IEEE Transactions on*, 47(2):853 – 860, 2011. ISSN 0093-9994. doi:10.1109/TIA.2010.2102997.

[39] D. Majstorovic, I. Ćelanovič, N.D. Teslic, N. Ćelanovič, y V.A. Katic. Ultralow-latency hardware-in-the-loopin-the-loop platform for rapid validation of power electronics designs. *Industrial Electronics, IEEE Transactions on*, 58(10):4708 – 4716, 2011. ISSN 0278-0046. doi:10.1109/TIE.2011.2112318.

[40] M. Matar y R. Iravani. FPGA implementation of the power electronic converter model for real-time simulation of electromagnetic transients. *Power Delivery, IEEE Transactions on*, 25(2):852 – 860, 2010. ISSN 0885-8977. doi:10.1109/TPWRD.2009.2033603.

[41] Pierre Bricaud Michael Keating. *REUSE METHODOLOGY MANUAL FOR SYSTEM -ON-A-CHIP DESIGNS*. KLUWER ACADEMIC PUBLISHERS, 3rd ed$\underline{\text{ón}}$., 2002.

[42] Mike Mintz y Robert Ekendahl. *Hardware Verification with SystemVerilog An Object-Oriented Framework*. Springer Science+Business Media, LLC, 2007.

[43] Aung Myaing y V. Dinavahi. FPGA-based real-time emulation of power electronic systems with detailed representation of device characteristics. *Industrial Electronics, IEEE Transactions on*, 58(1):358 – 368, 2011. ISSN 0278-0046. doi:10.1109/TIE.2010.2044738.

[44] D. Navarro, Ó. Lucía Gil, L. Barragán, I. Urriza, y Ó. Jiménez. High-level synthesis for accelerating the FPGA implementation of computationally-demanding control algorithms for power converters. *Industrial Informatics, IEEE Transactions on*, 9(3):1371 – 1379, 2013. ISSN 1551-3203. doi:10.1109/TII.2013.2239302.

[45] Oscar Go ni, Martin Vazquez, Gustavo Sutter, y Elias Todorovich. Experiences applying framework-based functional verification to a design for programmable logic. En *Proceedings of the VII IEEE Southern Conference on Programable Logic (SPL 2011)*. Cordoba, Argentina. 2011.

[46] G.G. Parma y V. Dinavahi. Real-time digital hardware simulation of power electronics and drives. *Power Delivery, IEEE Transactions on*, 22(2):1235 – 1246, 2007. ISSN 0885-8977. doi:10.1109/TPWRD.2007.893620.

[47] Per F. V. Hasle Peter hrstrm. *Temporal Logic: From Ancient Ideas to Artificial Intelligence*. Springer, 1995. ISBN 0792335864, 9780792335863.

[48] Amir Pnueli. The temporal logic of programs. En *FOCS*, págs. 46–57. 1977.

[49] Vaughan R. Pratt. Semantical consideration on floyo-hoare logic. *Foundations of Computer Science, 1976., 17th Annual Symposium on*, págs. 109–121, 1976. doi:10.1109/SFCS.1976.27.

[50] Arthur Norman Prior. *Time and Modality*. Oxford University Press, 1957.

[51] Sifakis J. Queille, J.P. Specification and verification of concurrent systems in cesar. En Montanari U. Dezani-Ciancaglini, M., ed., *Programming 1982*, tomo 137, pág. 337351. LNCS, Springer, Heidelberg, 1982.

[52] A. Sanchez, A. de Castro, y J. Garrido. A comparison of simulation and hardware-in-the-loop alternatives for digital control of power converters. *Industrial Informatics, IEEE Transactions on*, 8(99):491 – 500, 2012. ISSN 1551-3203. doi:10.1109/TII.2012.2192281.

[53] Rudolf Usselmann. *Floating Point Unit*, 2005. URL http://www.opencores.org/projects.cgi/web/fpu/overview.

[54] Moshe Y. Vardi. Automata-theoretic model checking revisited. En Cook, B and Podelski, A, ed., *Verification, Model Checking, and Abstract Interpretation, Proceedings*, tomo 4349 de *Lecture Notes in Computer Science*, págs. 137–150. SPRINGER-VERLAG BERLIN, HEIDELBERGER PLATZ 3, D-14197 BERLIN, GERMANY, 2007. ISBN 978-3-540-69735-0. ISSN 0302-9743. doi:10.1007/978-3-540-69738-1_10. URL http://www.springerlink.com/content/y7331395240nm6ll. 8th International Conference on Verification, Model Checking, and Abstract Interpretation, Nice, FRANCE, JAN 14-16, 2007.

[55] Moshe Y. Vardi. From monadic logic to PSL. En Avron, A and Dershowitz, N and Rabinovich, A, ed., *PILLARS OF COMPUTER SCIENCE*, tomo 4800 de *Lecture Notes in Computer Science*, págs. 656–681. 2008. ISBN 978-3-540-78126-4. ISSN 0302-9743. Computation Day Celebrating Boris Trakhtenbrots 85th Birthday, Jaffa, ISRAEL, APR 28, 2006.

[56] Srivatsa Vasudevan. *Effective Functional Verification: Principles and Processes*. Springer, Berlin, 2006.

[57] Martin Vazquez. IEEE754 - 2008 BCD adder/subtractor. Inf. téc., Intia Institute - Universidad Nacional del Centro de la Provincia de Buenos Aires, 2009.

[58] Bruce Wile, John Goss, y Wolfgang Roesner. *Comprehensive functional verification. The complete industry cycle.* Elsevier, 2005.

[59] Stephen Williams. Icarus verilog. URL http://iverilog.icarus.com/.

[60] Myoung-Keun You, Young-Jin Oh, y Gi-Yong Song. Case study: Functional verification of a reconfigurable systolic array using truss. págs. 694 –697. 2009. doi:10.1109/ASICON.2009.5351301.