Como enseñar a generar circuitos en aritmética de punto fijo y punto flotante a partir de Matlab

Santiago T. Pérez Suárez*
Departamento Señales y Comunicaciones, Universidad de Las Palmas de Gran Canaria, España

RESUMEN

Normalmente los algoritmos se implementan en aritmética de punto flotante sobre un computador o sistema microprocesador. Estos sistemas no siempre alcanzan las prestaciones necesarias, en tal caso deben implementarse sobre circuitos específicos. En esta ponencia se parte de los algoritmos descritos en Matlab, usando aritmética de punto flotante; a partir de estos algoritmos, con las metodologías apropiadas, se pueden implementar los circuitos digitales correspondientes, usando aritmética de punto fijo o de punto flotante. Las metodologías se describirán convenientemente, lo mismo que la tendencia de estas herramientas y la necesidad de modificar los planes de estudio.

Palabras clave: prototipado de algoritmos, dispositivos digitales, punto flotante, punto fijo, métodos de diseño, Matlab, Simulink, HDL

1. INTRODUCCIÓN

Actualmente los algoritmos tienen alta complejidad, como sucede por ejemplo en el procesado de señales de vídeo. Esto se traduce en una elevada necesidad de cálculo. Además, el formato de los datos de entrada y salida suele ser de gran tamaño. En estos casos diseñar el circuito puede ser una tarea de muy compleja. Entonces, es necesario recurrir a los últimos métodos de diseño, que son rápidos y flexibles, y permiten la exploración del espacio de soluciones; es decir, las posibles arquitecturas y los diferentes formatos de los datos. El formato de los datos no solo es configurable en las entradas y salidas, sino en los puntos intermedios del sistema. Por otro lado, en los circuitos digitales se ha aumentado de forma exponencial el número de recursos disponibles; por tanto, el problema es configurar los recursos del dispositivo para que soporte el algoritmo.

Esta ponencia se centra en el uso de dispositivos digitales programables por el diseñador (FPGA, *Field Programmable Gate Array*) [1], porque permiten su reconfiguración con la ayuda de un ordenador personal, sin necesidad de ser enviados a una fábrica; esto lo hace idóneo para el desarrollo de prototipos en entornos educativos. Por otro lado, los diseños obtenidos pueden ser transferidos a circuitos integrados de aplicaciones específicas (ASIC, *Application-Specific Integrated Circuit*), en los que los ficheros que describen los diseños deben ser enviados a una fábrica de circuitos integrados; lo que encarece y retrasa el prototipo, pero se consiguen mejores prestaciones de área, velocidad y potencia.

El reto para el diseñador es elegir el método de diseño apropiado. Incluso en diseños sobre FPGA son posibles múltiples métodos de descripción [2]. Con los métodos antiguos se necesita mucho tiempo de diseño; además, el tiempo de simulación y verificación suele exceder el tiempo de diseño. En sistemas complejos es necesario recurrir a métodos avanzados con los que se consigue disminuir estos tiempos. La mayor parte de las implementaciones se han desarrollado en aritmética de punto fijo [1], pero en los últimos años se han desarrollado métodos de diseño que generan circuitos en aritmética de punto flotante [3]. En esta ponencia se hará énfasis en los métodos de diseño que se apoyan en Matlab, y han aparecido en años recientes. Como método debe destacarse el uso de lenguajes de descripción hardware (HDL, *Hardware Description Language*), con los que se describen los sistemas usando ficheros de texto, lo que facilita su edición y modificación; datan de los años ochenta del siglo pasado. Uno de los HDL estándar es VHDL (*Very High Speed Integrates Circuit Hardware Description. Language*) [4], el otro es Verilog [5]. Los métodos avanzados generan la descripción del sistema en VHDL o Verilog, antes de su implementación final. Esta tendencia está consagrada por que permite la integración de diferentes sistemas generados con diferentes herramientas.

*santiago.perez@ulpgc.es; teléfono 34 928451277

2. PROTOTIPADO DESDE MATLAB Y SIMULINK

En la actualidad los métodos avanzados de diseño para FPGA se apoyan en Matlab [6] y Simulink [7]. Estos métodos aprovechan la funcionalidad y prestaciones de este paquete de cálculo matemático. Matlab es un paquete de programas que conforma un entorno de desarrollo integrado. Tienen su propio lenguaje de programación, el lenguaje M, que es interpretado. Además, Matlab tiene altas capacidades de representación gráfica. Las funciones de Matlab se agrupan en *Toolboxes*. Matlab incluye Simulink, que es un paquete gráfico de diseño y simulación. Los bloques de Simulink se agrupan en *Blocksets*, y permiten el diseño en forma de diagrama de bloques. Matlab y Simulink son ampliamente usados en entornos de investigación, académicos e industriales. Debe resaltarse la capacidad de la visualización de las señales en las simulaciones y la facilidad para su posterior análisis numérico.

Los principales suministradores de FPGA facilitan herramientas que funcionan sobre Simulink para la configuración de los dispositivos programables. Una vez que se han instalado aparecen unos *Blocksets* en Simulink propios del fabricante de FPGA. Estos *Blocksets* agrupan bloques configurables en aritmética de punto fijo y de punto flotante. Los principales suministradores de FPGA son Altera [8] y Xilinx [9], conviene resaltar que Altera fue comprada por Intel Corporation en el año 2.015. Debe resaltarse que usar los *Blocksets* de estos suministradores, *DSP Builder* [10] y *System Generator* [11], produce diseños óptimos para Altera y Xilinx respectivamente, pero es difícil la configuración de los tipos de datos desde Matlab. Por supuesto estos diseños solo son válidos para estos fabricantes. Se describirá la generación desde Matlab sin usar los *Blocksets* de estos suministradores, lo que facilita el manejo del formato de los datos y hace que el código HDL generado sea portable a cualquier suministrador de FPGA.

2.1 Sistema usado para mostrar las metodologías

Para mostrar el proceso se usará un sistema simple pero ilustrativo; en particular se implementará la función tangente hiperbólica, llamada *tansig* en la notación de Matlab. Su expresión matemática está dada por la ecuación 1, y su representación se muestra en la figura 2. Esta función es de tipo sigmoidea y se usa habitualmente como función de transferencia en inteligencia artificial; de hecho, es el cuello de botella en la implementación circuital de las redes neuronales artificiales. Debe destacarse que es una función no lineal; incluye exponentes y una división, operaciones de circuitería muy costosa.

$$y(x) = tansig(x) = \frac{e^{+x} - e^{-x}}{e^{+x} + e^{-x}} \tag{1}$$

2.2 Diseño del sistema en punto fijo desde Matlab y Simulink

La tangente hiperbólica no es directamente implementable en aritmética de punto fijo. Por este motivo se recurre a algún método de aproximación. Uno muy común, es el almacenar en una memoria muestras de la función que se desea aproximar. Para implementar este sistema se puede recurrir a un modelo en Simulink donde se incluyen los bloques de los que se puede generar el código en HDL. Estos bloques están incluidos en el *Blockset* llamado *HDL Coder*. Para esto se editó el sistema de la figura 1, donde la aritmética usada es en punto fijo en complemento a dos. El bloque *1-D Lookup Table* (LUT) es el elemento que almacena las muestras que aproximan la función. En el caso de la figura almacena 16 palabras, por lo que el bus de direcciones de la entrada es de 4 bits (ufix4, *4 bits unsigned fixed-point*). En el ejemplo las palabras almacenadas tienen 1 bit de signo, 1 bit para la parte entera y 7 bits fraccionarios, (sfix9_En7, *9 bits signed fixed-point 7 bits fractional*). El formato de la entrada tiene 1 bit de signo, 2 bits para la parte entera y 6 bits fraccionarios, (sfix9_En6, *9 bits signed fixed-point 6 bits fractional*); de esta forma en la entrada los valores representados tienden al rango [-4,+4) cuando crece el número de bits de la parte fraccionaria. El multiplicador y el sumador que siguen, junto con las constantes *G* y *C*, realizan una conversión lineal de la entrada a la dirección de la LUT. En este caso el formato en punto fijo de las señales se ha realizado de forma manual, pero el formato puede fijarse mediante el uso en Matlab de *Fixed-Point Designer* [12]. La utilidad *Fixed-Point Designer* permite manejar tipos de datos y herramientas para el desarrollo de algoritmos en punto fijo, usando código de Matlab, modelos de Simulink o el editor de diagramas de flujo. Esta herramienta propone automáticamente el número de bits y el método de redondeo. Conviene resaltar que los 16 valores de la función son especificados por el diseñador en punto flotante; es Simulink, con el formato de punto fijo, quien establece los valores que se almacenarán en el circuito. En la figura 3 se muestra la función tangente hiperbólica cuando la entrada varía en [-4, +4]. También se muestra la simulación en Simulink de la aproximación conseguida y el error. Debe destacarse que las muestras de la LUT están espaciadas uniformemente en el rango de entrada. El error máximo cometido vale 0,2497.

 ISBN 978-84-09-02374-5

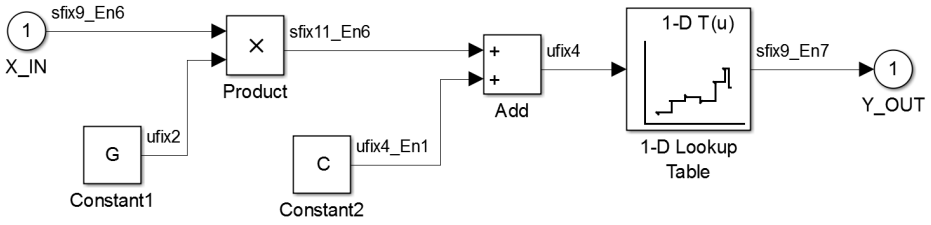

Figura 1. Diagrama de bloques de Simulink que realiza la aproximación.

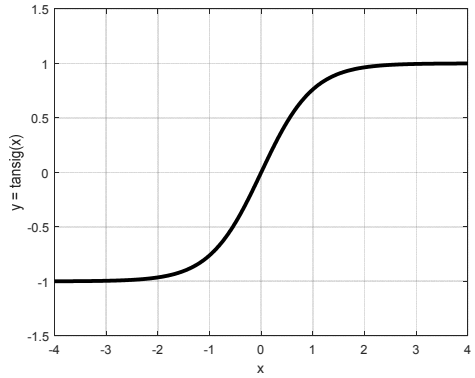

Figura 2. Representación de la función tangente hiperbólica.

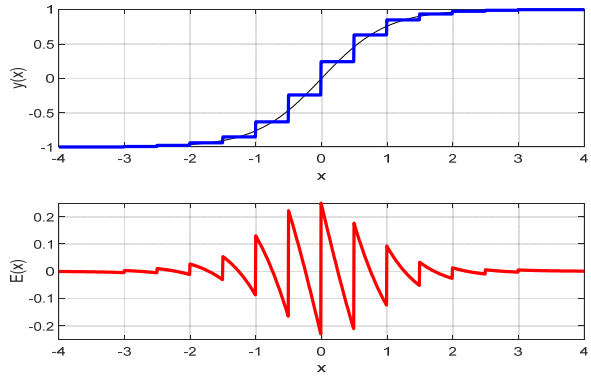

Figura 3. La función tangente hiperbólica, la aproximación conseguida y el error cometido.

Verificado el funcionamiento del modelo de Simulink, se procede a generar el sistema en un HDL en aritmética de punto fijo. Para ello se inicia en el modelo el *HDL Workflow Advisor* [13], mostrado en la figura 4. Con esta utilidad se llegó hasta *Create Project* que es donde se crea la descripción del circuito en el HDL elegido. Es decir, las tareas desde *Perform Synthesis* hasta la configutación del dipositivo se realizará en la herramienta de implementación *Quartus II* [14] de Altera. La utilidad *HDL Workflow Advisor* permite elegir una de las herramientas para sintetizar el circuito de las instaladas en la computadora; también permite elegir el fabricante y el modelo de FPGA. Es posible optar entre herramientas de síntesis de Altera o Xilinx en esta etapa. Con la opción *Set Basic Options* se eligió el lenguaje Verilog. En este caso se optó por el dispositivo EP4CE115F29C7N de la familia Cyclone IV E [15] de Altera.

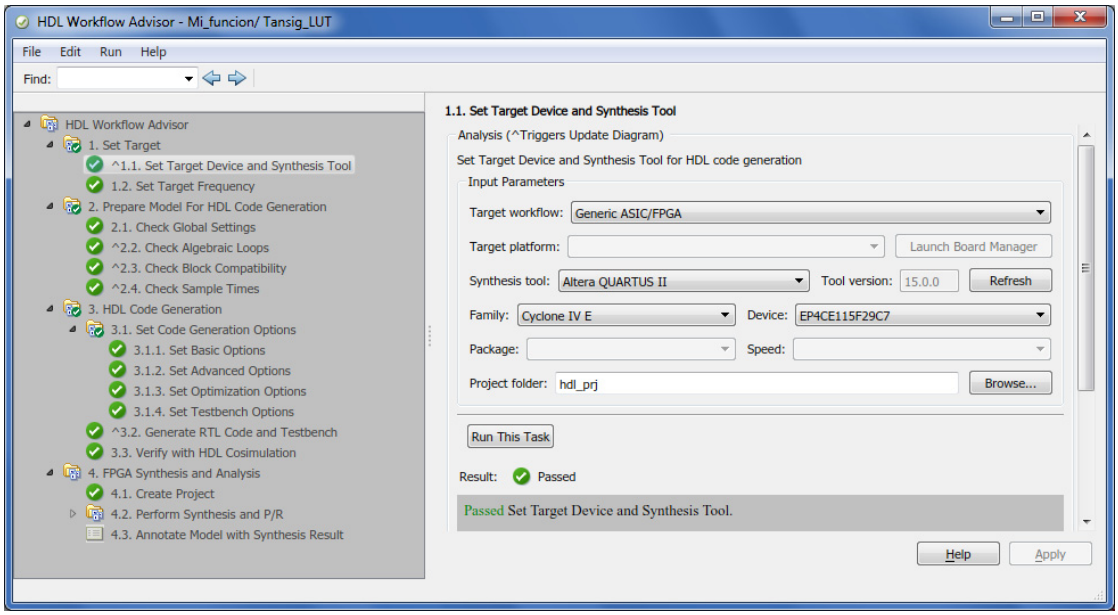

Figura 4. Utilidad *HDL Workflow Advisor* con las tareas finalizadas hasta *Create Project*.

Antes de proseguir con las nuevas técnicas de verificación, y con carácter aclaratorio, se procederá a la implementación del proyecto descrito en Verilog. El proyecto se simuló con *ModelSim-Altera Edition* y con el *Simulation Waveform Editor* incluidos en *Quartus II*. El circuito esquemático obtenido es el de la figura 5. La simulación con *ModelSim* quedó como muestra la figura 6, para una frecuencia de reloj de 25 MHz. Cuando se genera desde Simulink el proyecto en el HDL también se generan las señales de prueba (*testbench*). La de entrada en punto fijo se usa en la simulación del circuito con *ModelSim*.

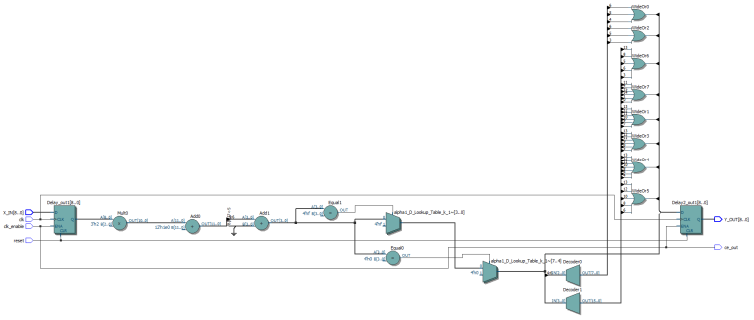

Figura 5. Circuito esquemático obtenido en punto fijo.

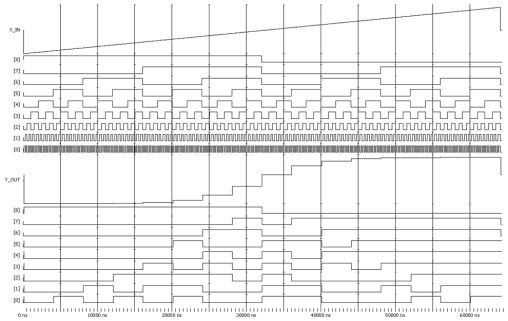

Figura 6. Simulación con *ModelSim*.

En *Quartus II* se obtuvieron los recursos hardware necesarios, evaluados como el número de elementos lógicos, se necesitaron 13 para su implementación. Con el analizador de tiempos de *Quartus II* se obtuvo que la frecuencia máxima de funcionamiento del sistema era de 762,2 MHz. Finalmente, con el analizador de potencia del *Quartus II* se obtuvo una potencia total de 140,44 mW. La estimación de potencia se hizo para una temperatura ambiente de 25º C, sin disipador de calor ni flujo de aire forzado.

2.2.1 La cosimulación del sistema en punto fijo desde Simulink

La cosimulación es una técnica de verificación que permite la simulación del modelo de referencia de Simulink y la simulación del HDL generado. Estas simulaciones se lanzan desde el entorno Simulink con una única orden. Por un lado, el simulador de HDL toma la señal de entrada que se usa en el modelo de referencia. Por otro lado, las salidas pueden compararse en Simulink, por que el simulador de HDL envía sus señales de salida a Simulink. Esta idea se muestra en la figura 7, y permite verificar que el código HDL realiza la misma función que el modelo de referencia; mediante la comparación de las dos salidas y generando una señal de error que debe ser nula, como muestra la figura 9. El sistema para la cosimulación es generado con el *HDL Workflow Advisor*, y se muestra en la figura 8. El simulador de HDL debe iniciarse antes de la cosimulación picando en el bloque *Start Simulator* de la figura 8.

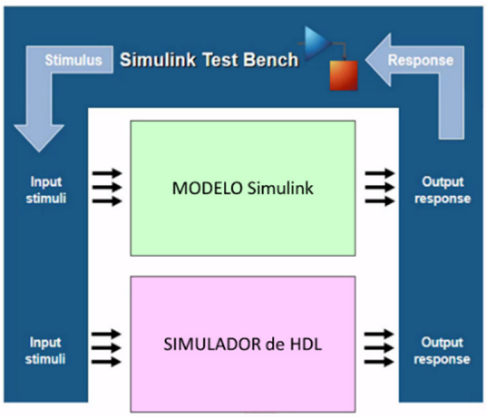

Figura 7. Flujo de la cosimulación del código HDL y del modelo de Simulink.

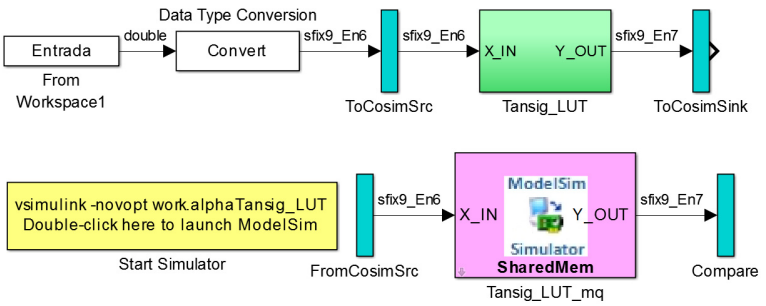

Figura 8. Modelo de Simulink que se genera para ejecutar la cosimulación.

ISBN 978-84-09-02374-5

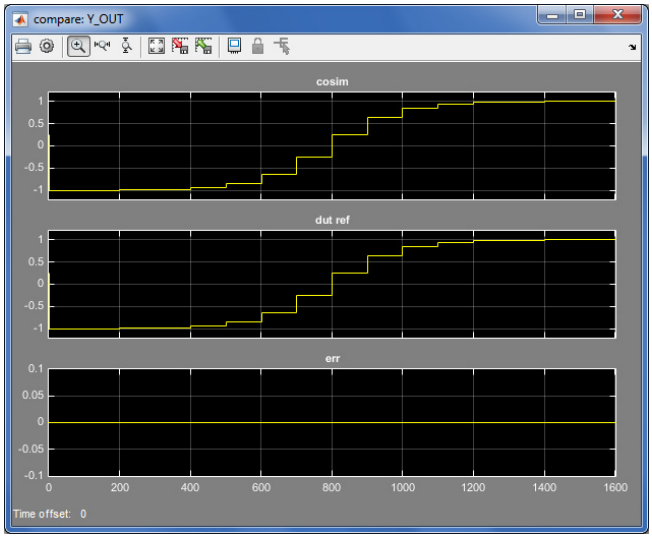

Figura 9. Visualización de las salidas de la cosimulación: superior, del código HDL; intermedia, del modelo de referencia en Simulink; inferior, la señal de error.

2.2.2 "FPGA en el bucle" del sistema en punto fijo desde Simulink

La técnica de verificación de la "FPGA en el bucle" es más conocida por su denominación en inglés *FPGA in the loop* [16]. En realidad es otra cosimulación, donde se simula de forma simultánea el modelo de referencia de Simulink y se compara con la salida obtenida en la FPGA. Primero desde Simulink debe configurarse la FPGA. Después de esto, la simulación del modelo de referencia en Simulink y el funcionamiento de la FPGA se invocan al simular el modelo. La placa usada es la DE2-115 de Terasic [17], que incluye la FPGA de Altera, a ella se envían las señales de entrada que se usan en el modelo de referencia. Por otro lado, las salidas de la FPGA se devuelven a Simulink. Estas ideas se muestran en las figuras 10 y 11, y permite verificar que la FPGA realiza la misma función que el modelo de referencia; mediante la comparación de las dos salidas y generando una señal de error que debe ser nula, de forma análoga a la figura 9. El sistema para *FPGA in the loop* es generado con el *HDL Workflow Advisor*, y se muestra en la figura 11. En el *HDL Workflow Advisor* se elige la placa para crear el sistema en Simulink para *FPGA in the loop*.

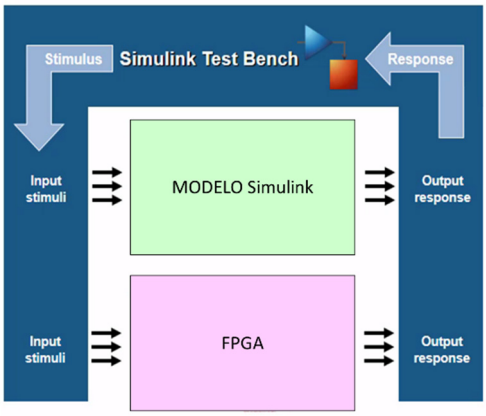

Figura 10. Flujo para ejecutar *FPGA in the loop* desde Simulink.

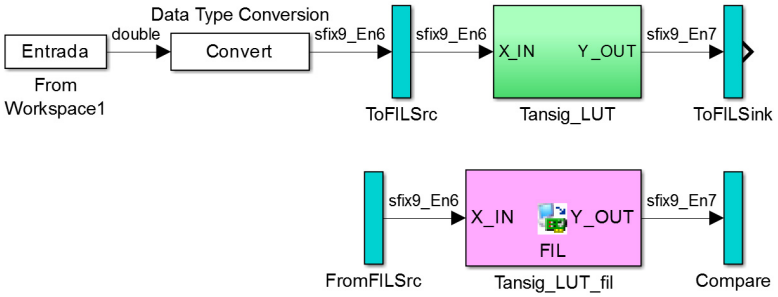

Figura 11. Sistema en Simulink para ejecutar *FPGA in the loop*.

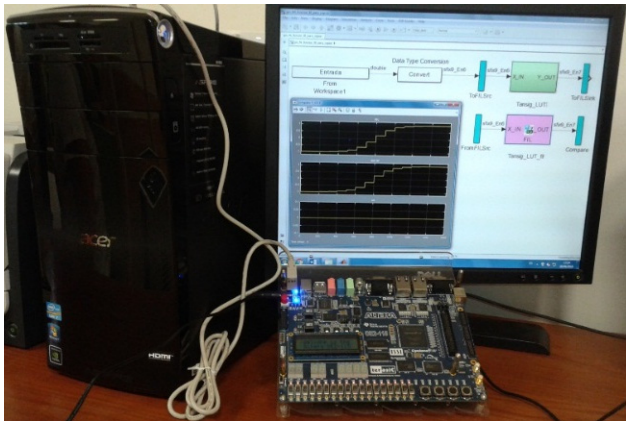

Figura 12. Montaje para invocar *FPGA in the loop*.

2.2.3 Edición de modelos y técnicas de verificación

Las técnicas de diseño presentadas se basan en la edición de modelos en Matlab y en el uso de técnicas de verificación. En la figura 13 se muestra como se integra este flujo [18]. Es posible, como muestra la figura 14, generar código en HDL desde un sistema en Simulink, desde código en Matlab; o de forma híbrida, donde el código M se incluye en un bloque de Simulink. La generación y simulación de los sistemas para estas técnicas de verificación se realizan en Matlab mediante su utilidad *HDL Verifier* [19].

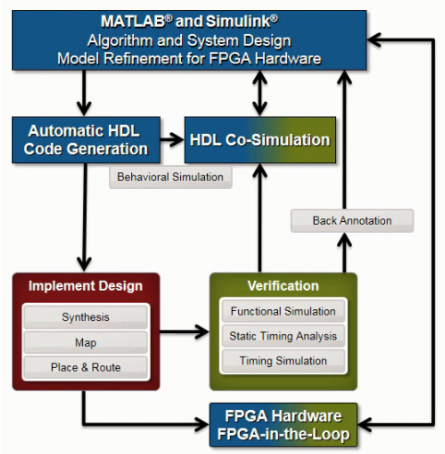

Figura 13. Flujo de diseño de Matlab y Simulink para FPGA basado en modelos y técnicas de verificación.

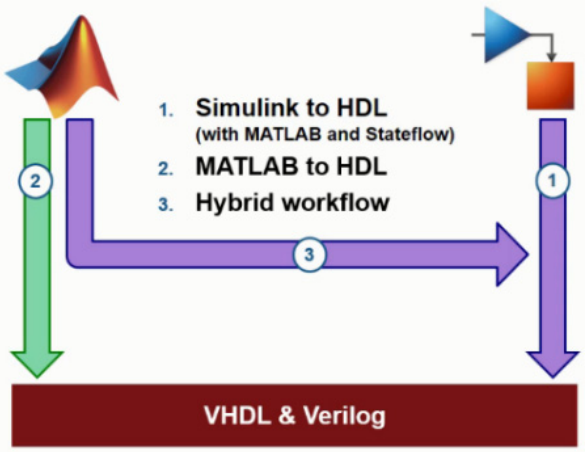

Figura 14. Generación de código HDL desde Simulink, desde Matlab o de forma híbrida.

2.3 Diseño del sistema en punto flotante desde Matlab y Simulink

El suministrador de Matlab indicó que en su versión R2016b incluía el *HDL Floating Point Operations* [20], esto es un *Blockset* en Simulink que permite generar circuitos en aritmética de punto flotante. Los formatos permitidos son en precisión simple o doble, que ocupan 4 y 8 octetos respectivamente. Se permite generar el HDL para librería genéricas, pero solo en formato simple y en lenguaje Verilog; esta es la opción elegida para mostrar el método. También integra librerías específicas de Altera y Xilinx. Las técnicas y método anteriores son aplicables. Solo se mostrará en un solo apartado la información relevante que cambia sustancialmente. Lo primero es que la expresión 1 es editada en un modelo que se muestra en la figura 15 de forma directa. En este caso la representación gráfica de la función y la salida del sistema Simulink se confunden porque el error máximo vale $1{,}2715 \cdot 10^{-07}$. Este tipo de representación aumenta el rango y mantiene el error relativo pequeño; por el contrario, aumenta el área ocupada y la potencia consumida, pero disminuye la frecuencia máxima. El circuito esquemático obtenido se muestra en la figura 16. Se necesitaron 6.743 elementos lógicos para su implementación. Se obtuvo una frecuencia máxima de funcionamiento de 72,05 MHz y una potencia de 189,54 mW.

ISBN 978-84-09-02374-5

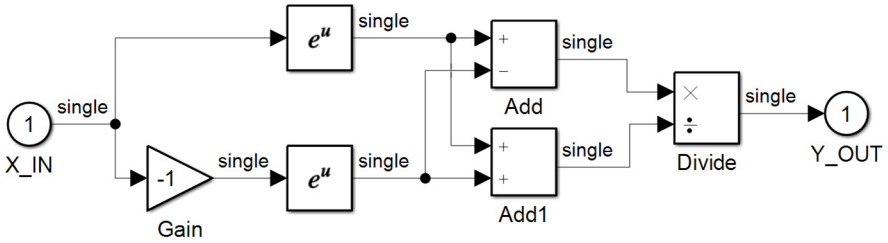

Figura 15. Modelo para la tangente hiperbólica en formato simple de punto flotante.

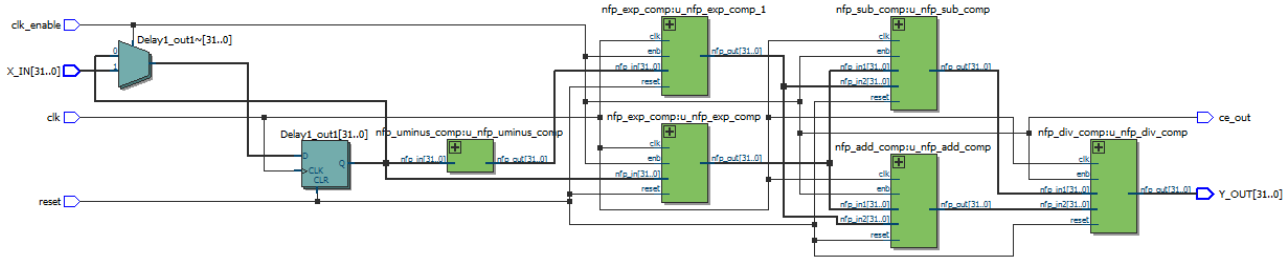

Figura 16. Circuito esquemático obtenido en punto flotante.

2.4 Escenario del diseño

Conviene resaltar el escenario y las utilidades usadas en este desarrollo: sistema operativo Windows 7 Home Premium, Matlab R2018a, Quartus II 15.0, ModelSim PE Student Edition 10.4a, ModelSim-Altera 10.3d; y la placa de desarrollo y educación DE2-115 de Terasic, que incluye el dispositivo EP4CE115F29C7N de la familia Cyclone IV E de Altera. El escenario planteado, respecto a los métodos habituales de diseño, tiene el coste añadido de la licencia de Matlab y sus utilidades. Esta licencia para la versión R2018b, con las mínimas opciones necesarias, se puede conseguir desde unos 104 euros en la versión de estudiante, y puede alcanzar para entidades educativas los 2.200 euros.

3. CONCLUSIONES

El entorno integrado desde Matlab y Simulink permite explorar el espacio de soluciones del diseño. Cuando se diseña un sistema son posibles múltiples arquitecturas y formatos de datos. El sistema debe cumplir con una funcionalidad dada, y en ocasiones cumplir requisitos de área, velocidad y potencia. Se llama área ocupada a los recursos hardware que necesita la implementación del diseño. El método de diseño basado en la edición de modelos y en técnicas de verificación permite la exploración de las posibles soluciones. Si en alguna de las fases de verificación (cosimulación o *FPGA in the loop*) no se alcanzan las expectativas del diseño, se puede rehacer el flujo de diseño de la figura 13. Estas técnicas permiten la generación automática de sistemas en HDL, en punto flotante o punto fijo. La rapidez de las simulaciones, su visualización y análisis, permite verificar la funcionalidad de forma rápida y fiable. En resumen, se acorta enormemente el tiempo de la descripción, simulación y verificación del sistema.

Los planes de estudio actuales en los grados y másteres de la Universidad de Las Palmas de Gran Canaria inciden en los métodos clásicos de diseño, basados sobre todo en la edición directa usando un HDL. No obstante, estos métodos que datan de hace una treintena de años, deben seguir impartiéndose. Los métodos sobre Matlab y Simulink, desarrollados en los últimos años, están relegados a actividades puntuales de investigación.

Por tanto, cabe una profunda reflexión relativa a los métodos de diseño, los métodos avanzados han de incluirse en los planes de estudio, principalmente en master y doctorado. Para finalizar se realiza una propuesta respecto al Máster Universitario en Ingeniería de Telecomunicación de la Universidad de Las Palmas de Gran Canaria [21], donde se dispone de tres asignaturas: Sistemas Integrados, Ingeniería de Sistemas y Desarrollo Hardware-Software de Productos Electrónicos. En ellas podría introducirse el método de diseño descrito para FPGA desde Simulink y Matlab.

Esto es solo una exposición de las posibilidades de los métodos descritos; en cualquier caso, sería necesario la redacción de un tutorial para el alumnado. Para la realización de este eventual tutorial, el sistema debe tener la versión correcta de Matlab y Simulink, e incluir *HDL Coder* y *HDL Verifier*. Por otro lado, se debe incluir una herramienta se síntesis para la implementación final, normalmente de Altera o Xilinx. Finalmente, debe disponerse de simuladores de HDL, tanto

ISBN 978-84-09-02374-5

ejecutables desde Matlab como desde la herramienta de síntesis. Para todo esto debe disponerse de las correspondientes licencias, tarea tediosa por provenir de diferentes corporaciones. Por si fuera poco, todo lo anterior debe funcionar sobre el sistema operativo adecuado, y deben ser versiones compatibles. Pero cuando se dispone de experiencia para configurar una plataforma las ventajas son espectaculares, a pesar del coste económico asociado a la licencia de Matlab.

4. AGRADECIMIENTOS

Se agradece a MathWorks Incorporated los permisos para usar sus gráficos en la redacción de esta ponencia. Igualmente se agradece a Altera Corporation la donación del software usado y sus correspondientes licencias; y la donación de la placa DE2-115, todo ello dentro de su *Intel FPGA University Program*.

REFERENCIAS

[1] Maxfield, C; The Design Warrior's Guide to FPGAs, Elsevier, 2004.
[2] Meeus, W.; Van, K.; Goedemé, T.; Meel, J.; Stroobandt, D.; "An overview of today's high-level synthesis tools", Design Automation of Embedded Systems, vol. 16, nº 3, pp. 31-51, 2012.
[3] IEEE Standard for Floating-Point Arithmetic, http://ieeexplore.ieee.org/servlet/opac?punumber=4610933, enlace permanente.
[4] IEEE Standard VHDL Language Reference Manual, http://ieeexplore.ieee.org/servlet/opac?punumber=4772738, enlace permanente.
[5] IEEE Standard for Verilog Hardware Description Language, http://ieeexplore.ieee.org/servlet/opac?punumber=10779, enlace permanente.
[6] MATrix LABoratory de MathWorks (Matlab), http://www.mathworks.com, (29 de octubre de 2018).
[7] Simulink de MathWorks, http://www.mathworks.com/products/simulink, (29 de octubre de 2018).
[8] Altera Corporation, http://www.altera.com, (29 de octubre de 2018).
[9] Xilinx Corporation, http://www.xilinx.com, (29 de octubre de 2018).
[10] DSP Builder, http://www.altera.com/products/software/products/dsp/dsp-builder.html, (29 de octubre de 2018).
[11] System Generator for DSP, http://www.xilinx.com/products/design-tools/vivado/integration/sysgen.html, (29 de octubre de 2018).
[12] Fixed-Point Designer, http://www.mathworks.es/products/fixed-point-designer, (29 de octubre de 2018).
[13] HDL Workflow Advisor, https://es.mathworks.com/help/hdlcoder/examples/basic-hdl-code-generation-with-the-workflow-advisor.html, (29 de octubre de 2018).
[14] Quartus II de Altera Corporation, https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/download.html, (29 de octubre de 2018).
[15] Familia Cyclone IV E II de Altera Corporation, https://www.intel.com/content/www/us/en/products/programmable/fpga/cyclone-iv.html, (29 de octubre de 2018).
[16] FPGA in the loop, http://es.mathworks.com/help/hdlverifier/ug/fpga-in-the-loop-fil-simulation.html, (29 de octubre de 2018).
[17] Altera DE2-115 Development and Education Board, https://www.intel.com/content/www/us/en/programmable/solutions/partners/partner-profile/terasic-inc-/board/altera-de2-115-development-and-education-board.html, (29 de octubre de 2018).
[18] Generación y Verificación de HDL para FPGAs, ASICs y SoCs desde MATLAB y Simulink, http://es.mathworks.com/videos/hdl-code-generation-and-verification-for-fpgas-asics-and-socs-from-matlab-and-simulink-22850.html?form_seq=conf756&elqsid=1475083337040&potential_use=Education&country_code=ES, (29 de octubre de 2018).
[19] HDL Verifier, http://www.mathworks.es/products/hdl-verifier, (29 de octubre de 2018).
[20] FPGA for DSP applications_Fixed Point Made Easy, https://es.mathworks.com/videos/fpga-for-dsp-applications-fixed-point-made-easy-1495129243550.html, (29 de octubre de 2018).
[21] Máster Universitario en Ingeniería de Telecomunicación de la ULPGC, https://www.eite.ulpgc.es/images/eite/docs/formacion/MUIT-Verifica%20EITE_ANECA2012.pdf, (29 de octubre de 2018).

 ISBN 978-84-09-02374-5