DISEÑO VLSI DE UN SUMADOR DE PUNTO FLOTANTE USANDO LAS HERRAMIENTAS DE ALLIANCE

VLSI DESIGN OF FLOATING POINT ADDER USING ALIANCE TOOLS

Elsa Denise Guerrero Eufracio
Universidad Autónoma de Ciudad Juárez, México
al157918@alumnos.uacj.mx

Jesús Alberto Carreón Rosales
Universidad Autónoma de Ciudad Juárez, México
al157970@alumnos.uacj.mx

Sofía Cordero Márquez
Universidad Autónoma de Ciudad Juárez, México
al168529@alumnos.uacj.mx

Alan Alfonso Cossio Silva
Universidad Autónoma de Ciudad Juárez, México
al157568@alumnos.uacj.mx

Juan Fernando Rodríguez Ramírez
Universidad Autónoma de Ciudad Juárez, México
al157855@alumnos.uacj.mx

Abimael Jiménez Pérez
Universidad Autónoma de Ciudad Juárez, México
abimael.jimenez@uacj.mx

Recepción: 28/octubre/2021 **Aceptación:** 26/enero/2022

Resumen

La suma es la operación clave en los sistemas digitales y el sumador de punto flotante se usa con frecuencia para la suma de números reales porque la representación de punto flotante proporciona un amplio rango dinámico. En este artículo se presenta el diseño VLSI (Very Large Scale Integration) de un circuito sumador de punto flotante. El algoritmo se implementó en lenguaje de descripción de hardware VHDL. Posteriormente se utilizan las herramientas de software libre de Alliance para realizar el proceso de síntesis, con el cual se obtuvo el layout del

circuito. El diseño presentó un consumo de área de 959,250 λ^2, un retardo de 20.8 ns y se utilizaron 4,604 transistores.

Palabras Clave: Área, diseño VLSI, PF, retardo, VHDL.

Abstract

Addition is the key operation in digital systems, and floating-point adder is frequently used for real number addition because floating-point representation provides a large dynamic range. This paper presents the VLSI (Very Large Scale Integration) design of a floating point adder circuit. The algorithm was implemented in hardware description language VHDL. Subsequently, Alliance free software tools were used to perform the synthesis process, obtaining the circuit layout in generic units of lambda. The design presented an area consumption of 959,250 λ^2, a delay of 20.8 ns and 4,604 transistors were used.

***Keywords*:** *Area, delay, floating point, VHDL, VLSI Design.*

1. Introducción

La complejidad computacional de las aplicaciones en ciencia e ingeniería se ha incrementado en los últimos años y es difícil el visualizar una infraestructura científica moderna sin una computación numérica. Muchas de las aplicaciones de la ciencia y la ingeniería utilizan la representación numérica de Punto Flotante (PF) para el cálculo de números reales, ya que proporcionan un amplio rango dinámico. El uso de PF en la informática se remonta a la operación de la primera máquina informática del mundo, la Z3, diseñada por Konrad Zuse, que incluía un cálculo binario de números en PF [Mathis, 2019].

Golberg demostró que la falta de cuidado en los diseños de PF puede generar resultados erróneos [Goldberg, 1991] y; por lo tanto, se requiere una investigación exhaustiva para los números de PF [Hassan, 2018]. Un ejemplo de falla en el diseño debido a cálculos inexactos es la falla del procesador Pentium de Intel. El profesor Thomas R. Nicely estaba trabajando en la suma del recíproco de dos números primos. Desarrolló varios algoritmos y los evaluó en diferentes tipos de procesadores. En 1994, cuando introdujo una máquina basada en el procesador

Pentium de Intel, notó que, para el algoritmo de división, producía un resultado de PF incorrecto. En un principio, Intel negó esa posibilidad, pero otros investigadores reportaron errores similares para diferentes aplicaciones. Después, el ingeniero Tim Coe comprueba que, para algunos casos, el cálculo con números de PF de doble precisión producía un error más grande que para los números de PF de precisión simple. El peor caso de error se generó cuando se utilizó una representación de PF de doble precisión para calcular la proporción de dos números, 4,195,835 y 3,145,727. El resultado correcto era 1.33382044 y el resultado calculado en el procesador Pentium era 1.33373906, que era preciso solo hasta el bit 14 y el error producido en este caso era mayor que su equivalente en precisión simple.

En febrero de 1991 [Behrooz, 2000], la representación inexacta de los números de PF pudo haber sido la causa de un accidente en el que murieron 28 soldados. Un misil Patriot estadounidense no pudo predecir el alcance de un misil Scud iraquí correctamente. El tiempo se expresó como un número entero, pero fue medido en 1/10 por el reloj interno del sistema, el cual es un número de PF. Todos los cálculos se realizaron en un registro de punto fijo de 24 bits; por lo tanto, los números después de 24 bits se truncaron, lo que dio lugar a un error de truncamiento no significativo. Este pequeño error de truncamiento calculado en el misil Patriot se convirtió en uno significativo cuando la batería del misil se utilizó de forma continua durante más de 100 horas.

Otro incidente similar ocurrió en junio de 1996 [Behrooz, 2000], cuando el cohete Ariane sin tripulación explotó 30 segundos después de su despegue debido a la conversión inexacta de números de PF. El proyecto, incluyendo el cohete y su carga, costó aproximadamente 500 millones de dólares, pero afortunadamente esta vez no se perdieron vidas. El sistema de cálculo fue incapaz de encontrar la conversión precisa de la velocidad del cohete representada por un número de PF de 64 bits en un entero con signo de 16 bits, ya que la velocidad resultante (en representación de entero con signo de 16 bits) era mayor a 32,767 (el mayor número representado por un entero con signo de 16 bits). Estas fallas de diseño indican que las operaciones con PF son muy importantes para diversas aplicaciones (procesamiento de imágenes, sistemas mecatrónicos, vehículos aeroespaciales,

drones, etc.) y necesitan realizar cálculos muy precisos para proporcionar el resultado correcto [Joldes, 2020].

Sumadores de punto flotante

Este artículo se enfoca en la implementación VLSI de Sumadores de Punto Flotante (SPF), ya que casi la mitad de las operaciones de PF están dominadas por operaciones de suma y resta [Oberman, 1997], como se muestra en la figura 1. Por ejemplo, el 60% de los algoritmos de procesamiento de señales requieren operación de suma [Pappalardo, 2004].

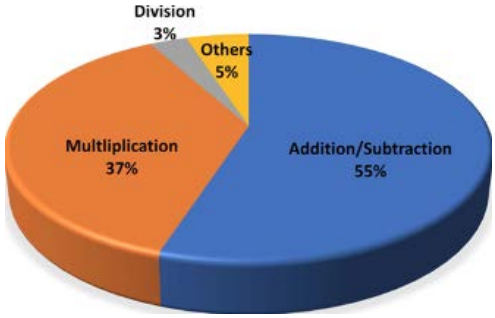


Figura 1 Distribución de instrucción de punto flotante [Oberman, 1997].

Varias patentes se ocupan del diseño e implementación de SPF [Haener, 2020]; [Yamada, 1997]; [Gorshtein, 1998]; [Kawaguchi, 1999]; [Iourcha, 2001]; [Resnick, 2003]. Los SPF tiene varias operaciones de latencia variable, la cual debe optimizarse [Seidel, 2003]; [Pangal, 2005]; ya sea determinando el exponente más pequeño que se restará del exponente más grande [Lutz, 2008] o usando un algoritmo de ruta dual [Nystad, 2015]; [Langhammer, 2018]; [Quinnell, 2018]; [Langhammer, 2019]; [Stiles, 1998]; [Eisen, 1998]; [Oberman, 2000] o considerando la operación de resta por separado [Nakayama, 1993]. Se proponen varias técnicas de diseño para mejorar el rendimiento de los SPF y aumentar su velocidad [Govindu, 2004]; [Malik, 2005]; [Karlstrom, 2006]; [Akkas, 2008]; [Tao, 2012], reducir el consumo de área en silicio [Quinnell, 2018], tratar con números desnormalizados [Ehiliar, 2014] y lógica de redondeo [Nannarelli, 2019] e implementar SPF en FPGAs [Hassan, 2018], [Villalba, 2018].

En 2004 John Thompson presentó un SPF decimal de 5 etapas que cumplía con el borrador de la revisión actual del estándar IEEE-754 [Thompson, 2004]. El sumador soporta operaciones con operandos de PF decimal de 64 bits (16 dígitos). Las pruebas de síntesis iniciales y la evaluación se llevaron a cabo utilizando el compilador de diseño de Synopsys y la librería de celdas estándar CMOS Gflxp de 0.11 micras. En 2005, Chi Huang presentó un SPF de doble precisión de alta velocidad que utilizaba macro-módulos diseñados a la medida. El número total de transistores de este sistema fue de 37,977 [Chi Huang, 2005]. Todos los resultados muestran que se hace un esfuerzo para reducir la latencia y el área e incrementar la resolución y la velocidad a través de nuevos algoritmos.

Formato de los números de punto flotante

La implementación de un SPF es más complicada que la de un sumador de enteros, ya que los números de PF no pueden realizar la suma/resta sin algunos pasos preliminares esenciales [Xu, 2011]. Los números de PF se representan utilizando el formato IEEE754 desarrollado por los Comités Técnicos de las Sociedades IEEE [Coonen, 1980]; [IEEE Computer Society, 2019], y es ampliamente aceptado por académicos e industrias.

En la figura 2 se muestra una representación estándar de PF con el bit de signo (s), el exponente (e) y el significando.

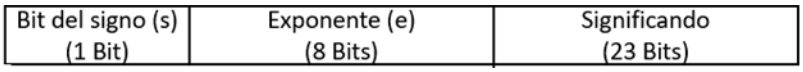


Figura 2 Formato de precisión simple (32 bits).

Hay dos representaciones principales de PF definidas en el formato IEEE-754. Precisión simple y doble, como se muestra en la figura 3. Un número de PF de precisión simple tiene una longitud de 32 bits: el bit más significativo (MSB, Most Significant Bit) es el bit de signo, los siguientes ocho bits corresponden al exponente y los 23 bits finales corresponden al significando. Un número de PF de doble precisión tiene 64 bits. El MSB representa el bit de signo, seguido de un exponente de 11 bits y un significando de 52 bits.

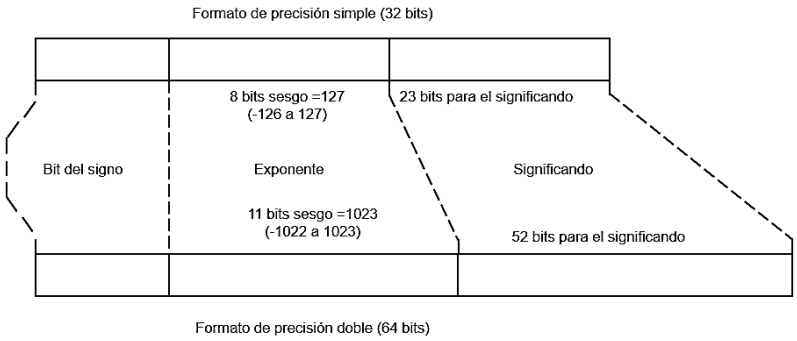


Figura 3 Comparación entre los formatos de precisión simple y doble.

2. Métodos

La suma de PF es un proceso más complejo en comparación con la multiplicación. El proceso implica la comparación de los exponentes, la alineación de la mantisa y los métodos de normalización. Todo esto requiere de un mayor tiempo de procesamiento. Este trabajo se enfoca en el diseño VLSI de un SPF, utilizando herramientas de diseño VLSI libres para analizar dos de los principales parámetros de diseño: área y retardo. En esta sección se describe la metodología de diseño utilizada.

Algoritmo del sumador de punto flotante

Con el mismo número de bits en formato de PF, el rango es mucho mayor que con el formato de enteros con signo. Aunque VHDL tiene un tipo de datos de PF incorporado, es demasiado complejo para sintetizarlo automáticamente. Por ello, es necesario desarrollar módulos de hardware específicos que implementen operaciones con números de PF.

En este algoritmo se utiliza un formato simplificado de 13 bits para los números de PF y no se considera el error de redondeo. La representación consiste en 1 bit de signo (s) que indica el signo del número (1 para negativo), un exponente de 4 bits y un significando (o mantisa M) de 8 bits que representa la fracción. En este formato el valor de una fracción de PF es $(-1)^s * f * 2^e$. $f * 2^e$ es la magnitud del número y $(-1)^s$ es solo una manera formal de establecer que $s = 1$ implica un número negativo. Como el bit de signo está separado del resto del número, la representación de PF se puede considerar como una variación del formato de signo y magnitud.

También se consideran las siguientes aproximaciones:

- Tanto el exponente como el significando están en el formato sin signo.
- La representación debe de ser normalizada o cero. La representación normalizada significa que el MSB del significativo debe ser 1. Si la magnitud del resultado es menor que la magnitud más pequeña no cero normalizada, $0.10000000 * 2^{0000}$, se debe convertir a cero.

El diseño del SPF sigue el método manual de la suma de números en notación científica. Este proceso se puede explicar mejor mediante los ejemplos de la tabla 1. Asumimos que los anchos del exponente y el bit más significativo son de 1 y 2 dígitos, respectivamente, En la tabla 1 se utiliza el formato decimal para mayor claridad y los cálculos se representan en las siguientes 4 etapas:

- Clasificación: se coloca el número de mayor magnitud en la parte superior y el número con la menor magnitud en la parte inferior (llamamos a los números ordenados "big number" y "small number").

Tabla 1 Ejemplos de sumas de PF en formato decimal.

	Ejemplo	Orden	Alineación	Suma/Resta	Normalización
Caso 1	-0.25E2	+3.65E1	+0.36E2	+0.36E2	+0.36E2
	+3.65E1	-0.25E2	-0.25E2	-0.25E2	-0.25E2
				-0.11E2	**-0.11E2**
Caso 2	-0.74E2	+0.775E2	+0.775E2	+0.775E2	+0.775E2
	+0.775E2	-0.74E2	-0.740E2	-0.740E2	-0.740E2
				+0.035E2	**+0.35E1**
Caso 3	-0.636E1	+0.644E1	+0.644E1	+0.644E1	+0.644E1
	+0.644E1	-0.636E1	-0.636E1	-0.636E1	-0.636E1
				+0.008E1	**+0.00E0**
Caso 4	-0.9E1	-0.9E1	-0.900E1	-0.900E1	-0.900E1
	-0.875E1	-0.875E1	-0.875E1	-0.875E1	-0.875E1
				+1.775E1	**+0.17E2**

- Alineación: se alinean los dos números para que tengan el mismo exponente. Esto se puede realizar ajustando el exponente del número pequeño para que coincida con el exponente del número grande. El significando del número pequeño tiene que desplazarse a la derecha, de acuerdo con la diferencia de los exponentes.

- Suma/resta: se suman o restan los significandos de los dos números alineados.
- Normalización: ajusta el resultado al formato normalizado. Se pueden presentar 3 casos:
    a. Después de una resta, el resultado puede contener ceros enfrente.
    b. Después de una resta, el resultado puede ser demasiado pequeño para normalizarse; por lo tanto, debe convertirse a cero.
    c. Después de una suma, el resultado puede generar un bit de acarreo.

En nuestro diseño del SPF binario utilizamos un algoritmo similar. Para simplificar la implementación, ignoramos el redondeo. Durante las etapas de alineación y normalización, los bits menos significativos (LSB, Least Significant Bit) del significando se descartarán cuando se desplacen. El diseño se divide en cuatro etapas, cada una de las cuales corresponde a una etapa del algoritmo como se muestra en la figura 4. En la primera etapa el circuito compara las magnitudes y enruta el número grande a las señales $signb$, $expb$ y $fracb$ y el número menor a las señales $signs$, $exps$ y $fracs$. La comparación se realiza entre $exp1$ y $frac1$ y, $exp2$ y $frac2$. Esto implica que primero se comparan los exponentes. Si son iguales se comparan los significados.

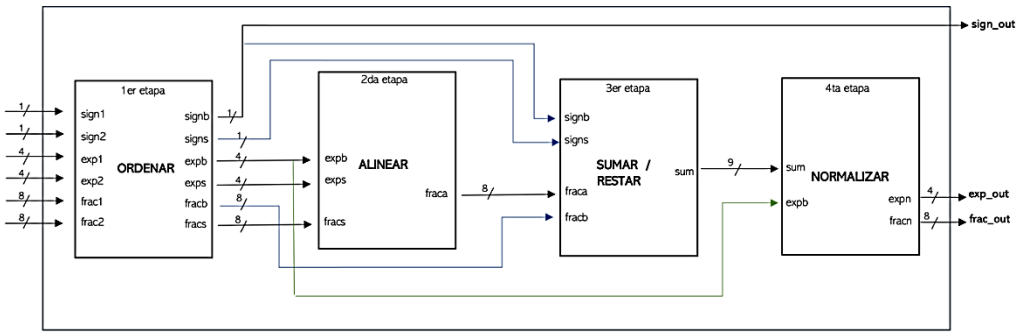


Figura 4 Diseño del diagrama de bloques del SPF propuesto.

En la segunda etapa se realiza la alineación. Primero se calcula la diferencia entre los dos exponentes ($expb - exps$) y después se desplaza el significado, $fracs$, a la derecha por esta cantidad. El significado alineado se etiqueta como $fraca$.

La tercera etapa realiza la suma de signo y magnitud, los operandos se extienden 1 bit para considerar el bit de acarreo.

La cuarta etapa realiza la normalización, que ajusta el resultado para que la salida final se ajuste al formato normalizado. La normalización del circuito está construida en tres fases. La primera cuenta el número de ceros a la izquierda. La segunda desplaza los significados a la izquierda por la cantidad indicada en el conteo de los ceros. La tercera comprueba las condiciones de acarreo de salida y cero.

Síntesis lógica y física con Alliance

En el proceso de síntesis se identifican dos etapas: síntesis lógica y síntesis física. En la primera se declara el circuito digital del SPF en VHDL para obtener una representación del circuito con arreglos de compuertas lógicas y, finalmente, se obtiene un circuito estructural con celdas estándar de la librería $sxlib$ de las herramientas de Alliance. La segunda inicia con el circuito obtenido a nivel estructural para después colocar cada celda estándar con los transistores correspondientes, el enrutamiento y la colocación de puertos de entrada / salida, obteniendo el layout final.

Se sintetizó el circuito del SPF con 8 bits para las mantisas de los números, en lugar de 24, para evitar problemas en el proceso de síntesis (los diseños en Alliance están limitados en el número de compuertas lógicas). La síntesis lógica inició con la traducción del comportamiento de cada bloque en lenguaje VHDL, mediante la herramienta VASY, a uno que puede ser interpretado por las herramientas de Alliance. Después, se realizó una simulación del comportamiento lógico del circuito con la herramienta ASIMUT y un archivo de patrones de entrada. Posteriormente, con la herramienta BOOM se minimizaron las expresiones booleanas del circuito. La herramienta BOOG convirtió la descripción de hardware a su equivalente estructural, utilizando las celdas estándar de Alliance. Después se redujeron las capacitancias y se introdujeron buffers con la herramienta LOON. En este punto, se instanciaron los diferentes bloques del sistema, utilizando la librería GENLIB (desarrollada en lenguaje C) y se inició la etapa de síntesis física con la colocación de las celdas estándar con la herramienta OCP. Después se realizó la interconexión

de las celdas, con dos niveles de metal, utilizando la herramienta NERO. Finalmente, con la herramienta COUGAR se obtuvo el layout preliminar para la fabricación del circuito integrado en unidades genéricas de lambda (λ).

3. Resultados

En la figura 5 se observan los resultados del diseño propuesto. Cada resultado representa un caso especial de la tabla 1 y se comprobaron a través de simulaciones con la herramienta ASIMUT a través de un archivo de patrones de prueba.

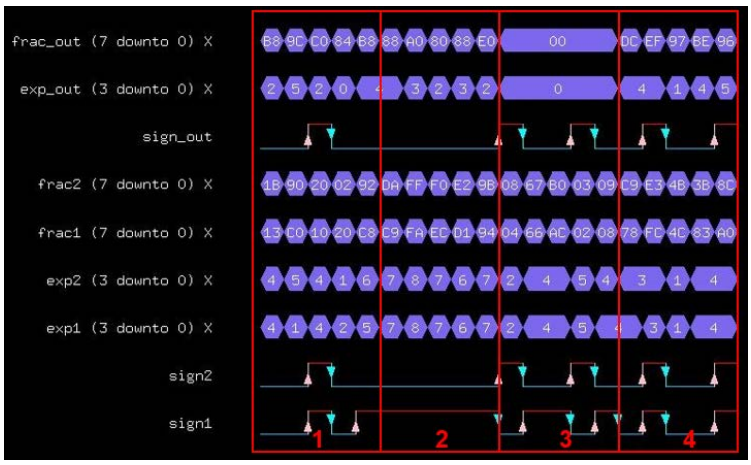
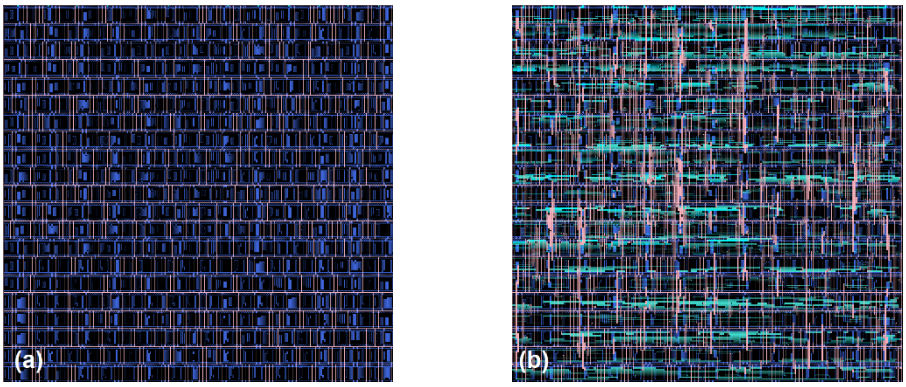


Figura 5 Resultados de simulación del diseño con ASIMUT de los 4 casos de la tabla 1.

El resultado de simulación se visualizó con la herramienta XPAT. En el caso 1 de la tabla 1 se tiene la suma de dos números de PF. En el caso 2 se suman dos números con exponentes de signos opuestos que tienen como resultado un valor con ceros adelante. En el caso 3, ver tabla 1, se obtiene un resultado con valor muy pequeño para poder normalizarlo por lo que se sustituye con 0. Finalmente, en el caso 4, ver tabla 1, se observa la situación en la cual el resultado de la suma tiene un bit de acarreo. Como resultado del proceso de síntesis física en la figura 6a se observa el layout obtenido después de la colocación de celdas estándar generada por la herramienta OCP y en la figura 6b se observa el layout final del core, ruteado con dos capas de metal, del circuito integrado del SPF. El layout fue generado por la herramienta NERO en unidades de λ.



a) Layout obtenido por la herramienta OCP. b) Layout generado por la herramienta NERO.

Figura 6 SPF, colocación de celdas con OCP y ruteado con dos niveles de metal con Nero.

Durante el proceso de síntesis física se obtuvieron los siguientes parámetros de diseño: ruta crítica (retardo) de 20.8 ns, área 959,250 λ^2, 4,604 transistores y 1151 celdas lógicas (4 transistores por celda).

4. Discusión

En la figura 4 se observa que la arquitectura del circuito SPF es completamente modular y el diseño se puede ampliar fácilmente a un mayor número de bits para obtener versiones en formatos de precisión simple o doble. Sin embargo, esto estará limitado por la capacidad que tengan las herramientas OCP y NERO para colocar y rutear una mayor cantidad de celdas estándar. Es por lo que en este trabajo se diseñó un circuito SPF simplificado con números de PF de 13 bits y no se consideró el error de redondeo.

A través de los resultados de simulación de la figura 5 se observa que las 4 etapas (clasificación, alineación, suma/resta y normalización) del circuito del SPF funcionan correctamente. El archivo de patrones incluyó diferentes casos además de los 4 analizados en la tabla 1.

Con las herramientas de síntesis de Alliance se logró obtener el layout del core del circuito integrado. Para poder fabricar este circuito primero es necesario escalar el layout a algún nodo de tecnología CMOS (por ejemplo, tecnología de 0.35 o 0.18 micras y; posteriormente, agregar los pads de entrada y salida correspondientes a la tecnología de fabricación seleccionada.

5. Conclusiones

Se diseñó un circuito integrado VLSI de un sumador de punto flotante con herramientas de software libre de Alliance. El diseño se realizó utilizando un esquema modular, lo que lo hace escalable y portable. Cada módulo fue validado de forma individual, así como su desempeño global en el circuito. Los resultados de simulación corroboran que la implementación del algoritmo de la suma de punto flotante es correcta. Fue necesario implementar una versión simplificada de 13 bits para que las herramientas de Alliance pudieran realizar el proceso de síntesis lógica y física correctamente. El layout del circuito integrado tuvo un área de 152,150 λ^2 y se utilizaron 4,950 transistores. Este diseño podría ser fabricado en alguna tecnología CMOS (0.35 o 0.18 micras). El objetivo principal de este trabajo fue desarrollar un recurso de diseño para que los diseñadores implementen un sumadores de punto flotante en FPGAs o en circuitos integrados. Se han reportado muy pocos trabajos de este tipo y consideramos que será de gran ayuda en la implementación y el diseño personalizados de sumadores de punto flotante.

6. Bibliografía y Referencias

[1] Akkas, A. Dual-mode floating-point adder architectures. J. Syst. Arch. 54, 1129–1142, 2008.

[2] Behrooz, P. Computer Arithmetic: Algorithms and Hardware Designs; Oxford University Press: New York, NY, USA, Volume 19, pp. 512583–512585, 2000.

[3] Chi Huang, Xinyu Wu, Junmei Lai, Chengshou Sun and Gan Li., (2005). A Design of High Speed Double Precision Floating Point Adder Using Macro Modules. Design Automation Conference, Proceedings of the ASP-DAC. Vol2, pp D11-D12. Asia and South Pacific DOI:10.1109/ASPDAC.

[4] Coonen, J. T. Special Feature an Implementation Guide to a Proposed Standard for Floating-PointArithmetic. Computer 1980, 13, 68–79, 1980.

[5] Ehliar, A. Area efficient floating-point adder and multiplier with IEEE-754 compatible semantics. In Proceedings of the 2014 International Conference on Field-Programmable Technology (FPT), Shanghai, China. pp. 131–138, 10–12, December 2014.

[6] Eisen, L.E., Elliott, T.A., Golla, R.T., Olson, C.H., Method and System for Performing a High Speed Floating Point Add Operation. U.S. Patent No. 5,790,445, 4 August 1998.

[7] Goldberg, D., What every computer scientist should know about floating-point arithmetic. ACM Comput. Surv., 23, 5–48. (CSUR) 1991.

[8] Gorshtein, V.Y., Grushin, A.I., Shevtsov, S.R., Floating Point Addition Methods and Apparatus. U.S. Patent 5, 808, 926, 15 September 1998.

[9] Govindu, G., Zhuo, L., Choi, S., Prasanna, V. Analysis of high-performance floating-point arithmetic on FPGAs. In Proceedings of the 18th International Parallel and Distributed Processing Symposium, Santa Fe, NM, USA, 26–30, p. 149. April 2004.

[10] Haener, T., Roetteler, M., Svore, K., Quantum Circuit Libraries for Floating-Point Arithmetic. U.S. Patent No. 10,699,209, 30 June 2020.

[11] Hassan, H. S.; Ismail, S. M., CLA based Floating-point adder suitable for chaotic generators on FPGA. In Proceedings of the 2018 30th International Conference on Microelectronics (ICM), Sousse, Tunisia. pp. 299–302, 16–19 December 2018.

[12] IEEE Computer Society, (2019). IEEE Standard for Floating-Point Arithmetic. IEEE STD 754-2019. IEEE. pp. 1–84. doi:10.1109/IEEESTD.2019.8766229. ISBN 978-1-5044-5924-2. IEEE Std 754-2019.

[13] Lourcha, K. I., Nguyen, A., Hung, D., Fast Adder/Subtractor for Signed Floating Point Numbers. U.S. Patent 6,175,851, 16 January 2001.

[14] Joldes, M., Muller, J. M., Algorithms for Manipulating Quaternions in Floating-Point Arithmetic. In Proceedings of the IEEE 27th Symposium on Computer Arithmetic (ARITH), Portland, OR, USA, 7–10 June 2020.

[15] Karlstrom, P., Ehliar, A., Liu, D. High performance, low latency fpga based floating point adder and multiplier units in a virtex 4. In Proceedings of the 2006 NORCHIP, Linkoping, Sweden. pp. 31–34, 20–21 November 2006.

[16] Kawaguchi, T., Floating Point Addition and Subtraction Arithmetic Circuit Performing Preprocessing of Addition or Subtraction Operation Rapidly. U.S. Patent 5, 931, 896, 3 August 1999.

[17] Langhammer, M., Variable Precision Floating-Point Adder and Subtractor. U.S. Patent No. 10,055,195, 21 August 2018.

[18] Langhammer, M., Pasca, B., Floating-Point Adder Circuitry with Subnormal Support. U.S. Patent Application No. 15/704,313, 14 March 2019.

[19] Lutz, D.R., Hinds, C. N., Data Processing Apparatus and Method for Performing Floating Point Addition. U.S. Patent No. 7,433,911, 7 October 2008.

[20] Malik, A., Ko, S.-B., Effective implementation of floating-point adder using pipelined LOP in FPGAs. In Proceedings of the Canadian Conference on Electrical and Computer Engineering, Saskatoon, SK, Canada, pp. 706–709. 1–4 May 2005.

[21] Mathis, B., Stine, J. A., Well-Equipped Implementation: Normal/Denormalized Half/Single/Double Precision IEEE 754 Floating-Point Adder/Subtracter. In Proceedings of the 2019 IEEE 30th International Conference on Application-specific Systems, Architectures and Processors (ASAP), New York, NY, USA. Volume 2160, pp. 227–234. 15–17 July 2019.

[22] Nakayama, T., Hardware Arrangement for Floating-Point Addition and Subtraction. U.S. Patent No. 5,197,023, 23 March 1993.

[23] Nannarelli, A., Tunable Floating-Point Adder. IEEE Trans. Comput. 68, 1553–1560. 2019.

[24] Nystad, J., Floating-Point Adder. U.S. Patent No. 9,009,208, 14 April 2015.

[25] Oberman, S. F., Floating Point Arithmetic Unit Including an Efficient Close Data Path. U.S. Patent No. 6,094,668, 25 July 2000.

[26] Oberman, S. F., Flynn, M. J., Design issues in division and other floating-point operations. IEEE Trans. Comput. 46, 154–161. 1997.

[27] Pangal, A., Somasekhar, D., Vangal, S. R., Hoskote, Y. V., Floating Point Adder. U.S. Patent No. 6,889,241, 3 May 2005.

[28] Pappalardo, F., Visalli, G., Scarana, M., An application-oriented analysis of power/precision trade-off in fixed and floating-point arithmetic units for VLSI processors. In Circuits, Signals, and Systems; Citeseer: Clearwater Beach, FL, USA. pp. 416–421, 2004.

[29] Quinnell, E. C., High Performance Floating-Point Adder With Full in-Line Denormal/Subnormal Support. U.S. Patent No. 10,108,398, 23 October 2018.

[30] Resnick, D. R., Moore, W. T., Floating-Point Adder Performing Floating-Point and Integer Operations. U.S. Patent 6,529,928, 4 March 2003.

[31] Seidel, P. M., Even, G., Fast IEEE Floating-Point Adder. U.S. Patent 10/138,659, 20 March 2003.

[32] Stiles, D., Method and Apparatus for Performing Floating Point Addition. U.S. Patent No. 5,764,556, 9 June 1998.

[33] Tao, Y., Deyuan, G., Xiaoya, F., Xianglong, R., Three-operand floating-point adder. In Proceedings of the 2012 IEEE 12th International Conference on Computer and Information Technology, Chengdu, China. pp. 192–196 27–29 October 2012.

[34] Thompson J., Nandini Karra & Schulte, M. J., A 64-bit Decimal Floating-point Adder. IEEE Computer Society Annual on VLSI Systes Design. DOI: 0-7695-2097-9/04. 2004.

[35] Villalba, J., Hormigo, J., González Navarro, S., Fast HUB Floating-point Adder for FPGA. IEEE Trans. Circuits Syst. II Express Briefs, 66, 1028–1032. 2018.

[36] Xu, J., Wang, H., Desynchronize a legacy floating-point adder with operand-dependant delay elements. In Proceedings of the 2011 IEEE International Symposium of Circuits and Systems (ISCAS), Rio de Janeiro, Brazil, pp. 1427–1430, 15–18 May 2011.

[37] Yamada, H., Murabayashi, F., Yamauchi, T., Hotta, T., Sawamoto, H., Nishiyama, T., Kiyoshige, Y., Ido, N., Hitachi Ltd. Floating-Point Addition/substraction Processing Apparatus and Method Thereof. U.S. Patent 5,684,729, 4 November 1997.