Revista EIA

UNIVERSIDAD EIA
Ser, Saber y Servir

Unidad aritmética de punto flotante: diseño e implementación con portabilidad

Revista EIA
ISSN 1794-1237
e-ISSN 2463-0950
Año XIX/ Volumen 20/ Edición N.39
Enero-Junio de 2023
Reia3905 pp. 1-20

Publicación científica semestral
Universidad EIA, Envigado, Colombia

Para citar este artículo / To reference this article /
Patarroyo Gutierrez, L.; Gil Hernández C.; Gutiérrez Meléndez, I.
Unidad aritmética de punto flotante: diseño e implementación con portabilidad.
Revista EIA, 20(39), Reia3905.
pp. 1-20.
https://doi.org/10.24050/reia.v20i39.1609

Autor de correspondencia:

Patarroyo Gutierrez, L. (Luis)
Magister en Ingeniería
Universidad Pedagógica y Tecnológica de Colombia
Correo electrónico:
luis.patarroyo@uptc.edu.co

Recibido: 03-08-2022
Aceptado: 12-09-2022
Disponible online: 01-01-2023

Luis David Patarroyo Gutierrez[1]
Carlos Andrés Gil Hernández[1]
Iván Ricardo Gutiérrez Meléndez[1]

1. Universidad Pedagógica y Tecnológica de Colombia

Resumen

El uso de las unidades de punto flotante (FPU) en el procesamiento digital de señales se ha incrementado dada la alta precisión y rango de números que se pueden representar. En el procesamiento de imágenes, filtros digitales con respuesta infinita al impulso (IIR), respuesta finita al impulso (FIR) y controladores digitales, se requieren este tipo de unidades para obtener resultados más precisos y evitar respuestas inestables, sin embargo, para implementar estas, algunos procesadores tienen unidades incorporadas lo que implica una dependencia tecnológica de los fabricantes para desarrollar prototipos. Para evitar dicha dependencia, en este artículo se presenta el diseño de los módulos para las operaciones más usadas en el procesamiento digital de señales: multiplicación y la suma/resta. Se presentan los pasos y consideraciones a tener en cuenta como las excepciones, redondeo y normalización de operandos, para lograr implementar estas operaciones en cualquier Filed Programmable Gate Array (FPGA). Se comprueban resultados utilizando el banco de pruebas MODELSIM® y se determinó la tasa de error, utilizando MATLAB®.

Palabras claves: Aritmética de punto flotante, Unidad de Punto Flotante, MatLab, Portabilidad, Sumador, Multiplicador, Procesadores Digitales de Señal.

Floating point unit: design and implementation with portability

Abstract

The use of floating-point units (FPU) in digital signal processing has increased due to the range and the high precision numbers that can be represented. These units are required in image processing, digital filters with infinite impulse response (IIR), finite impulse response (FIR), and digital controllers, to obtain accurate results and avoid unstable responses, however, to implement them, some processors have built-in units, this implies a technological dependency of the manufacturers when prototypes are developed. To avoid such dependency, this article presents the design of modules for the most used operations in digital signal processing: multiplication and addition/subtraction. To be able to implement these operations in any Field Programmable Gate Array (FPGA), the steps and considerations to be aware of are presented, such as exceptions, rounding, and normalization of operands. Results are verified using the MODELSIM® test bench and the error rate was determined using MATLAB®.

Key Words: Floating-point arithmetic, Float Point Unit, MatLab, Portability, Adder, Multiplicator, Digital Signal Processors.

1. Introducción

Existen dos representaciones numéricas que permiten realizar cálculos computacionales como lo son punto fijo y punto flotante. Estas se seleccionan dependiendo el rendimiento y la precisión de la aplicación; en procesamiento digital de señales y sensado remoto se requiere de alto rendimiento, mientras que en sistemas de reconocimiento de objetos con alta velocidad y en computadoras de muy alto rendimiento se usa la representación en punto flotante, debido a que se puede representar una gama más amplia de valores numéricos (Malkapur y Rajput, 2020). Los sistemas de alto rendimiento que realizan operaciones con números reales, como los computadores y las Unidades de Punto Flotante (FPU) dedicadas a aplicaciones específicas de procesamiento de señales, usan la representación binaria en punto flotante según el estándar IEEE-754 para representar números reales. En las aplicaciones científicas, en la electrónica industrial, en el procesamiento de imágenes y en unidades aritméticas de microprocesadores se usan FPUs dado que es necesario calcular una gran cantidad de datos que se usan con frecuencia (Joshi y Gawali, 2016), como en el control de sistemas dinámicos y la clasificación de patrones usualmente se presentan operaciones aritméticas complejas con grandes demandas en términos de precisión y rango dinámico, también en la optimización de puntos de operación en generadores fotovoltaicos a menudo requiere la linealización de

Revista EIA https://doi.org/10.24050/reia.v20i39.1609

funciones exponenciales en las relaciones de corriente y voltaje (I-V), con el fin de estimar los parámetros requeridos para los propios algoritmos de optimización. Un caso en el que se requiere necesariamente la aritmética en punto flotante es la simulación e implementación de filtros IRR, debido a que los coeficientes del filtro deben contar con precisión para evitar inestabilidad a la salida del filtro, por lo tanto, la aritmética de punto flotante es obligatoria en las aplicaciones que requieren precisión y rango dinámico (Barry y Crowley, 2012; Kaur *et al.*, 2015).

Una manera de implementar FPUs consiste en hacer uso de una FPGA (BOLAÑOS y BERNAL, 2008; Álvarez y Lindig B., 2008), ya que el diseño y desarrollo de módulos de aplicación específica mediante el uso de lenguajes de descripción de Hardware (HDL), se ha convertido en una posibilidad para dar solución al gran crecimiento en la complejidad de aplicaciones que involucran cálculos densos y donde se requieren procesar datos de manera precisa, confiable y altas frecuencias operativas (Hamid *et al.*, 2010; Ramyarani, N. Subbiah, V. Deepa, 2019).

Las FPGA modernas son excelentes componentes que permiten el desarrollo de prototipos gracias a su hardware configurable (Sandoval-Ruiz, 2019; López, Restrepo y Tobón, 2020). Con cada generación de FPGA se agregan características que permiten hacer posible el realizar cálculos con menor latencia, sin embargo, se realizan en punto fijo careciendo de soporte para cálculos en punto flotante (Karlstrom, Ehliar y Liu, 2008). Otra alternativa para realizar operaciones en punto flotante es usar circuitos integrados de aplicación específica (ASICs) ya que son muy eficientes en las operaciones de punto flotante, sin embargo, carecen de programabilidad y flexibilidad que se requieren en algunos casos y adicionalmente el costo de un ASIC es significativamente alto. Además, los procesadores de propósito general con unidades punto flotante (FPU) son programables y son capaces de operar a altas tasas de reloj, pero el rendimiento es limitado por la carencia de hardware modificable. Luego como se indica en (Chong y Parameswaran, 2011) las FPGA son plataformas muy atractivas para aplicaciones aunque presenten las limitaciones mencionadas.

Finalmente, el diseño de elementos de procesamiento en punto flotante (FPPE) se convierte en un requerimiento para llevar acabo implementaciones que minimicen el error en las operaciones realizadas (Wilton, 2012; Lasith y Thomas, 2014; José *et al.*, 2014; Zoni, Galimberti y Fornaciari, 2021)y teniendo en cuenta que en el procesamiento de señales las operaciones de multiplicación, suma y resta son el 94% del total de las instrucciones (Hamid *et al.*, 2010), en este artículo se presenta el diseño de una FPU con estas operaciones, usando lógica combinacional bajo el estándar IEEE 754 en precisión sencilla. Este

articulo brinda un procedimiento para describir unidades en punto flotante de acuerdo al estándar, permitiendo abordar los procesos de normalización, redondeo, ajuste de mantisas y casos especiales que se presentan al efectuar operaciones. Dado que este diseño es totalmente combinacional es escalable y por lo tanto se puede ajustar a doble precisión, sin embargo, se requieren más recursos lógicos para su implementación, además, es un punto de partida para llevar a cabo la evaluación de *test* sobre diferentes arquitecturas de circuitos digitales que realizan operaciones en representación punto flotante (Cantoro *et al.*, 2016; Condia *et al.*, 2020; Guerrero-Balaguera, Condia y Reorda, 2021)we describe techniques for the generation of SBST programs to be run on-line by an embedded microprocessor to detect faults in the Floating-Point Unit (FPU. Una ventaja del diseño totalmente combinacional es que se puede paralelizar fácilmente usando técnicas de pipeline, ya que el diseño combinacional permite evaluar diferentes niveles de pipeline, así, el diseño propuesto se aplica para evaluar fallos en implementaciones futuras pipeline y no pipeline.

El principal aporte de este artículo consiste en que el diseño propuesto para la implementación de las unidades de suma/resta y multiplicación se puede aplicar en cualquier FPGA, así como su incorporación en SoCs y ASICs, debido a que el diseño se realizó empleando lenguajes de descripción de hardware ofreciendo portabilidad para aplicación en sistemas reconfigurables, ya que únicamente se usan librerías que pertenecen al estándar IEEE, facilitando el uso específico de la tecnología en la etapa de síntesis de diseño, lo que permite implementar algoritmos de alto rendimiento en punto flotante y describirlos en hardware (Zhang y Zhao, 2017; Veeranki y Nakkeeran, 2013)many scientific applications require floating point arithmetic because of high accuracy in their calculations. Therefore, an attempt is made to explore FPGA implementations in Institute of Electrical and Electronics Engineers (IEEE, permitiendo la evaluación de *test* en futuras implementaciones.

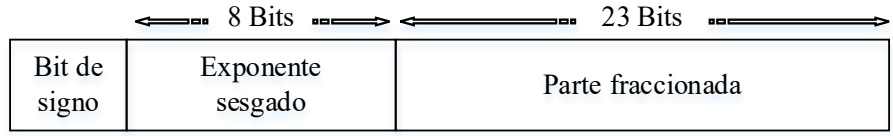
Este artículo está organizado de la siguiente manera: en la sección 2 se da una descripción del estándar IEEE 754 haciendo énfasis en la representación de números en precisión sencilla. Los módulos que forman la unidad propuesta junto con los bloques que hacen parte de cada módulo y sus funciones se describen en la sección 3. En la sección 4 se presentan el análisis de resultados junto con recursos lógicos requeridos para la descripción de los dos módulos. Finalmente se muestran las conclusiones en la sección 5.

Revista EIA https://doi.org/10.24050/reia.v20i39.1609

1.1. ESTÁNDAR IEEE 754.

Un formato en punto flotante, se entiende como un sistema para representar un número como un vector de 32 bits, para precisión simple, y de 64 bits, para precisión doble, que consta con representaciones para el signo, exponente y mantisa (Russinoff, 2019), en la Figura 1 se muestra la representación del estándar en precisión simple utilizado en este proyecto.

Figura 1. Representación de un número en el estándar IEEE 754-2008 precisión simple



Para simplificar los cálculos realizados con números representados en coma flotante, el estándar IEEE 754 hace mención de números normalizados, un número se considera normalizado cuando el primer bit después de la coma flotante es 1 y se resta el valor al exponente un valor de sesgo (número entero sin signo de 8 bits) dado por donde *k* es el número de bits que conforman el exponente (STALLINGS, 2005; Hamid *et al.*, 2010). La Ecuación 1 expresa la representación de números de coma flotante bajo el estándar:

$$f=(-1)^{signo}\bullet 2^{exp+bias}\bullet 1.matisa \quad \text{(Ecuación 1)}$$

Este tipo de representación fue publicada en 1985 por el Institute of Electrical and Electronics Engineers (IEEE), con el objetivo de realizar una especificación en la precisión y formato de números en punto flotante, este estándar ha tenido una amplia acogida en los procesadores y coprocesadores actuales (STALLINGS, 2005). Este formato incluye una lista de operaciones que se pueden realizar con estos números, además del tratamiento de algunos casos especiales, como el manejo de números infinitos y valores no numéricos. Este estándar se diseñó con el fin de otorgar una mayor facilidad a la portabilidad de los programas de un coprocesador a otro y alentar el desarrollo de programas numéricos avanzados (Cervantes *et al.*, 2016), en este se definen:

Representación numérica

Se define como el conjunto de datos binarios en punto flotante, que consisten de números finitos, infinitos y valores que no representan ningún número (NaN - Not a Number), un valor que esté por encima de los rangos representables (números finitos), se evidencia teniendo los 8 bits de exponente en '1' y la representación entera de la mantisa en '0' (Russinoff, 2019), como son definidos en la Figura 2.

Figura 2. Representación binaria de infinito

0	11111111	00000000000000000000000

Algoritmo de redondeo

El estándar IEEE indica cuatro formas para realizar el redondeo más cercano a par (RNE), redondeo a cero (RTZ), redondeo a infinito (y redondeo a menos infinito (. El redondeo es necesario en las operaciones con números de coma flotante, para hacer aproximaciones al número representable más cercano o cero, con el fin de no afectar la precisión del resultado (STALLINGS, 2005).

Excepciones

Estas hacen referencia a los números que no forman parte de los rangos representables que dicta el estándar en precisión sencilla, estos rangos de números van desde a y desde a (Russinoff, 2019). Las excepciones indican si se presentan casos como desbordamiento (Overflow) que ocurre en el instante que la magnitud del resultado excede la magnitud a representar, desbordamiento hacia cero (Underflow) en el caso de que el resultado es muy pequeño y no se puede representar o números denormalizados. Estos últimos son incluidos en el estándar para disminuir las situaciones de desbordamiento a cero de los exponentes (STALLINGS, 2005). En el estándar IEEE-754 de precisión simple, un Overflow ocurre si el exponente es mayor que 128, y un Underflow si el número es tan pequeño con un exponente menor a -127 que su representación se acerca a cero.

Revista EIA https://doi.org/10.24050/reia.v20i39.1609

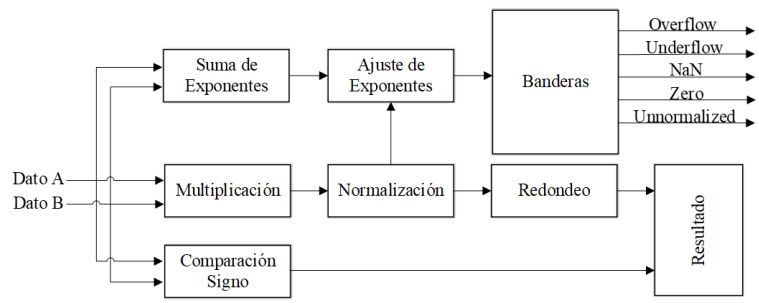
2. Metodología

En esta sección se presenta el diseño de la FPU propuesta, se inicia con el módulo de multiplicación y seguido de este el módulo de suma-resta, lo anterior mediante descripción comportamental aplicando diseño RTL.

2.1. Módulo de multiplicación

Esta operación consiste en realizar los productos de las mantisas y sumas de los exponentes de los operandos, además de llevar a cabo la normalización del resultado para evitar desbordamiento de la mantisa y finalmente el redondeo del número de acuerdo al estándar IEEE 754. El diseño de esta unidad se contempló en el resultado real de la operación y las excepciones. El bloque de excepciones se describe debido a que al realizar la multiplicación de dos números con exponentes o mayores, o con exponentes o menores se generaba una excepción. En la Figura 3 se muestra un esquema general de la unidad.

Figura 3. Diagrama de un multiplicador en punto flotante



Los bloques de esta unidad se encargan de realizar los procesos necesarios para obtener un resultado, con un porcentaje de error bajo, a continuación, se describe cómo funcionan los bloques que conforman este módulo:

Bloque Multiplicación

Este bloque cuenta con dos entradas de 24 bits para cargar los datos de ambos operandos y una única salida de 48 bits, estos son procesados por el bloque de normalización, una particularidad de este formato es

que se puede realizar la operación de las mantisas como si fuese una multiplicación entre números enteros.

Bloque Banderas

Este bloque está formado únicamente por compuertas lógicas AND, OR y NOT, el bloque de banderas cuenta con dos entradas de 8 bits para cargar los exponentes de cada operando, aquí se generan 5 señales que representan cada una de las excepciones indicadas en el estándar IEEE 754.

Bloque Suma de exponentes

Para obtener el dato de salida del exponente, este bloque realiza la suma entre los dos exponentes teniendo en cuenta un acarreo de entrada, al resultado de la suma se resta, o suma en complemento un sesgo de 127 (Hamid et al., 2010), para generar los indicadores de excepción cuando se operan dos datos con exponentes o mayores, o o menores se debe cumplir lo presentado en la Tabla 1, teniendo en consideración el acarreo de salida y el bit más significativo de la parte entera representable del exponente.

Tabla 1. Tabla de descripción de indicadores de excepción.

Acarreo de salida	MSB	Infinito	Cero
0	0	0	1
0	1	0	0
1	0	0	0
1	1	1	0

Bloque Normalización

La normalización del resultado se lleva a cabo con este bloque, dado el caso que se presente un desbordamiento de la mantisa; dicho de otra forma, cuando se genera un segundo bit después del punto decimal (además de la normalización) se lleva a cabo el trunqueo del dato, es decir se ignoran los 23 bits menos significativos del resultado de la multiplicación. Utilizando el bit más significativo (MSB) del vector de bits resultantes del trunqueo para generar un ajuste tanto en el exponente como en la mantisa del resultado.

Revista EIA https://doi.org/10.24050/reia.v20i39.1609

Bloque Redondeo

El redondeo se debe llevar a cabo como se define en el estándar IEEE 754, para lograrlo este bloque busca hacer una aproximación de la representación más cerca del resultado, siempre y cuando el bit menos significativo de este sea cero.

2.2. Módulo De Suma/Resta

La suma consiste en una operación más compleja que la multiplicación, debido a la cantidad de desplazamientos que deben realizarse en la mantisa de los datos. Este módulo tiene la capacidad de comprobar valores cero, ajustar partes significativas, sumar o restar las partes significativas y normalizar el resultado. El diseño de este módulo se sintetiza mediante la descripción de cuatro bloques para satisfacer los requisitos mencionados anteriormente, también se deben determinar excepciones y hacer el respectivo redondeo del resultado. Estos bloques están interconectados como se muestra en la Figura 4. Los puertos de entrada y salida se resumen en la Tabla 2.

Figura 4. Distribución y conexión de los bloques funcionales del circuito combinacional de la unidad de suma/resta en punto flotante

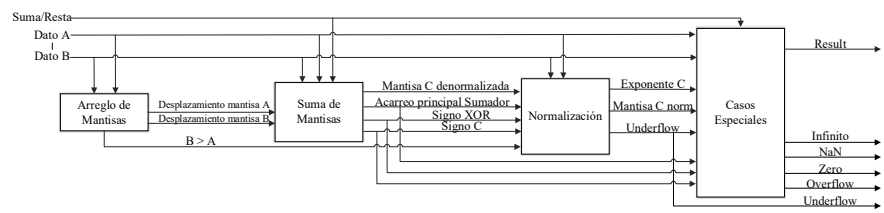


Tabla 2. Distribución y conexión de los bloques funcionales del circuito combinacional de la unidad suma/resta en punto flotante.

Puerto	Tipo	Descripción
DATA A	IN	Dato de entrada de 32 bits concerniente a un número en punto flotante bajo el estándar IEEE-754.
DATA B	IN	Dato de entrada de 32 bits concerniente a un número en punto flotante bajo el estándar IEEE-754.
ADD/ SUBSTRACT	IN	Señal de selección, si es '0' la unidad opera DATA B + DATA A, si es '1' DATA B – DATA A.
RESULT	OUT	Dato de salida de 32 bits
OVERFLOW	OUT	Determina si se presenta Overflow en la operación.
UNDERFLOW	OUT	Determina si se presenta Underflow en la operación.
ZERO	OUT	Determina si el resultado es 0.
NAN	OUT	Determina si el resultado es NaN.
INFINIT	OUT	Determina si el resultado es infinito

Cada bloque y sus funciones se describen a continuación:

Bloque de arreglo de mantisas

Para realizar la operación suma se cargan en este bloque los dos datos de 32 bits, y seguido de esto se ajustan las partes significativas de ambos números, para lograrlo se desplazan hacia la derecha los bits del menor de los datos (Hamid *et al.*, 2010), hasta un máximo de 15 desplazamientos (en caso de requerirse más, el dato se iguala a cero). Este dato se selecciona a partir de la comparación de los exponentes de cada número, mediante la señal ($B > A$) que determina cuál de los exponentes es el mayor. Esta señal se usa como línea de selección de multiplexores que seleccionan la mantisa del número con el menor exponente para ser ajustada. El bit implícito es añadido a las mantisas de ambos datos.

La cantidad de desplazamientos se definen restando el menor exponente del mayor, obteniendo un número positivo o cero (en este caso no se realiza ningún ajuste). Este resultado es comparado y si es mayor a 15, se determina que el menor de los datos es insignificante con respecto al otro, por lo que se considera como cero para evitar hacer desplazamientos innecesarios. Al final de este bloque se organizan las mantisas resultantes de ambos números, tanto la que ha sido ajustada, como la que permanece inalterada (junto con el bit 24). Por tanto,

Revista EIA https://doi.org/10.24050/reia.v20i39.1609

este bloque tiene como salida dos datos de 24 bits (*Desplazamiento de mantisa A* y *Desplazamiento de mantisa B*) así como una señal (*B > A*).

Bloque de suma de mantisas

Este bloque opera las mantisas que provienen del bloque arreglo de mantisas (*Desplazamiento de mantisa A* y *Desplazamiento de mantisa B*), llevando a cabo operaciones binarias de suma y de resta. Para establecer qué operación se lleva a cabo se requiere la señal *ADD/SUBSTRACT* (que únicamente modifica el signo del dato A), y también los signos de cada número, de acuerdo a la Tabla 3.

Tabla 3. Tabla de tipos de operación en el módulo suma/resta.

Puerto	Tipo	Descripción
0	0	+1 * (B + A)
0	1	+1 * (B – A)
1	0	-1 * (B - A)
1	1	-1 * (B + A)

Como se presenta en la Tabla 3, de las 4 operaciones posibles teniendo en cuenta únicamente los signos de los datos, es posible resumirlas en una suma o una resta binaria, con su correspondiente factor multiplicativo. Este factor se asocia con el estado lógico del signo B, siendo '0' para una constante (+1) y '1' para una constante de (-1). Para determinar si el dato A es sumado o restado del dato B, se realiza una operación XOR entre los bits de signo de ambos números y la señal ADD/SUBSTRACT. Esta operación da como resultado la señal denominada (Signo XOR), si es '0' se realiza B+A y si es '1' se realiza B-A. Una vez realizada la operación binaria, se obtiene un dato de 24 bits correspondiente a la mantisa resultante sin normalizar (Mantisa C denorm). El signo del resultado (Signo C), es determinado de acuerdo a lo presentado en la Tabla 4, la cual depende de las señales (Signo XOR), (Signo B) y (Acarreo principal Sum.), esta última señal corresponde al acarreo de salida (carry) del sumador de 24 bits donde se realizan las operaciones binarias, el acarreo determina si hubo desbordamiento en operaciones de suma binaria, o si la resta binaria da un resultado negativo o positivo.

Tabla 4. Tipos de operación en el módulo suma/resta.

Signo B	Signo XOR	Acarreo principal sum	Signo C
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Bloque de normalización

Se encarga de normalizar la mantisa resultado del bloque anterior (Mantissa C denorm.), mediante el método L.O.D (leading zero detecting) (Hamid et al., 2010). Este método consiste en contar el número de ceros lógicos antes del uno lógico más significativo, para realizar desplazamientos en la mantisa a la cantidad de ceros lógicos previamente contados, ajustando a la vez el exponente del resultado. El método L.O.D se utiliza al presentarse una resta binaria, si se produce una suma binaria los desplazamientos dependerían de un desbordamiento de la parte significativa, producto del acarreo resultante de la operación (Acarreo principal Sum) (STALLINGS, 2005). El tipo de normalización, de acuerdo a la operación, se determina mediante la señal (Signo XOR), utilizada como línea de selección de datos. Lo anterior se resume en la Tabla 5. En este bloque se lleva a cabo el redondeo del resultado (RNE), justo después de completar el proceso de normalización.

Tabla 5. Normalización del resultado dependiendo del tipo de operación.

NORMALIZACIÓN			
Adición	Desbordamiento	10,xxxxxxx…	1 desplazamiento a la derecha
	No desbodamiento	01,xxxxxx…	Sin desplazamientos
Sustracción	L.O.D	X0,00001xxx…	n desplazamientos a la izquierda

Revista EIA https://doi.org/10.24050/reia.v20i39.1609

Por otra parte, el exponente se normaliza teniendo en cuenta también estos dos posibles escenarios. Si se realiza una suma, el exponente resultante (*Exponente C*) es igual al exponente del mayor número, si no se presenta desbordamiento. Si, por el contrario, se presenta desbordamiento en la operación, el exponente resultante será igual al exponente del mayor numero . Si se realiza una resta, al exponente más grande se le resta los *n* desplazamientos. En el determinado caso en que el número de desplazamientos sea mayor a dicho exponente, se presentara *Underflow.* Los exponentes durante todo el proceso están sesgados.

Bloque de casos especiales

Este bloque se encarga de revisar las excepciones. Mediante 5 banderas incluidas en la Tabla 2, estas permiten indicar si se presenta alguno de estos casos especiales. A este bloque se cargan las señales signo, exponente y mantisa normalizada resultantes de los bloques anteriores. De acuerdo a si se presentan excepciones, omite estos resultados y directamente, a la par de la señalización mediante la bandera, se presenta un resultado acorde al caso. Si no se presenta ninguna excepción (es decir, se llevó a cabo una operación común entre números normalizados), se conforma el dato de salida de 32 bits (Result), con los datos (Signo C), (Exponente C) y (Mantisa C norm). En este bloque además se revisa si alguno de los datos de entrada (o ambos) es cero, para determinar el resultado omitiendo el proceso de ajuste, suma de partes significativas y la normalización.

3. Resultados

Para determinar el funcionamiento de los módulos descritos, se realizó simulación Test Bench en el software MODELSIM®. Para ello se realizaron pruebas entre amplios rangos de números, con alrededor de 10.000 muestras por cada prueba. Generando dos grupos de números aleatorios en MATLAB®, que posteriormente son multiplicados o sumados en pares ordenados, hasta realizar 10.000 operaciones. Los resultados de estas operaciones arrojadas por el Test Bench son comparadas en MATLAB®, donde se determina el error porcentual absoluto medio o MAPE como se muestra en la Ecuación 2.

$$APE=\sum_{i=1}^{n} ET-ER/|ET|/n\bullet 100 \qquad \text{(Ecuación 2)}$$

Donde, n es la cantidad de resultados a comparar, ET son los resultados obtenidos al operar los grupos de números aleatorios en MATLAB® y ER son los resultados obtenidos en el Test Bench de los módulos. Se realizaron 5 pruebas en 2 rangos diferentes, tanto para el módulo de multiplicación (Figuras 5 y 6), como para el de suma/resta (Figuras 7 y 8). En estas se puede observar el MAPE calculado para cada caso.

Figura 5. Resultados de Error de la unidad de multiplicación en el rango 1

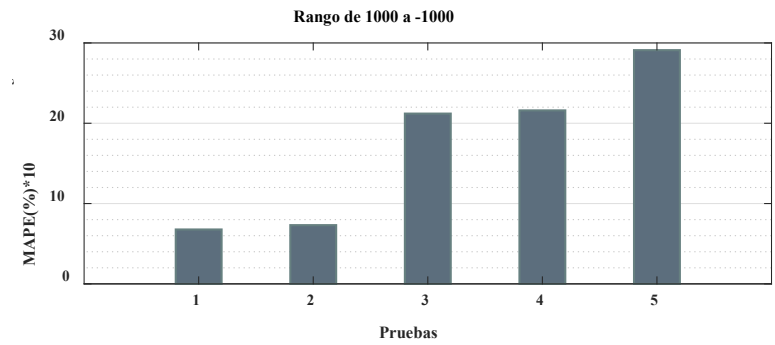
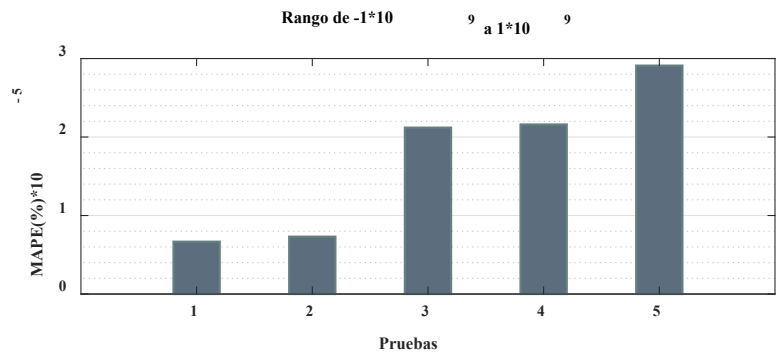


Figura 6. Resultados de Error de la unidad de multiplicación en el rango 2



https://doi.org/10.24050/reia.v20i39.1609

Figura 7. Resultados de Error de la unidad de suma/resta en el rango 1

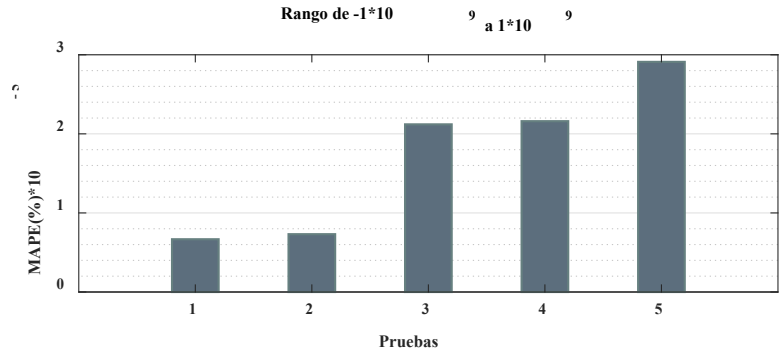
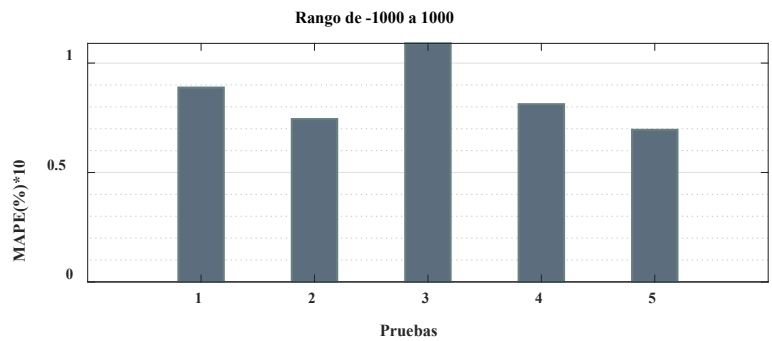


Figura 8. Resultados de Error de la unidad de suma/resta en el rango 2



Los resultados obtenidos pueden determinar que, tanto la unidad de multiplicación como la de suma/resta poseen un error menor al , debido a que se basa expresamente en lógica combinacional (como es el caso del módulo de suma/resta) que sintetiza los pasos para realizar cada operación. Cada bloque (Figuras 3 y 4) se conforma estrictamente de compuertas lógicas y circuitos como multiplexores descritos en VHDL. Esto permite identificar el flujo de datos correctamente, y se propicia para conocer exactamente cómo funciona cada bloque, dado que el diseño es transparente al basarse en hardware digital únicamente.

Así mismo, de acuerdo a la revisión de la literatura, se tienen en cuenta todos los aspectos, como el desplazamiento de la mantisa, las técnicas de normalización y de redondeo, fundamentales para asegurar resultados acordes al estándar IEEE 754. Como consideración especial, ambos módulos tienen en cuenta en sus operaciones las excepciones mencionadas anteriormente, al dedicarse hardware específico para atender estos casos. En términos

de recursos lógicos utilizados para implementar cada diseño, según los reportes de compilación del software QUARTUS® usado para sintetizar los diseños en la FPGA Cyclone® IV EP4CE22F17C6N de Altera con 22.320 elementos lógicos totales, se obtuvo los siguientes resultados para el módulo de suma/resta.

En la Tabla 6 se presentan los recursos lógicos utilizados por la unidad de suma/resta, aproximadamente el 48% de estos elementos se utilizan en el **bloque de arreglo de mantisas**, debido a la cantidad de desplazamientos que deben realizarse. Mientras que los **bloques de suma de mantisas** y **normalización** ocupan menos del 19%. Cerca del 32% restante de los elementos lógicos son usados por el **bloque de casos especiales**, con el fin de determinar los estados de las banderas, y tener en cuenta las excepciones al momento de obtener el resultado. El hardware adicional de este bloque no interfiere en el cálculo normal de números normalizados (como el mostrado en las Figuras 5, 6, 7 y 8), pero comprueba si alguno de los datos de entrada es cero, para evitar hacer cálculos innecesarios y así incrementar el rendimiento de la unidad.

Tabla 6. Normalización de recursos lógicos módulo de suma/resta.

	Elementos lógicos
Sumador punto flotante	1024
Suma de mantisas	89
Casos especiales	335
Normalización	100
Arreglo de mantisa	500

En la Tabla 7 se presentan los recursos lógicos utilizados por la unidad de multiplicación, a diferencia de la unidad de suma/resta, el multiplicador hace uso de 7 elementos de Procesamiento digital de señales (DSP) o multiplicadores embebidos, esto se debe a la técnica utilizada para realizar la operación de la mantisa, la cual fue descrita utilizando el signo de multiplicación permitiendo utilizar solo 154 elementos lógicos en total, a diferencia de otra técnica utilizada como la descripción de un multiplicador matricial de acarreo anticipado, el cual necesitaba de 1541 recursos lógicos, para realizar solo la operación entre las mantisas.

Revista EIA https://doi.org/10.24050/reia.v20i39.1609

Tabla 7. Utilización de recursos lógicos módulo de multiplicación.

	Elementos lógicos	Elementos DSP
Multiplicador punto flotante	154	0
Exponente	25	0
Casos especiales	14	0
Normalización	53	0
Multiplicación de mantisas	55	7

4. Conclusiones

- En esta investigación se diseñó una FPU descrita usando lógica combinacional combinado con el uso de técnicas de normalización, ajuste y redondeo adecuadas (mencionadas por la literatura), se obtuvo resultados con errores inferiores al en la suma y en la multiplicación. Así mismo, son diseños que presentan la facilidad de ser revisados y perfeccionados, reduciendo posiblemente la utilización de elementos lógicos al utilizar técnicas diferentes a las mencionadas y considerando que, al haberse descrito en VHDL, es posible configurarlos para un uso más específico. Esto significa prescindir de ciertos bloques o expandir el bus de datos para aumentar la precisión, debido a que son diseños escalables.
- Durante el diseño del bloque de multiplicación, fue posible obtener una disminución significativa en el uso de recursos lógicos, planteando una técnica diferente a la utilización de un multiplicador matricial con acarreo anticipado, empleando en cambio, una conversión de formato sencilla de los datos a tratar.
- Los datos obtenidos por los reportes de compilación de Quartus® pueden usarse para determinar futuras aplicaciones para estos módulos, teniendo en cuenta que, gracias a su portabilidad, pueden usarse en cualquier FPGA sin limitaciones.
- El diseño de la FPU es un punto de partida para llevar a cabo la evaluación de test sobre diferentes arquitecturas de circuitos digitales que realizan operaciones en el formato IEEE 754. Como este diseño totalmente combinacional se puede paralelizar fácilmente usando técnicas de pipeline, ya que el diseño combinacional permite evaluar diferentes niveles de pipeline, así, el diseño propuesto se aplica para evaluar fallos en implementaciones futuras pipeline y no pipeline.

5. Referencias

Álvarez, J. A. y Lindig B., M. (2008) 'Diseño de un Coprocesador Matemático de Precisión Simple usando el Spartan 3E', *Polibits*, 38. Available at: https://www.redalyc.org/articulo.oa?id=402640451010.

Barry, P. y Crowley, P. (2012) 'Chapter 5 - Embedded Processor Architecture', in Barry, P. and Crowley, P. B. T.-M. E. C. (eds). Boston: Morgan Kaufmann, pp. 99–152. doi: https://doi.org/10.1016/B978-0-12-391490-3.00005-9.

BOLAÑOS, F. y BERNAL, Á. (2008) 'Una implementación hardware optimizada para el operador exponenciación modular.', *Dyna*, (156), pp. 55–63.

Cantoro, R. *et al.* (2016) 'In-field functional test programs development flow for embedded FPUs', *2016 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, DFT 2016*, pp. 107–110. doi: 10.1109/DFT.2016.7684079.

Cervantes, A. *et al.* (2016) 'Implementation of an open core IEEE 754-based FPU with non-linear arithmetic support', in *2016 IEEE 36th Central American and Panama Convention (CONCAPAN XXXVI)*, pp. 1–6. doi: 10.1109/CONCAPAN.2016.7942354.

Chong, Y. J. y Parameswaran, S. (2011) 'Configurable Multimode Embedded Floating-Point Units for FPGAs', *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 19(11), pp. 2033–2044. doi: 10.1109/TVLSI.2010.2072996.

Condia, J. E. R. *et al.* (2020) 'Design and Verification of an open-source SFU model for GPGPUs', in *2020 17th Biennial Baltic Electronics Conference (BEC)*, pp. 1–6. doi: 10.1109/BEC49624.2020.9276748.

Guerrero-Balaguera, J.-D., Condia, J. E. R. y Reorda, M. S. (2021) 'On the Functional Test of Special Function Units in GPUs', in *2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*, pp. 81–86. doi: 10.1109/DDECS52668.2021.9417025.

Hamid, L. S. A. *et al.* (2010) 'Design of Generic Floating Point Multiplier and Adder/Subtractor Units', in *2010 12th International Conference on Computer Modelling and Simulation*, pp. 615–618. doi: 10.1109/UKSIM.2010.117.

José, W. *et al.* (2014) 'Efficient implementation of a single-precision floating-point arithmetic unit on FPGA', in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, pp. 1–4. doi: 10.1109/FPL.2014.6927391.

Joshi, M. N. y Gawali, D. H. (2016) 'Floating point unit core for Signal Processing applications', in *2016 Online International Conference on Green Engineering and Technologies (IC-GET)*, pp. 1–6. doi: 10.1109/GET.2016.7916650.

Karlstrom, P., Ehliar, A. y Liu, D. (2008) 'High-performance, low-latency field-programmable gate array-based floating-point adder and multiplier units in a Virtex 4', *IET Computers & Digital Techniques*, 2(4), pp. 305–313. doi: 10.1049/iet-cdt:20070075.

Kaur, P. *et al.* (2015) 'Double Precision Floating Point Arithmetic Unit Implementation- A Review', *International Journal of Engineering Research & Technology (IJERT)*, 4(7), pp. 992–994. Available at: http: / /dx.doi.org/10.17577/IJERTV4IS070766.

Lasith, K. K. y Thomas, A. (2014) 'Efficient implementation of single precision floating point processor in FPGA', in *2014 Annual International Conference on Emerging Research Areas: Magnetics, Machines and Drives (AICERA/iCMMD)*, pp. 1–5. doi: 10.1109/AICERA.2014.6908269.

López, J., Restrepo, J. y Tobón, J. (2020) 'Parametric Decimal Division using Hardware Description Language', *Revista EIA*, 17, pp. 1–6. doi: https://doi.org/10.24050/reia.v17i33.1318.

Malkapur, S. B. y Rajput, R. P. (2020) 'Design of Generic Floating Point Pipeline Based Arithmetic Operation for DSP Processor', in *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*, pp. 1059–1064. doi: 10.1109/ICIRCA48905.2020.9182948.

Ramyarani, N. Subbiah, V. Deepa, P. (2019) 'Design of area-efficient IIR filter using FPPE', *Turkish Journal of Electrical Engineering & Computer Sciences*, 27(3), pp. 2321–2330. doi: :10.390.

Russinoff, D. M. (2019) *Formal Verification of Floating-Point Hardware Design*. first edit. Austin, TX, USA: Springer. doi: 10.1007/978-3-319-95513-1.

Sandoval-Ruiz, C. (2019) 'Modelo VHDL de Control Neuronal sobre tecnología FPGA orientado a Aplicaciones Sostenibles ', *Ingeniare. Revista chilena de ingeniería* . scielocl , pp. 383–395.

STALLINGS, W. (2005) *Organización y arquitectura de computadores*. Séptima Ed. Madrid: PEARSON EDUCACIÓN.

Veeranki, Y. R. and Nakkeeran, R. (2013) 'Spartan 3E Synthesizable FPGA Based Floating-Point Arithmetic Unit', *International Journal of Computer Trends and Technology (IJCTT)*, 4, pp. 751–755. Available at: www.ijcttjournal.org.

Wilton, C. Y. A. M. S. W. L. P. H. W. L. S. J. E. (2012) 'Optimizing Floating Point Units in Hybrid FPGAs', *IEEE TRANSACTIONS ON VERY*

LARGE SCALE INTEGRATION (VLSI) SYSTEMS, 20(7), pp. 1295–1303. doi: 10.1109/TVLSI.2011.2153883.

Zhang, B. y Zhao, J. (2017) 'Elementary Function Computing Method for Floating-Point Unit', *Journal of Signal Processing Systems*, 88(3), pp. 311–321. doi: 10.1007/s11265-016-1166-x.

Zoni, D., Galimberti, A. y Fornaciari, W. (2021) 'An FPU design template to optimize the accuracy-efficiency-area trade-off', *Sustainable Computing: Informatics and Systems*, 29, p. 100450. doi: https://doi.org/10.1016/j.suscom.2020.100450.