Instituto Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Diseño e implementación de una unidad aritmético-lógica de coma flotante para un procesador de aplicación específica

Informe de Proyecto de Graduación para optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatura

Diego Andrés Rodríguez Valverde

22 de Enero del 2016

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía, he procedido a indicar las fuentes mediante las respectivas citas bibliográficas.

En consecuencia, asumo la responsabilidad por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

Diego Rodríguez Valverde

Cartago, 22 de Enero, 2016 Firma del autor

Diego Andrés Rodríguez Valverde

Cédula: 1-1478-0688

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería en Electrónica
Proyecto de Graduación
Tribunal Evaluador

Proyecto de Graduación defendido ante el presente Tribunal Evaluador como requisit
optar por el título de Ingeniero en Electrónica con el grado académico de Licenciatu
Instituto Tecnológico de Costa Rica.

Miembros del Tribunal

Ing. Javier Pérez Rodríguez
Profesor Lector

Dr. Mauricio Muñoz Arias
Profesor Lector

Dr. Alfonso Chacón Rodríguez
Profesor Asesor

Los miembros de este tribunal dan fe que el presente proyecto de graduación para optar por el título de Licenciado en Ingeniería Electrónica, ha sido aprobado y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica.

Cartago, 22 de Enero, 2016

Resumen

En este trabajo se establecen las bases de una unidad aritmética de coma flotante (FPU) basada en el estándar IEEE 754. Para dicha unidad se desarrollaron los módulos de suma, resta y multiplicación utilizando arquitecturas de 32 y 64 bits. Las operaciones aritméticas se implementaron en una placa de prototipos (FPGA) utilizando el lenguaje de descripción de hardware (HDL) conocido como Verilog. Se realizó una verificación funcional de las operaciones a nivel de simulación y sobre la FPGA, lográndose resultados con un porcentaje de error menor a un 1% respecto a un modelo de referencia elaborado en el software GNU Octave. Finalmente, se presentan los resultados obtenidos de las simulaciones *Post Place & Route* referentes al consumo de potencia, reportes de tiempos y uso de recursos de hardware.

Palabras clave: FPU, IEEE 754, suma, resta, multiplicación, FPGA, HDL, RTL.

Abstract

In this work, a basic floating point arithmetic unit (FPU) based on the IEEE 754 standard is design and tested. Arithmetic operations of addition, subtraction and multiplication, using 32 and 64 bits architectures were implemented on a prototyping board (FPGA) using the hardware description language known as Verilog. Functional verification was done both on simulation and over the FPGA, the results obtained had a percent of error less than 1% respect to a reference model elaborated on the GNU Octave software. Finally, the *Post Place & Route* results are presented in relation to the power requirements, timing reports and the use of hardware resources.

Keywords: FPU, IEEE 754, addition, subtraction, multiplication, FPGA, HDL, RTL.

A mi querida familia

Agradecimientos

En primer lugar a Dios, quién me ha dado fuerzas para seguir adelante y me ha dado sabiduría ante situaciones difíciles que se han presentado a lo largo del camino.

A mi familia, por ser el motor de mi vida y por haberme hecho la persona que soy hoy. Agradezco por la educación académica que me brindaron, por los valores que me inculcaron, y por el apoyo brindado todos estos años.

Al Ing. Alfonso Chacón Rodríguez, quién ha sido un pilar en mi formación académica, agradezco la paciencia, los consejos y la confianza brindada durante la ejecución de este proyecto.

Agradezco a mi buen amigo el Ing. Carlos Salazar García, quién me demostró que con trabajo duro y determinación se pueden lograr grandes cosas. Gracias por todo el conocimiento compartido, la ayuda prestada, y el apoyo brindado hasta el final del proceso.

Finalmente, agradezco a las personas que de una u otra forma creyeron en este proceso.

Diego Andrés Rodríguez Valverde

Cartago, 22 de Enero, 2016

ÍNDICE GENERAL

Índice de Figuras

Índice de tablas

Lista de abreviaciones y símbolos vii

Abreviaciones

ASP	Procesador de aplicación específica
DCI Lab	Laboratorio de diseño de circuitos integrados
FPGA	Del inglés Field Programmable Gate Array
FPU	Del inglés Floating Point Unit
HDL	Del inglés Hardware Description Language

HMM	Del inglés Hidden Markov Models
SoC	Del inglés System on chip
RTL	Del inglés Register-Transistor Logic

Capítulo 1

Introducción

1.1 Entorno del proyecto

En el Instituto Tecnológico de Costa Rica, la Escuela de Ingeniería en Electrónica cuenta con el Laboratorio de Diseño de Circuitos Integrados o DCI Lab, en el cual se desarrollan proyectos de investigación y desarrollo en los que participan profesores y estudiantes.

Uno de estos proyectos de investigación es el SiRPA (Sistema de Reconocimiento de Patrones Acústicos), el cual forma parte de un proyecto de la Escuela de Ingeniería en Electrónica denominado Sistema Electrónico Integrado en Chip (SoC), cuya finalidad es detectar disparos de armas de fuego y de motosierras, mediante el reconocimiento de patrones acústicos [3]. El SoC pretende contribuir a la protección del medio ambiente a través de la detección de actividades de tala y caza ilegal en zonas protegidas del país.

El SiRPA cuenta con un bloque de clasificación encargado de calcular las probabilidades de que una señal acústica corresponda a un disparo de arma de fuego, al sonido de una motosierra, o a un sonido común del bosque [17]. Para llevar a cabo dicha función, la etapa de clasificación utiliza un modelo acústico conocido como Modelos Ocultos de Markov (HMM) [8], cuyas representaciones estadísticas requieren del procesamiento de datos en coma flotante.

La correcta implementación del HMM requiere de un procesador de aplicación específica (ASP) capaz de ejecutar operaciones aritméticas de coma flotante. Sobre esta necesidad de procesamiento aritmético se sitúa el desarrollo de la tesis.

1.2 Descripción del problema y justificación

El procesamiento de datos en coma flotante es un punto clave para la correcta implementación del HMM. En la actualidad, el DCI Lab cuenta con un módulo de FPU sintetizado en un lenguaje HDL, el cual se obtuvo del sitio web OpenCores [15]. Anteriormente, este coprocesador numérico se utilizó en una implementación en hardware del SiRPA, con el fin de evaluar el modelo acústico HMM [1]. Sin embargo, los resultados obtenidos de dicha implementación divergieron respecto a los resultados esperados. Esta situación hizo imposible clasificar los patrones acústicos de manera correcta.

La poca documentación existente para dicha unidad, y la falta de experiencia en el manejo de aritmética de coma flotante, llevó al DCI Lab a tomar la decisión de desarrollar una FPU propia. Esta unidad sería utilizada para implementarse en diferentes proyectos del laboratorio, incluido el SiRPA.

El coprocesador numérico a desarrollar, debía satisfacer los siguientes requerimientos:

- Estar basado en el estándar IEEE 754 [4], el cual es un estándar para el cómputo de aritmética en coma flotante.
- Utilizar una arquitectura de 32 y 64 bits.
- Poseer una interfaz que le permita integrarse a un procesador de aplicación específica.
- Ser diseñado e implementado usando un lenguaje HDL.
- Utilizar la menor cantidad de recursos de hardware posible.

El desarrollo de una FPU funcional supone un proceso a largo plazo. Por esta razón, el aporte de este trabajo se centrará en implementar las operaciones básicas de suma, resta y multiplicación en coma flotante, las cuales son suficientes para cubrir las necesidades de procesamiento del HMM, y servirán como punto de partida para la implementación futura de funciones aritméticas más complejas.

1.3 Síntesis del problema

¿Cómo implementar los módulos de suma, resta y multiplicación según el estándar IEEE 754, de modo que puedan integrarse a un procesador de aplicación específica, y ser desarrollados utilizando la menor cantidad de recursos posible?

1.4 Enfoque de la solución

Inicialmente, se llevó a cabo un proceso de investigación exhaustivo sobre aritmética de coma flotante basada en el estándar IEEE 754. Posteriormente, se discutió la manera de utilizar una versión reducida de dicho estándar, basándose en las características de precisión del SiRPA, y en la necesidad de utilizar la menor cantidad de recursos posible.

Una vez definido lo anterior, se procedió a desarrollar los módulos aritméticos utilizando una estrategia de diseño modular, a través de la cual se definió la interfaz de entrada-salida, y la estrategia de interconexión de los bloques que constituyen las diferentes operaciones aritméticas.

Posteriormente, los módulos aritméticos se implementaron utilizando el lenguaje HDL Verilog, y se verificaron utilizando un proceso de comparación entre los resultados obtenidos de simulaciones Post Place & Route, y los resultados obtenidos del software GNU Octave.

Por último, se integraron los módulos aritméticos al procesador de aplicación específica, sobre el cual se ejecutaron rutinas matemáticas complejas para corroborar el funcionamiento conjunto de ambas unidades. A este punto, los resultados fueron satisfactorios, por lo que se procedió a ejecutar el algoritmo HMM.

1.5 Meta

- Desarrollar un sistema de reconocimiento de patrones acústicos para la detección de disparos y motosierras en zonas protegidas del país.

1.6 Objetivos y estructura

1.6.1 Objetivo general

Diseñar una primera versión de FPU que cuente con un estándar propio para el cómputo de aritmética en coma flotante, que posea una interfaz de entrada-salida bien definida, que sea escalable, que permita ejecutar las operaciones aritméticas de suma, resta y multiplicación y que se pueda integrar a un procesador de aplicación específica.

1.6.2 Objetivos Específicos

- Definir una interfaz de comunicación entre el ASP y la FPU para el manejo de flujo de datos y excepciones.
- Diseñar e implementar en HDL las operaciones aritméticas de suma, resta, y multiplicación en coma flotante basándose en el estándar IEEE 754.

1.6.3 Estructura de la tesis

El capítulo 2 ofrece un resumen de los fundamentos teóricos utilizados para el desarrollo del proyecto. En el capítulo 3 se detalla el desarrollo de los módulos aritméticos, para los cuales se presentan diagramas de bloques y explicaciones asociadas a su funcionamiento. Además, se resume en una serie de pasos el algoritmo de control utilizado por la FSM. En el capítulo 4 se presentan los resultados obtenidos de la implementación de los módulos, y se realiza una discusión justificando cada uno. Finalmente, en el capítulo 5 se ofrecen

conclusiones del trabajo realizado, y recomendaciones a considerar para futuros proyectos.

Capítulo 2

Marco teórico

En este capítulo se presentan las bases teóricas necesarias para el desarrollo del proyecto. Los fundamentos que se pueden encontrar en esta sección son los referentes al estándar IEEE 754 para el manejo de aritmética en coma flotante, en particular los relativos a los algoritmos de suma, resta y multiplicación.

2.1 Estándar IEEE 754

2.1.1 Introducción al estándar IEEE 754

En los sistemas computacionales todos los valores con una parte fraccionaria diferente de cero, son almacenados como valores de coma flotante. Definir un método para representar estos números de modo que puedan ser interpretados por un computador es un proceso que requiere considerar aspectos como rango, precisión, eficiencia temporal, consumo de área, y el hecho de que cada número tenga una representación única.

Los diseñadores del estándar IEEE 754 le dieron solución a todas esas necesidades utilizando el método de notación científica [18]. La notación científica permite representar un número factorizándolo en dos partes; la primera corresponde a una magnitud, mientras que la segunda corresponde a una potencia. Por lo general, en ingeniería se utiliza la notación científica en base 10, donde la magnitud z del número que se desea representar es tal que $1 \leq z < 10$, y la potencia que se utiliza es el número 10. Por ejemplo, la representación del número 87564 usando notación científica en base 10 corresponde a $8{,}7564 \times 10^4$. La misma idea aplica para el estándar IEEE 754, con la excepción de que se utilizan potencias de 2, dada la naturaleza binaria de las computadoras actuales. Además, es importante mencionar que la representación binaria de números de coma flotante utiliza el formato signo-magnitud, en el cual el bit más significativo de la cadena representa el signo del número, y el resto de bits representan la magnitud (ver la figura 2.1).

Figura 2.1 Formato de bits para la representación signo-magnitud de números binarios

2.1.2 Representación binaria de un número en coma flotante

Un número binario en coma flotante es un arreglo de bits que se caracteriza por tener tres componentes: un *signo*, un *exponente* y una *mantisa* o parte fraccionaria. Cada uno de estos componentes posee un número de bits específico según el formato de precisión con el que se trabaje.

Para obtener el patrón de bits que representa un número en coma flotante, el producto magnitud-potencia (de la notación científica) debe definirse según la siguiente relación:

$$\mathrm{D} = (-1)^{\text{Bit de Signo}} \times (1 + \text{Fracción}) \times 2^{\text{Exponente Normalizado}-\text{Bias}} \qquad (1)$$

En esta ecuación se pueden observar cuatro factores de interés:

- **D**: valor decimal de determinado número en coma flotante.

- **Bit de Signo**: si el bit de signo es un valor lógico de 0, entonces el número es positivo $(-1)^0 = 1$. Si el bit de signo es un valor lógico de 1, entonces el número es negativo $(-1)^1 = -1$.

- **1 + Fracción**: según lo descrito en la sección 2.1, se sabe que este factor posee un valor numérico tal que $1 \leq (1 + \text{Fracción}) < 2$ (notación científica de base 2). Además note que solo el parámetro "Fracción" varía $(0 \leq \text{Fracción}) < 1$), por lo que solo este valor se representa como una cadena de bits.

- **Exponente Normalizado**: este valor es un número entero y se encuentra normalizado. Cuando un número se representa en notación científica, la potencia de 2 puede estar elevada a un exponente positivo o negativo. En un sistema computacional este exponente debería ser almacenado utilizando una representación en complemento a 2 (método más común para representar cantidades enteras con signo). Sin embargo, para fines de comparación entre valores, comparar valores en complemento a 2 es más complejo que comparar valores que no poseen signo, por esta razón los diseñadores del estándar

decidieron sumar un valor constante al exponente para ubicarlo independientemente de su signo en un rango de valores positivos y poder llevar a cabo comparaciones dónde únicamente se considera la magnitud de los números. Al valor constante que se suma al exponente se le conoce como sesgo, y a la suma del exponente y el valor de sesgo se le conoce como exponente normalizado, estas cantidades se relacionan según la ecuación (3). El exponente normalizado se representa como una cadena de bits.

$$\text{Exponente Normalizado} = \text{Exponente} + \text{Bias} \qquad (3)$$

Una vez que se tiene la representación binaria del signo, de la parte fraccionaria, y del exponente, se construye una cadena de bits de tamaño fijo como se muestra en la figura 2.2. Esta cadena de bits es la representación binaria del número en coma flotante utilizando el estándar IEEE 754.

Figura 2.2 Acomodo de bits para representación binaria de un número en coma flotante utilizando el estándar IEEE 754

La cantidad de bits que poseen los campos del exponente y la mantisa, define el rango de números representables y su precisión respectivamente. En el estándar IEEE 754 la cantidad de bits que se asigna a cada uno de los campos está definida por los formatos de precisión. Estos formatos también definen el valor del sesgo con el cual se normaliza el exponente.

Además, el estándar posee patrones de bits predeterminados para representar números especiales, entre estos se encuentran el cero, los infinitos (positivo y negativo), los números subnormales, y aquellos números que son imposibles de representar.

2.1.3 Formatos de precisión del estándar IEEE 754

Los formatos de precisión (o formatos de almacenamiento) especifican la forma en que se almacena un valor de coma flotante en memoria, y se utilizan para representar un subconjunto finito de números reales. Se caracterizan por la ubicación del punto decimal, el rango del exponente, la precisión y el hecho de que cada uno puede representar únicamente un determinado conjunto de números en punto flotante.

En el estándar IEEE 754 existen formatos binarios, formatos decimales, y formatos extendidos. Los formatos decimales y extendidos no forman parte del desarrollo del documento, por lo tanto no se hace una mención significativa de los mismos.

Los formatos binarios también son conocidos como formatos de precisión básicos y se caracterizan al igual que los formatos decimales y extendidos por tener un ancho de palabra definido. Los que se tratan en este documento corresponden al formato de precisión simple y al formato de precisión doble [4].

2.1.3.1 Formato de precisión simple

Este formato posee un ancho de palabra de 32 bits, distribuido en los tres campos especificados en la figura 2.2. En este formato se utiliza un bit para representar el signo del número, ocho bits conforman el campo del exponente normalizado, y se dedican 23 bits para representar la mantisa o parte fraccionaria, tal y como se muestra en la figura 2.3.

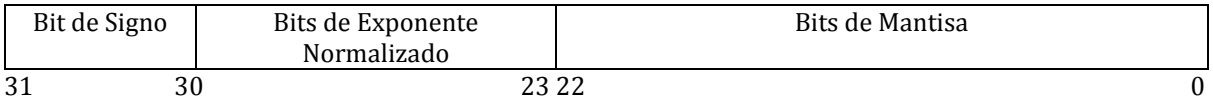

Figura 2.3. Ordenamiento de bits en el formato de precisión simple.

La normalización del exponente en este formato de precisión se lleva a cabo con un valor de sesgo de 127.

2.1.3.2 Formato de precisión doble

Este formato posee un ancho de palabra de 64 bits, al igual que en el formato de precisión simple los bits se distribuyen en los tres campos de la figura 2.2. Se utiliza un bit para representar el signo del número, once bits constituyen el campo del exponente normalizado, y 52 bits representan la parte fraccionaria, tal y como se muestra en la figura 2.4.

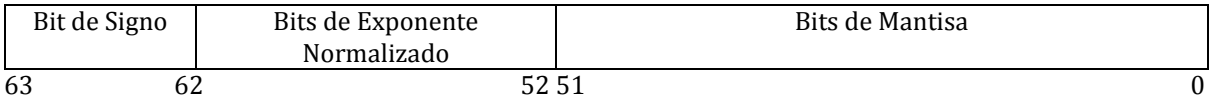

Figura 2.4. Ordenamiento de bits en el formato de precisión doble.

La normalización del exponente en este formato de precisión se lleva a cabo con un valor de sesgo de 1023.

2.1.3.3 Patrones de bits de valores representables por el formato IEEE 754

En las tablas 2.1 y 2.2, se muestra la relación existente entre los patrones de bits y el valor decimal de un número representado con el estándar IEEE 754. Dependiendo del patrón de bits, los números se pueden clasificar como números normales o subnormales [4], o representar valores numéricos especiales.

Los formatos de precisión definen un rango de valores representables a través de la ecuación (1); estos valores se conocen como números normales. Entre el valor mínimo de los números normales y el número cero, existe un hueco o vacío de valores que por razones de precisión no pueden representarse. Para llenar este vacío se utilizan los números subnormales (ver figura 2.5).

El conjunto de números subnormales se utiliza para representar números menores al mínimo número normal representable. Estos números son de vital importancia en aplicaciones que requieren de mucha precisión alrededor del valor cero. Sin embargo, debido a las características de precisión del SiRPA en este proyecto no fue necesario el uso de este conjunto de valores. De igual manera, en las tablas 2.1 y 2.2 se presenta la ecuación que permite relacionar la representación binaria y decimal de estos números.

0	VACÍO Dominio de los números subnormales	Mínimo valor normal representable

Figura 2.5. Representación del vacío que existe entre el valor cero y el mínimo valor normal representable.

Finalmente, se introduce el concepto de bit implícito. El bit implícito debe su nombre a que no aparece en la codificación de palabra utilizada por los formatos de precisión. Este corresponde al bit más significativo del campo de mantisa, y su valor siempre es un 1 para los números normales. La invariabilidad en el valor lógico de este bit fue la razón por la cual los diseñadores del estándar decidieron no representarlo dentro de los 32 y 64 bits que manejan los formatos de precisión (estrategia para ahorrar recursos de hardware).

Sin embargo, el bit implícito se vuelve relevante cuando se desean ejecutar operaciones aritméticas, en dónde debe incluirse si se desean obtener resultados correctos.

De lo anterior, se puede derivar que la cadena de bits que describe la precisión de un número posee en realidad un ancho de 24 bits para el formato de precisión simple, y de 53 bits para el formato de precisión doble.

Tabla 2.1. Interpretación de patrones de bits del estándar IEEE 754 según el formato de precisión simple.

Patrón de bits para formato de precisión simple	Valor numérico / **Descripción
Tipo: Números Normales Signo = 0 ó 1 00_h < Exponente Normalizado < ff_h Fracción = Cualquier combinación de bits	$(-1)^{\text{Bit de Signo}} \times 2^{\text{Exponente Normalizado}-127} \times (1 + \text{Fracción})$
Tipo: Números Subnormales Signo = 0 ó 1 Exponente Normalizado = 00_h Fracción ≠ 00000_h	$(-1)^{\text{Bit de Signo}} \times 2^{-126} \times (0 + \text{Fracción})$
Tipo: Caso Especial - Cero con Signo Signo = 0 ó 1 Exponente Normalizado = 00_h Fracción = 000000_h	$(-1)^{\text{Bit de Signo}} \times 0.0$
Tipo: Caso Especial -Infinito Positivo Signo = 0 Exponente Normalizado = ff_h Fracción = 00_h	+INF
Tipo: Caso Especial - Infinito Positivo Signo = 1 Exponente Normalizado = ff_h Fracción = 00_h	-INF

Tipo: Caso Especial - Números no representables (Not a Number) Signo = 0 ó 1 Exponente Normalizado = ff_h Fracción ≠ 00000_h	NaN

Tabla 2.2 Interpretación de patrones de bits del estándar IEEE 754 según el formato de precisión doble.

Patrón de bits para formato de precisión simple	Valor numérico / **Descripción
Tipo: Números Normales Signo = 0 ó 1 000_h < Exponente Normalizado < $7ff_h$ Fracción = Cualquier combinación de bits	$(-1)^{\text{Bit de Signo}} \times 2^{\text{Exponente Normalizado}-1023} \times (1 + \text{Fracción})$
Tipo: Números Subnormales Signo = 0 ó 1 Exponente Normalizado = 000_h Fracción ≠ 0000000000000_h	$(-1)^{\text{Bit de Signo}} \times 2^{-1022} \times (0 + \text{Fracción})$
Tipo: Caso Especial - Cero con Signo Signo = 0 ó 1 Exponente Normalizado = 000_h Fracción = 0000000000000_h	$(-1)^{\text{Bit de Signo}} \times 0.0$
Tipo: Caso Especial - Infinito Positivo Signo = 0 Exponente Normalizado = $7ff_h$ Fracción = 0000000000000_h	+INF
Tipo: Caso Especial - Infinito Negativo Signo = 1 Exponente Normalizado = $7ff_h$ Fracción = 0000000000000_h	-INF

Tipo: Caso Especial - Número no representable (Not a Number) Signo $= 0$ ó 1 Exponente Normalizado $= 7\text{ff}_\text{h}$ Fracción $\neq 0000000000000_\text{h}$	NaN

2.1.2 Rango y precisión del estándar IEEE 754

2.1.4.1 Rango decimal representable por los formatos de precisión

Tomando como referencia los conceptos ligados al estándar, se puede demostrar que, el rango de valores normales positivos en formato decimal representables por el formato de precisión simple, se encuentra entre 1,17549435e-38 y 3,40282347e+38, mientras que para el formato de precisión doble este rango de valores corresponde a números ubicados entre 2,2250738585072014e-308 y 1,7976931348623157e+308. El rango representable de valores negativos es exactamente el mismo que el de los números positivos; esta es una característica de simetría alrededor del valor cero propia del formato signo-magnitud que utiliza el estándar.

La obtención de dichos rangos (y de cualquier número normal) se realiza utilizando la ecuación (1), y utilizando la tabla 2.1 o 2.2, según el formato de precisión con el que se trabaje. Por ejemplo, para el formato de precisión simple un número normal debe cumplir en su campo de exponente la siguiente condición:

$$00_\text{h} < \text{Exponente Normalizado} < \text{ff}_\text{h}$$

De lo anterior se puede inferir que el patrón de bits que representa el valor normal mínimo positivo viene dado por 00800000_h, dónde:

$$\text{Bit de Signo} = 0$$
$$\text{Bits de Exponente Normalizado} = 01_\text{h} = 1_{10}$$

$$\text{Bits de Mantisa} = 000000_h = 0_{10}$$

Sustituyendo los equivalentes decimales de los tres campos en la ecuación (1) se obtiene lo siguiente:

$$\mathrm{D} = (-1)^0 \times (1 + 0) \times 2^{1-127} = 1.17549435\mathrm{e} - 38$$

De manera similar, el patrón de bits que representa el valor normal máximo positivo viene dado por 7f7fffff$_h$, dónde:

$$\text{Bit de Signo} = 0$$
$$\text{Bits de Exponente Normalizado} = \mathrm{fe_h} = 254_{10}$$
$$\text{Bits de Mantisa} = 7\mathrm{ffff}f_h = 0{,}9999998807907104_{10}$$

El valor decimal que se obtiene sustituyendo las representaciones decimales de los diferentes campos en la ecuación (1) es 3,40282347e+38. El mismo procedimiento se puede seguir para derivar el rango representable del formato de precisión doble.

En la tabla 2.3 se presenta un resumen de los valores máximos y mínimos que pueden ser representados por ambos formatos de precisión.

Tabla 2.3 Patrones de bits correspondientes a los valores normales positivos mínimos y máximos representables en ambos formatos de precisión.

Formato de Precisión Simple		Formato de Precisión Doble	
Valor Normal Mínimo Positivo	Valor Normal Máximo Positivo	Valor Normal Mínimo Positivo	Valor Normal Máximo Positivo
00800000_h	$7f7fffff_h$	0010000000000000_h	$7fefffffffffffff_h$
Valor Normal Mínimo Negativo	Valor Normal Máximo Negativo	Valor Normal Mínimo Negativo	Valor Normal Máximo Negativo
$ff7fffff_h$	80800000_h	$ffefffffffffffff_h$	8010000000000000_h

2.1.4.2 Dígitos decimales precisos en el estándar IEEE 754

Esta interrogante se puede contestar utilizando un proceso de conversión de un número decimal a su correspondiente formato binario IEEE 754, y luego convirtiéndolo de nuevo a formato decimal. Este procedimiento se explica a través del siguiente ejemplo:

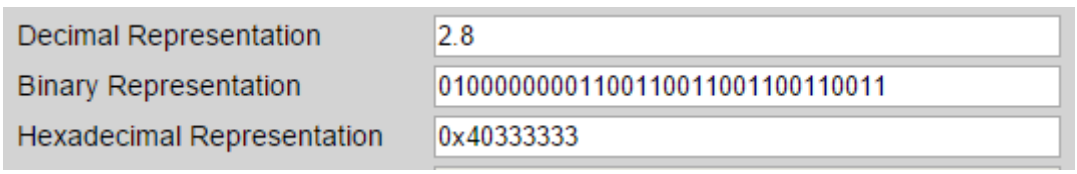

Figura 2.6 Representación en formato IEEE 754 del valor numérico 2,8

En la figura 2.6 se muestra el valor decimal 2,8 junto a su representación binaria y hexadecimal en formato de precisión simple. La idea es transformar la representación binaria de nuevo a su representación decimal, para esto hay que separar los tres campos que conforman la palabra y encontrar sus equivalentes decimales tal y como se muestra a continuación:

$$\text{Bit de Signo} = 0$$
$$\text{Bits de Exponente} = 80_{\text{h}} = 128_{10}$$
$$\text{Bits de Mantisa} = 333333_{\text{h}} = 0{,}3999999762_{10}$$

Ahora se sustituyen los equivalentes decimales de los tres campos en la ecuación (1) para obtener el equivalente decimal de la representación binaria,

$$\text{D} = (-1)^0 \times (1 + 0{,}3999999762\ ) \times 2^{128-127} = 2{,}799999952316284$$

Otro ejemplo se ilustra en la figura 2.7.

Decimal Representation	150.42
Binary Representation	01000011000101100110101110000101
Hexadecimal Representation	0x43166b85

Figura 2.7. Representación en formato IEEE 754 del valor numérico 150,42

Siguiendo un procedimiento similar al del ejemplo anterior se obtiene que el valor decimal de la representación binaria de la figura 2.7, es la siguiente:

$$\text{D} = 150{,}4199981689453$$

La diferencia entre los valores mostrados en los ejemplos se muestra a continuación:

$$2{,}8 - 2{,}799999952316284 = 0{,}000000047683716$$
$$150{,}42 - 150{,}4199981689453 = 0{,}0000018310547$$

La representación binaria de 2,8 utilizando el estándar IEEE 754 es ocho órdenes de magnitud menor que el valor original, mientras que en el caso de 150,42 la representación binaria es seis órdenes de magnitud menor. Estos resultados son un indicador de cuantos dígitos decimales se pueden representar en forma precisa, siendo ocho dígitos decimales en el caso del número 2,8, y de seis dígitos decimales en el caso del número 150,42. Por lo general en el estándar IEEE 754 se puede asegurar que la cantidad de dígitos significativos que se pueden representar de manera precisa está entre

seis y nueve dígitos para el formato de precisión simple, y entre quince y diecisiete dígitos para el formato de precisión doble [4].

La conversión entre formatos en la mayoría de casos es inexacta dada la diferencia existente entre la representación numérica en base 10 y la representación numérica en base 2. Sin embargo, existen casos en el estándar IEEE 754 dónde la conversión es exacta, es decir dónde todos los dígitos significativos de un número se pueden representar de manera exacta.

2.2 Operaciones aritméticas en el estándar IEEE 754

El estándar IEEE 754 define procedimientos para el manejo de aritmética en coma flotante. Entre estos procedimientos se encuentran los algoritmos para ejecutar operaciones básicas entre las que se encuentran la suma, la resta, la multiplicación, la división, y la operación de cálculo de residuo. Además, se definen estrategias de redondeo, y el manejo de casos excepcionales que pueden ocurrir durante el proceso de ejecución de alguna de las operaciones.

En esta sección se discuten las operaciones de suma, resta y multiplicación (operaciones que se trabajaron en este proyecto), además se hace mención de los diferentes modos de redondeo que utiliza el estándar, así como casos excepcionales y su respectivo manejo.

2.2.1 Modos de redondeo

El proceso de ejecución de operaciones aritméticas que utilizan el estándar IEEE 754 requiere del uso de bits adicionales para propósitos de precisión [16]. Estos bits corresponden al bit implícito y a tres bits extra llamados bit de guarda, bit de redondeo y el sticky bit. Para propósitos de este documento, solo se utilizaron los bits de redondeo y guarda.

El bit implícito corresponde al bit más significativo del campo fraccionario y se utiliza para representar de manera correcta la relación establecida por la ecuación (1), mientras que el bit de guarda y el bit de redondeo se ubican a la derecha del bit menos significativo de la mantisa en forma de ceros, y se utilizan para evitar la pérdida de precisión en los diferentes procesos de normalización requeridos por operaciones aritméticas, y para tomar decisiones de redondeo.

El estándar define cuatro modos de redondeo:

- Redondeo hacia +infinito: el resultado se redondea por exceso hacia el infinito positivo.
- Redondeo hacia -infinito: el resultado se redondea por defecto hacia el infinito negativo.
- Redondeo hacia cero: si el resultado es un número positivo redondea hacia abajo, si el resultado es un número negativo redondea hacia arriba. Nótese que lo anterior corresponde a una función de truncamiento. Este modo de redondeo es el más utilizado.
- Redondeo al par más próximo: el resultado se redondea al valor para más cercano. Si el resultado se encuentra igualmente distanciado de dos valores pares, se redondea hacia aquel valor que posee un 0 como bit menos significativo.

Para el desarrollo del proyecto solo se utilizaron los primeros tres modos de redondeo.

El redondeo se lleva a cabo mediante la adición de un uno al bit menos significativo de la mantisa (sin contar los bits de guarda y redondeo). Un ejemplo del proceso de modificación del campo de mantisa en función del bit de guarda y el bit de redondeo se muestra en la tabla 2.4. Para este ejemplo se considera un formato dónde el campo de mantisa posee un ancho de cuatro bits, y solo se ejemplifican algunos casos (los bits enmarcados en color rojo corresponden al bit de redondeo y al bit de guarda respectivamente).

Tabla 2.4. Modificación de la mantisa en el proceso de redondeo según el bit de guarda y redondeo.

Valor actual de la Mantisa	Hacia +INF Nuevo valor de Mantisa	Hacia –INF Nuevo valor de Mantisa	Hacia cero Nuevo valor de Mantisa
+0001.00	0001	0001	0001
+0010.01	0011	0010	0010
+0011.11	0100	0011	0011
+0100.10	0101	0100	0100
-1010.01	1010	1011	1010
-1000.11	1000	1001	1000
-0100.00	0100	0100	0100
-1110.10	1110	1111	1110

2.2.2 Excepciones

Una excepción es una condición de interrupción que ocurre cuando se lleva a cabo una operación aritmética que incurre en un resultado que carece de sentido (calcular la raíz cuadrada de un número negativo). Toda excepción debe indicarse en el momento en que

se presenta, para hacerlo se establecen banderas de estado que registran la ocurrencia de dichas condiciones.

El estándar maneja cinco tipos de excepciones, que se resumen en la tabla 2.5.

Tabla 2.5. Excepciones del Estándar IEEE 754

Tipo de Excepción	Descripción	Algunos operaciones generan la excepción	Resultado Generado
Operación Inválida	Llevar a cabo una operación aritmética dónde alguno de los operandos o ambos produzcan resultados no aceptados.	$0 \times \pm\infty$ $0 \div 0$ Raíz cuadrada de números negativos $\infty - \infty$ $-\infty + \infty$ Logaritmo de un número negativo	NaN
Desborde	La magnitud del resultado es más grande que la magnitud del mayor número que puede representar el formato.	Para el formato de precisión simple, cualquier operación que genere un resultado mayor a 3,40282347e+38 Para el formato de precisión doble, cualquier operación que genere un resultado mayor a 1,7976931348623157e+308.	$\pm\infty$
Sub-desborde	La magnitud del resultado es más pequeña que la magnitud del menor número que puede representar el formato.	Para el formato de precisión simple, cualquier operación que genere un resultado menor a 1,17549435e-38 Para el formato de precisión doble, cualquier operación que genere un resultado menor a 2,2250738585072014e-308	Número subnormal o cero

Operación Inexacta	El resultado redondeado de determinada operación es diferente del resultado exacto.	Cualquier operación cuyo resultado después del proceso de redondeo sea diferente al valor exacto.	El resultado de la operación

Un aspecto particular ocurre cuando se da una excepción de sub-desborde (underflow). Existen varias maneras de manejar esta condición: la más común es conocida como almacenamiento de cero y consiste precisamente en representar el resultado de la operación como un cero. Sin embargo, el método preferido por el estándar IEEE 754 se conoce como sub-desborde gradual, que consiste en representar con números subnormales resultados que sean menores al mínimo valor normal representable con el fin de proveer de mayor precisión a los diferentes procesos aritméticos. En este proyecto se utilizó el método de almacenamiento de cero para el manejo de este tipo de excepción.

Para el desarrollo de este proyecto solo se consideraron las condiciones de sub-desborde y desborde (overflow).

2.2.3 Suma, resta y multiplicación según el estándar IEEE 754

Dado que la operación resta es en realidad una suma de números con el signo opuesto, el algoritmo que se explica en esta sección es válido para ambas operaciones. Este algoritmo sigue una metodología similar a la suma (y resta) en formato decimal de dos números representados en notación científica. El algoritmo se presenta como diagrama de flujo en la figura 2.8.

De manera similar, el proceso de multiplicación es parecido al que se lleva a cabo en formato decimal entre dos números representados en notación científica. Este algoritmo se muestra en la figura 2.9.

Cabe destacar que los algoritmos que se muestran en esta sección, están basados en los algoritmos básicos que pone a disposición el estándar IEEE 754, y son los que se utilizaron para el desarrollo del proyecto[6].

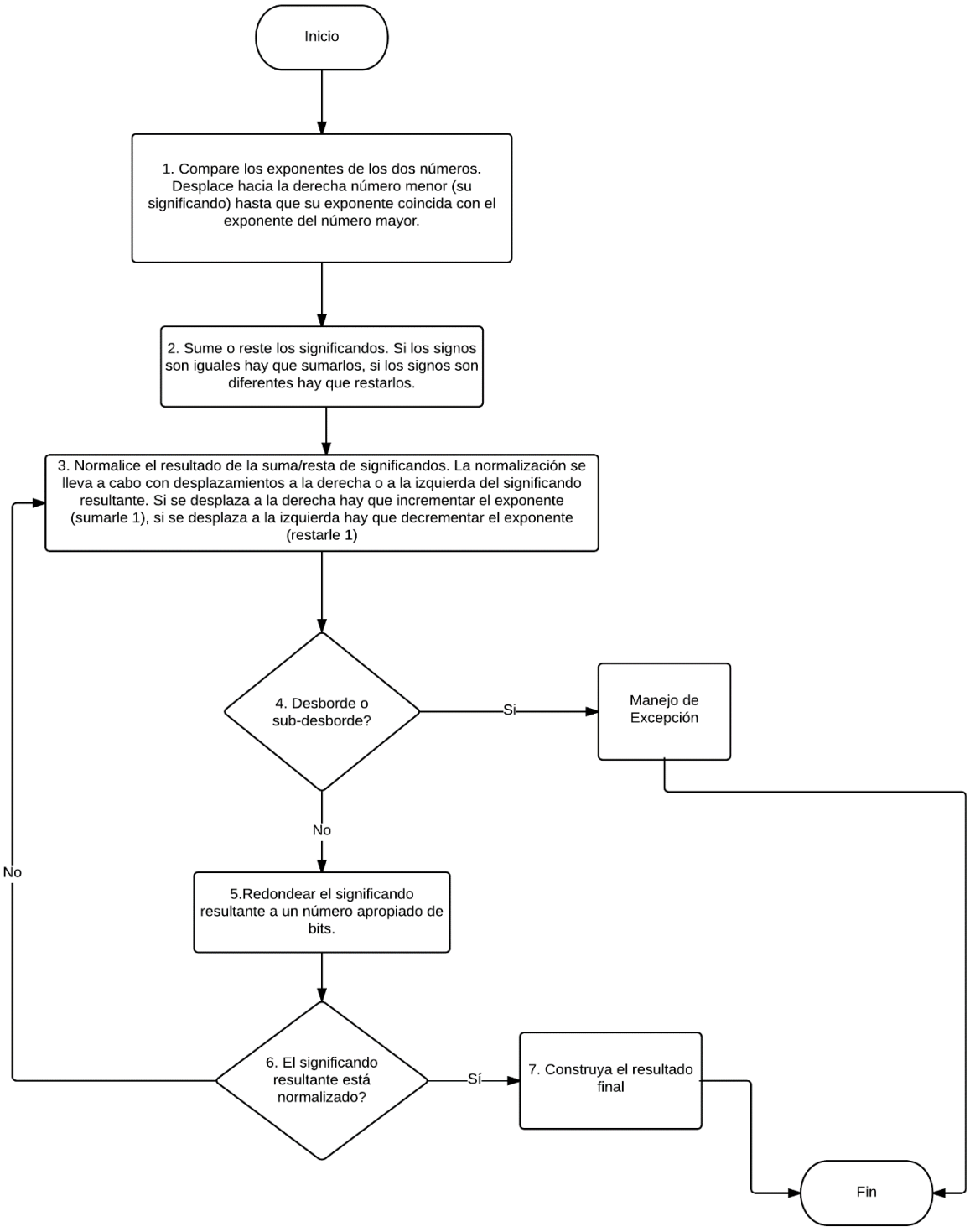

Figura 2.8. Diagrama de flujo del procedimiento para llevar a cabo la suma binaria de dos números en formato IEEE 754

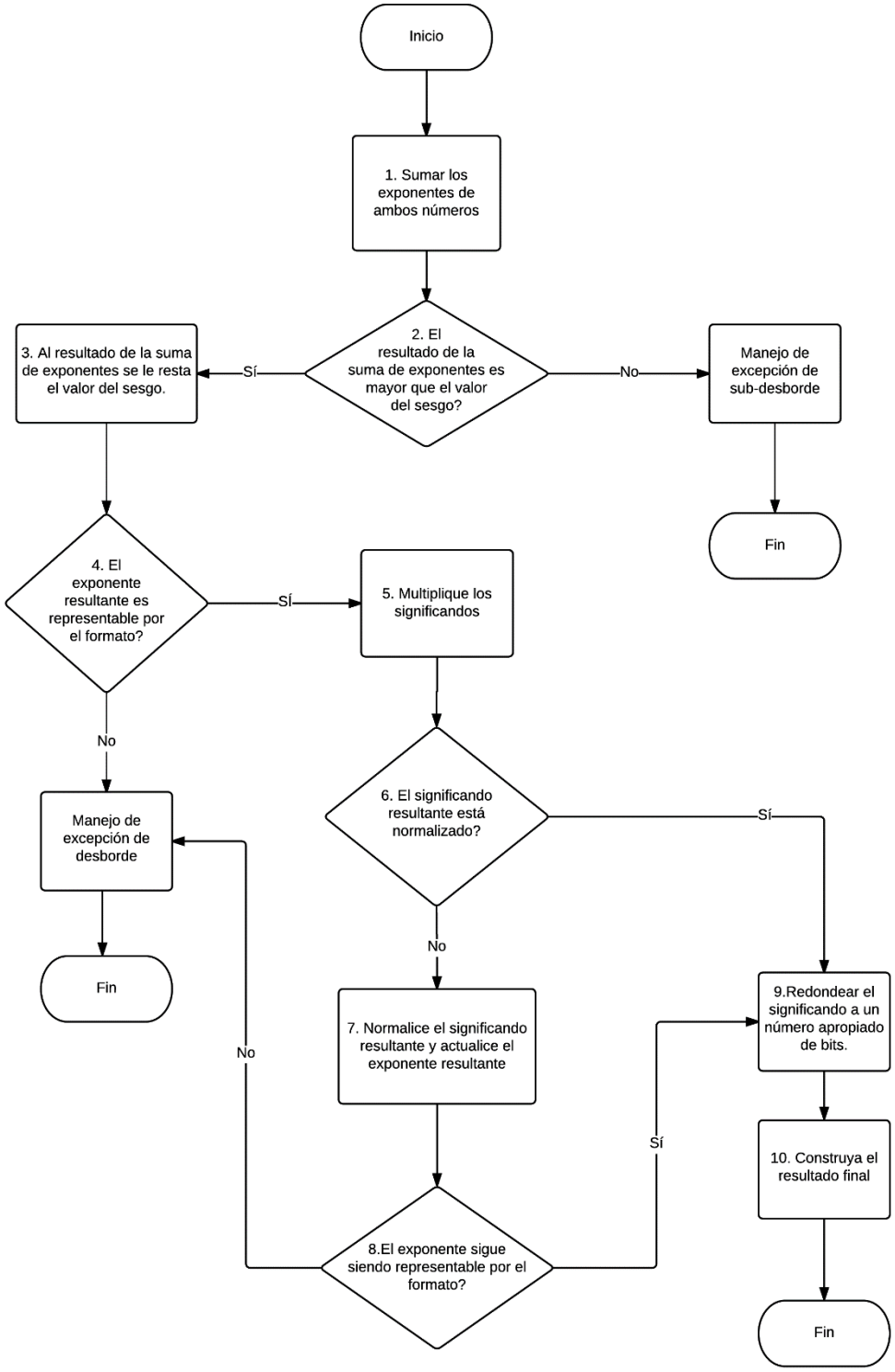

Figura 2.9. Diagrama de flujo del procedimiento para llevar a cabo la multiplicación binaria de dos números en formato IEEE 754

Capítulo 3

Diseño e implementación en hardware de los módulos aritméticos

En este capítulo se discuten aspectos ligados al diseño e implementación de las operaciones aritméticas de suma, resta y multiplicación en coma flotante.

3.1 Generalidades

Los módulos aritméticos desarrollados utilizan principios establecidos por el estándar IEEE 754, sin embargo no son compatibles en su totalidad con el mismo. La razón de la incompatibilidad se debe a decisiones de diseño tomadas por el DCI Lab en base a las necesidades del SiRPA. Algunas de estas decisiones fueron:

- No implementar todos los modos de redondeo: los valores que procesan los módulos aritméticos no son precisos (estos valores provienen de otros módulos que introducen errores de precisión).
- No manejar números subnormales: el algoritmo HMM no requiere tanta precisión alrededor del valor cero.
- Considerar solo algunos casos excepcionales: la excepción de operación inexacta no se utiliza dado que los valores que se procesan ya poseen errores de precisión, mientras que la excepción de operación inválida es un proceso que se controla desde las rutinas de software que se corren en el procesador de aplicación específica desarrollado en paralelo con este proyecto.
- Manejar solo dos formatos de precisión: el HMM no requiere utilizar formatos de precisión con un ancho de palabra mayor a 64 bits.

Las operaciones aritméticas se realizan siguiendo el ordenamiento Data_X $\pm$ Data_Y, y Data_X $\times$ Data_Y.

Finalmente, respecto a la metodología de diseño e implementación de los módulos aritméticos se utilizó una estrategia de diseño modular de arriba hacia abajo.

3.2 Módulo de suma y resta

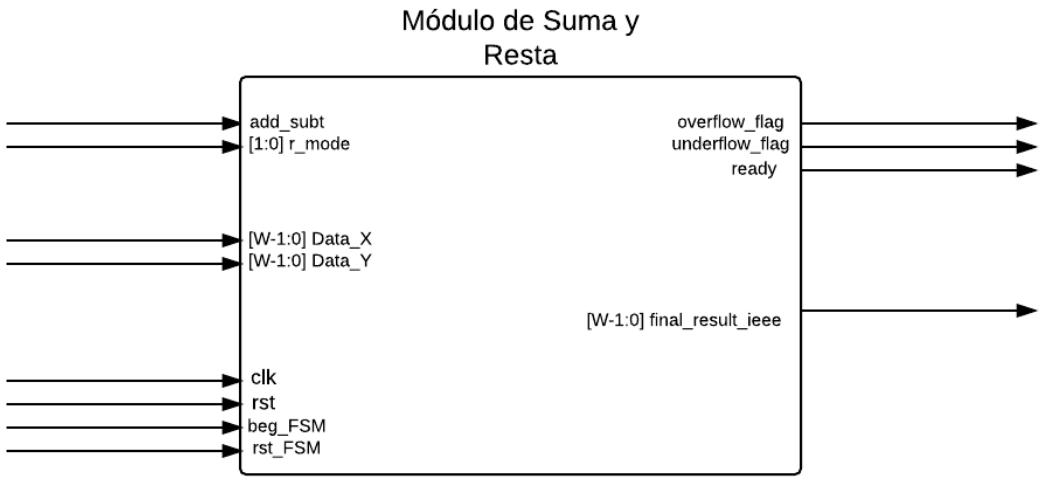

Figura 3.1. Diagrama de entradas y salidas del módulo de suma-resta

El módulo de suma-resta realiza los procesos de adición y sustracción en coma flotante según el estándar IEEE 754. Para llevar a cabo dichos procedimientos se propone el diagrama de entradas-salidas mostrado en la figura 3.1. La estructura interna de este módulo se presenta en la figura 3.2.

Las señales de entrada del módulo se enuncian a continuación:

- **add_subt:** ofrece información al sistema referente a la operación aritmética que se debe ejecutar. Se utiliza un valor lógico de 1 para efectuar la resta y un valor lógico de 0 para la suma.
- **r_mode:** establece el modo de redondeo con el que se va a trabajar. Esta codificación viene dada según las siguientes secuencias de bits:

    - 00 : redondeo hacia cero
    - 01: redondeo hacia –infinito
    - 10: redondeo hacia +infinito
    - 11: no se utiliza

- **Data_X, Data_Y:** corresponden a los números en formato IEEE 754 sobre los cuales se va a llevar a cabo la operación aritmética de suma o resta.
- **clk:** reloj del sistema, 100 MHz (Placa FPGA Artix-7 Nexys 4).
- **rst:** reinicio de la FSM encargada de controlar el flujo de datos.
- **beg_FSM:** indica al sistema en que momento debe iniciar el proceso de suma o resta.
- **rst_FSM**: forma en que un sistema externo le comunica al módulo de suma-resta que ha recibido con éxito el resultado de la operación.

Las señales de salida corresponden a:

- **overflow_flag:** informa sobre la ocurrencia de un caso excepcional de desborde durante el proceso de ejecución del algoritmo.
- **underflow_flag:** ocurrencia del caso excepcional de sub-desborde.
- **ready:** indica la finalización del algoritmo, el resultado está listo para ser recuperado.
- **final_result_ieee:** resultado final de la operación aritmética según el estándar IEEE 754.

3.2.1 Descripción de los bloques que conforman el módulo aritmético de suma-resta

En esta sección se discuten los diferentes bloques de la figura 3.2. Para cada bloque (a excepción de la FSM) se presenta un diagrama RTL y se describe su rol durante la ejecución del algoritmo. Para la máquina de estados finitos, se presenta una descripción general de las funciones que ejecuta a través de una serie de pasos.

Se recomienda entender el código de colores presentado en la esquina superior derecha de la figura 3.2, cuyo objetivo es generar una mayor comprensión respecto a la interconexión de bloques. Además, se aclara que las señales rotuladas como **W**, **W_Exp**, y **W_Sgf** son dependientes del formato de precisión con el que se trabaja, en dónde W corresponde al ancho total de la palabra, **W_Exp** al ancho de bits del exponente y **W_Sgf** al ancho de bits de la mantisa.

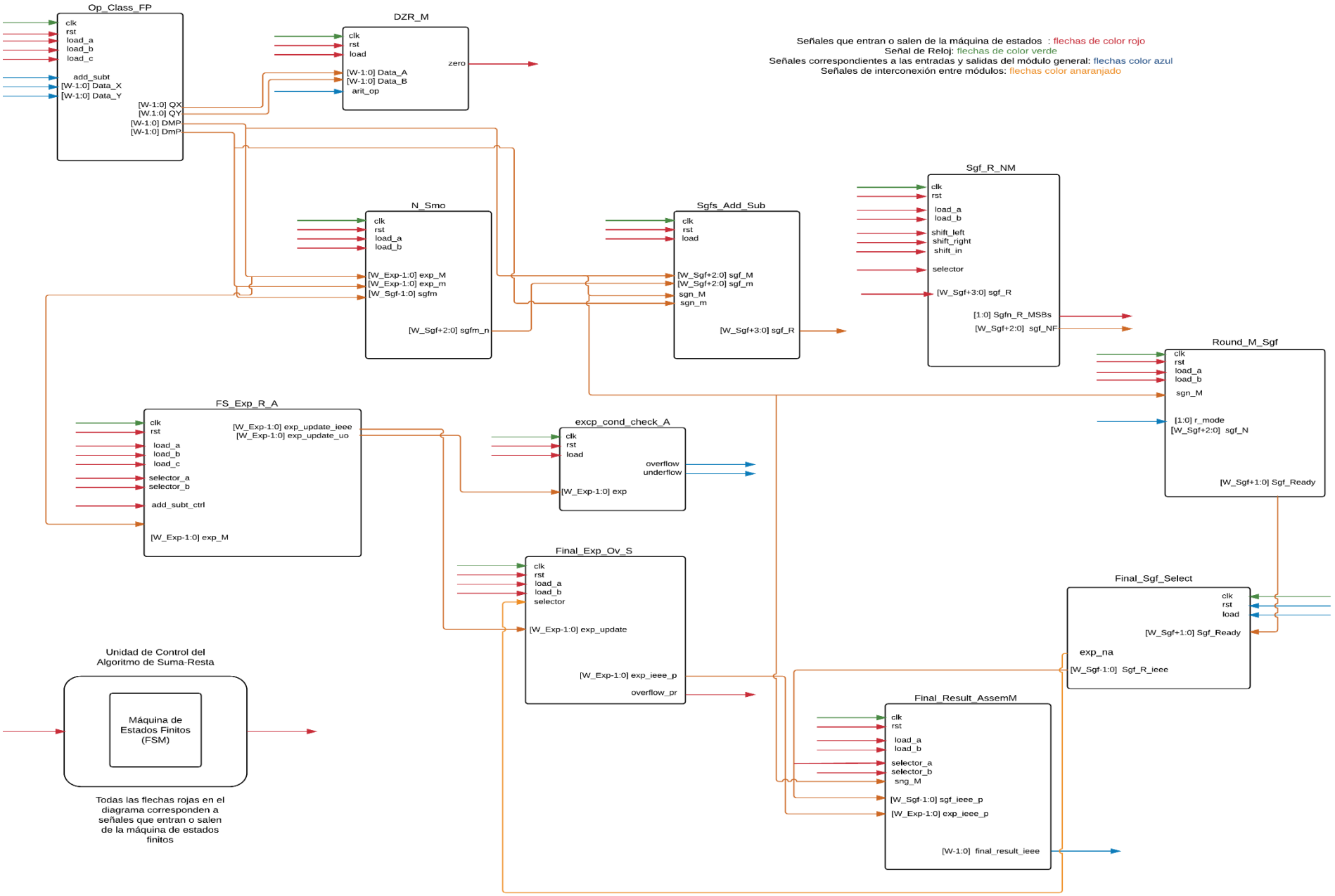

Figura 3.2.Arquitectura completa de flujo de datos para la unidad de suma-resta en coma flotante.

3.2.1.1 Clasificación de operandos (Op_Class_FP)

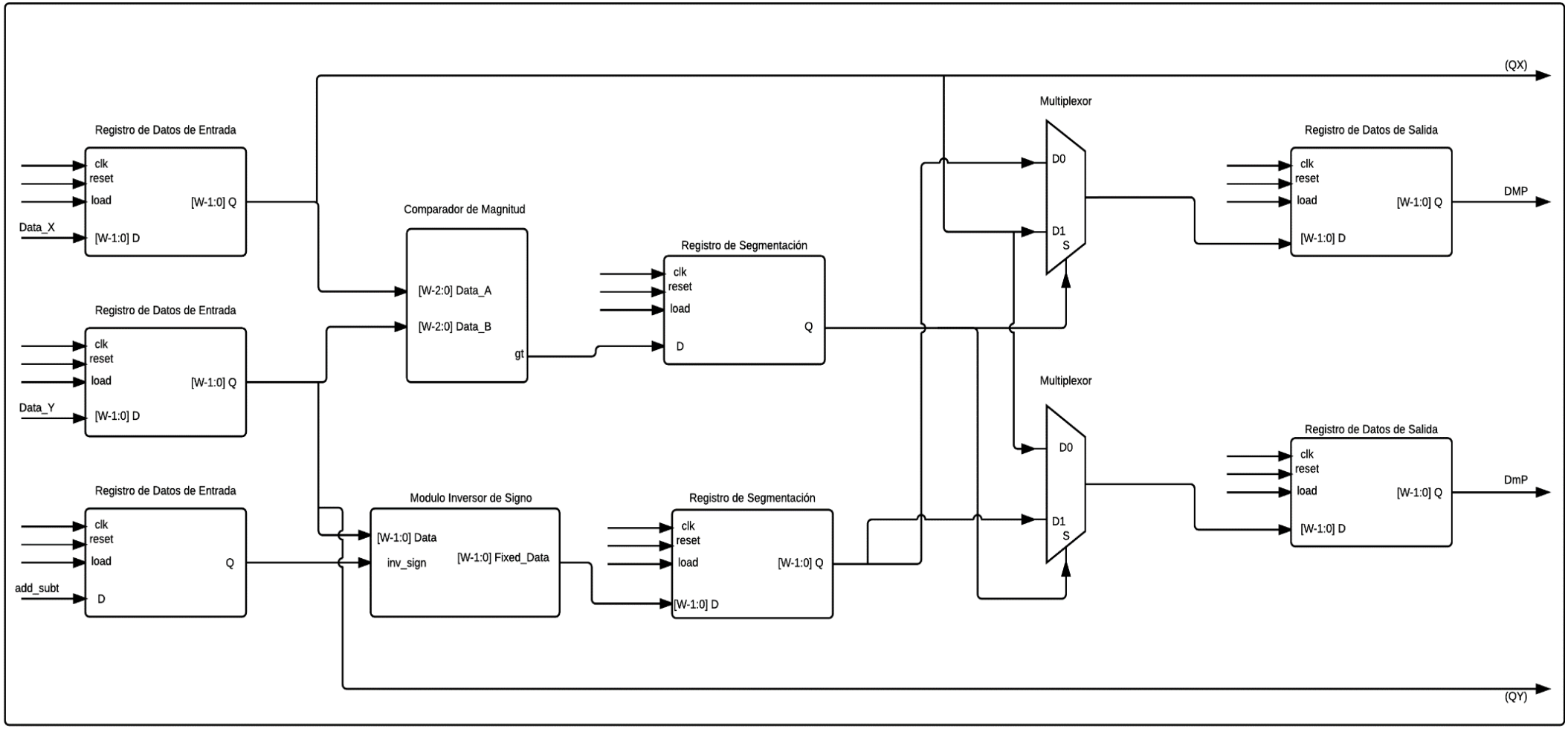

Figura 3.3. Diagrama RTL del bloque Op_Class_FP

El módulo clasificador de operandos cumple con dos funciones:

- Ajustar el signo del segundo operando si la operación aritmética a realizar es una resta.
- Clasificar los operandos en un operando mayor y un operando menor.

Para llevar a cabo estos procedimientos se diseñó la estructura mostrada en la figura 3.3.

El bloque recibe las señales **Data_X**, **Data_Y** y **add_subt** provenientes del ASP. Cuando comienza la ejecución del algoritmo, estas señales son almacenadas en tres registros de entrada. Luego de este proceso de carga, los operandos y el operador se propagan a través de bloques de lógica combinacional, que son respectivamente un comparador de magnitud y un inversor de signo. El comparador comprueba qué magnitud de operando es mayor, mientras que el módulo inversor de signo corrige el bit de signo del operando **Data_Y** en caso de efectuarse una resta. Los resultados de estos módulos son almacenados en registros de segmentación. Finalmente, estos resultados se propagan junto a la señal **Data_X** a través de dos multiplexores cuya función es direccionar los operandos a los registros de salida que contendrán al número mayor (**DMP**) y al número menor (**DmP**).

3.2.1.2 Detector de resultado Cero (DZR_M)

La función de este módulo es determinar si el resultado final de la operación aritmética va a ser igual a cero. Para llevar a cabo este proceso se diseñó la estructura mostrada en la figura 3.4.

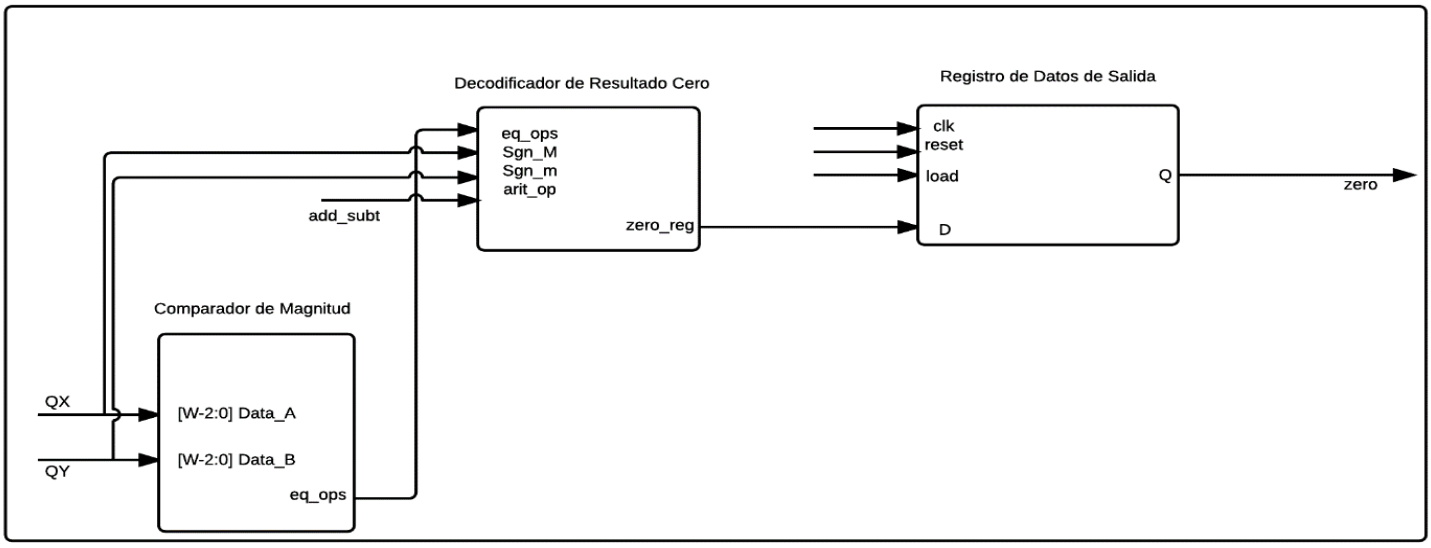

Figura 3.4. Diagrama RTL del módulo DZR_M

El bloque recibe las señales **QX** y **QY** provenientes del módulo *Op_Class_FP* (ver la figura 3.3) y la señal **add_subt** suministrada por el ASP. Inicialmente se realiza un proceso de comparación de magnitud entre las señales **QX** y **QY**. El resultado de esta comparación se suministra a un decodificador de 4 a 1 junto a otras señales, que corresponden a los signos de los operandos (**Sgn_M**, **Sgn_m**) y al tipo de operación que se está realizando (**add_subt**). El proceso de decodificación indica si el resultado de la operación aritmética es igual a cero. Finalmente, esta información es almacenada en un registro de salida y corresponde a la señal **zero**.

3.2.1.3 Normalizador del significando del número menor (N_Smo)

La adición o sustracción utilizando el método de notación científica requiere que los exponentes de ambos operandos sean idénticos. Para satisfacer dicha condición se diseñó la estructura mostrada en la figura 3.5, cuya función es desplazar el significando del número menor (se desplazan ceros hacia la derecha) hasta alcanzar la condición de igualdad de los exponentes. Este proceso de desplazamientos normaliza el significando del número menor.

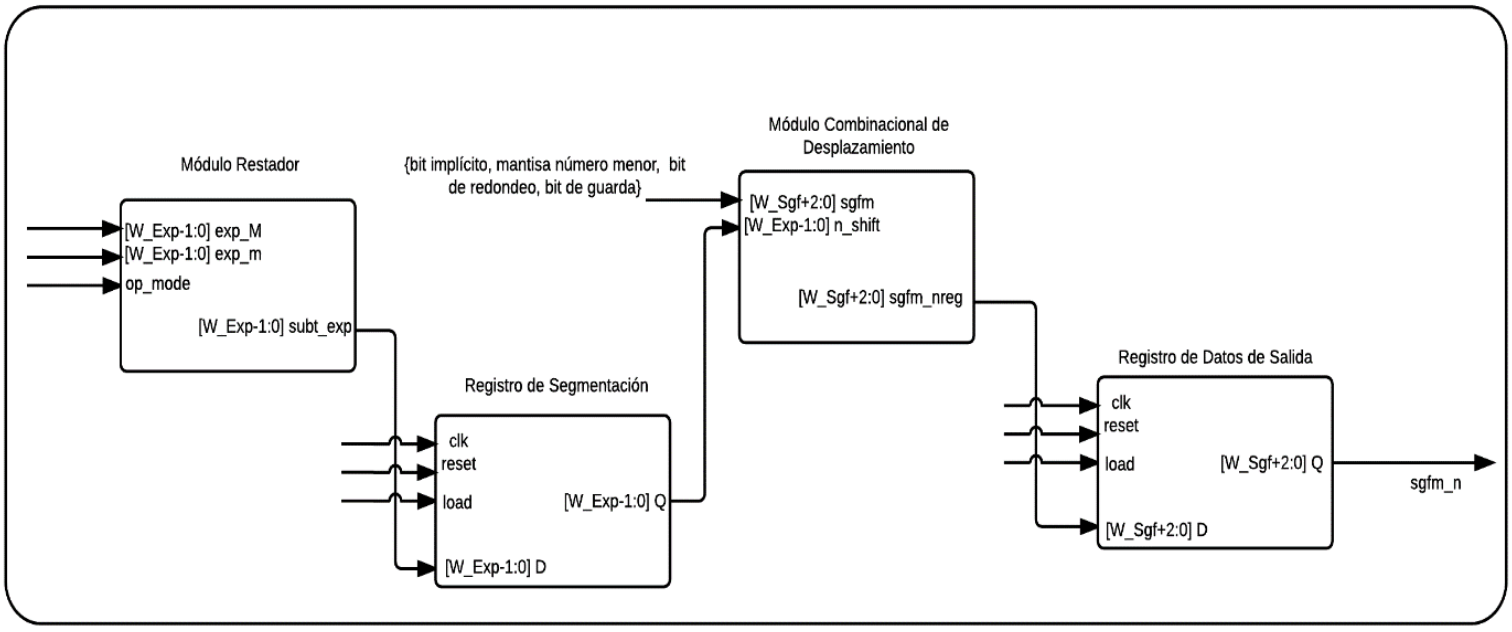

Figura 3.5 Diagrama RTL del módulo N_Smo

Inicialmente, el módulo construye el significando del número menor (señal **sgfm**) y efectúa la resta de los exponentes de ambos operandos (señales **exp_M** y **exp_M**), cuyo resultado es almacenado en un registro de segmento. Posteriormente, estas señales se

alimentan a un módulo combinacional de desplazamientos dónde se concluye el proceso de normalización. Finalmente, se almacena el significando normalizado del número menor en un registro de salida (señal **sgfm_n**).

3.2.1.4 Sumador-restador de significandos (Sgfs_Add_Sub)

En la figura 3.6 se presenta el diagrama RTL del módulo encargado de sumar o restar los significandos.

Inicialmente, el módulo construye el significando del número mayor, y compara los signos de ambos operandos (señales **sgn_M** y **sgn_m**) para definir la operación aritmética a ejecutar (por medio de la señal **op_mode**). Finalmente, se efectúa el proceso de suma o resta de los significandos y se almacena el resultado en un registro de salida (señal **Sgf_R**).

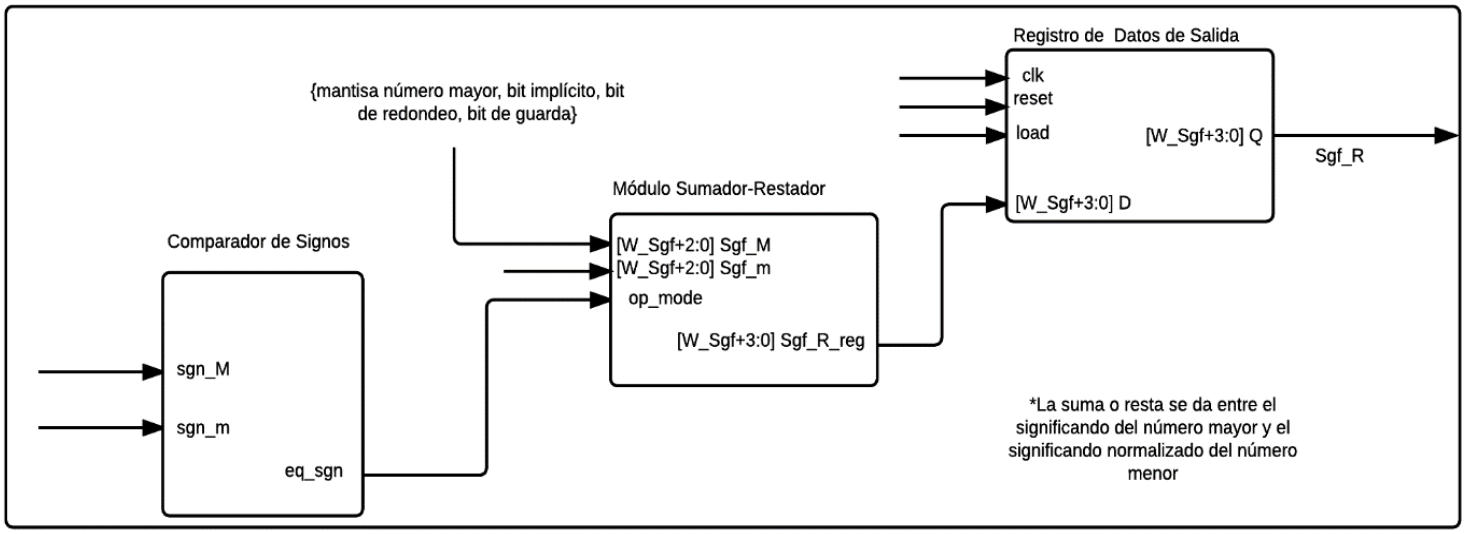

Figura 3.6 Diagrama RTL del módulo Sgfs_Add_Sub

3.2.1.5 Normalizador del significando resultante (Sgf_R_NM)

El significando que resulta del proceso aritmético efectuado en el módulo *Sgfs_Add_Sub* debe normalizarse. Este proceso de normalización implica que el patrón de bits del significando debe ser de la forma 1.Fracción (ver sección 2.1.3.3). La forma de generar dicha cadena es a través de procesos de desplazamiento de ceros en el significando. Esto último implica que el proceso de normalización es un proceso iterativo.

Para llevar a cabo lo anterior, se diseñó la estructura mostrada en la figura 3.7. El módulo cuenta con un multiplexor, un registro universal de desplazamiento y un registro de salida. El multiplexor se encarga de seleccionar inicialmente el significando resultante del

módulo *Sgfs_Add_Sub* (señal **Sgf_R**), y posteriormente significandos que han sido sometidos a procesos de desplazamiento. El registro de desplazamiento universal se encarga de desplazar ceros hacia la derecha o hacia la izquierda del significando y de brindar información a través de sus señales de salida (señal **Sgfn_R_MSBs**) sobre el estado (normalizado o no) del significando. Finalmente, cuando concluye el proceso de desplazamientos se almacena el significando normalizado en el registro de salida (señal **Sgf_NF**).

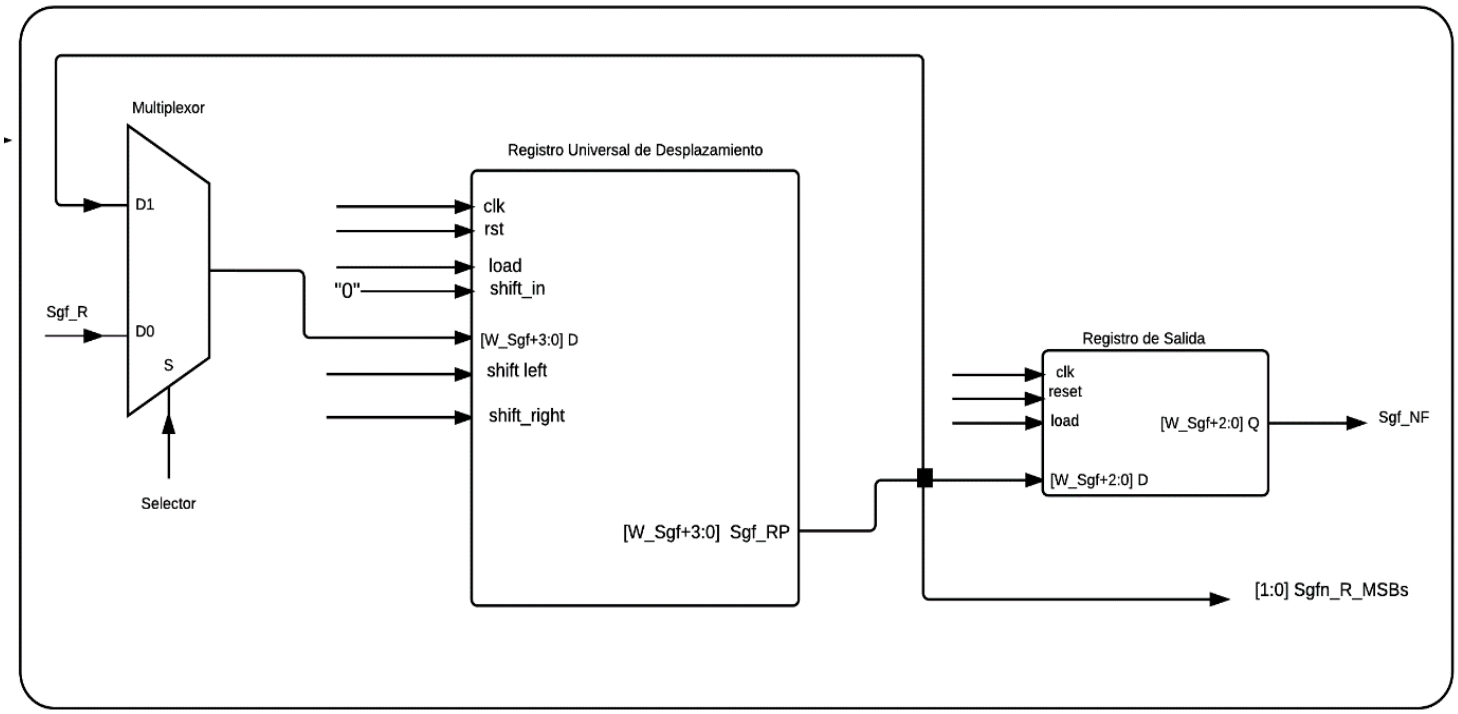

Figura 3.7. Diagrama RTL del módulo Sgf_R_NM

3.2.1.6 Primera etapa de actualización del exponente resultante (FS_Exp_R_A)

Cada desplazamiento efectuado durante el proceso de normalización implica sumar o restar un 1 al exponente del número mayor (señal **exp_M** proveniente del módulo *Op_Class_FP*). Este proceso aritmético se conoce como proceso de actualización del exponente. El diagrama RTL del módulo que realiza esta función se presenta en la figura 3.8.

La estructura del módulo cuenta con dos multiplexores, dos registros de segmentación, un módulo de suma o resta y un registro de salida. Los multiplexores se encargan de seleccionar el exponente a actualizar y el exponente resultante del proceso de actualización. Los registros de segmento se utilizan para almacenar resultados intermedios y el registro de salida para almacenar en primera instancia el exponente del

resultado final (señal **exp_update_ieee**). Además, el módulo ofrece una señal de salida denominada **exp_update_uo**, que corresponde al exponente actualizado en cada iteración y es utilizado para verificar casos de desborde y sub-desborde en el módulo *excp_cond_check_A*.

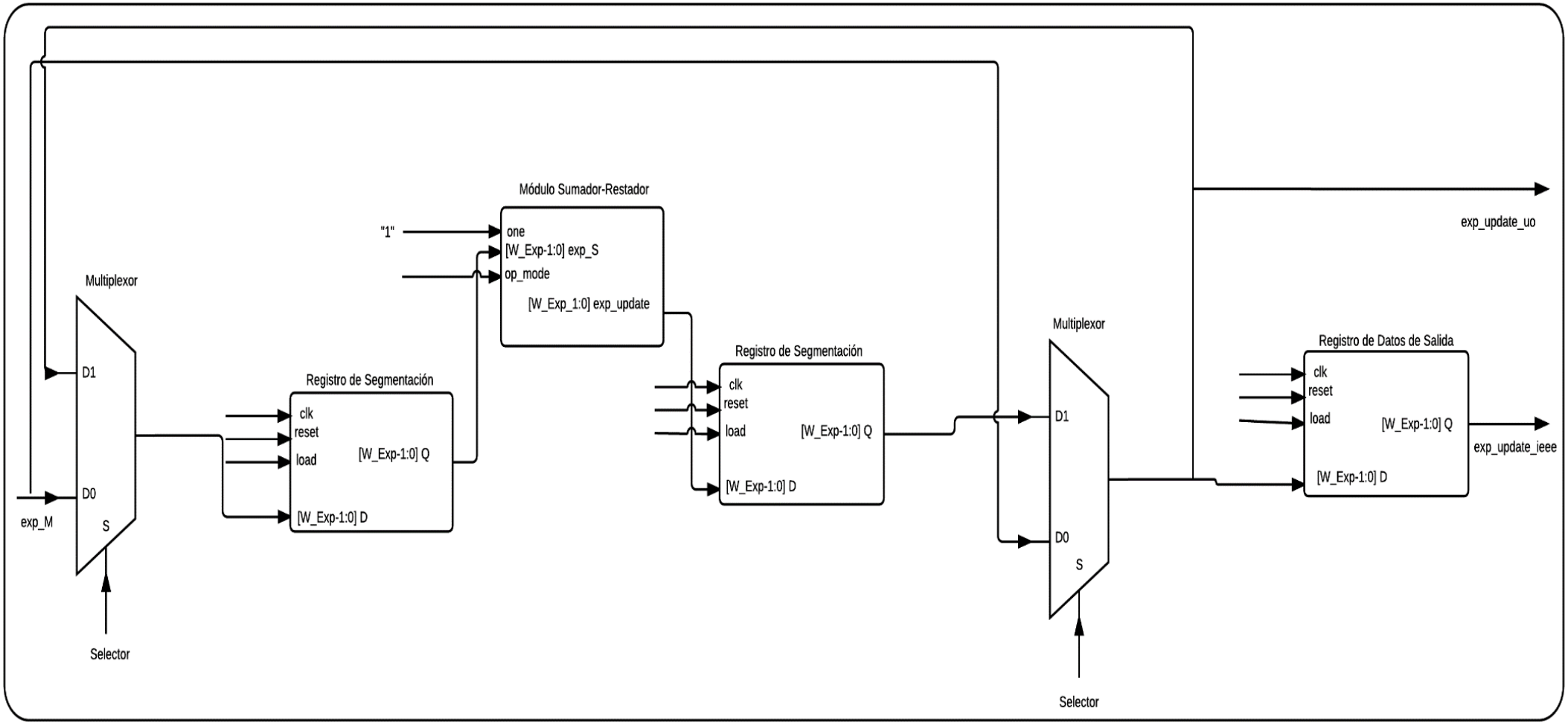

Figura 3.8. Diagrama RTL del módulo FS_Exp_R_A

3.2.1.7 Primera etapa de verificación de casos excepcionales (excp_cond_check_A)

Cada vez que se actualiza el exponente en el módulo FS_Exp_R_A es necesario verificar que su patrón de bits sigue siendo representable por el formato de precisión. Para llevar a cabo esta verificación, se diseñó el módulo que se muestra en la figura 3.9.

El bloque recibe la señal **exp_update_uo** proveniente del módulo *FS_Exp_R_A* y la compara con los valores de umbral correspondientes al exponente del mayor y menor número normal representable por el formato de precisión (ver la tabla 2.3). Los resultados del proceso de comparación son almacenados en registros de salida (señales **overflow_a** y **underflow**) a partir de los cuales la FSM puede determinar si existe un caso de desborde y sub-desborde.

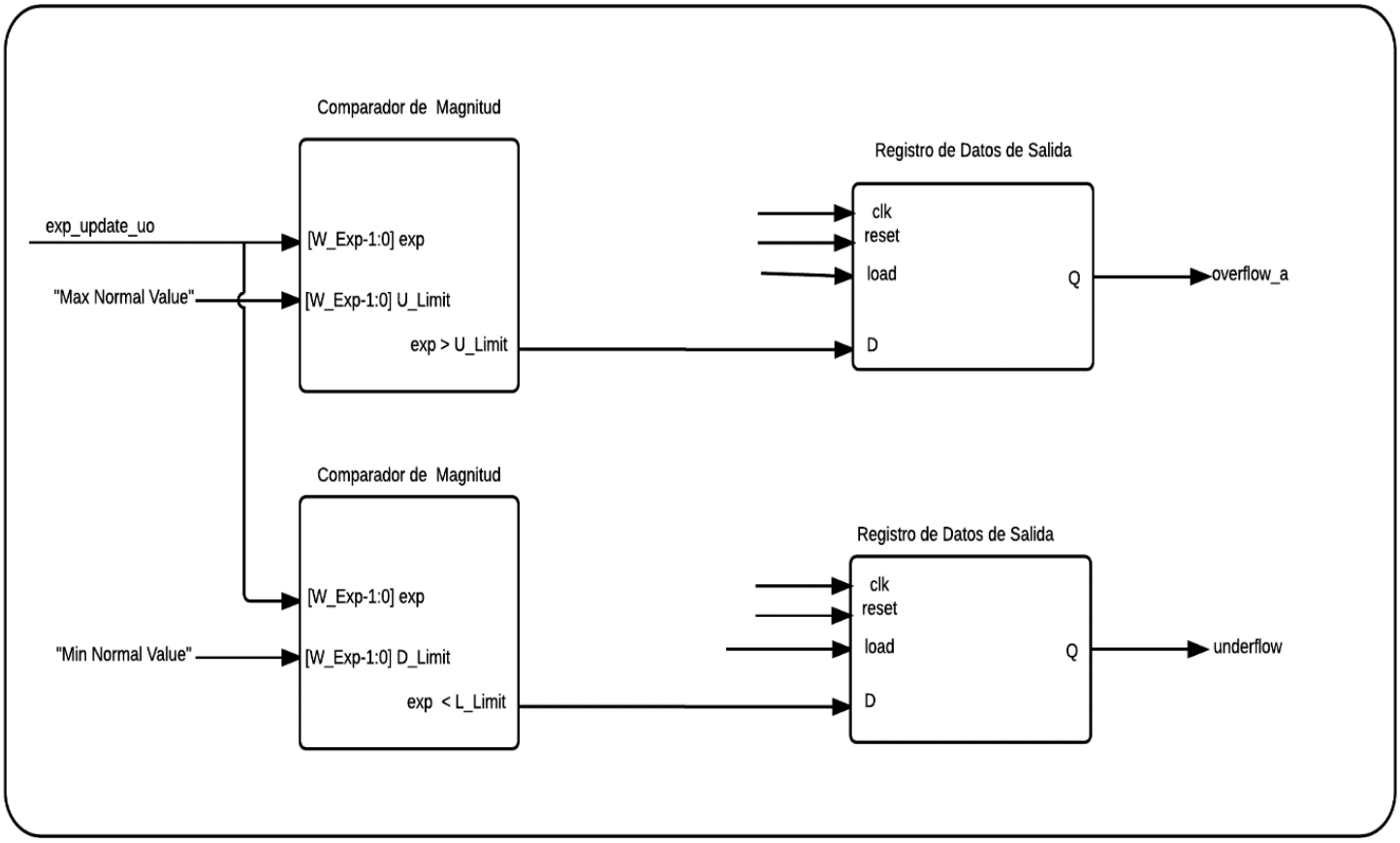

Figura 3.9. Diagrama RTL del módulo excp_cond_check_A

3.2.1.8 Módulo de redondeo (Round_M_Sgf)

En la figura 3.10 se muestra la estructura diseñada para llevar a cabo el proceso de redondeo. Esta codificación se efectúa sobre el significando obtenido en el módulo *Sgf_R_NM* (señal **Sgf_NF**), y según lo indicado en la tabla 2.4.

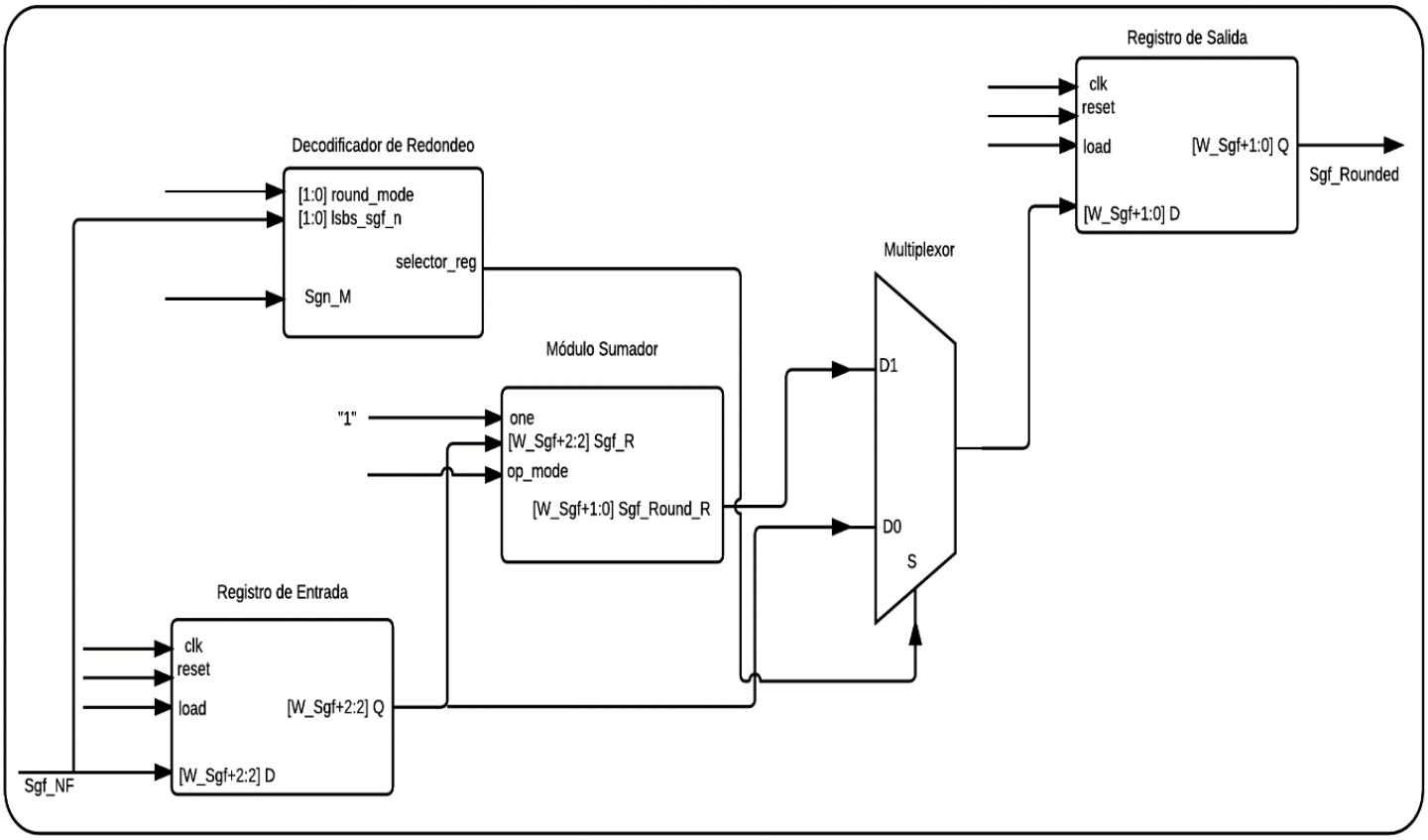

Figura 3.10. Diagrama RTL del módulo Round_M_Sgf

El módulo recibe la señal **Sgf_NF**, que se almacena en un registro de entrada. Esta última señal se alimenta a una de las entradas del multiplexor y a una unidad aritmética de suma. El módulo de suma adiciona un 1 al significando, y el resultado es conectado a la otra entrada del multiplexor. Posteriormente, el módulo decodificador de redondeo permite al multiplexor seleccionar el valor correspondiente al significando redondeado, basado en el signo del exponente mayor (señal **Sgn_M**), en los bits de redondeo y guarda del significando (señal **lsbs_sgf_n**) y en el modo de redondeo que se esté utilizando (señal **round_mode**). Finalmente, el significando redondeado (señal **Sgf_Rounded**) es almacenado en un registro de salida.

3.2.1.9 Selección de la mantisa del resultado final (Final_Mant_Select)

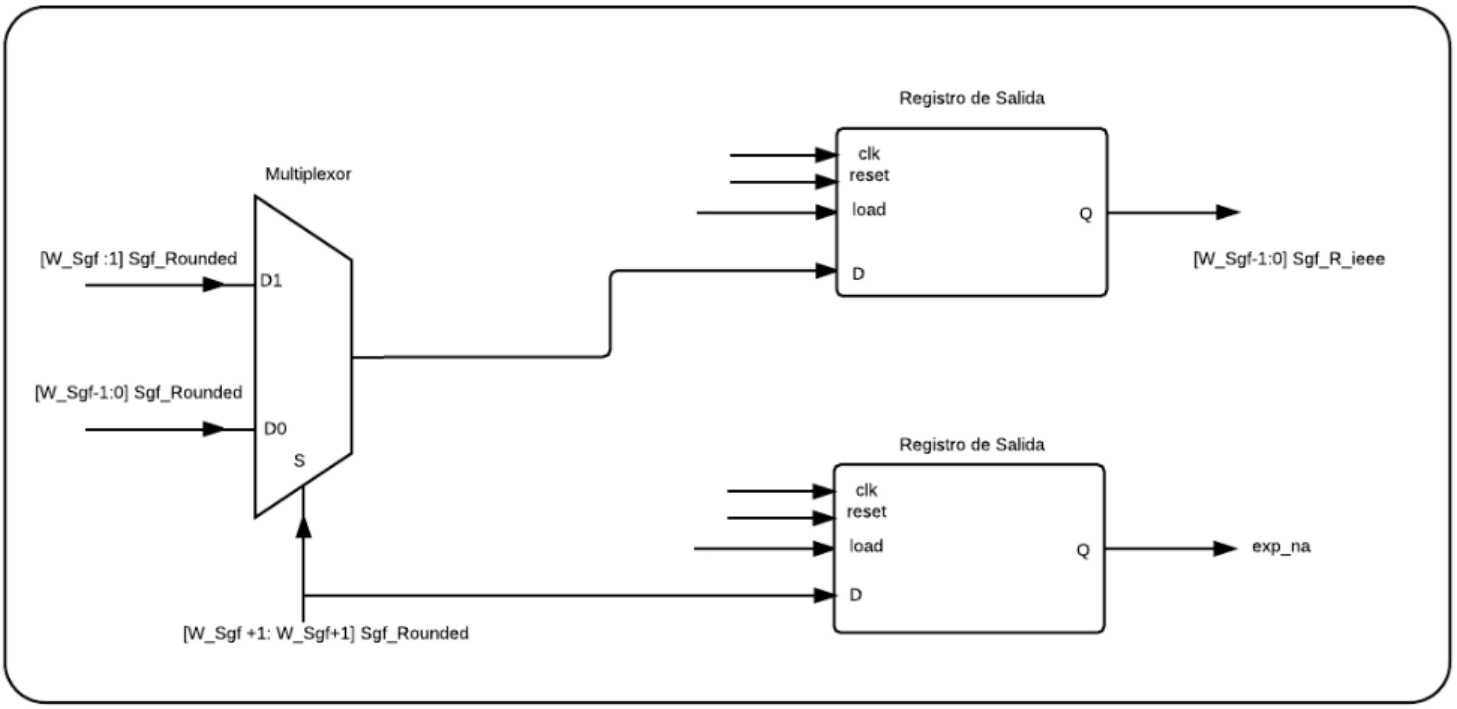

Figura 3.11. Diagrama RTL del módulo Final_Mant_Select

Después de aplicar el proceso de redondeo cabe la posibilidad de que el significando se desnormalice; bajo esta situación hay que efectuar nuevamente el proceso de normalización. Convenientemente, esta nueva normalización se lleva a cabo en una sola iteración y se realiza mediante un desplazamiento hacia la derecha del significando. Para efectuar este procedimiento, se diseñó la estructura de la figura 3.11.

El módulo utiliza un multiplexor cuyas entradas se conectan estratégicamente a la señal **Sgf_Rounded** proveniente del módulo *Round_M_Sgf*. Este esquema de conexión permite seleccionar de manera correcta la mantisa (bit implícito suprimido) del resultado final, que se almacena en un registro de salida (señal **Sgf_R_ieee**). Cabe destacar que aunque el acrónimo utilizado para esta señal es similar a los utilizados para describir significandos, éste hace referencia a un valor de mantisa.

Finalmente, el módulo cuenta con una señal de salida llamada **exp_na** cuya función es indicar si se llevó a cabo una nueva normalización del significando. Esto último es de vital importancia para saber si se debe actualizar nuevamente el exponente.

3.2.1.10 Selección del exponente del resultado final y segunda etapa de verificación de casos excepcionales (Final_Exp_Ov_S)

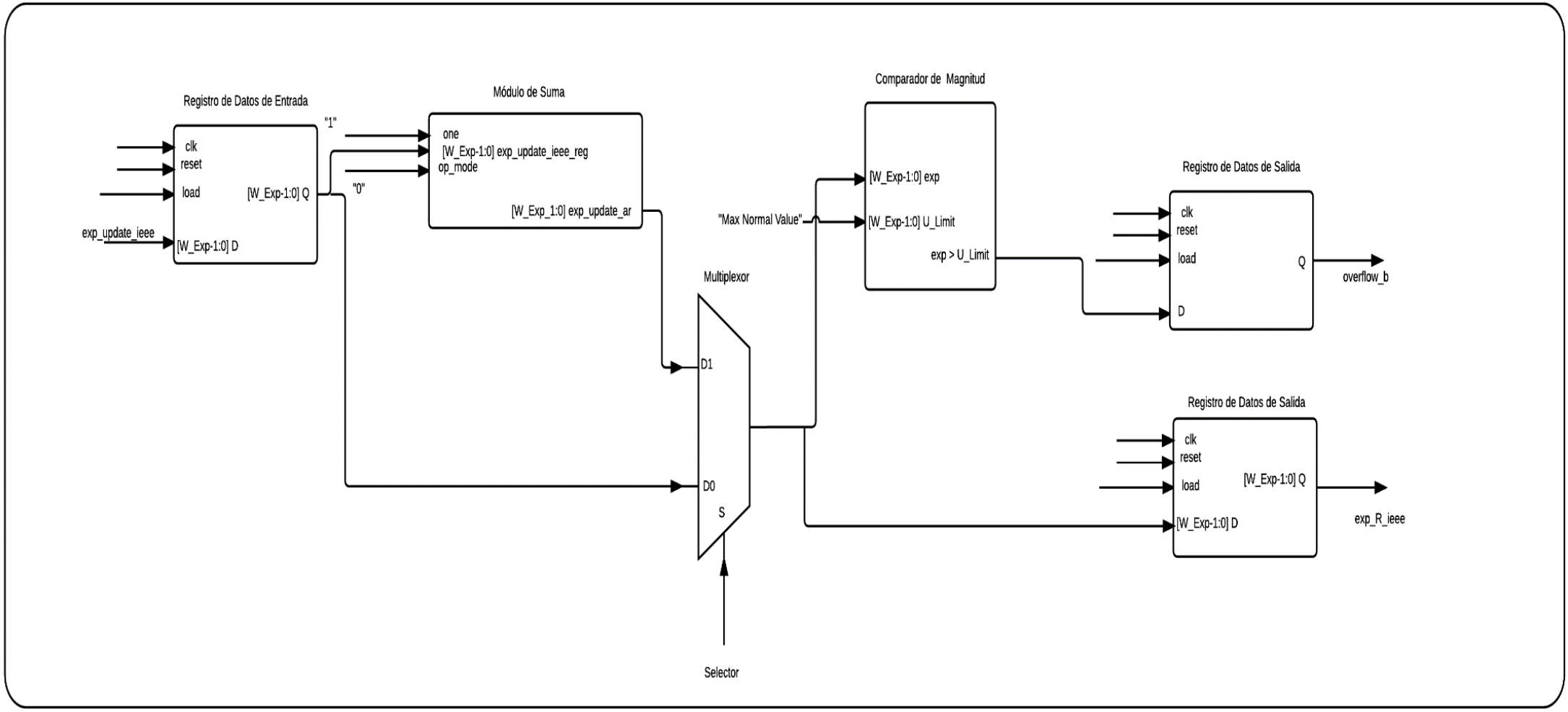

Figura 3.12. Diagrama RTL del módulo Final_Exp_Ov_S

La estructura mostrada en la figura 3.12 tiene como función seleccionar el exponente del resultado final y verificar si este sigue siendo representable por el formato de precisión (esto último ante una eventual normalización post-redondeo en el módulo *Final_Mant_Select*).

Si se da un caso de normalización post redondeo, el módulo debe actualizar la señal **exp_update_ieee** (proveniente del módulo *FS_Exp_R_A*), seleccionarla como exponente del resultado final **(señal exp_R_ieee**), e informar si sigue siendo representable por el formato de precisión (esto se logra a través de la señal **overflow_b**, pues en este punto la única excepción que puede ocurrir es la de desborde). En caso contrario, el módulo selecciona la señal **exp_update_ieee** como exponente del resultado final.

3.2.1.11 Ensamblado del resultado final (Final_Result_AssemM)

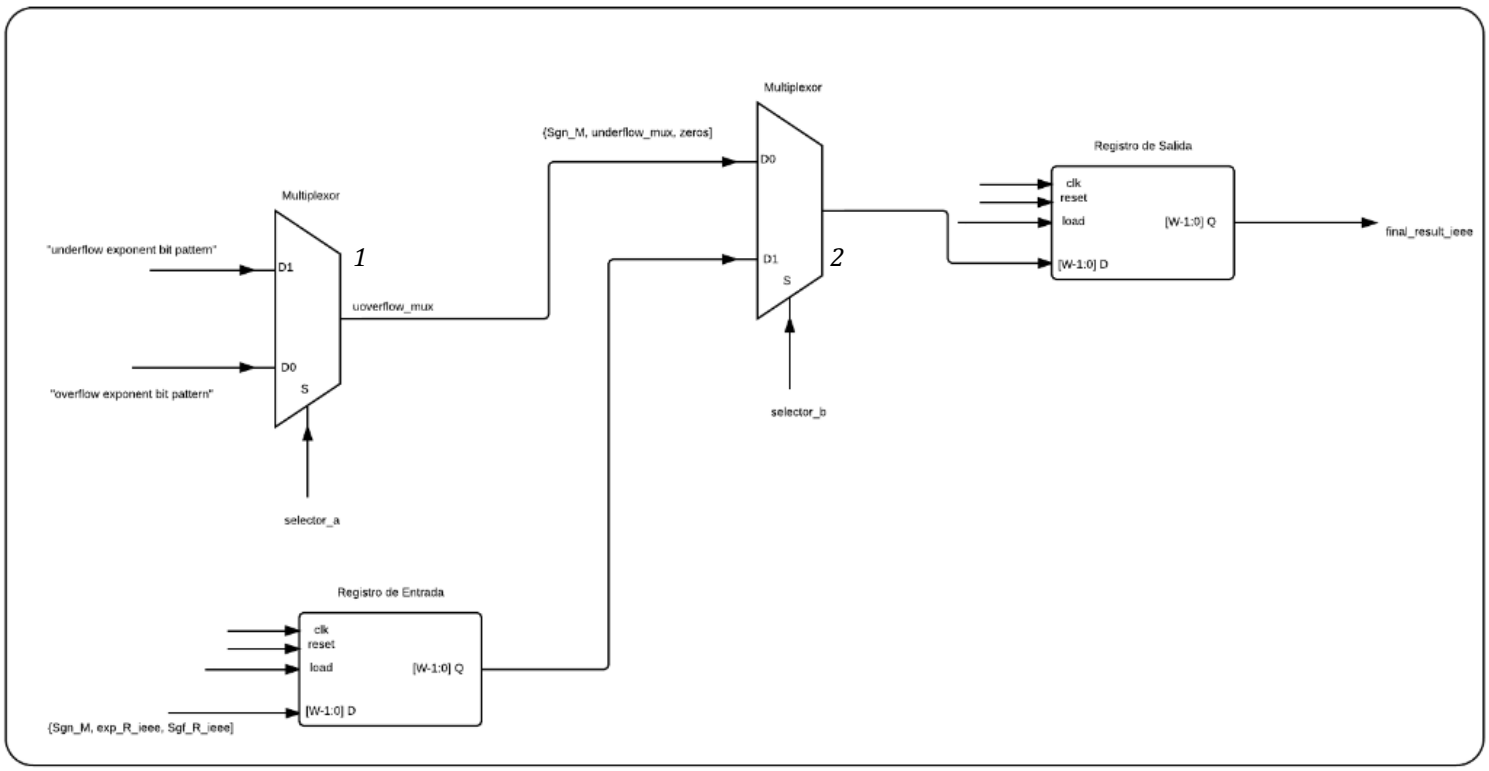

Figura 3.13 Estructura interna del módulo Final_Result_AssemM

Este módulo se encarga de ensamblar el resultado final de la operación aritmética. Para llevar a cabo este procedimiento se diseñó la estructura de la figura 3.13. La estructura posee un registro de entrada en el cual se carga el signo del número mayor (señal **Sgn_M**),

el exponente obtenido del módulo *Final_Exp_Ov_S,* y la mantisa obtenida del módulo *Final_Mant_Select*. La concatenación de estos tres campos corresponde al resultado final de la operación aritmética en formato IEEE 754. Este resultado se selecciona a través del multiplexor 2 en caso de que no se haya dado algún caso excepcional durante el proceso de ejecución, y se almacena en el registro de salida (señal **final_result_ieee**). Por otro lado, cuando se genera un caso excepcional, el resultado final de la operación corresponde a patrones de bits predeterminados (para el caso de desborde el resultado final puede ser el infinito positivo o infinito negativo, mientras que en el caso de sub-desborde el resultado final corresponde a un cero). Estos patrones se construyen utilizando un proceso de multiplexado (multiplexor 1) y un proceso de concatenación. Igualmente, este resultado es seleccionado por el multiplexor 2 y posteriormente se almacena en el registro de salida (señal **final_result_ieee**).

3.2.1.12 Máquina de estados finitos para el algoritmo de suma-resta

El algoritmo de suma-resta es una estructura secuencial bien definida y que requiere de una gran cantidad de pasos durante el proceso de ejecución. Estos pasos se implementaron con ayuda de una FSM de 48 estados. La función de la máquina de estados finitos es conducir de manera correcta el flujo de datos a través de los diferentes módulos presentados en la figura 3.2.

En esta sección, el proceso de control del flujo de datos se describe a partir de una serie de pasos, con el fin de presentar una descripción compacta e informal sobre el funcionamiento del módulo de suma-resta desarrollado. Los pasos que son ejecutados por el módulo de suma-resta son los siguientes:

1. Clasificar los operandos en un operando mayor y un operando menor.
2. Construir y normalizar el significando del número menor.
3. Construir el significando del número mayor.
4. Sumar o restar los significandos obtenidos en los pasos 1 y 2.
5. Si se requiere, normalizar el significando resultante del paso 4.
6. Actualizar el exponente del número mayor ante un eventual proceso de normalización en el paso 5.
7. Verificar que el exponente obtenido en el paso 6 sea representable por el formato de precisión (casos de sub-desborde y desborde).
8. Redondear el signficando resultante del paso 5.
9. En caso de que el proceso de redondeo desnormalice el significando, se debe volver a normalizar.
10. Actualizar el exponente ante una eventual normalización en el paso 9.
11. Verificar que el exponente del paso 10 sea representable por el formato.
12. Ensamblaje del resultado final.

3.3 Módulo de multiplicación

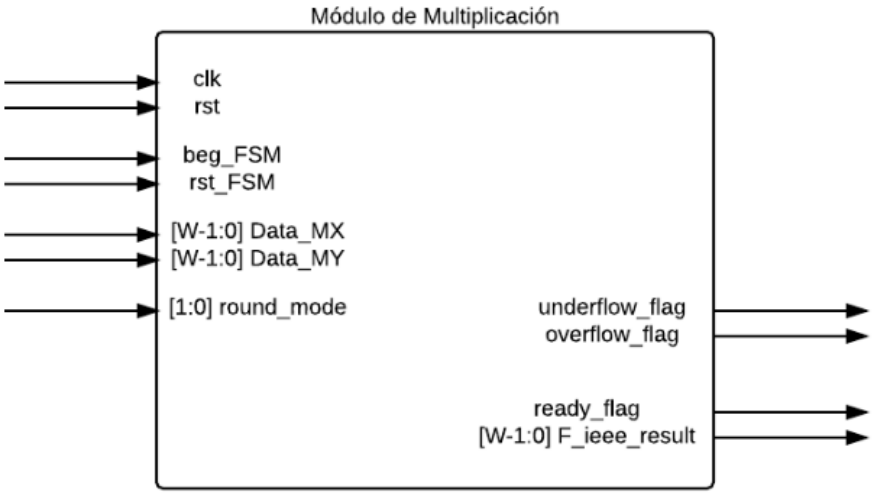

Figura 3.14 Diagrama de entradas-salidas del módulo de multiplicación

El módulo de multiplicación lleva a cabo el producto de dos operandos según el estándar IEEE 754. Para implementar esta unidad aritmética se propuso el diagrama de la figura 3.14, cuya estructura interna se presenta en la figura 3.15.

Las señales de entrada del módulo son:

- **r_mode**: establece la codificación de redondeo.
- **Data_MX, Data_MY**: operandos en formato IEEE 754.
- **clk**: reloj del sistema, 100 MHz(Placa FPGA Artix-7 Nexys 4).
- **rst**: reinicio de la FSM que controla la ejecución del algoritmo.
- **beg_FSM**: indica al sistema el momento en que debe iniciar el proceso de multiplicación.
- **rst_FSM**: manera en que un sistema puede comunicarle al módulo de multiplicación que ha recibido con éxito el resultado de la operación.

Las señales de salida corresponden a:

- **overflow_flag**: caso excepcional de desborde.
- **underflow_flag**: caso excepcional de sub-desborde.
- **ready_flag**: señal que indica que la ejecución del algoritmo ha terminado.
- **F_ieee_result**: resultado final de la operación según el estándar IEEE 754.

Figura 3.15. Arquitectura completa de flujo de datos para la unidad de multiplicación en coma flotante

3.3.1 Descripción de los bloques que conforman la unidad aritmética de multiplicación

En esta sección se describen los bloques presentados en la figura 3.15. La metodología que se utiliza para llevar a cabo tal descripción es la misma que la presentada para el módulo de suma-resta.

3.3.1.1 Módulo de almacenamiento de operandos (Op_Load_Reg)

El módulo Op_Load_Reg tiene como función almacenar en dos registros de entrada los operandos (señales **Data_MX** y **Data_MY**) a multiplicar. La estructura diseñada para llevar a cabo dicha función se muestra en la figura 3.16.

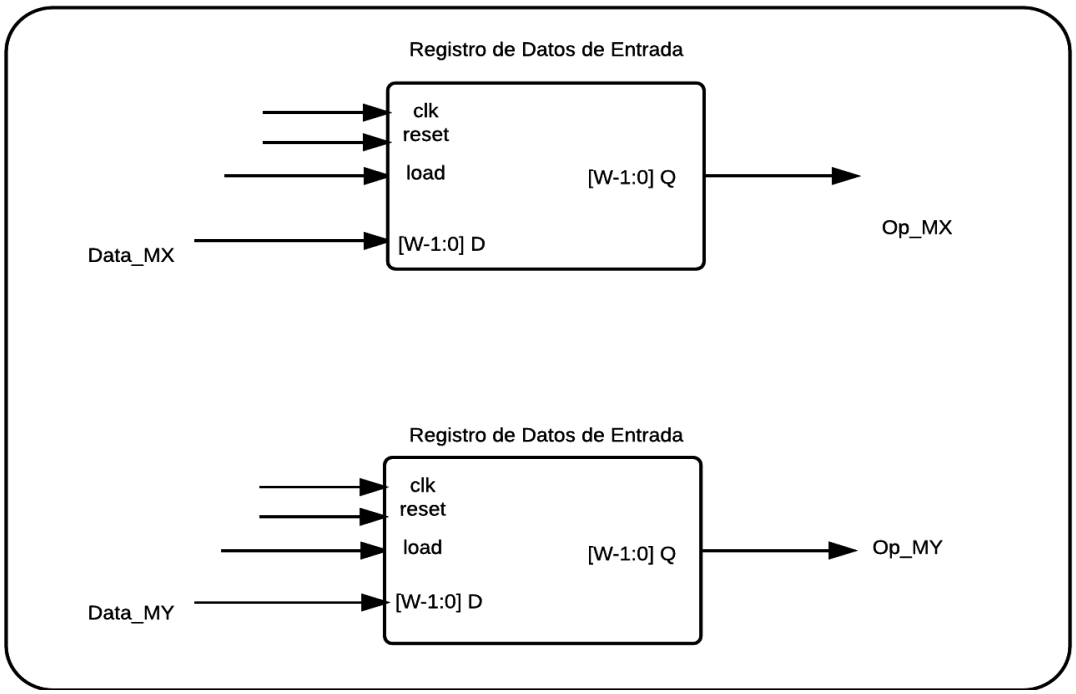

Figura 3.16. Diagrama RTL del módulo Op_Load_Reg

3.3.1.2 Detector de resultado cero (Zero_Result_D)

La estructura presentada en la figura 3.17 tiene como función identificar si alguno de los operandos es igual a cero. El módulo recibe las señales **Op_MX** y **Op_MY** provenientes del módulo Op_Load_Reg, que se someten a un proceso de comparación para determinar si

alguna o ambas poseen un valor igual a cero. Finalmente, el resultado de esta comparación es almacenado en un registro de salida (señal **zero_m_flag**).

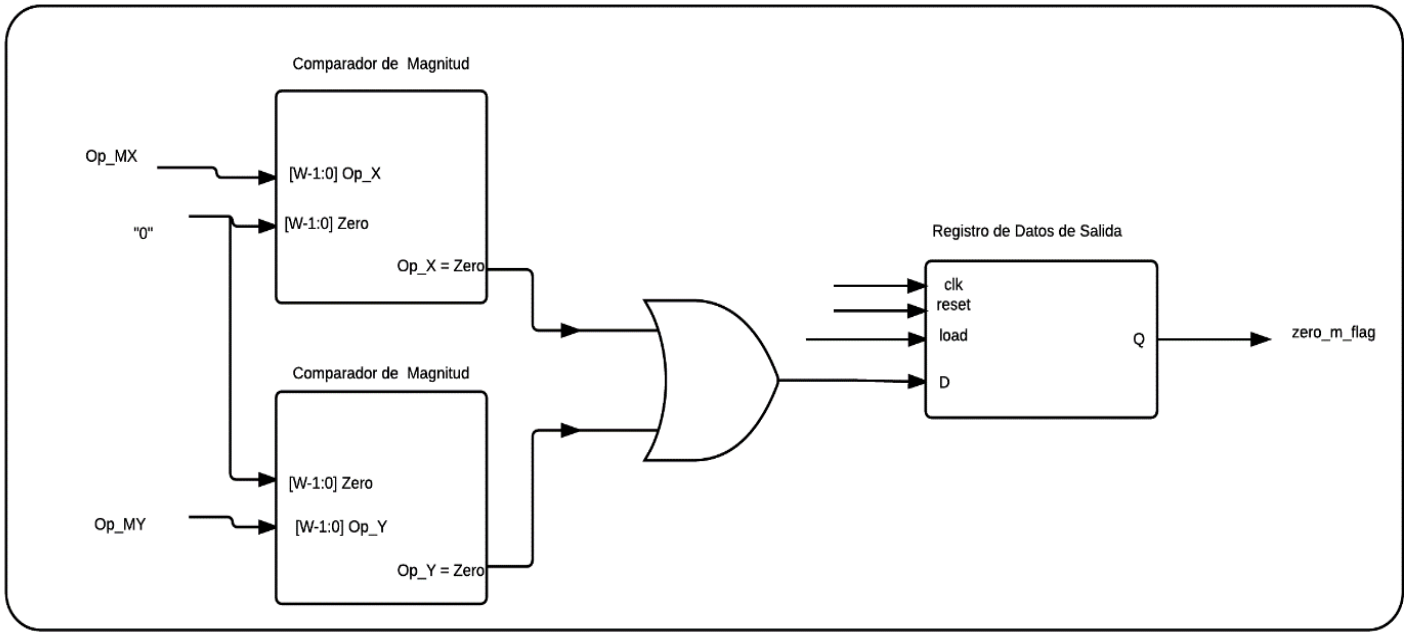

Figura 3.17. Diagrama RTL del módulo Zero_Result_D

3.3.1.3 Etapa de verificación de sub-desborde (Uflow_State_M)

Por la naturaleza del algoritmo de multiplicación, se comienza verificando si existe el caso excepcional de sub-desborde. Para llevar a cabo este proceso se diseñó la estructura mostrada en la figura 3.18.

El módulo recibe los exponentes de ambos operandos (señales **Exp_X** y **Exp_Y**), los cuales se toman de las señales **Op_MX** y **Op_MY** provenientes del módulo *Op_Load_Reg*. Estos exponentes se suman y el resultado se almacena en un registro de segmento. Este resultado se compara con el valor de sesgo del formato de precisión. Si la suma de los exponentes es menor que el valor de sesgo entonces existirá un caso excepcional de sub-desborde. Finalmente, se almacena el resultado de la comparación en un registro de salida (señal **underflow_flag**). Cabe destacar que la suma de los exponentes (señal **Exp_Add**) es una señal que se necesita en el módulo *UnBias_ExpAdd*.

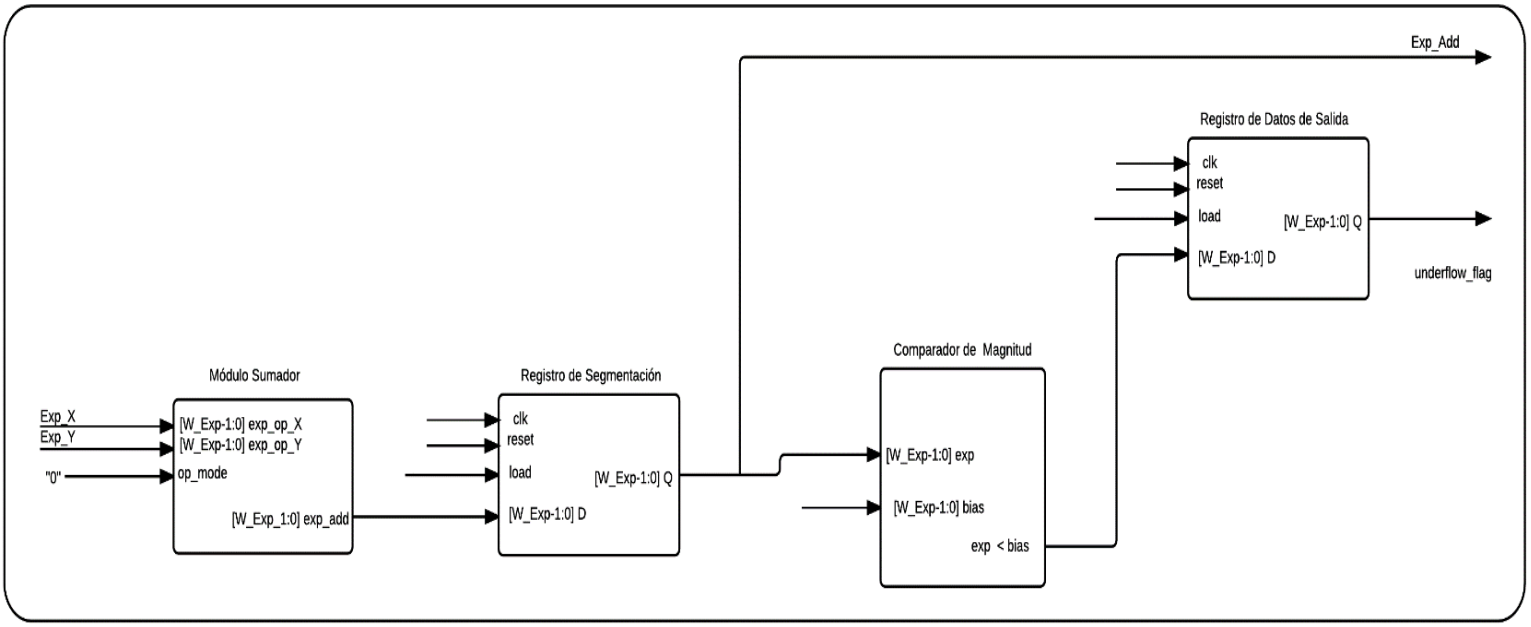

Figura 3.18 Diagrama RTL del módulo Uflow_State_M

3.3.1.4 Normalizador de suma de exponentes (UnBias_ExpAdd)

El proceso de suma de exponentes llevado a cabo en el módulo *Uflow_State_M* genera un exponente desnormalizado. Esto se debe a que la suma de los exponentes adiciona dos veces el valor de sesgo. Por lo tanto, es necesario sustraer el sesgo de la señal **Exp_Add**. Para llevar a cabo este procedimiento se diseñó la estructura mostrada en la figura 3.19.

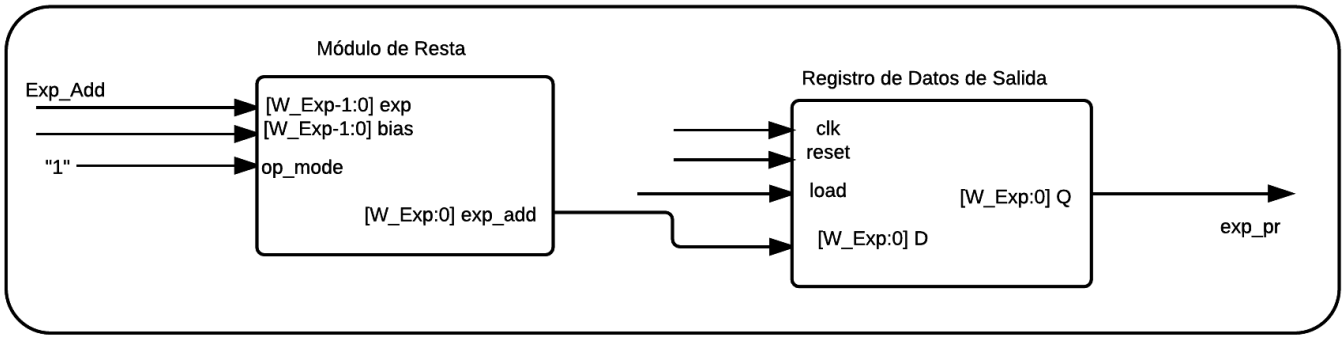

Figura 3.19 Diagrama RTL del módulo UnBias_ExpAdd

El bloque recibe la señal **Exp_Add** proveniente del módulo *Uflow_State_M,* a la cual se le sustrae el valor del sesgo utilizando una unidad de resta. Finalmente, el resultado del proceso de sustracción se almacena en un registro de salida (señal **exp_pr**).

3.3.1.5 Etapa de verificación de desborde (Oflow_State_M)

El resultado de restar el valor de sesgo a la suma de los exponentes (señal **exp_pr** proveniente del módulo *UnBias_ExpAdd*) debe ser verificado para detectar un posible caso de desborde. Para llevar a cabo esta verificación se diseñó la estructura que se muestra en la figura 3.20.

Existen dos posibles casos de desborde que se pueden generar en este algoritmo. El primero está relacionado al bit de acarreo de la señal **exp_pr**, mientras que el segundo puede generarse si el patrón de bits de esta señal no es representable por el formato de precisión. Para detectar estos casos, se utilizó una estrategia de conexión de la señal **exp_pr**, que permite almacenar en dos registros de salida información referente a los posibles casos de desborde (señales **overflow_comp** y **overflow_cout_flag**). Además, se almacena en un tercer registro lo que en primera instancia corresponde al exponente del resultado final (señal **exp_r**).

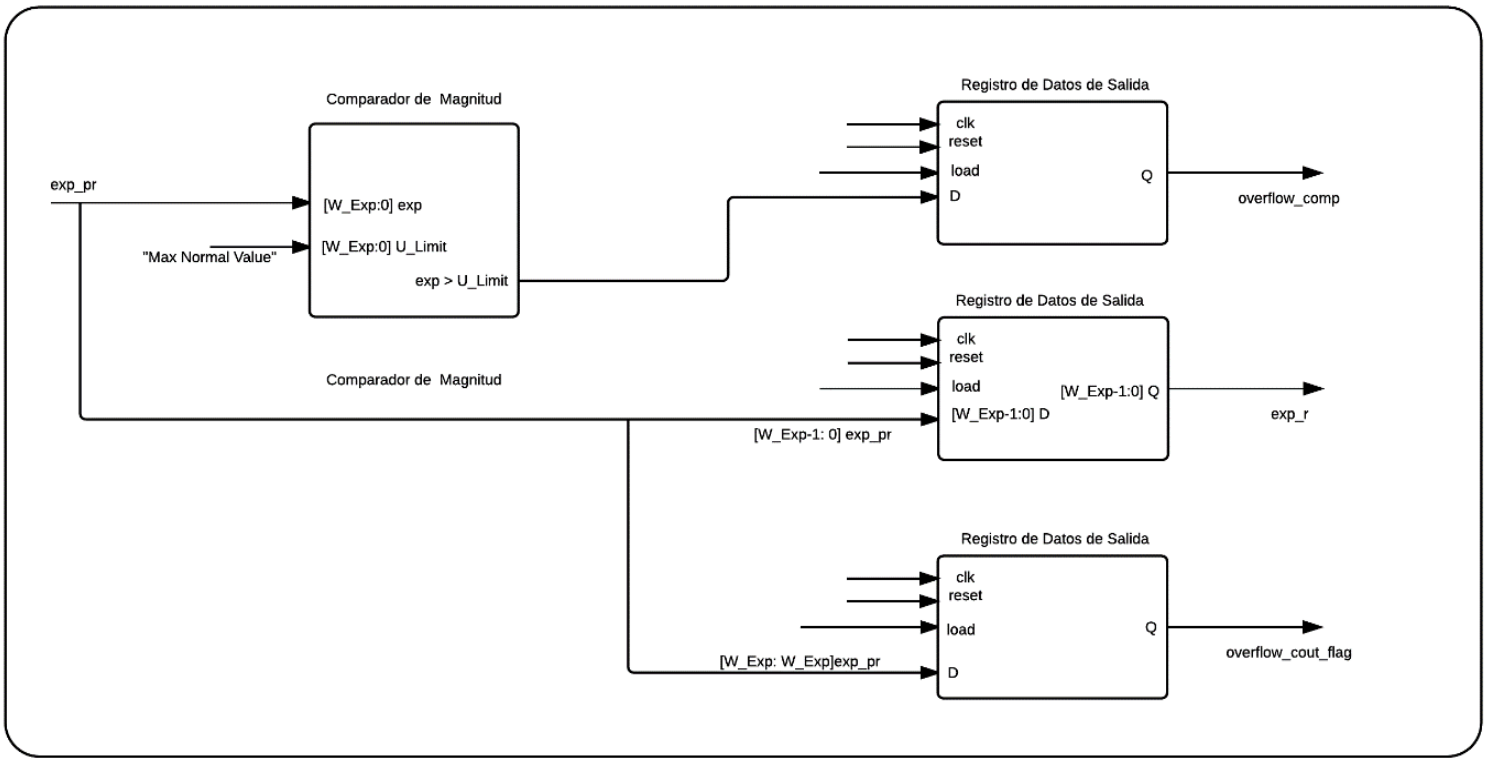

Figura 3.20. Diagrama RTL del módulo Oflow_State_M

3.3.1.6 Multiplicación de significandos (Sgf_Mult_M)

El bloque que se presenta en la figura 3.21 se encarga de multiplicar los significandos de los operandos (señales **Sgf_X** y **Sgf_Y**, que se extraen del módulo *Op_Load_Reg*), cuyo

resultado produce la señal **P_Sgf.** Esta señal es utilizada en otros módulos para generar el significando del resultado final.

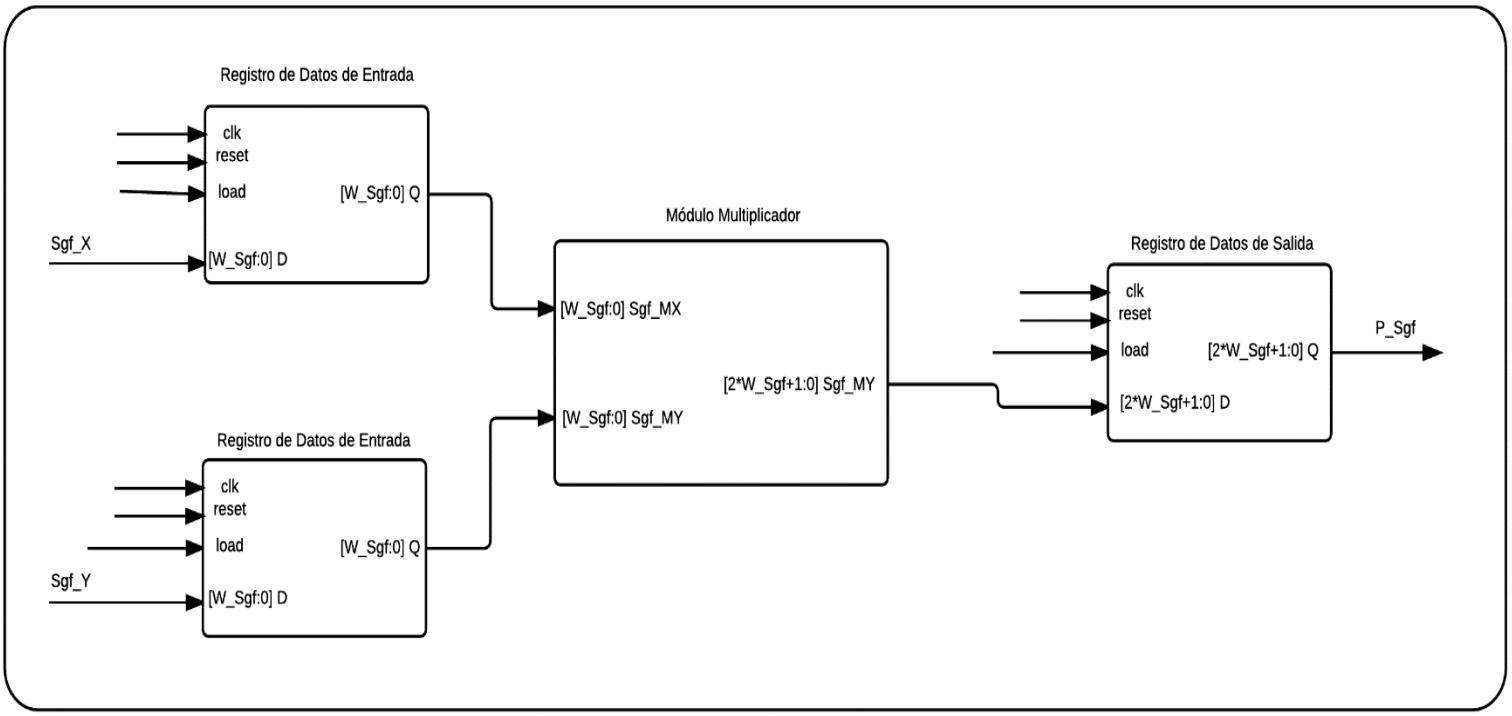

Figura 3.21 Diagrama RTL del módulo Sgf_Mult_M

Note que el ancho de palabra de la señal **P_Sgf** es dos veces el ancho de palabra de los significandos que se multiplican. Por ejemplo, si se trabaja en formato de precisión simple, la señal **P_Sgf** tendrá un ancho de palabra de 48 bits al ser los significandos de 24 bits cada uno. El significando del resultado final debe ser también de 24 bits, razón por la que sólo una fracción de la señal **P_Sgf** puede usarse para construirlo. A continuación se muestra la distribución de bits utilizada para construir en primera instancia el significando del resultado final:

Bits Utilizados Para Construir el Significando	Bits para propósito de redondeo
47 23	22 0

Figura 3.22 Distribución de bits para el producto de significandos.

3.3.1.7 Normalización del significando resultante (Sgf_Norm_M)

El módulo de la figura 3.23 utiliza los bits más significativos de la señal **P_Sgf** proveniente del módulo *Sgf_Mult_M* (señal **Sgf_FP**) para construir y normalizar en primera instancia

el significando del resultado final (señal **Sgf_F**). También se encarga de normalizar el significando después del proceso de redondeo (señal **Sgf_P_Round** proveniente del módulo *Sgf_Round_Sgn_Info*). Además, el módulo brinda información referente a la actualización del exponente (señal **exp_nu**) debido a una eventual normalización del significando.

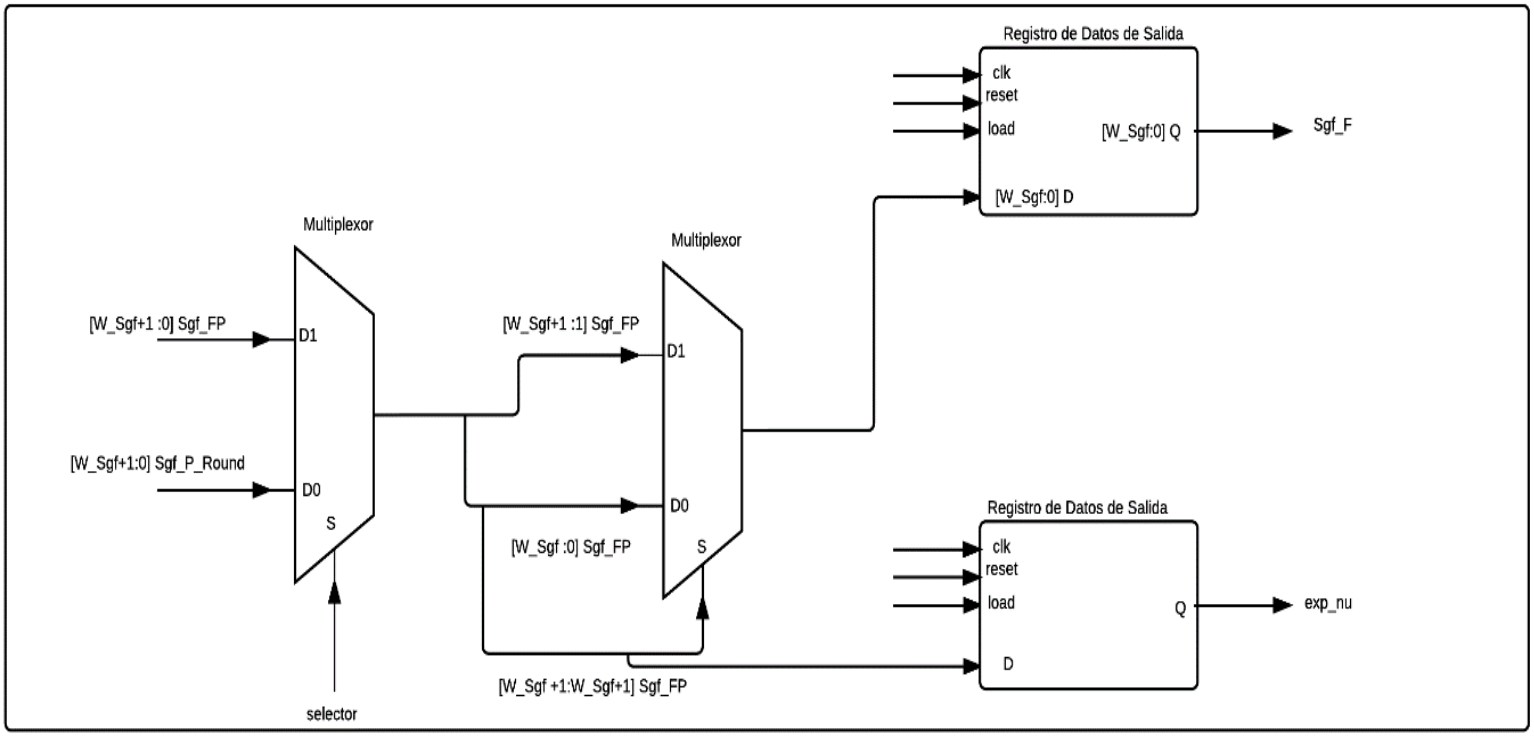

Figura 3.23. Diagrama RTL del módulo Sgf_Norm_M

Por la naturaleza del algoritmo de multiplicación, el proceso de normalización se ejecuta un máximo de dos veces, y se realiza a través de desplazamientos de ceros hacia la derecha del significando.

3.3.1.8 Actualización del exponente y verificación de desborde (Exp_Act_Ov_M)

La estructura que se muestra en la figura 3.24 tiene como función actualizar el exponente (señal **exp_pr** proveniente del módulo *UnBias_Exp_Add*) ante un eventual proceso de normalización en los módulos *Sgf_Norm_M* y *Sgf_Round_Sgn_Info.* Además, se encarga de verificar que el exponente actualizado siga siendo representable por el formato de precisión (a través de la señal **overflow_b**). El exponente resultante de este módulo (señal **exp_act**) corresponde al campo de exponente del resultado final.

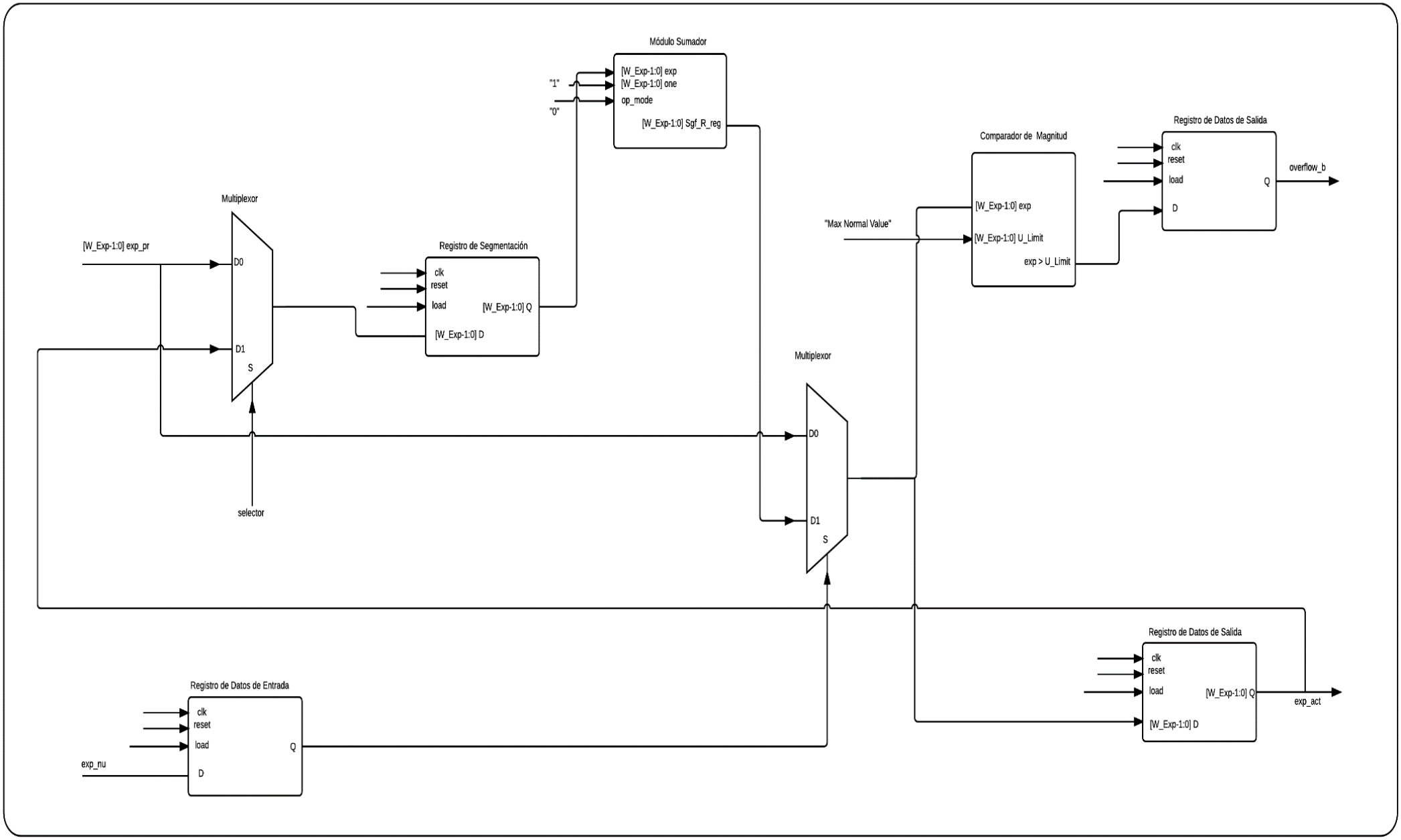

Figura 3.24. Diagrama RTL del módulo Exp_Act_Ov

3.3.1.9 Etapa de redondeo del significando (Sgf_Round_Sgn_Inf)

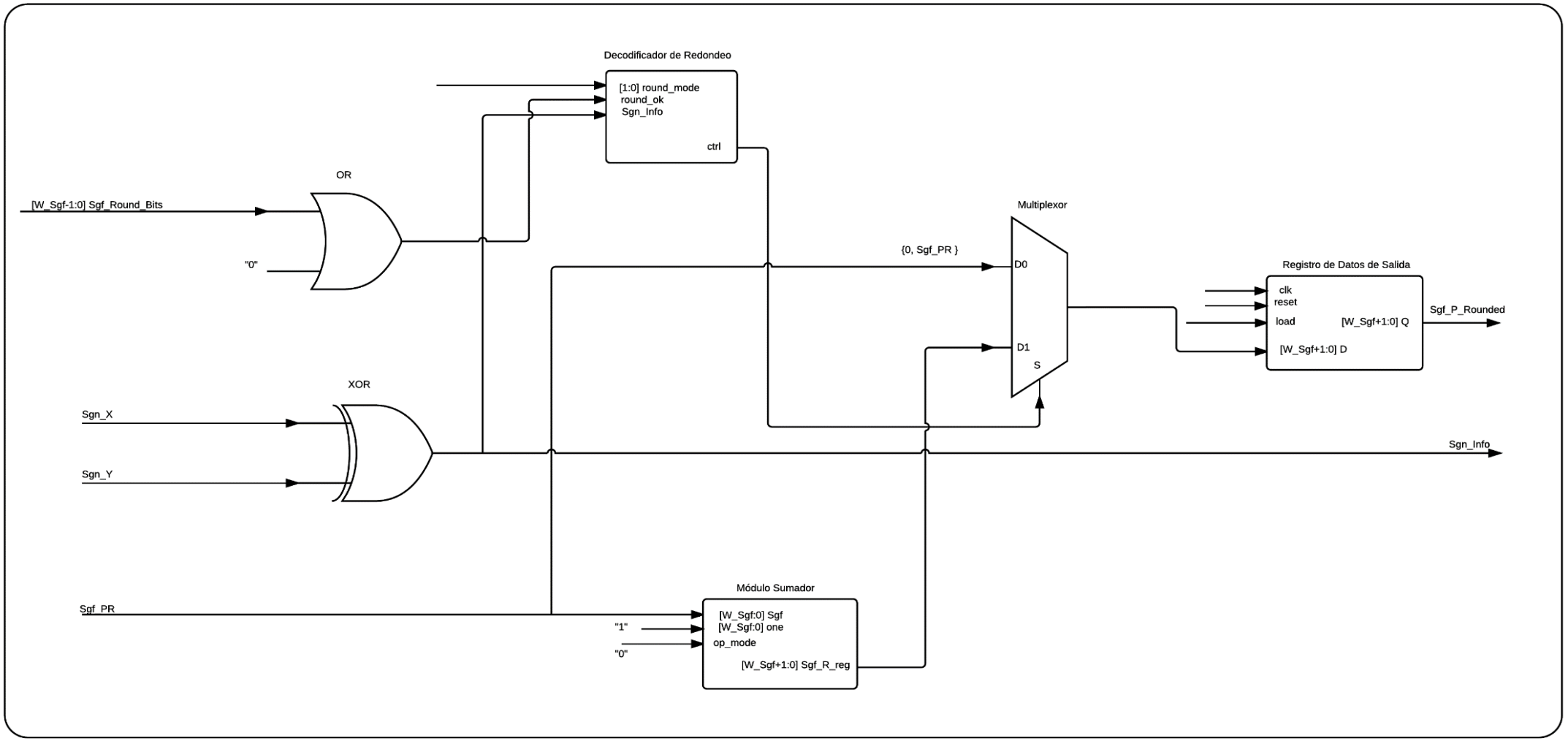

Figura 3.25 Diagrama RTL del módulo Sgf_Round_Sgn_Inf

La estructura presentada en la figura 3.25 se encarga del proceso de redondeo del significando proveniente del módulo *Sgf_Norm_M* (señal **Sgf_F**). Además, se encarga de generar el signo del resultado de la multiplicación (señal **Sgn_Info**).

El proceso de redondeo utiliza información referente a los bits de propósito de redondeo obtenidos en el módulo *Sgf_Mult_M* (ver figura 3.21), el signo del resultado final (generado a través de una compuerta XOR), y la codificación de redondeo (señal **round_mode**) para decidir si sumar un 1 a la señal **Sgf_F**. El significando que se obtiene de este módulo (señal **Sgf_P_Rounded**) se somete nuevamente al proceso de normalización llevado a cabo por el módulo *Sgf_Norm_M*. Después de este nuevo proceso de normalización, se obtiene el significando del resultado final. Este significando corresponde a la señal **Sgf_F** perteneciente al módulo *Sgf_Norm_M***.**

3.3.1.10 Etapa de ensamblado de resultado Final (Final_Result_M_AssemM)

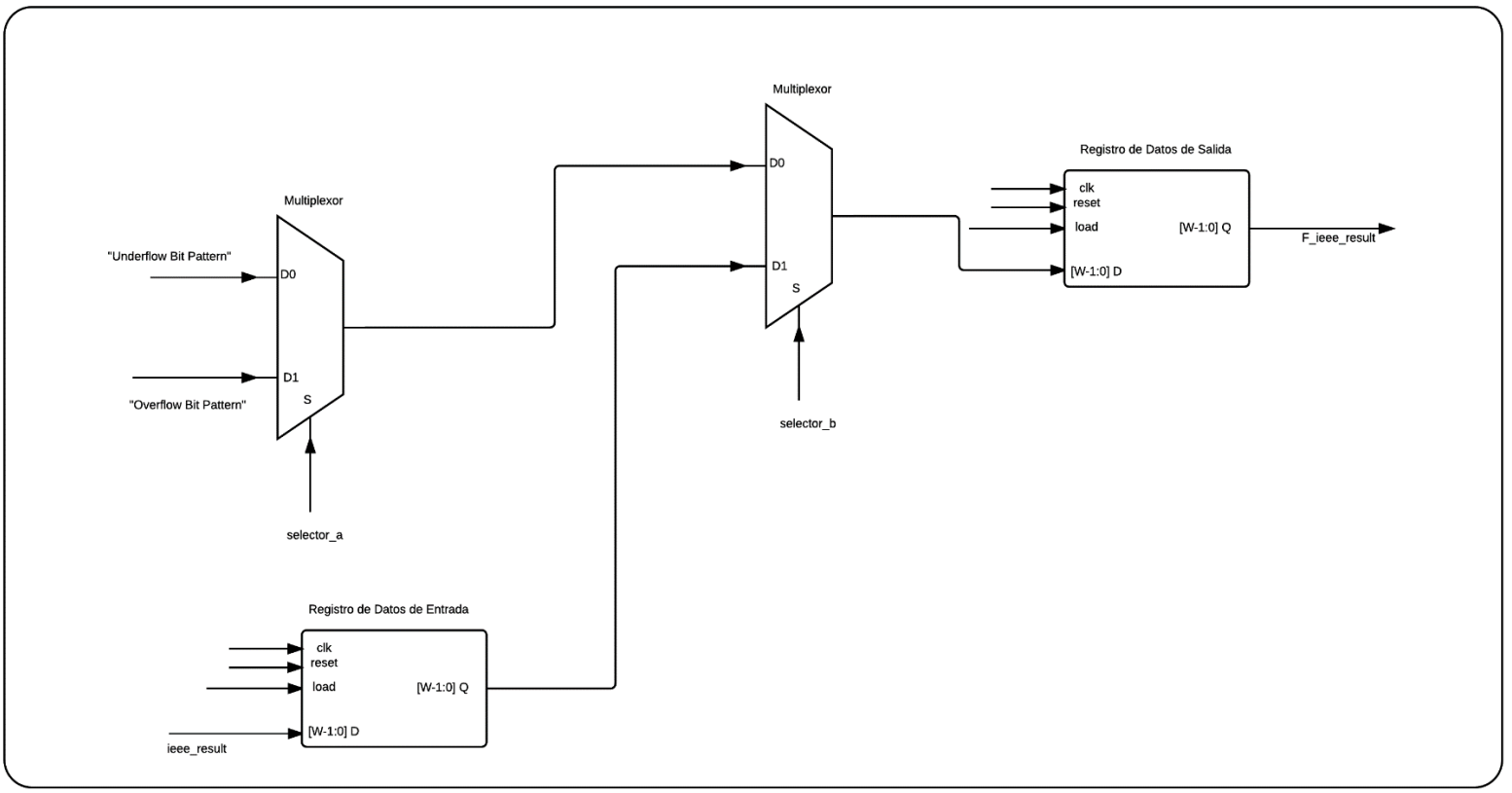

Figura 3.26 Diagrama RTL del módulo Final_M_Result_AssemM

Esta etapa corresponde al ensamblado del resultado final de la operación aritmética de multiplicación. El módulo encargado de ejecutar esta función se muestra en la figura 3.26.

El procedimiento de ensamblado del resultado final es similar al descrito para el algoritmo de suma-resta. Si no se genera un caso excepcional, el resultado final se construye utilizando los campos de signo (señal **Sgn_Info** proveniente del módulo *Sgf_Round_Sgn_Inf*), exponente (señal **exp_act** proveniente del *módulo Exp_Act_Ov_M*), y mantisa (se obtiene al extraer el bit implícito de la señal **Sgf_F** proveniente del módulo *Sgf_Norm_M*) obtenidos durante la ejecución del algoritmo. En caso de presentarse un caso de desborde o sub-desborde, la respuesta final corresponde al patrón de bits característico para cada una de estos casos.

3.3.1.11 Máquina de estados finitos para el algoritmo de multiplicación

El manejo del flujo de datos a través de los bloques mostrados en la figura 3.15 se llevó a cabo utilizando una FSM de 45 estados. Este algoritmo de control se describe utilizando líneas de pseudocódigo, cuyo objetivo es brindar una compresión general del módulo aritmético de multiplicación. Dicho pseudocódigo sigue la misma metodología de descripción que el presentado para la operación aritmética de suma-resta.

1. Verificar si alguno de los exponentes es igual a cero. Si esto es afirmativo, se finaliza la operación y se devuelve como resultado un cero. En caso contrario se procede con el paso 2.
2. Sumar los exponentes.
3. Verificar que el resultado obtenido en el paso 2 sea mayor que el valor de sesgo. Si el resultado es menor, se genera una excepción de sub-desborde y se finaliza el algoritmo devolviendo como resultado un cero. En caso contrario se procede con el paso 4.
4. Restar el valor del sesgo al resultado del paso 2.
5. Verificar que el exponente obtenido en el paso 4 no genere un caso de desborde. Si esta excepción ocurre, se establece la bandera de desborde respectiva y se finaliza el algoritmo devolviendo como resultado el valor de infinito positivo o infinito negativo. En caso contrario se sigue con el paso 6.
6. Multiplicar los significandos.
7. Normalizar (en caso de necesitarse) el significando resultante del paso 6.
8. Actualizar (en caso de necesitarse) el exponente obtenido del paso 4.
9. Redondear el significando resultante del paso 7.
10. Si se desnormaliza el significando luego del redondeo (paso 9), se debe normalizar nuevamente.
11. Actualizar (en caso de necesitarse) el exponente del paso 8 ante un eventual proceso de normalización en el paso 10.
12. Construir el resultado final.

CAPÍTULO 4

Resultados y análisis

En este capítulo se presentan y discuten los resultados obtenidos del desarrollo de los módulos aritméticos. El capítulo ha sido dividio en tres secciones, en las cuales se describe el ambiente de verificación utilizado, y se presentan resultados referentes a la implementación en hardware de las operaciones de suma, resta y multiplicación.

Cabe destacar que los resultados que se presentan corresponden a las arquitecturas de 32 bits de los módulos aritméticos, ya que la metodología de verificación para las arquitecturas de 64 bits es la misma. Sin embargo, la funcionalidad de estas últimas puede ser consultada en la tesis de maestría del Ingeniero Carlos Salazar [17], el cual desarrolló el procesador de aplicación específica al que debían integrarse los módulos aritméticos desarrollados en este proyecto, para la ejecución del HMM. Los resultados obtenidos de dicha integración fueron satisfactorios ya que se logró ejecutar de manera precisa dicho modelo acústico.

4.1 Metodología de verificación

Inicialmente, se generaron dos vectores de datos de 2^n elementos, correspondientes a los operandos. Para esto se utilizó una rutina encargada de generar números aleatorios (desarrollada en el software GNU Octave), en la que podía ajustarse el parámetro *n* para definir las longitudes de los vectores, además de poder establecer el rango de valores en el que se ubicarían. Una vez generados los operandos, a estos se aplicaron las diferentes operaciones aritméticas, tanto a nivel de software como a nivel de hardware, con el fin de generar vectores de datos correspondientes a los resultados teóricos y experimentales, respectivamente.

Para generar los resultados teóricos se utilizó GNU Octave, en el cual se desarrolló una rutina que realizaba el proceso de lectura de los operandos, y efectuaba sobre estos las operaciones de suma, resta o multiplicación. Por último, estos resultados se almacenaron en un vector de datos.

Respecto a los resultados experimentales, se generó un banco de pruebas utilizando el simulador ISE Simulator (ISIM) de la herramienta de software ISE, producida por Xilinx. Dentro del banco de pruebas se desarrolló una rutina que tuvo las siguientes funciones:

- Establecer la frecuencia de operación de los módulos.
- Realizar el proceso de lectura de los vectores de datos correspondientes a los operandos.
- Generar las señales de estímulo necesarias para inicializar o finalizar la ejecución de los módulos de suma, resta o multiplicación.
- Almacenar en un vector de datos los resultados *Post Place & Route* obtenidos del proceso de ejecución de los módulos aritméticos.

Finalmente, se desarrolló una rutina en GNU Octave encargada de generar representaciones gráficas referentes a los resultados teóricos, experimentales y a los porcentajes de error existentes entre estos resultados. El porcentaje de error se calculó según la siguiente relación:

$$\%\, Error = \frac{|Valor\ teórico - \ \ Valor\ expermiental|}{Valor\ teórico} \times\ 100$$

Estas representaciones gráficas se utilizaron como punto de partida para verificar la funcionalidad de los módulos, estimar un rango de precisión, y discutir la aceptabilidad de los resultados obtenidos.

La aceptabilidad de los resultados se discutió con el Ing. Carlos Salazar, definiéndose que el porcentaje de error entre los resultados teóricos y experimentales no podía exceder un 5%.

4.1.1 Consideraciones importantes del proceso de verificación.

En el proceso de verificación, se efectuaron pruebas utilizando vectores de datos (referentes a los operandos) de diferente longitud, y ubicados en diferentes rangos numéricos. Como se mencionó anteriormente (sección 4.1), los parámetros referentes a la longitud y el rango de los operandos, podían predefinirse en la rutina encargada de generarlos. Los rangos y longitudes utilizados en este proyecto se muestran en las tablas 4.1 y 4.2.

Tabla 4.1: Longitud del vector de datos para los diferentes operandos. La cantidad de valores que se pueden generar depende del parámetro *n*.

Valor del parámetro n	$Longitud\ del\ vector = 2^n$ valores
5	32
6	64
7	128
8	254
9	512
10	1024
11	2048
12	4096
13	8192

Tabla 4.2: Rangos utilizados para generar los operandos. Los operandos pueden tomar cualquier valor dentro del rango y poseer signo positivo o negativo.

Rangos numéricos de los operandos
$\pm[0 - 0.0000001]$
$\pm[0 - 0.000001]$
$\pm[0 - 0.00001]$
$\pm[0 - 0.0001]$
$\pm[0 - 0.001]$
$\pm[0 - 0.01]$
$\pm[0 - 0.1]$
$\pm[0 - 0.5]$
$\pm[0 - 1]$
$\pm[0 - 10]$
$\pm[0 - 20]$
$\pm[0 - 30]$
$\pm[0 - 40]$
$\pm[0 - 50]$
$\pm[0 - 100]$
$\pm[0 - 1000]$
$\pm[0 - 10000]$
$\pm[0 - 100000]$
$\pm[0 - 1000000]$

La selección de rangos mostrada en la tabla 4.2 permitió evaluar la respuesta de los módulos ante estímulos (operandos) de diferentes órdenes de magnitud, mientras que utilizar diferentes longitudes en los vectores de datos (tabla 4.1), permitió de manera gradual aumentar la densidad de resultados obtenidos del proceso de ejecución. Esto

último sirvió como herramienta de depuración, ya que al efectuar más operaciones podría identificarse algún caso en el que el cómputo aritmético fuese erróneo.

Durante el proceso de verificación, se realizaron simulaciones exhaustivas sobre los módulos aritméticos, cuyos operandos se generaron a partir de combinaciones entre los diferentes rangos y las diversas longitudes de vector. Por ejemplo, en una simulación *Post Place & Route* del módulo de resta se realizaban cinco iteraciones utilizando el mismo rango pero con diferentes longitudes de vector. Esta metodología se aplicó a todos los rangos, y la validez de los resultados se verificó utilizando el procedimiento descrito en la sección 4.1.

4.2 Resultados del desarrollo del módulo de suma-resta.

A continuación, se ofrecen los resultados relacionados a la implementación en hardware del módulo de suma-resta.

4.2.1 Resultados de la operación aritmética de suma

Para el módulo de suma, se realizaron dos simulaciones que utilizan vectores de 2048 y 1024 valores como operandos, cuyos rangos son de $\pm[0-10]$ y $\pm[0-100000]$ respectivamente. Para estas simulaciones se utilizó la codificación de redondeo hacia cero.

Los resultados obtenidos se muestran en las figuras 4.1 y 4.2, las cuales corresponden a representaciones gráficas de los resultados teóricos, los resultados experimentales, y el porcentaje de error existente entre estos. En estos gráficos cada punto hace referencia al resultado obtenido tras ejecutar el proceso de suma. Como puede observarse, este proceso se ejecutó tantas veces como valores tienen los vectores de datos.

Los gráficos correspondientes a los resultados teóricos y experimentales, presentan patrones de puntos similares. Para cuantificar esta similitud, se realizó un gráfico del porcentaje de error, en el cual se puede observar que los mayores porcentajes obtenidos corresponden aproximadamente a 6×10^{-5} para la simulación con 2048 valores, y a $3{,}5 \times 10^{-5}$ para la simulación con 1024 valores. Estos porcentajes de error representan un grado de precisión aceptable en los resultados obtenidos, según lo discutido con el Ing.

Carlos Salazar. Nótese que los porcentajes de error corresponden a valores menores a un 5%.

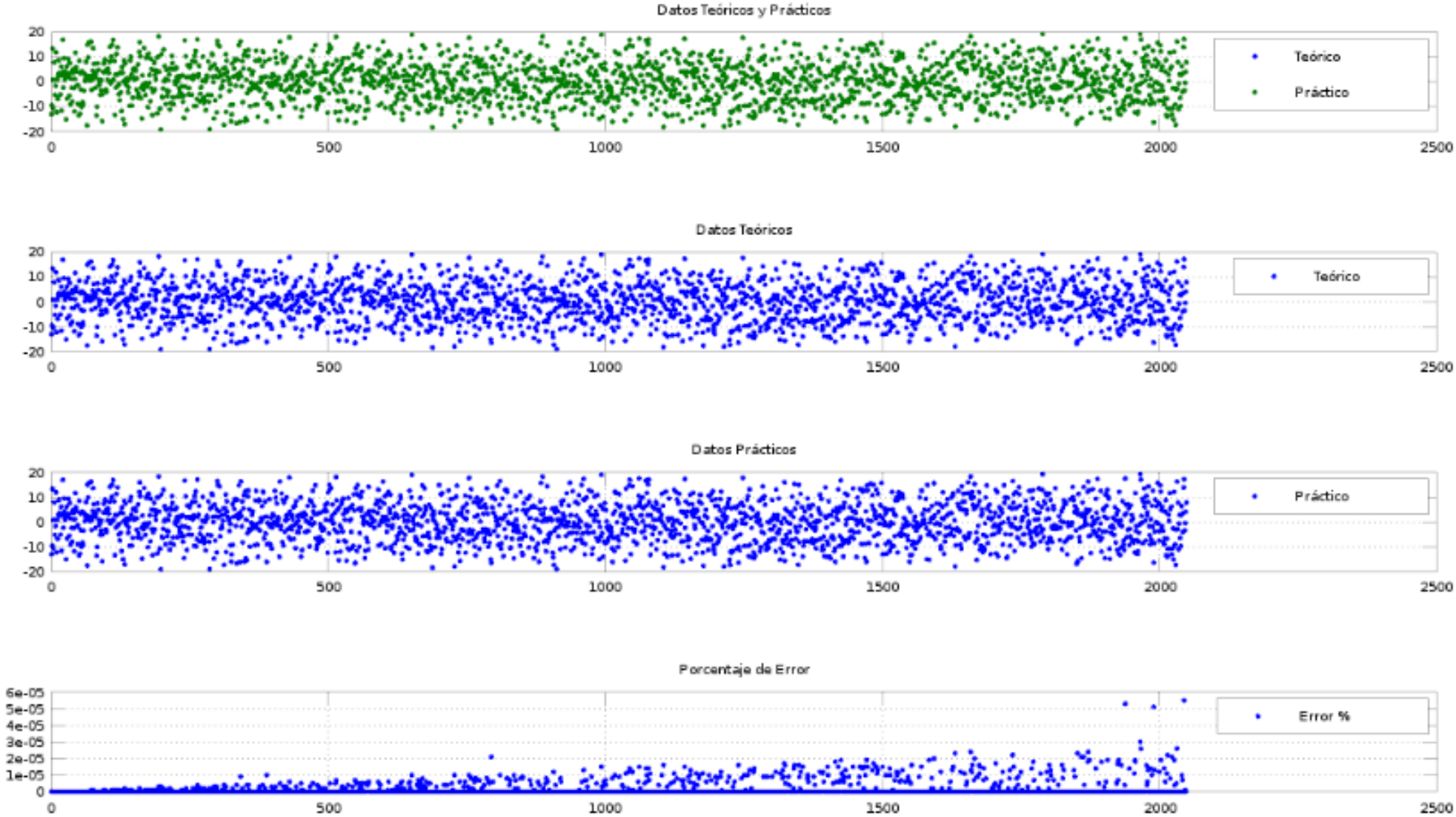

Figura 4.1. Representaciones gráficas del porcentaje de error, y de los resultados teóricos y prácticos obtenidos durante el proceso de suma para el rango de $\pm[0-10]$

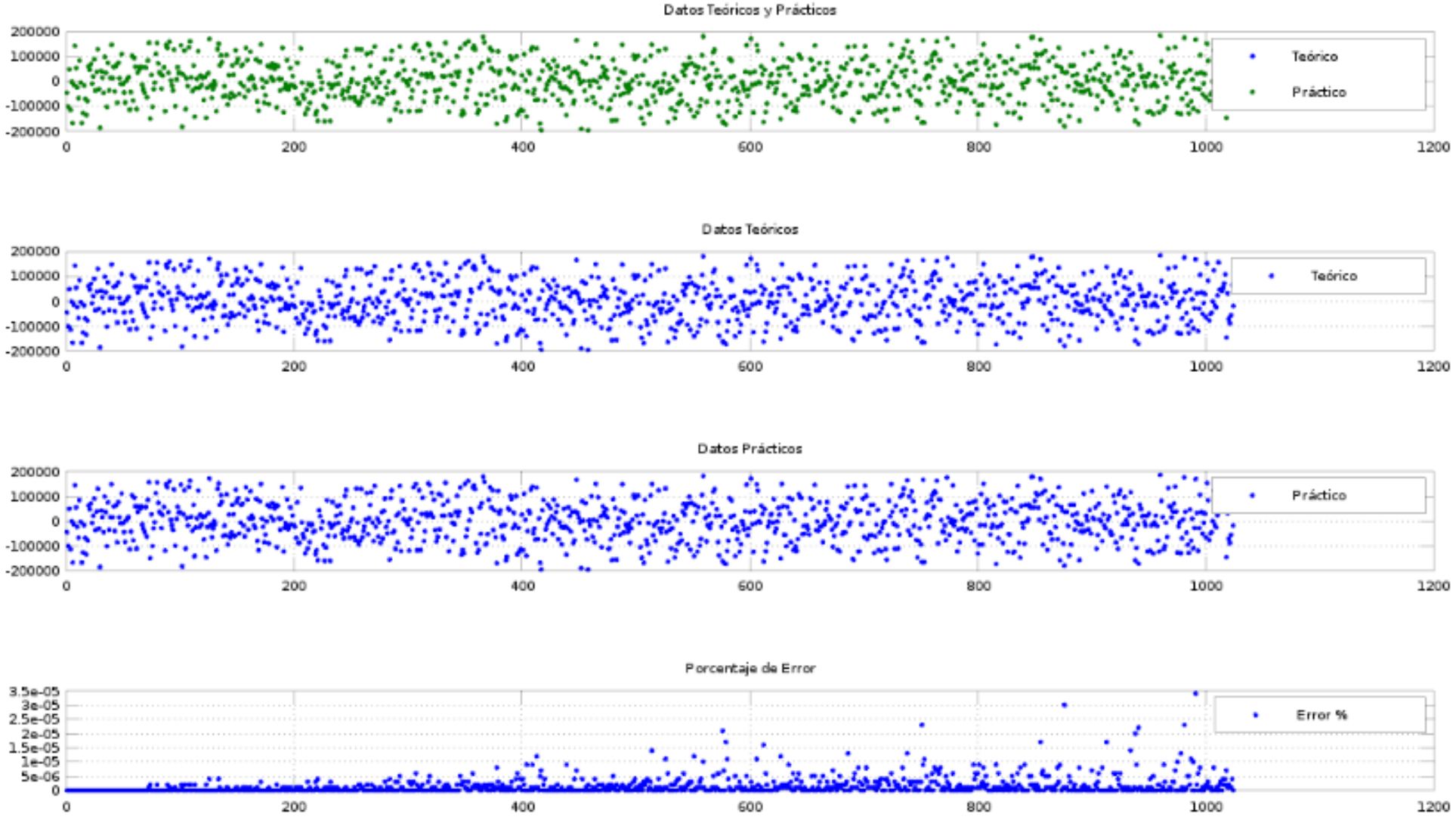

Figura 4.2. Representaciones gráficas del porcentaje de error, y de los resultados teóricos y prácticos obtenidos durante el proceso de suma para el rango de $\pm[0 - 100000]$.

4.2.2 Resultados de la operación aritmética de resta

Para mostrar los resultados de esta operación, se realizaron dos simulaciones utilizando la codificación de redondeo hacia menos infinito. Para estas simulaciones se utilizaron como operandos 1024 valores en un rango de $\pm[0 - 0{,}5]$ y 2048 valores en un rango de $\pm[0 - 1000000]$.

Se generaron representaciones gráficas para analizar y cuantificar el grado de similitud entre los resultados (figuras 4.3 y 4.4), en las cuales se observa que los resultados teóricos y experimentales son similares. Los porcentajes de error asociados corresponden aproximadamente a $2{,}5 \times 10^{-4}$ para la simulación con 1024 valores, y a $2{,}5 \times 10^{-5}$ para la simulación con 2048 valores. Por lo tanto, se puede afirmar que los resultados generados por el módulo de resta son aceptables, al ser los porcentajes de error menores a un 5%.

4.2.3 Rango del porcentaje de error para el módulo de suma-resta

Simulaciones como las realizadas en las secciones 4.2.1 y 4.2.2 se llevaron a cabo de manera exhaustiva, combinando cada rango (ver tabla 4.2) con diferentes longitudes de vector (ver tabla 4.1). Los resultados obtenidos respecto al porcentaje de error permitieron estimar un rango para este, entre 1×10^{-6} y 1×10^{-3}. Asegurando un grado de precisión aceptable en los resultados obtenidos de la ejecución del módulo de suma-resta (porcentajes de error menores a un 5%).

Los mayores porcentajes de error que se observan, se deben al uso de una versión reducida del estándar IEEE 754, específicamente en lo que corresponde a los bits para propósitos de redondeo. Para este proyecto por cuestiones de recursos y precisión no se utilizó el sticky bit, cuya ausencia genera la pérdida de precisión en determinados resultados durante el proceso de redondeo.

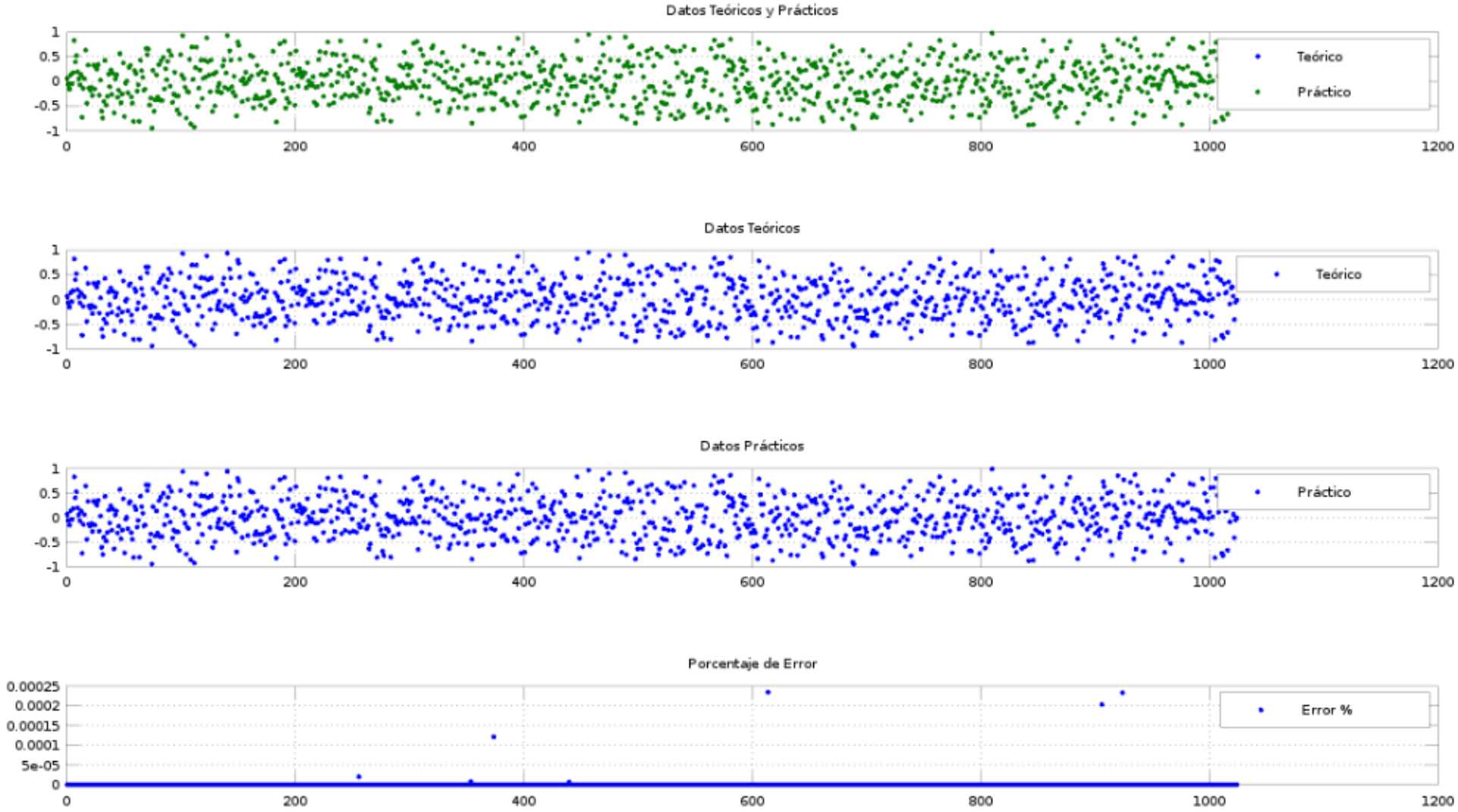

Figura 4.3. Representaciones gráficas del porcentaje de error, y de los resultados teóricos y prácticos obtenidos durante el proceso de resta para el rango de $\pm[0 - 0{,}5]$

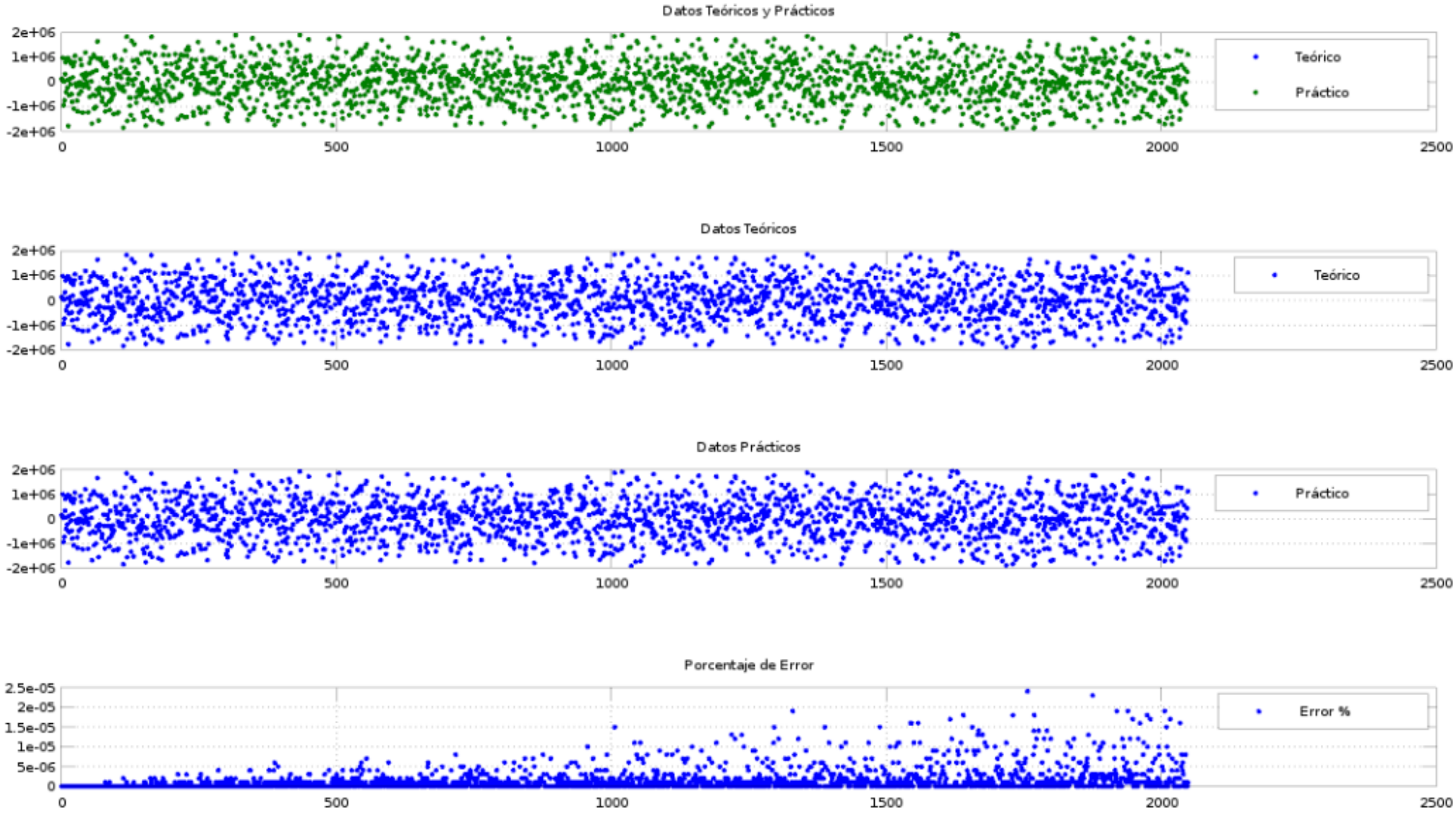

Figura 4.4. Representaciones gráficas del porcentaje de error, y de los resultados teóricos y prácticos obtenidos durante el proceso de resta para el rango de $\pm[0 - 1000000]$.

4.2.4 Casos excepcionales

Dada la poca probabilidad de que los operandos generados aleatoriamente en el proceso de verificación produjeran una operación cuyo resultado no fuera representable por el formato de precisión, se decidieron generar dos casos puntuales en dónde se puede visualizar el comportamiento del módulo de suma-resta ante un caso excepcional de desborde o sub-desborde.

En la figura 4.5 se realiza la resta de los números $1{,}17549435 \times 10^{-38}$ y $1{,}1754945 \times 10^{-38}$, la cual genera un resultado muy pequeño para ser representado por el formato de precisión simple (caso excepcional de sub-desborde). Por otro lado, en la figura 4.6 se realizó la suma de los valores 2×10^{38} y 3×10^{38}, cuyo resultado es un número muy grande para ser representado por el formato (caso excepcional de desborde).

Note que en ambas figuras las banderas de desborde y sub-desborde están establecidas en 1, y que los resultados finales corresponden a los patrones de bits referentes a estos casos (0x00000000 para el caso de sub-desborde, y 0x7f800000 para el caso de desborde).

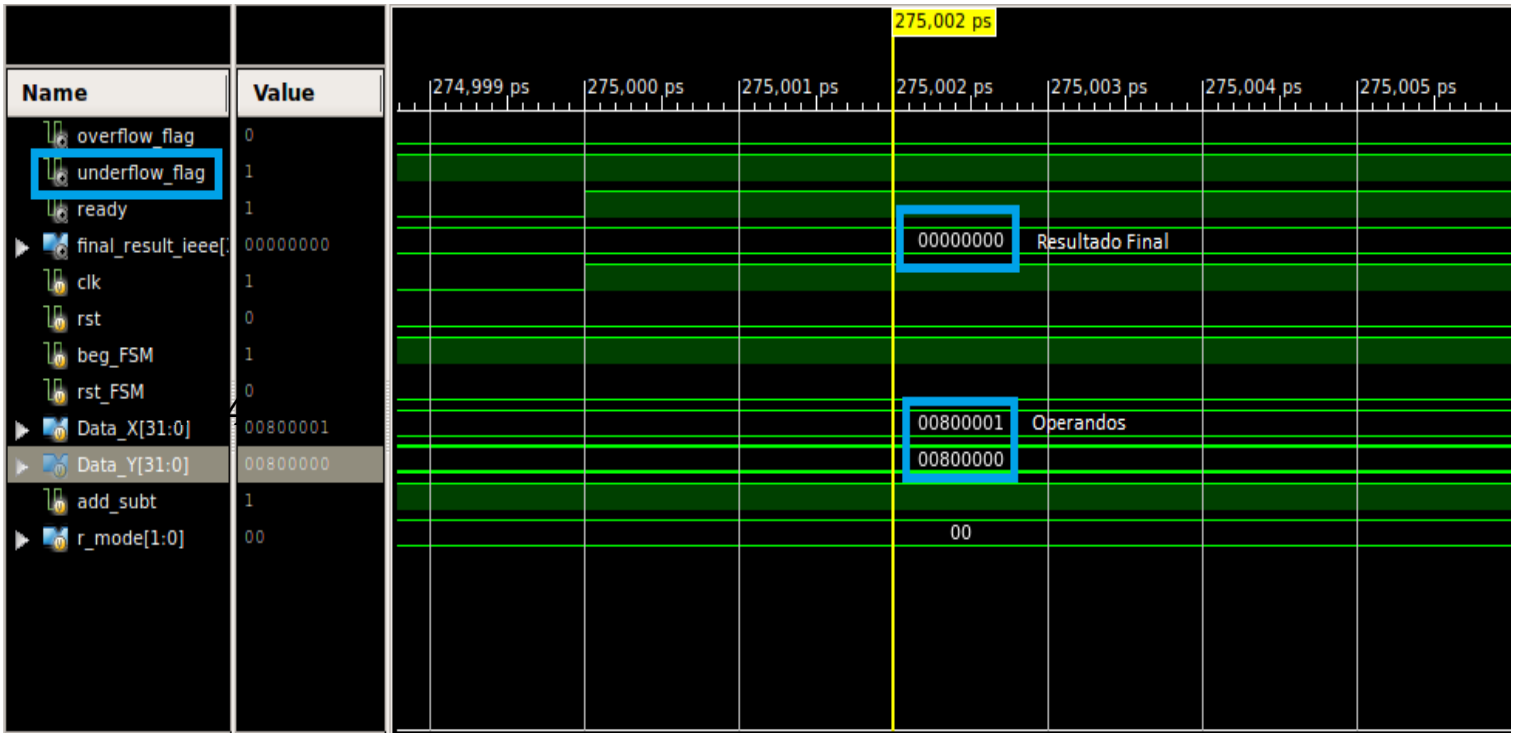

Figura 4.5. Caso puntual de la resta que genera una condición de sub-desborde.

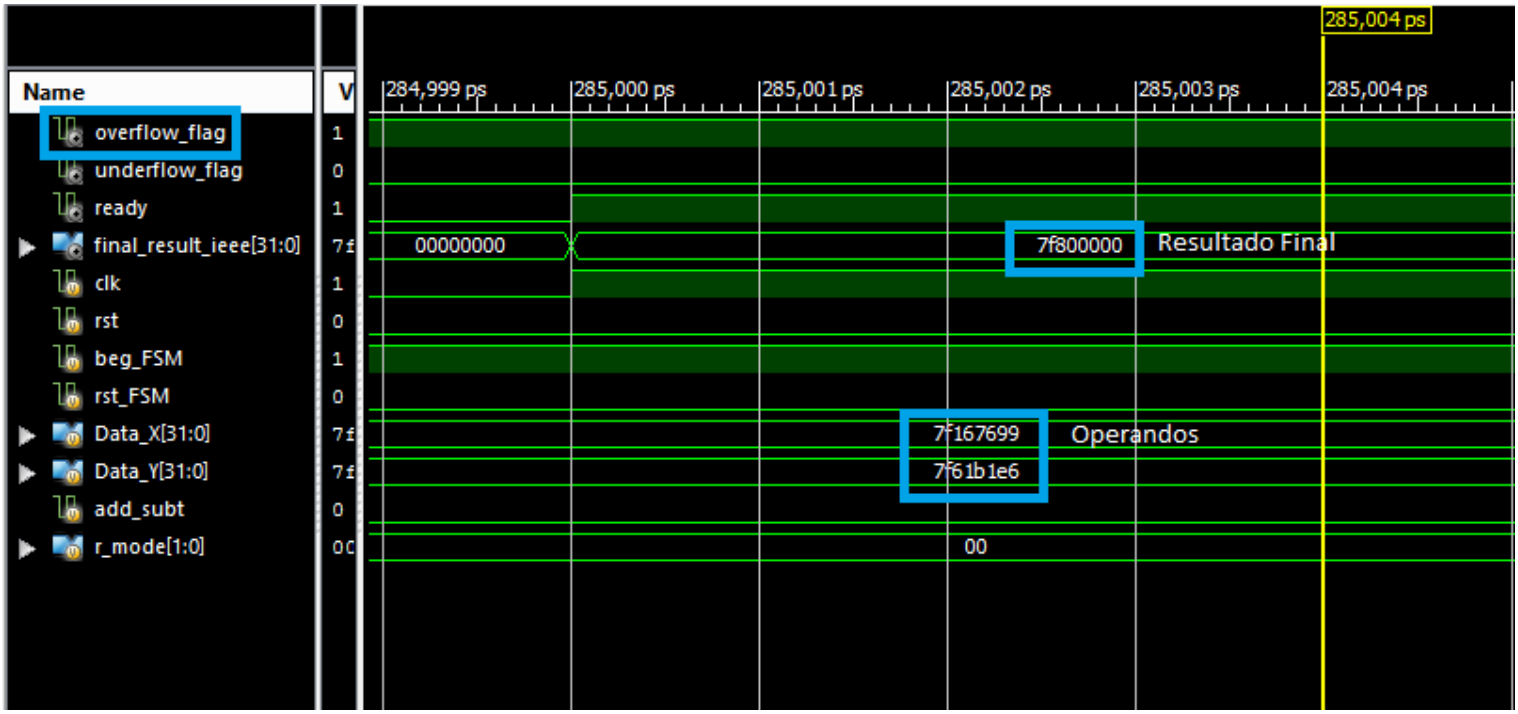

Figura 4.6. Caso puntual de la suma que genera una condición de desborde.

4.2.5 Información relevante para las secciones de recursos utilizados, consumo de potencia y reporte de tiempos.

Antes de proceder con la descripción de las secciones mencionadas, es necesario aclarar la metodología que se utilizó para llegar a los resultados obtenidos. Las herramientas utilizadas para generar reportes de consumo de recursos, de potencia y de tiempos, necesitan que todas las entradas y salidas de los módulos aritméticos estén mapeadas a pines de la FPGA. Sin embargo, la interconexión de las señales de entrada y salida de los módulos desarrollados, únicamente se realiza a lo interno de la placa de prototipos, por lo que carece de sentido mapearlas a nivel de pines. Esto último se corroboró al mapear los pines de las arquitecturas de 64 bits, y generar el reporte *Post Place & Route* del consumo de recursos, en el cual se observó que en la síntesis de los módulos se utilizó un 95% de los buffers de entrada-salida, lo cual no representa un consumo de área real.

Para evitar la situación descrita anteriormente y poder generar resultados aproximados satisfactorios, se desarrolló un módulo de cobertura para las unidades aritméticas (incluído el multiplicador). Este módulo tiene como función permitir la conexión interna de los módulos aritméticos, y ofrecer una interfaz de entrada-salida cuyo consumo de recursos (uso de buffers de entrada-salida) sea despreciable. Esto con el fin de obtener resultados más confiables respecto al manejo de recursos, consumo de potencia, y reporte de tiempos por parte de los módulos aritméticos.

4.2.6 Recursos utilizados

En las tablas 4.3 y 4.4 se presenta un resumen del uso de recursos por parte de las arquitecturas de 32 y 64 bits, respectivamente. Esta información fue extraída de un informe Post Place & Route generado por la herramienta ISE.

Tabla 4.3. Resumen del reporte *Post Place & Route* del uso de dispositivos generado por la herramienta ISE, para la arquitectura de 32 bits del módulo de suma-resta

Slice Logic Utilization	Available Devices	Used Devices	Percent of utilization
Number of slice registers used as Flip Flops	126800	576	1%
Number of slice LUTs	63400	412	1%
Number of occupied slices	15850	159	1%
Number of LUT flip flop pairs used with an unused flip flop	559	120	21%
Number of LUT flip flop pairs used with an unused LUT	559	147	26%
Number of fully used LUT FF pairs	559	292	52%
DSP48E1's	240	0	0%

Tabla 4.4. Resumen del reporte *Post Place & Route* del uso de dispositivos generado por la herramienta ISE, para la arquitectura de 64 bits del módulo de suma-resta

Slice Logic Utilization	Available Devices	Used Devices	Percent of utilization
Number of slice registers used as Flip Flops	126800	1085	1%
Number of slice LUTs	63400	821	1%
Number of occupied slices	15850	282	1%
Number of LUT flip flop pairs used with an unused flip flop	1032	260	25%
Number of LUT flip flop pairs used with an unused LUT	1032	211	20%

Number of fully used LUT FF pairs	1032	561	54%
DSP48E1's	240	0	0%

Como se puede observar, el consumo de recursos (dispositivos utilizados) aumenta considerablemente cuando se sintetiza la arquitectura de 64 bits respecto a la de 32 bits.

4.2.7 Consumo de Potencia

Haciendo uso de la herramienta gráfica XPower Analyzer (integrada en la herramienta ISE), se generó un reporte referente al consumo de potencia estática y dinámica de las dos arquitecturas del módulo de suma-resta. Los resultados obtenidos se resumen en la tablas 4.5 y 4.6.

Tabla 4.5. Resumen del reporte generado por la herramienta XPower Analizer que indica el consumo estático, dinámico y total de potencia para ambas arquitecturas del módulo suma-resta.

Arquitectura de 32 bits			Arquitectura de 64 bits		
Consumo de potencia estática (mW)	Consumo de potencia dinámica (mW)	Consumo total de potencia (mW)	Consumo de potencia estática (mW)	Consumo de potencia dinámica (mW)	Consumo total de potencia (mW)
88,20	8,02	96,22	88,22	12,04	100,26

Tabla 4.6. Resumen del reporte generado por la herramienta XPower Analizer que indica el consumo de potencia On-Chip de diversos elementos en el módulo de suma-resta.

	Arquitectura de 32 bits	Arquitectura de 64 bits
On-Chip	Power(mW)	Power (mW)
Clocks	2,18	4,51
Logic	1,07	1,96
Signals	1,26	2,51
IOs	3,51	3,07
Static Power	88,20	88,22
Total	96,22	100,26

De la tabla 4.5, se puede observar que la potencia estática de ambas arquitecturas se mantiene relativamente constante. El consumo de potencia estática es propio de la FPGA y es independiente del diseño que se implemente en ella. Respecto al consumo de potencia dinámica, se puede apreciar (tabla 4.5) un incremento en la arquitectura de 64 bits respecto a la de 32 bits, dicho incremento se puede justificar observando la tabla 4.6, en el cual se presenta un desgloce de la potencia On-Chip consumida por diferentes elementos.

La potencia consumida por las señales de entrada salida (IOs), es irrelevante por lo mencionado en la sección 4.2.5. Esto es un claro indicador de que la potencia dinámica se sobreestima en la estimación presentada en las tablas 4.5 y 4.6. Por otro lado, la potencia consumida por los relojes en la arquitectura de 64 bits es 2,07 veces mayor que en la arquitectura de 32 bits, mientras que la potencia consumida por los bloques de lógica y las señales de los módulos aproximadamente se duplica respecto a la arquitectura de 32 bits. Además, note la similitud en el consumo estático de ambas arquitecturas.

4.2.8 Reporte de tiempos

Utilizando la herramienta *Timing Analyzer* integrada en el ISE , se realizó un reporte de tiempos para la implementación Post Place & Route de las arquitecturas del módulo de suma-resta. Este reporte corresponde a una estimación, cuyo resultados se resumen en las tablas 4.7 y 4.8.

Tabla 4.7. Resumen del reporte de tiempos para la arquitectura de 32 bits del módulo de suma-resta. Este reporte se generó utilizando la herramienta Timing Analyzer.

Design statistics	
Minimum Period	3,178 ns
Maximum frequency	314,663 MHz
Maximum combinational path delay	4,257 ns
Minimum input required time before clock	0,954 ns
Maximum output delay after clock	9,679 ns

Tabla 4.8. Resumen del reporte de tiempos para la arquitectura de 64 bits del módulo de suma-resta. Este reporte se generó utilizando la herramienta Timing Analyzer.

Design statistics	
Minimum Period	3,947 ns
Maximum frequency	253,357 MHz
Maximum combinational path delay	4,311 ns
Minimum input required time before clock	1,415ns
Maximum output delay after clock	9,800 ns

El período mínimo corresponde al menor tiempo de retraso entre un elemento síncrono y otro. De esto último, se vuelve notoria la importancia de segmentar rutas combinacionales a través del uso de registros, ya que se le permite a dichas rutas correr a determinadas frecuencias. Por ejemplo, en el caso de la arquitectura de 32 bits, el período mínimo corresponde a 3,178 ns, permitiéndole a rutas combinaciones ubicadas entre registros ejecutarse a una frecuencia máxima de 314,663 MHz. Para el caso de la

arquitectura de 64 bits la máxima frecuencia a la que puede correr una ruta combinacional segmentada es de 253,357 MHz.

Por otro lado, se presenta información referente al tiempo máximo de propagación a través de una ruta combinacional. Este tiempo aplica para rutas que empiezan en la entrada de un bloque y terminan en determinada salida del mismo, sin atravesar por algún registro. En los módulos que se desarrollaron, existen rutas combinacionales no segmentadas, razón por la que se aprecian estos tiempos en el reporte. Estos valores corresponden a 4,257 ns para la arquitectura de 32 bits y 4,311 ns para la de 64 bits, notándose que la diferencia de estos valores es mínima entre arquitecturas.

Además, se obtuvieron resultados referentes al tiempo mínimo en el que alguna de las entradas globales del módulo debe ser válida antes de que ocurra un flanco de reloj positivo. Estos tiempos corresponden a 0,954 ns para la arquitectura de 32 bits y a 1,415 ns para la arquitectura de 64 bits. Nótese que la arquitectura de 32 bits ofrece una ventana de tiempo de 9,046 ns para que las señales de entrada se validen o estabilicen, mientras que para la arquitectura de 64 bits esta ventana corresponde a 8,585 ns.

Finalmente, se presentan los tiempos máximos que le toma a la salida establecerse después del flanco de reloj positivo. Estos tiempos corresponden a 9,679 ns y 9,800 ns para las arquitecturas de 32 y 64 bits respectivamente. Nótese que no existe una diferencia significativa entre estos tiempos.

4.3 Resultados del desarrollo del módulo de multiplicación.

La metodología utilizada para presentar los resultados de este módulo, es exactamente la misma que la expuesta para el módulo de suma-resta. Los resultados que se muestran e las figuras 4.7 y 4.8 corresponden a dos simulaciones, en las cuales se utilizaron 1024 y 2048 valores como operandos, ubicados en los rangos de $\pm[0 - 0{,}5]$ y $\pm[0 - 100]$ respectivamente.

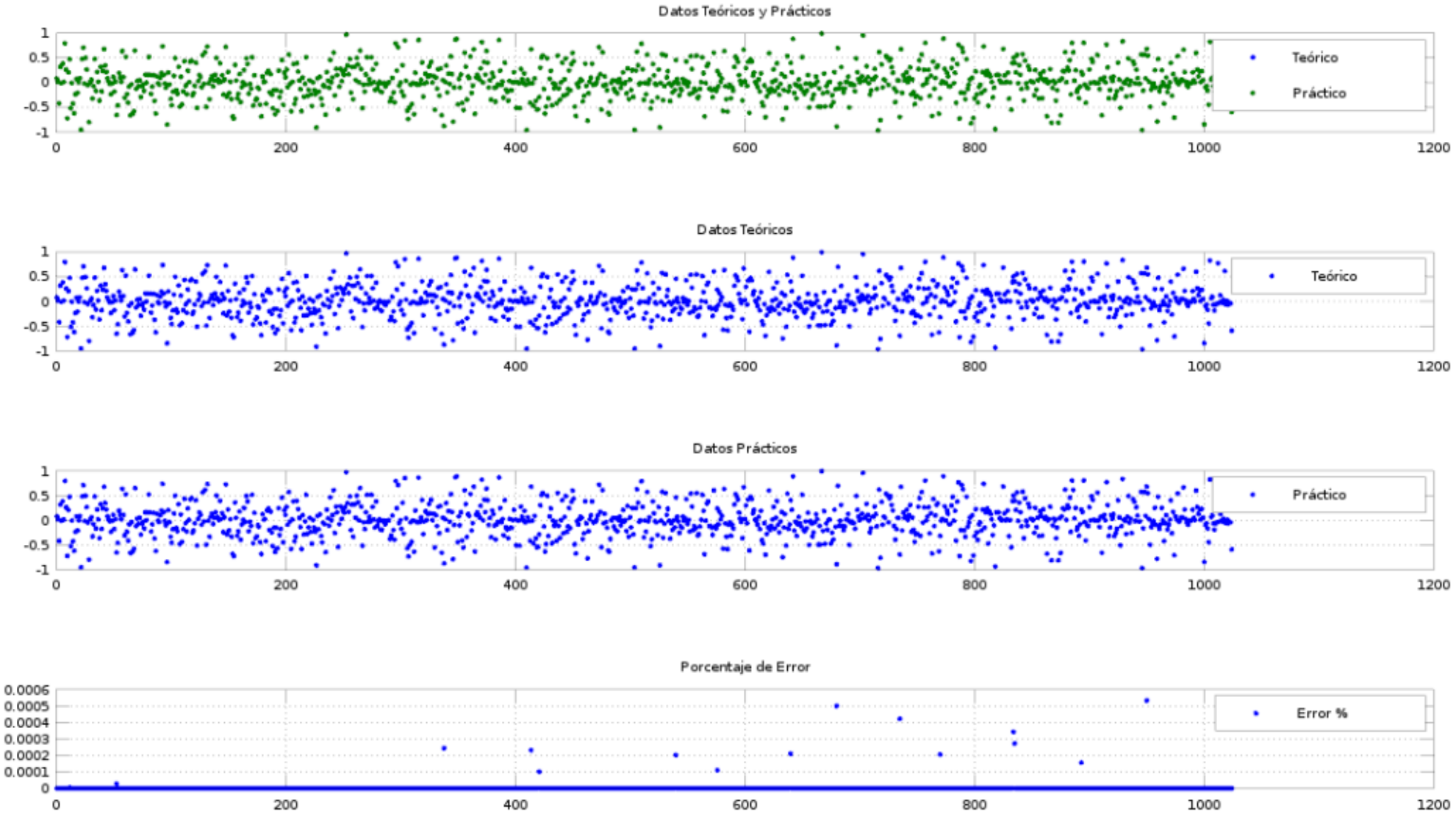

Figura 4.7. Representaciones gráficas del porcentaje de error, y de los resultados teóricos y prácticos obtenidos durante el proceso de multiplicación para el rango de $\pm[0 - 0{,}5]$.

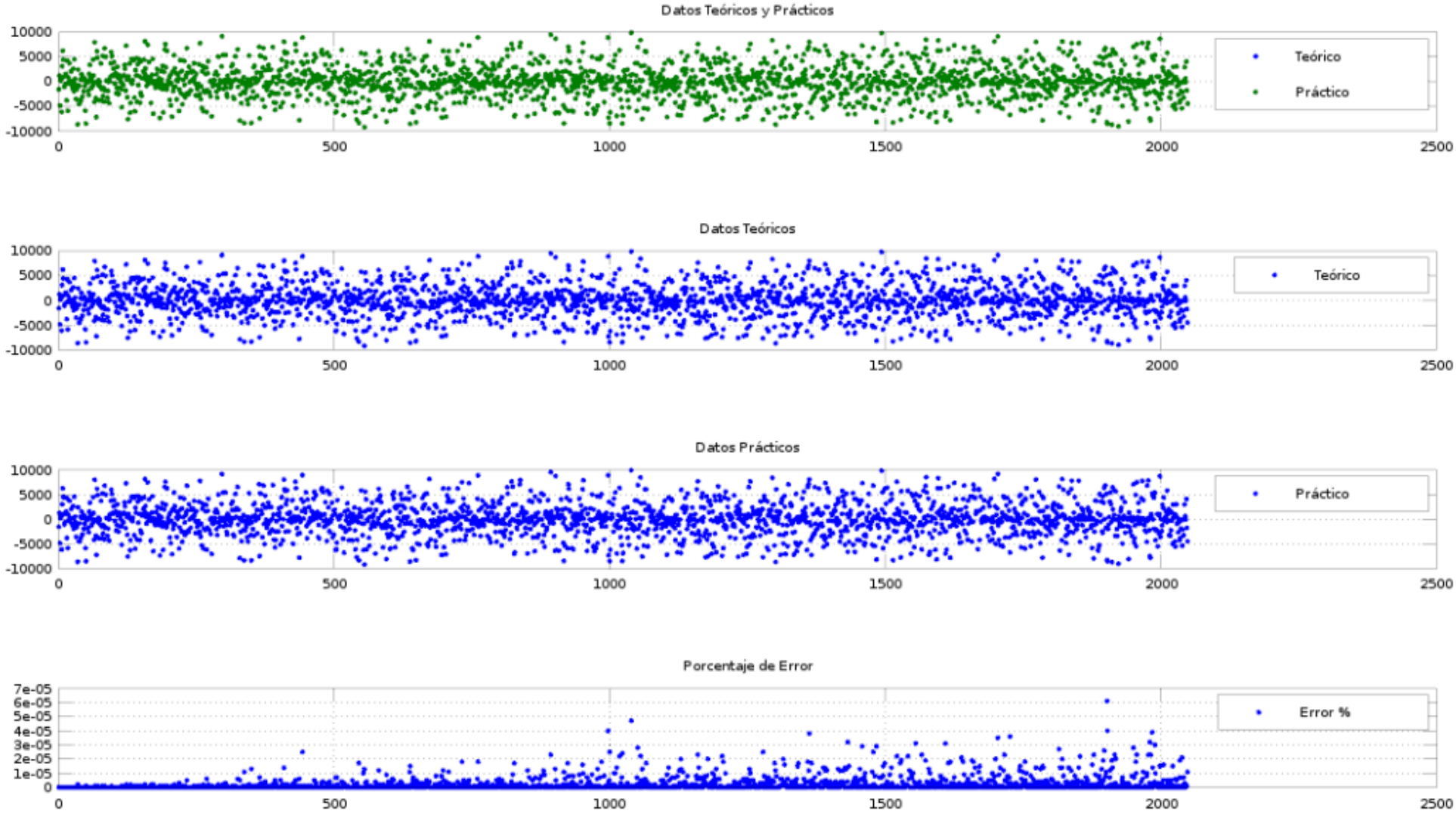

Figura 4.8. Representaciones gráficas del porcentaje de error, y de los resultados teóricos y prácticos obtenidos durante el proceso de multiplicación para el rango de $\pm[0-100]$.

Note que los porcentajes de error máximos que se obtuvieron, corresponden aproximadamente a 6×10^{-4} para la simulación con 1024 valores, y a 6×10^{-5} en la simulación con 2048 valores. Un aspecto a destacar, es que cuando se multiplican números cuya magnitud es menor a 1, se obtienen porcentajes de error igual a cero (véase la figura 4.7).

4.3.1 Rango del porcentaje de error para el módulo de multiplicación.

Al igual que en el módulo de suma-resta, las simulaciones utilizadas para obtener los resultados de la sección 4.3, se realizaron de manera exhaustiva combinando diferentes rangos con diferentes longitudes de vector. Este proceso de verificación, permitió estimar un rango para el porcentaje de error entre 0 y 1×10^{-6}, que representa un grado de precisión aceptable para los resultados generados por el módulo de multiplicación, según lo discutido con el Ing. Carlos Salazar.

Nuevamente, los mayores porcentajes de error se atribuyen a la ausencia del sticky bit para propósitos de redondeo.

4.3.2 Casos excepcionales

A continuación, se presentan dos casos puntuales para el módulo de multiplicación en los que se generaron las condiciones excepcionales de desborde y sub-desborde. En la figura 4.9 se realiza el producto de los números $1{,}23 \times 10^{24}$ y $5{,}2 \times 10^{19}$, cuyo resultado es muy grande para ser representado por el formato, provocando la condición de desborde. Por otro lado, en la figura 4.36 se multiplicaron los números $2{,}548 \times 10^{-25}$ y $1{,}258 \times 10^{-26}$, cuyo resultado genera la condición de sub-desborde. Nótese que para estas condiciones excepcionales las banderas de excepción se establecen de manera correcta, y que los resultados corresponden a los patrones de bits referentes a estos casos.

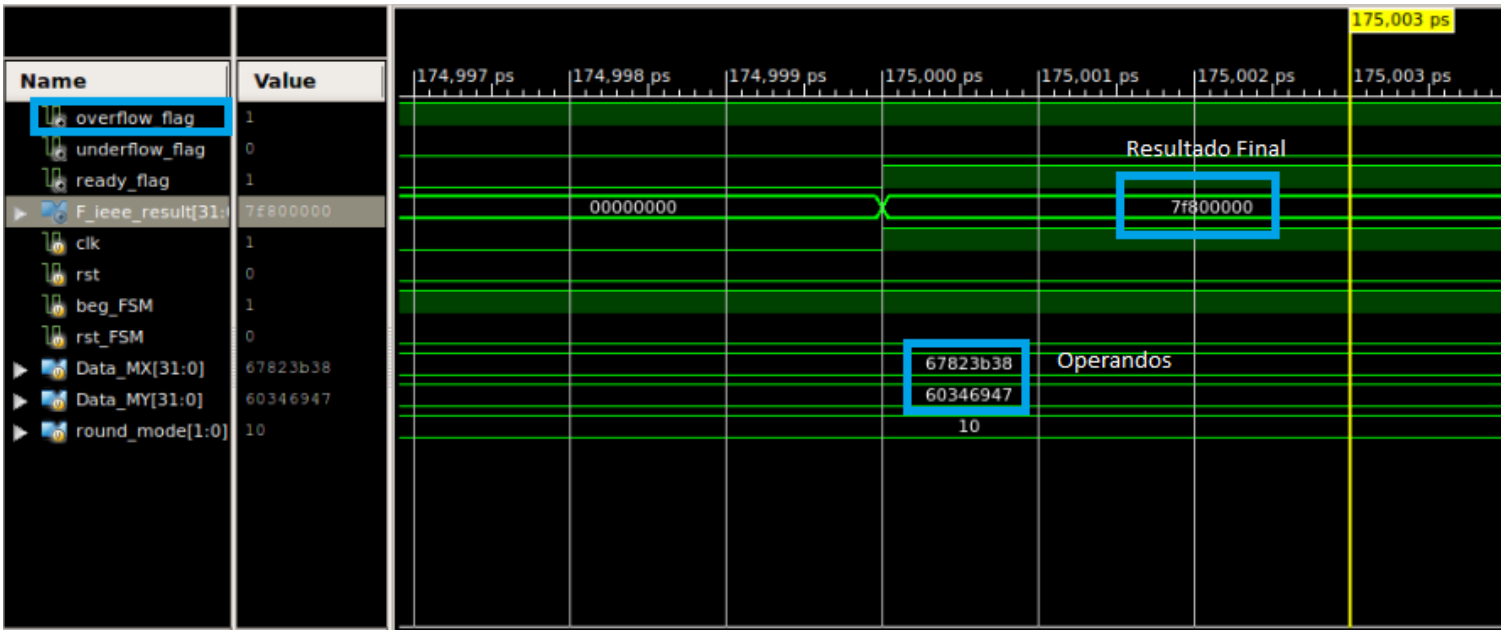

Figura 4.9. Caso puntual de la multiplicación que genera un caso excepcional de desborde.

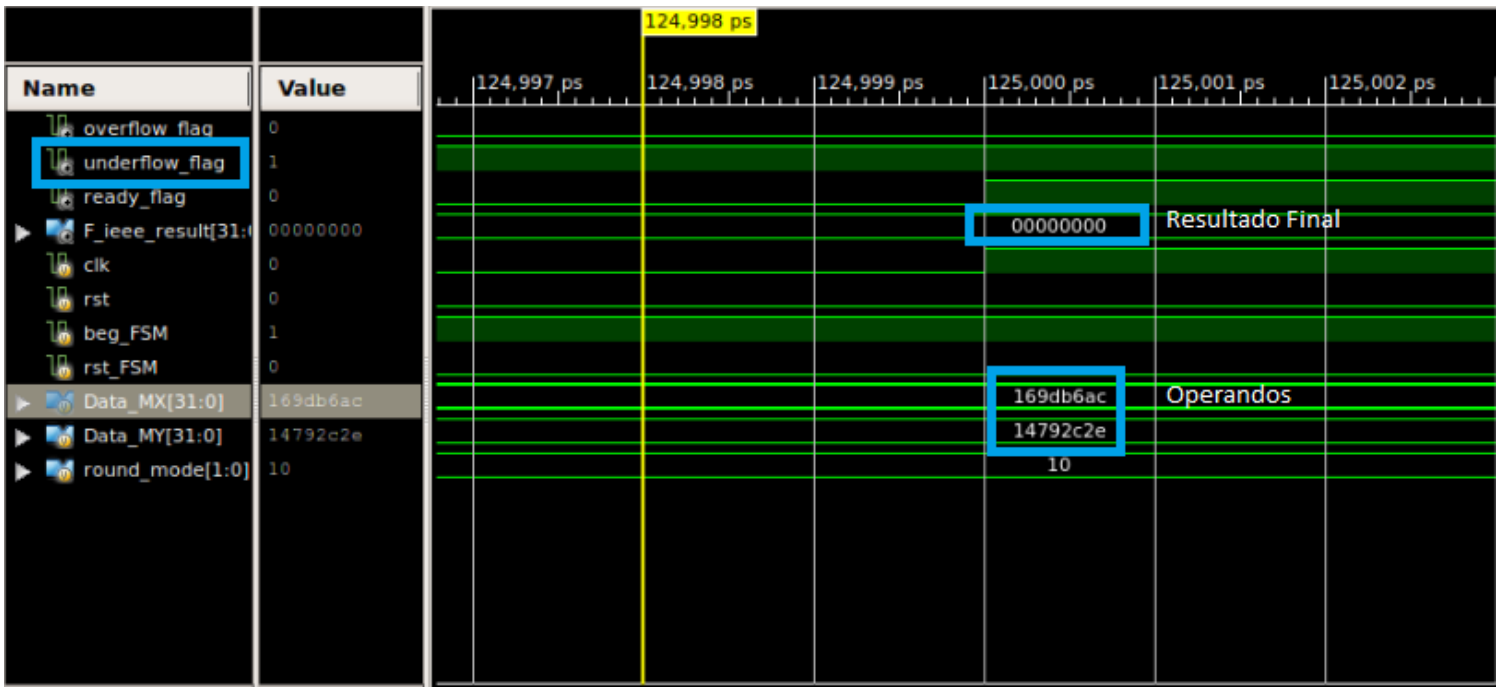

Figura 4.10. Caso puntual de la multiplicación que genera un caso excepcional de sub-desborde.

4.3.3 Recursos utilizados

Del informe Post Place & Route generado en el ISE, se obtuvo información referente al consumo de recursos por parte de las arquitecturas de 32 y 64 bits del multiplicador. Estos resultados se resumen en las tablas 4.9 y 4.10.

Tabla 4.9. Resumen del reporte Post Place & Route del uso de dispositivos generado por la herramienta ISE, para la arquitectura de 32 bits del módulo de multiplicación.

Slice Logic Utilization	Available Devices	Used Devices	Percent of utilization
Number of slice registers used as Flip Flops	126800	433	1%
Number of slice LUTs	63400	236	1%
Number of occupied slices	15850	119	1%
Number of LUT flip flop pairs used with an unused flip flop	407	35	8%
Number of LUT flip flop pairs used with an unused LUT	401	171	42%
Number of fully used LUT FF pairs	407	201	49%
DSP48E1's	240	2	1%

Tabla 4.10. Resumen del reporte Post Place & Route del uso de dispositivos generado por la herramienta ISE, para la arquitectura de 64 bits del módulo de multiplicación.

Slice Logic Utilization	Available Devices	Used Devices	Percent of utilization
Number of slice registers used as Flip Flops	126800	815	1%
Number of slice LUTs	63400	554	1%
Number of occupied slices	15850	277	1%
Number of LUT flip flop pairs used with an unused flip flop	961	215	22%
Number of LUT flip flop pairs used with an unused LUT	961	407	42%

Number of fully used LUT FF pairs	961	339	35%
DSP48E1's	240	15	6%

De manera similar al módulo de suma-resta, existe un aumento considerable en el uso de dispositivos cuando se trabaja con la arquitectura de 64 bits del multiplicador.

4.3.4 Consumo de Potencia

Con la ayuda de la herramientaXPower Analyzer, se obtuvo un informe referente al consumo de potencia estática y dinámica para las dos arquitecturas del módulo multiplicador. Los resultados obtenidos se presentan en las tablas 4.11 y 4.12.

Tabla 4.11. Resumen del reporte generado por la herramienta XPower Analizer que indica el consumo estático, dinámico y total de potencia para ambas arquitecturas del módulo multiplicador.

Arquitectura de 32 bits			Arquitectura de 64 bits		
Consumo de potencia estática (mW)	Consumo de potencia dinámica (mW)	Consumo total de potencia (mW)	Consumo de potencia estática (mW)	Consumo de potencia dinámica (mW)	Consumo total de potencia (mW)
88,20	6,99	95,19	88,22	12,32	100,54

Tabla 4.12. Resumen del reporte generado por la herramienta XPower Analizer que indica el consumo de potencia On-Chip de diversos elementos en el módulo multiplicador.

	Arquitectura de 32 bits	Arquitectura de 64 bits
On-Chip	Power(mW)	Power (mW)
Clocks	1,84	4,35
Logic	0,47	1,08
Signals	1,01	2,47
DSPs	0,31	1,57
IOs	3,35	2,85
Static Power	88,20	88,22
Total	95,19	100,54

Según lo discutido en la sección 4.2.7, la potencia estática se mantiene constante indiferentemente del diseño que se implemente en la FPGA, por lo que su análisis no es relevante. Por otro lado, el consumo de potencia dinámica es mayor en la arquitectura de 64 bits (ver tabla 4.11). Esto nuevamente se puede justificar en base a la tabla 4.12, en la cual se observa que el consumo de potencia en las señales, relojes, DSPs y bloques de lógica aumenta considerablemente respecto a la arquitectura de 32 bits.

4.2.8 Reporte de tiempos

Con ayuda del *Timing Analyzer*, se generó un resumen de tiempos para la implementación Post Place & Route de las arquitecturas del multiplicador. Estos resultados se presentan en las tabla 4.13 y 4.14.

Tabla 4.13. Resumen del reporte de tiempos para la arquitectura de 32 bits del módulo multiplicador. Este reporte se generó utilizando la herramienta *Timing Analyzer*.

Design statistics	
Minimum Period	6,227 ns
Maximum frequency	160,591 MHz
Maximum combinational path delay	4,238 ns
Minimum input required time before clock	0,853 ns
Maximum output delay after clock	9,757 ns

Tabla 4.14. Resumen del reporte de tiempos para la arquitectura de 64 bits del módulo multiplicador. Este reporte se generó utilizando la herramienta *Timing Analyzer*.

Design statistics	
Minimum Period	11,324 ns
Maximum frequency	88,308 MHz
Maximum combinational path delay	4,304 ns
Minimum input required time before clock	3,231 ns
Maximum output delay after clock	10,657 ns

Para la arquitectura de 32 bits el período mínimo corresponde a 6,227 ns, lo que le permite a bloques de lógica combinacional segmentados correr a frecuencias de hasta 160,591 MHz. Por otro lado, el tiempo de propagación a través de rutas combinacionales no segmentadas corresponde a 4,238 ns, mientras que el tiempo mínimo en el que las señales de entrada globales deben ser válidas antes de que ocurra el flanco de reloj corresponde a 0,853 ns, permitiéndose una ventana de tiempo de 9,147ns para que los datos se estabilicen. Finalmente, se obtuvo un tiempo máximo de establecimiento de las salidas después del flanco de reloj positivo de 9,757 ns.

Una situación relevante ocurrió en la arquitectura de 64 bits, en la cual el período mínimo de operación es de 11,324 ns. Note que este valor es mayor que el período de operación de la FPGA y fuerza al módulo a trabajar a una frecuencia máxima de 88,308 MHz. Esta situación fue provocada por el módulo encargado de multiplicar las mantisas (el cual es

un bloque segmentado), en dónde se identificó una ruta crítica en la que el tiempo de propagación de los datos corresponde a 11,232 ns. Por esta razón, se definió una frecuencia de trabajo de 50 MHz para el módulo. Además, el tiempo mínimo para el que las señales de entrada deben ser válidas antes del flanco de reloj positivo corresponde a 3,231 ns (ventana de tiempo de 6,769 ns para que las entradas sean estables). Finalmente, el tiempo de establecimiento de las salidas corresponde a 10,657 ns por la misma situación de la frecuencia máxima de operación.

Capítulo 5

Conclusiones y recomendaciones

5.1 Conclusiones

Se demostró que es posible sintetizar unidades aritméticas de precisión aceptable utilizando versiones reducidas del estándar IEEE 754.

Llevando a cabo una verificación funcional de los módulos aritméticos, se comprobó que el porcentaje de error de los resultados experimentales respecto a los resultados teóricos fue menor a un 1%.

Basándose en los reportes de tiempo obtenidos de la herramienta *Timing Analyzer*, se concluye que las arquitecturas de 32 y 64 bits del módulo sumador-restador pueden operar a frecuencias mayores de 100 MHz, mientras que en el multiplicador solo la arquitectura de 32 bits puede correr a frecuencias mayores, ya que en la de 64 bits existe una ruta crítica combinacional que limita la frecuencia de operación.

Se determinó que se pueden mejorar los tiempos de ejecución en el módulo de suma-resta, modificando o integrando nuevo hardware en los módulos encargados de ejecutar procesos de normalización.

Se concluyó que en general el módulo de suma-resta consume más recursos que el módulo de multiplicación, dado que a nivel de ejecución e implementación, el algoritmo de adición-sustracción es más complejo que el de multiplicación.

5.2 Recomendaciones

Existen muchas posibilidades de mejora dentro de los módulos aritméticos. En el módulo de suma es posible mejorar la velocidad de ejecución, mediante la integración de una unidad de anticipación de ceros [11] (conocido en inglés como Leading-zero anticipator ó por sus siglas LZA) y un desplazador de barril [13] (conocido en inglés como barrel

shifter). El uso de estas unidades permitirá agilizar el proceso de normalización del significando llevado a cabo en el módulo *Sgf_R_NM*, ya que el LZA permite anticipar la cantidad de desplazamientos necesarios para obtener un significando normalizado, mientras que el desplazador de barril es capaz de desplazar "n" cantidad de bits hacia la derecha o hacia la izquierda en un solo ciclo de reloj. Otro módulo sobre el que se podría utilizar el desplazador de barril es en el que se normaliza el significando del número menor, ya que este proceso requiere de desplazamiento de ceros hacia la derecha para llevarse a cabo.

Para el módulo de multiplicación se recomienda investigar estrategias de interconexión de los multiplicadores internos de la FPGA, con el fin de poder alcanzar una frecuencia de operación de 100 MHz en la arquitectura de 64 bits.

Finalmente, se recomienda integrar de manera gradual más características del estándar IEEE 754, como por ejemplo los modos de redondeo que quedaron pendientes, el uso del sticky bit, el manejo de otros casos excepcionales, y si las proyecciones a futuro incluyen proyectos en el área de procesadores gráficos se debería considerar desarrollar el hardware necesario para el manejo del set de números subnormales.

Bibliografía

[1] Alfaro Hidalgo, L. A. (2013). Implementacion en hardware del Sistema de Reconocimiento de Patrones Acu sticos (SiRPA). Tesis de pregrado. Instituto Tecnológico de Costa Rica, Cartago, Costa Rica. Recuperado el 1 de abril del 2015, desde: http://repositoriotec.tec.ac.cr/handle/2238/3092

[2] Butler, M. (2003). Hidden Markov Model Clustering of Acoustic Data. Recuperado el 1 de mayo del 2015, desde: http://www.inf.ed.ac.uk/publications/thesis/online/IM030057.pdf

[3] Chacón Rodríguez, A., & Alvarado Moya, J. P. (2014). Sistema electrónico integrado en chip (SoC) para el reconocimiento de patrones de disparos y motosierras en una red inala mbrica de sensores para la protección ambiental (Documento I). Recuperado el 25 de marzo del 2015, desde: http://www.ie.itcr.ac.cr/achacon/VIE/sirpa-doc1.pdf

[4] Chapter 2 IEEE arithmetic. (Sin Fecha). Recuperado el 13 de abril del 2015, desde: http://docs.oracle.com/cd/E19957-01/806-3568/ncg_math.html

[5] Chapter 7 – floating point arithmetic. (Sin Fecha). Recuperado el 13 de abril del 2015, desde: http://pages.cs.wisc.edu/~smoler/x86text/lect.notes/arith.flpt.html

[6] D. Patterson J. Hennessy. Computer Organization Design. The hardware/software interface, volume 1. Morgan Kaufmann, 3 edition, 2005.

[7] Floating Point Representation. (Sin Fecha). Recuperado el 1 de agosto del 2015, desde: http://pages.cs.wisc.edu/~smoler/x86text/lect.notes/arith.flpt.html

[8] Ghahramani, Z. (2001). An Introduction to Hidden Markov Models and Bayesian Networks. Recuperado el dí a 2 de mayo del 2015, desde: http://mlg.eng.cam.ac.uk/zoubin/papers/ijprai.pdf

[9] Guralnik, E., Aharoni, M., Birnbaum, A. J. & Koyfman, A. (Sin Fecha). Simulation based verification of floating point division. Recuperado el 10 de abril del 2015, desde: https://www.research.ibm.com/haifa/projects/verification/fpgen/papers/Simulation_Based_Verification_of_Floating_Point_Division.pdf

[10] Goldberg, D. (1991). What every computer scientist should know about floating-point arithmetic. Recuperado el 10 de abril del 2015, desde: http://docs.oracle.com/cd/E19957-01/806-3568/ncg_goldberg.html

[11] Hokenek. E, & Montoye, R. K. (sin fecha). Leading-zero anticipator (LZA) in the IBM RISC System/6000 floating-point execution unit. Recuperado el 2 de septiembre del 2015, desde: http://www.csee.umbc.edu/~phatak/645/supl/lza/lza-ibmrisc-1990.pdf

[12] Kahan, W. (1997). IEEE standard 754 for binary floating-point arithmetic. Recuperado el 13 de abril del 2015, desde: http://www.eecs.berkeley.edu/~wkahan/ieee754status/IEEE754.PDF

[13] Rudolf Pillmeier, M. (2001). Barrel shifter design, optimization, and analysis. Recuperado el 25 de julio del 2015, desde: http://preserve.lehigh.edu/cgi/viewcontent.cgi?article=1714&context=etd

[14] Monniaux, D. (2008). The pitfalls of verifying floating-point computations. Recuperado el 15 de abril del 2015, desde: https://hal.archives-ouvertes.fr/file/index/docid/281429/filename/floating-point-article.pdf

[15] OpenCores. (1999). Recuperado el 10 de abril del 2015, desde: http://opencores.org/

[16] Operations on floating point numbers. (Sin Fecha). Recuperado el 10 de abril del 2015, desde: https://cs.nyu.edu/courses/fall03/G22.2420-001/lec4.pdf

[17] Salazar García, C. Implementación de un microprocesador de aplicación específica para la ejecución del algoritmo de modelos ocultos de Markov para el reconocimiento de patrones ac_usticos. Tesis de Maestría. Escuela de Ingeniería Electrónica, ITCR, Cartago, Costa Rica, Diciembre 2015.

[18] The Oxford Math Center. (Sin Fecha). IEEE 754 Format. Recuperado el 14 de Julio del 2015, desde: http://www.oxfordmathcenter.com/drupal7/node/43