IMPLEMENTACIÓN Y EVALUACIÓN DE UNA UNIDAD DE FUNCIONES ESPECIALES TOLERANTE A FALLAS BASADA EN APROXIMACIÓN POLINOMIAL POR PARTES

EDWAR JAVIER PATIÑO NÚÑEZ

UNIVERSIDAD PEDAGÓGICA Y TECNOLÓGICA DE COLOMBIA
FACULTAD SECCIONAL TUNJA
ESCUELA DE INGENIERÍA ELECTRÓNICA
TUNJA
2022

IMPLEMENTACIÓN Y EVALUACIÓN DE UNA UNIDAD DE FUNCIONES ESPECIALES TOLERANTE A FALLAS BASADA EN APROXIMACIÓN POLINOMIAL POR PARTES

EDWAR JAVIER PATIÑO NÚÑEZ

MONOGRAFÍA

DIRECTOR: M.Sc. JUAN DAVID GUERRERO BALAGUERA
CODIRECTOR: Ph.D. JOSIE ESTEBAN RODRÍGUEZ CONDIA

UNIVERSIDAD PEDAGÓGICA Y TECNOLÓGICA DE COLOMBIA
FACULTAD SECCIONAL TUNJA
ESCUELA DE INGENIERÍA ELECTRÓNICA
TUNJA
2022

Nota de Aceptación

_____

_____

_____

_____

Presidente del Jurado

_____

Jurado

_____

Jurado

Tunja, 23 de abril de 2022

A mi familia, a mis padres y a mi hermana.

AGRADECIMIENTOS

En primer lugar, quiero agradecer al director de este trabajo M.Sc. Juan David Guerrero Balaguera y al codirector Ph.D. Josie Esteban Rodríguez Condia, por la confianza ofrecida para desarrollar este trabajo, por el apoyo brindado y por la guía para culminar con éxito esta etapa de aprendizaje.

También agradezco a mis compañeros por todas esas horas de trabajo compartidas a lo largo de nuestra formación y a mis docentes por el conocimiento brindado.

Gracias a mis padres y mi hermana que además de la ayuda económica, me brindaron soporte, paciencia y comprensión para recorrer esta etapa hasta el final y que hoy me permite decir, a todos, muchas gracias.

CONTENIDO

LISTA DE TABLAS

LISTA DE FIGURAS

LISTA DE ANEXOS

Nota: *Los anexos se encuentran en la carpeta "Anexos".*

RESUMEN

En este trabajo se presenta la implementación lógica de un método basado en tablas para la aproximación de funciones de alta velocidad en formato IEEE-754. La aproximación se centra en las funciones $sin(x)$, $cos(x)$, $\log_2(x)$, $1/x$, $\sqrt{x}$, $1/\sqrt{x}$ y 2^x. El algoritmo usado combina la búsqueda en tablas de los coeficientes de un polinomio de segundo orden, calculados con una aproximación cuadrática minimax y una evaluación optima del polinomio usando una unidad de elevación al cuadrado especializada, multiplicadores rápidos y un sumador de varios operandos. El trabajo presenta un enfoque en los sistemas tolerantes a fallos con redundancia de hardware, en donde se hace uso de un sistema secundario de acceso libre para modificarlo y comparar resultados con el sistema implementado. Los sistemas se modifican con diferentes configuraciones de redundancia en hardware y se hace un análisis de confiabilidad, se estiman los tiempos de ejecución y los requisitos de la cantidad de elementos lógicos necesarios por cada una de las diferentes configuraciones de redundancia.

PALABRAS CLAVE: Funciones elementales, aproximación polinomial, Unidad de Funciones Especiales (SFU), hardware de propósito especial, minimax.

1. INTRODUCCIÓN

En la actualidad, los avances y descubrimientos en diferentes campos de la ciencia y la tecnología han cambiado la forma de vivir de la sociedad moderna permitiendo mejorar la calidad de vida de las personas. Esto ha exigido el desarrollo de soluciones tecnológicas cada vez más eficientes, adoptando temas de vanguardia que usualmente incluyen inteligencia artificial (IA), conectividad 5G e internet de las cosas (IoT) [1]. La complejidad resultante de la integración de estos desarrollos y que comúnmente hacen uso de sistemas embebidos, implica un esfuerzo de la industria de semiconductores en el desarrollo de nuevos dispositivos con alta capacidad de procesamiento, baja latencia y alta confiabilidad para responder a las necesidades que exige el mundo moderno. Estas exigencias han incrementado la cantidad de información que se debe procesar, haciendo que los circuitos y sistemas digitales empleados deban implementar funciones matemáticas cada vez más complejas y eficientes, con lo cual, es común que los sistemas modernos incluyan uno o varios módulos de hardware dedicados a estos cálculos conocidos como Unidades de Funciones Especiales (*SFUs – Special Function Units*).

Las SFUs están mayoritariamente presentes en Unidades de Procesamiento Gráfico (*GPU – Graphics Processing Unit*), pero también pueden ser incluidas como coprocesadores en Sistemas en Chip (*SoC - Systems on a Chip*) o en aplicaciones personalizadas con FPGAs. Sin embargo, la optimización en rendimiento y consumo de potencia de las SFU requieren del uso de tecnologías de integración nuevas lo cual las hace más propensas a sufrir errores o fallas durante su funcionamiento. Debido a esto, es relevante evaluar técnicas de tolerancia a fallas sobre estas unidades para su mitigación, sin embargo, en la actualidad hay poca disponibilidad de modelos de simulación de SFUs y escasos detalles técnicos de su implementación en sistemas comerciales. En este trabajo se implementa un modelo de SFU basada en Aproximación Polinomial por Partes (*PPA – Piecewise Polynomial Approximation*) y se hace una evaluación de su rendimiento y confiabilidad frente a otros modelos de SFU disponibles implementando una técnica de tolerancia a fallas conocida como redundancia modular triple (*TMR – Triple Modular Redundancy*).

En los siguientes dos capítulos se presentan los objetivos, el planteamiento y la justificación de este trabajo. Posteriormente en el capítulo 4 se expone una contextualización de las SFUs junto con diferentes nociones acerca de su diseño, confiabilidad y sistemas tolerantes a fallos. En el capítulo 5 se presenta el proceso de implementación de la SFU y contiguo en el capítulo 6 se valida su funcionamiento. En el capítulo 7 se explica la realización de la técnica de tolerancia a fallas a la SFU implementada y al modelo disponible de acceso libre y, por último, en el capítulo 8 se presentan las conclusiones del trabajo realizado.

2. OBJETIVOS

2.1. OBJETIVO GENERAL

Implementar una SFU basada en Aproximación Polinomial por Partes y evaluar su rendimiento y confiabilidad frente a otra arquitectura como guía en el diseño y uso en aplicaciones tolerantes a fallas.

2.2. OBJETIVOS ESPECÍFICOS

Modelar el comportamiento de un modelo arquitectural de SFU basada en Aproximación Polinomial por Partes, empleando una descripción en software de alto nivel (*Golden Model*), e implementar dicha SFU mediante un lenguaje de descripción de hardware a partir de su *Golden Model*.

Verificar la descripción de la SFU empleando técnicas de verificación funcional.

Aplicar y evaluar una técnica de tolerancia a fallas a la SFU implementada y a los modelos de SFUs disponibles para comparación.

Estimar la confiabilidad de los modelos de SFUs con la técnica de tolerancia a fallas seleccionada.

3. PLANTEAMIENTO DEL PROBLEMA

3.1. DEFINICIÓN DEL PROBLEMA

Hoy en día los sistemas digitales complejos como GPUs, son usados en una gran variedad de tareas críticas como: líneas producción [2], sistemas de maquinaria autónomos, satélites [3], aplicaciones automotrices [4], etc. Donde la confiabilidad del dispositivo en un factor clave en el desempeño de la aplicación. Estos dispositivos están compuestos de varias unidades de ejecución, entre ellas están las SFUs, las cuales son importantes en varias aplicaciones realizando cálculos optimizados para el alto rendimiento. Sin embargo, los circuitos integrados (*IC - Integrated Circuits*) pueden presentar fallas en su fabricación o durante su ciclo de vida, por lo que puede reducirse su rendimiento e incrementar la posibilidad de un mal funcionamiento. Las fallas que se puedan presentar se pueden dividir principalmente en tres grupos de acuerdo a su duración y ocurrencia así: fallas permanentes, fallas intermitentes y fallas transitorias. Las primeras ocurren, pero no desaparecen, las segundas ocurren y desaparecen repentinamente y las últimas se presentan, pero son de poca o finita duración. Existen diferentes factores relacionados con la ocurrencia de fallas, algunos factores intrínsecos se relacionan con la fabricación, fatiga o envejecimiento de los IC, algunos factores externos se encuentran relacionados con ambientes de operación exigentes o errores del operario y otros factores conjuntos se relacionan con el aumento del uso de tecnologías nanométricas en su fabricación, que tienen un alto rendimiento y bajo consumo energético, pero aumentan su sensibilidad a la radiación [5].

En la actualidad los circuitos aritméticos y sistemas digitales son diseñados y fabricados dependiendo de la aplicación y de las condiciones de su entorno de operación. De esta forma, se pueden definir dos enfoques básicos de diseño y fabricación de dispositivos: i) sistemas no tolerantes a fallos y ii) sistemas tolerantes a fallos. En el primer caso, y como su nombre lo indica, son dispositivos que pueden llegar a fallar y generar errores durante la ejecución de una tarea. Este tipo de aplicaciones se caracterizan por la baja o media exigencia de confiabilidad en los dispositivos ya que es posible reemplazar un módulo dañado o el sistema completo. Además, los errores que se presenten en la aplicación no constituyen daños críticos de recursos económicos, medio ambientales o vidas humanas. Por otra parte, los sistemas digitales tolerantes a fallos son especialmente diseñados con algún tipo de redundancia y están orientados a satisfacer las necesidades de confiabilidad en ambientes de operación complejos (p. ej. ambientes con alta radiación o altas temperaturas) y en aplicaciones de seguridad crítica, donde los posibles efectos de una falla pueden provocar graves daños económicos, daños ambientales y heridas o pérdida de vidas humanas. Estos sistemas, comúnmente incluyen métodos de redundancia de hardware, software, tiempo, información o una combinación de estas [6], [7].

3.2. JUSTIFICACIÓN

Los efectos de fallas en GPUs, SFUs y en general en los IC, son indeseables en ambientes de aplicaciones de seguridad crítica, por lo tanto, las soluciones que logran mitigar y evitar sus efectos son comúnmente requeridos. Sin embargo, en la actualidad, son pocos los estudios que se centren en el diseño, exploración y evaluación de las características de confiabilidad en las SFUs. Además, muchos factores limitan la exploración y evaluación de la confiabilidad en SFUs, incluyendo la poca disponibilidad de modelos de simulación de este tipo de circuitos y los escasos detalles técnicos de implementación en sistemas comerciales.

Debido a la poca disponibilidad de descripciones a nivel de microarquitectura (RTL) de las SFU en la literatura, no se han explorado, evaluado o implementado técnicas de tolerancia a fallas y no se han analizado los efectos en costo de hardware, potencia y rendimiento que tendría la implementación de alguna técnica de tolerancia a fallas basada en redundancia en este circuito aritmético de propósito especial. Por tanto, no hay un precedente de la evaluación del comportamiento y el diseño de estas unidades. En este trabajo, se propone implementar un modelo de SFU basada en PPA (presente en algunos modelos de GPU comerciales) y hacer una evaluación de su rendimiento y confiabilidad junto a otros modelos de SFU disponibles de acceso libre implementando una técnica de tolerancia a fallas redundante.

4. MARCO TEÓRICO

En este capítulo se presentan los principales conceptos alrededor del diseño de las SFUs, así mismo se presentan conceptos sobre confiabilidad y sistemas tolerantes a fallos. En primer lugar, se exponen algunos conceptos relacionados con los SoCs y las GPUs, los cuales son sistemas electrónicos que incorporan diferentes unidades de procesamiento entre las cuales se encuentran la SFUs. Posteriormente se abordarán los métodos de diseño más comunes para evaluar funciones elementales en hardware, de los cuales, uno será seleccionado para la implementación de la SFU de este trabajo. Por último, se proporciona una vista general de los tipos de fallas que se presentan en los sistemas y los métodos de tolerancia a fallas más comunes para aumentar su confiabilidad.

4.1. SISTEMAS-EN-CHIP

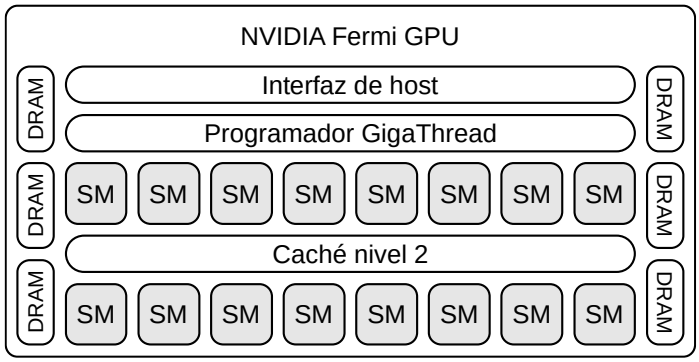
Un sistema en chip es un circuito complejo que integra los principales elementos funcionales de un sistema electrónico en un único circuito integrado: procesador, memoria, controladores de dispositivos de hardware, conversores ADC/DAC, sistemas ópticos, sistemas microelectromecánicos (MEMS), etc. [8], [9]. Con lo cual, pueden interactuar con dispositivos periféricos y/o con el mundo real [10]. Los avances en la tecnología de los sistemas y procesos de fabricación han permitido aumentar su complejidad y con esto, su panorama de aplicaciones pueden ir desde la inteligencia artificial hasta la computación cuántica [11].

Los SoCs son parte integral de los productos de computación, comunicación, entretenimiento, médicos, logísticos y de transporte. Donde son usados en la generación de gráficos en 2D y 3D, transcodificación de video, reconocimiento de voz, etc. Estos problemas tienen un alto grado de paralelismo intrínseco por lo que los SoCs incorporan GPUs [12], ya que son la mejor herramienta para solucionar rápidamente este tipo de problemas gracias a su estructura altamente paralela.

4.2. UNIDADES DE PROCESAMIENTO GRÁFICO

Una GPU es un microprocesador con una arquitectura centrada en una gran cantidad de procesadores diseñados para la computación en paralelo. Consiste en varios multiprocesadores, cada uno con un conjunto de núcleos los cuales operan en forma de Única-Instrucción Datos-Múltiples (*SIMD - Single-Instruction Multiple-Data*), es decir, procesan sincrónicamente una única instrucción sobre múltiples datos [13], [14], [15], [16]. Aunque en un inicio las GPUs fueron empleadas principalmente para renderizar imágenes, en los últimos años han sido utilizadas para resolver problemas relacionados con el procesamiento paralelo [17].

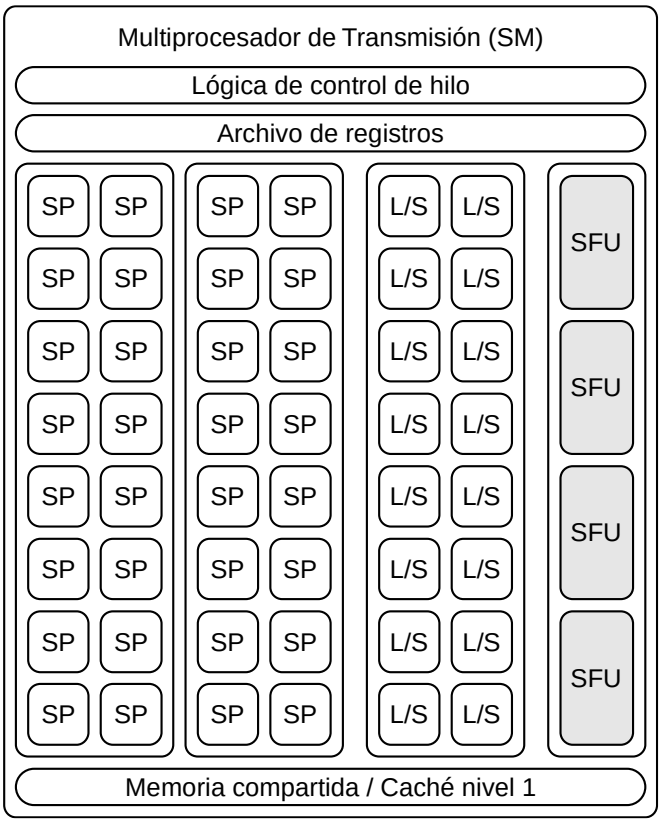
Figura 1. Diagrama de bloques simplificado de la GPU Fermi de NVIDIA



Fuente: Adaptado de [18] **y** [19]

Más en detalle, una GPU está compuesta de diferentes unidades de hardware funcionales. En base a las GPUs de NVIDIA con arquitectura CUDA (cuyo nombre clave es Fermi, lanzada en el 2010), se pueden mencionar algunos módulos como: multiprocesadores de transmisión (*SM – Streaming Multiprocessor*), programador GigaThread, interfaz de host, memoria cache de nivel 2 y múltiples interfaces DRAM, como se puede observar en la Figura 1.

Figura 2. Diagrama de bloques de un multiprocesador de transmisión (SM)



Fuente: Adaptado de [18] **y** [19]

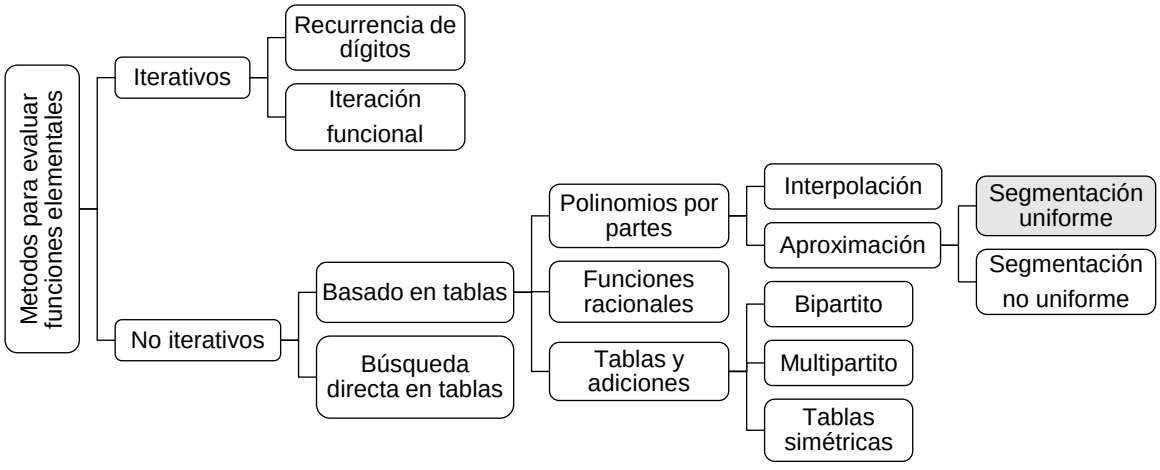
Cada uno de los SMs, como se muestra en la Figura 2, está conformado por procesadores de transmisión (*SP - Streaming Processor*), unidades de carga/almacenamiento de memoria (*L/S - load-store*), memoria caché de nivel 1, lógica de control y Unidades de Funciones Especiales (*SFU - Special Function Units*). Los SP internamente tienen una unidad aritmético lógica (*ALU - Arithmetic Logic Unit*) y una unidad de punto flotante (*FPU – Floating Point Unit*) para ejecutar las operaciones lógicas y aritméticas requeridas, pero es la SFU la encargada de computar las operaciones de funciones elementales como: $sin(x)$, $cos(x)$, $log_2(x)$, $1/x$, $\sqrt{x}$, $1/\sqrt{x}$, 2^x, entre otras; necesarias en aplicaciones que requieren de funciones matemáticas complejas [20], [21].

Las SFUs son aceleradores de hardware que se empezaron a integrar en las GPUs de NVIDIA a partir del modelo Fermi [22], su implementación ayuda a mejorar el rendimiento de las GPUs, por lo que la mayoría de los fabricantes las suelen incluir para procesar parte de las operaciones más complejas [23]. Su implementación se puede llevar a cabo haciendo uso de diferentes metodologías, las cuales se explorarán en más detalle a continuación.

4.3. EVALUACIÓN DE FUNCIONES ELEMENTALES EN HARDWARE

Existen diferentes métodos para evaluar funciones elementales que, al llevarlos a una implementación en hardware, se pueden considerar como diferentes formas de diseñar una SFU; estos métodos se pueden dividir (como se indica en la Figura 3) en dos clases principales: iterativos y no iterativos. Entre las diferentes características deseables que pueden tener estas unidades se tiene: velocidad, precisión, bajos recursos de hardware y bajo consumo de energía, entre otras [24].

Figura 3. Taxonomía de los métodos de generación de funciones elementales



Fuente: Elaboración propia

4.3.1. MÉTODOS ITERATIVOS

Este método incluye principalmente dos tipos de algoritmos. El primero son algoritmos iterativos de recurrencia de dígitos (*digit-recurrence*) como los propuestos en los trabajos [25], [26], sin embargo, solo se han usado para evaluar dos funciones elementales que son: $1/x$ y $1/\sqrt{x}$. Por otra parte, se tienen los algoritmos de iteración funcional como CORDIC (*COordinate Rotation DIgital Computer*) [23], [27], Newton-Raphson [28] y Goldschmidt [29], los cuales se han usado para computar una mayor cantidad de funciones elementales como: $\sin(x)$, $\cos(x)$, $\log_2(x)$, $\sqrt{x}$ y 2^x, entre otras. Los diseños iterativos suelen ser compactos en hardware, pero tienen latencia en el cálculo de las funciones y están orientados a tener una alta exactitud y precisión.

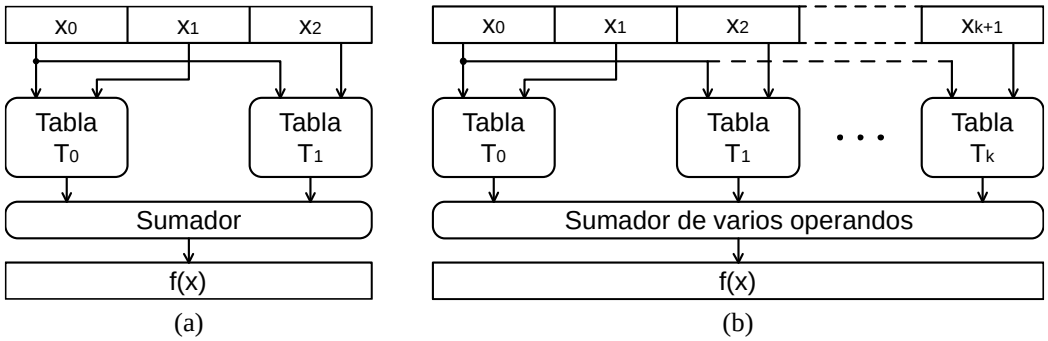
4.3.2. MÉTODOS NO ITERATIVOS

Los métodos no iterativos consisten en diferentes técnicas para la evaluación de polinomios de la forma (1), los cuales están basados en tablas como: búsquedas directas (no muy usados), tablas-y-adiciones [30], evaluación de polinomios por partes [31] y aproximaciones racionales [32]. Estos métodos están orientados a la optimización de hardware y el consumo de energía, los cuales son más rápidos que los iterativos, pero necesitan una gran cantidad de memoria. Normalmente la función a computar es evaluada en un intervalo de entrada específico y se hace uso de la reducción del rango para los datos que están fuera del intervalo.

$$\begin{aligned} f(x) &\cong C_0 + C_1 x + C_2 x^2 + \cdots + C_{k-1} x^{k-1} + C_k x^k \\ &= T_0 + T_1 + T_2 + \cdots + T_{k-1} + T_k \end{aligned} \tag{1}$$

El método de búsqueda directa en tablas equivale a evaluar un polinomio de orden cero ya que se almacenan los valores de $f(x)$ para cada uno de los valores de entrada x. El principal inconveniente de este método es que el aumento en el tamaño del dato de entrada requiere una gran cantidad de memoria para almacenar todos los valores $f(x)$, para un dato de 8 bits se requieren 2^8 filas de n columnas, si se quiere aumentar el tamaño del dato en 1 bit, la cantidad de filas requeridas será del doble y así, el aumento del tamaño de memoria requerido se vuelve exponencial a medida que el tamaño del dato aumenta. En los métodos basados en tablas-y-adiciones, cada valor de T_i, $i = \{0,1,..,k\}$ expresado en (1), es almacenado y direccionado por una combinación de diferentes partes del dato de entrada como se observa en la Figura 4 y las salidas de las tablas son sumadas [33]. Dependiendo de la cantidad de tablas usadas este método se puede dividir en bipartito y multipartito.

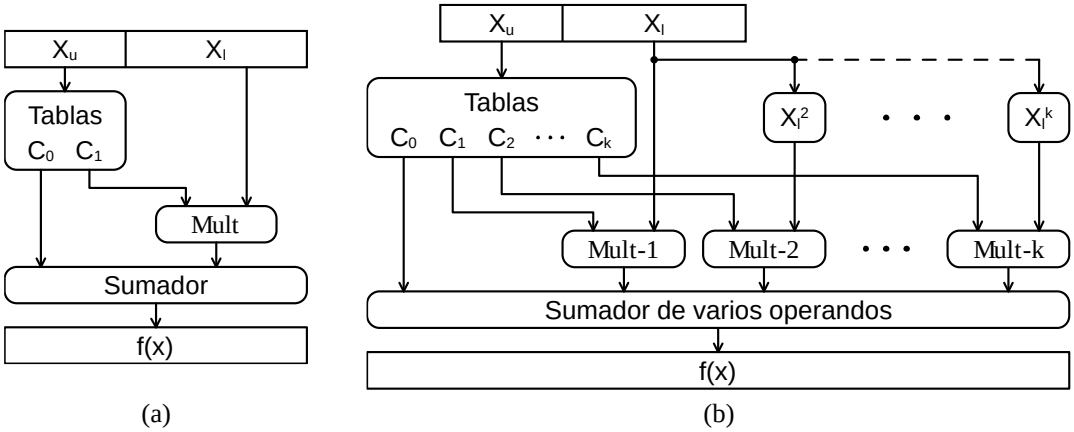
Figura 4. Diagrama generalizado tablas-y-adiciones: (a) bipartito, (b) multipartito



Fuente: Adaptado de [33]

Usando operaciones básicas como sumas, restas y multiplicaciones, se pueden evaluar polinomios por partes y al agregar divisiones, se pueden evaluar funciones racionales por partes [24]. El método de evaluación de polinomios por partes divide el intervalo de entrada de la función a evaluar en varios segmentos y cada uno es representado por un polinomio de la forma (1), normalmente se usan polinomios de grado uno o dos. Este método se puede dividir en dos subcategorías: aproximación [31] e interpolación [34], y los coeficientes C_i, $i = \{0,1,..,k\}$ se determinan en función de la subcategoría implementada. El método de interpolación almacena los valores de los puntos finales de cada segmento y estos son usados para computar los coeficientes C_i (lo cual ocupa tiempo de ejecución), posteriormente se evalúa el polinomio [35]. En el método de aproximación, los coeficientes C_i son almacenados en tablas y son direccionados por algunos de los bits más significativos (*MSBs*) del dato de entrada, una vez los coeficientes son direccionados, se evalúa el polinomio. Las aproximaciones más comunes incluyen las series de Taylor, la aproximación de Chebyshev y los polinomios Minimax.

Figura 5. Diagrama generalizado del método PPA de: (a) grado uno, (b) grado k



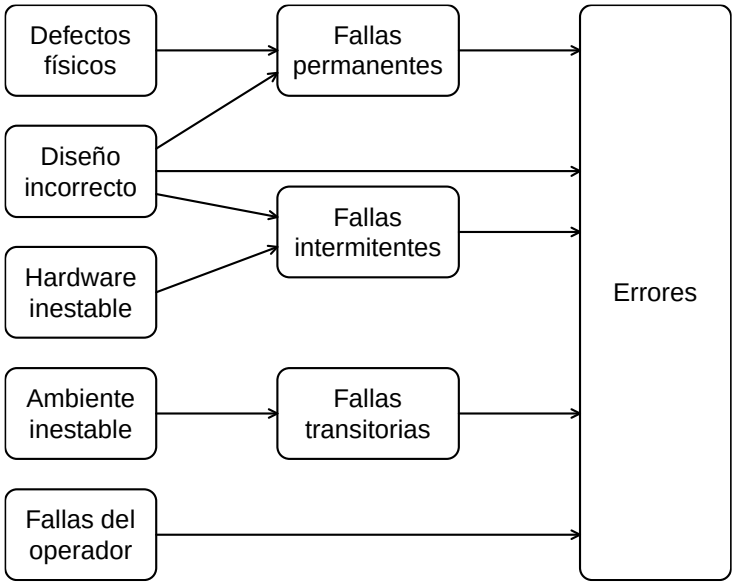
Fuente: Adaptado de [36]

El método de aproximación para la evaluación polinomios por partes, a su vez, se puede subdividir en dos categorías según el tipo de segmentación: uniforme o no uniforme [37]. La segmentación no uniforme usa un codificador de direcciones para ajustar los valores de entrada y es eficaz para calcular funciones altamente no lineales [38], [39]. La segmentación uniforme no requiere de un codificador para ajustar el dato de entrada ya que el subintervalo de entrada se divide en segmentos de igual longitud. Para un dato X de entrada con n bits, este es dividido en dos partes: los m bits más significativos, X_u, y los $n-m$ bits menos significativos (LSBs), X_l, como se puede observar en la Figura 5. El número total de segmentos es de 2^m, correspondiente al número de entradas de la tabla de coeficientes.

4.4. FALLAS

Una falla es un estado incorrecto en alguna parte de un sistema, resultante de interferencias del entorno físico, errores de los operarios o un mal diseño. Un error es la manifestación de una falla en un sistema, ocasionando un funcionamiento incorrecto del mismo [40], pero la presencia de una falla no necesariamente conduce a que se presente un error en el sistema, la falla se puede presentar en alguna zona que no esté siendo utilizada o que no se exteriorice, con lo cual la respuesta del sistema es correcta.

Figura 6. Fuentes de errores



Fuente: Adaptado de [6]

Las fallas en un sistema digital se pueden dividir en tres grupos principales de acuerdo a su duración y ocurrencia (ver Figura 6): fallas permanentes que ocurren,

pero no desaparecen, fallas intermitentes que ocurren y desaparecen repentinamente y fallas transitorias que a menudo son de duración finita.

1. Fallas permanentes: Una falla permanente hace alusión a una falla que es continua y estable en el tiempo, está relacionada con un defecto físico (p. ej. malas conexiones eléctricas, conexiones faltantes, conexiones de más, etc.) o un mal diseño del sistema. Estas se pueden presentar en mayor medida durante el proceso de fabricación, pero también durante el funcionamiento cuando el sistema empieza a desgastarse y se presentan cables de chip quemados o desgaste en las conexiones. Una falla permanente tiene la característica de que una vez que ocurre no desaparece, sin embargo, puede quedar enmascarada por algún método de tolerancia a fallas o de lo contrario, es necesario descartar o reemplazar el sistema defectuoso.

2. Fallas intermitentes: Las fallas intermitentes describen un tipo de falla que se presenta ocasionalmente debido a hardware inestable como conexiones sueltas, variación de resistencia, capacitancia e inductancia, ruido eléctrico, etc. o a un diseño inadecuado. Este tipo de fallas pueden preceder la aparición de una falla permanente y son difíciles de detectar ya que pueden ocurrir solo bajo ciertas condiciones ambientales o por alguna combinación específica a su entrada. La ocurrencia de estas fallas puede quedar enmascaradas por algún método de tolerancia a fallos, también se puede desviar la parte defectuosa o, por último, descartar o reemplazar el sistema defectuoso.

3. Fallas transitorias: Las fallas transitorias o fallas suaves se pueden describir como el cambio espontaneo de un solo bit que, cuando se prueba más adelante en el tiempo, funciona de forma correcta [41]. Su aparición es aleatoria por lo cual, son de difícil detección, pero no causan ningún daño, el bit que se modifica se puede corregir con una sobreescritura. Estas fallas se presentan momentáneamente por alguna condición ambiental temporal relacionada con fenómenos externos como: humedad, temperatura, presión, vibración, rayos cósmicos, polución, interferencia electromagnética, etc. o ruido generado en otra parte del sistema [40].

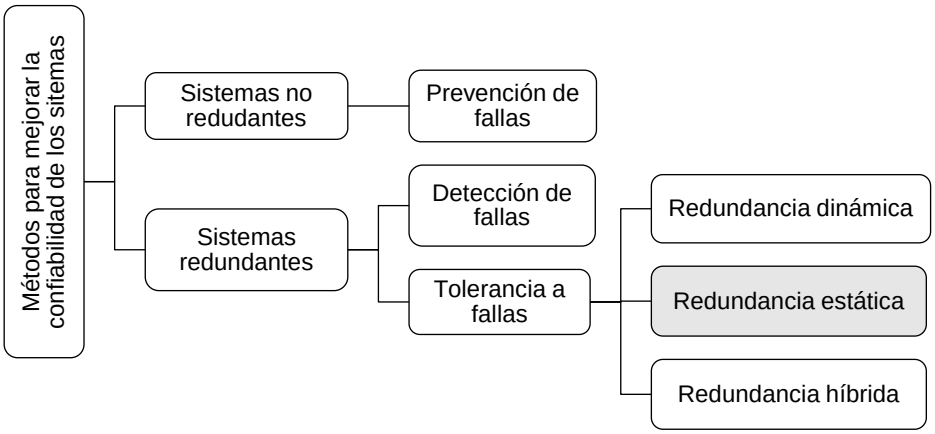
4.5. CONFIABILIDAD EN LOS SISTEMAS

La confiabilidad hace referencia a la expectativa de que un circuito realice su función prevista y está relacionado con el nivel de aceptación de calidad (partes defectuosas por millón), tasa de fallos hora, características de ruido, etc. [40]. Los sistemas digitales son ocupados en una gran variedad de aplicaciones donde la perdida de información en tiempo real podría afectar severamente su desempeño, por lo que, el aumento en la complejidad de los sistemas ha traído consigo la necesidad de un alto nivel de confiabilidad de los mismos. Aparte de las

condiciones ambientales y las especificaciones de los sistemas, la confiabilidad involucra aspectos matemáticos como probabilidad y estadística, lo cual se abordará más adelante en la sección 7.

Caben mencionar dos enfoques básicos para mejorar la confiabilidad de los sistemas (ver Figura 7), el primer enfoque es llamado prevención de fallas, asociado a sistemas intolerantes a fallas y el segundo se denomina tolerancia a fallas, asociado a sistemas que pueden tolerar la presencia de fallas. Por otro lado, las estrategias de detección de fallas se utilizan solamente para detectar eventos que necesiten de corrección o reparación, por ende, no proporcionan en si tolerancia a fallas, solo advierten de la presencia de estas. Dentro de este método se pueden mencionar algunas técnicas como: codificación de información (p. ej. paridad, sumas de comprobación), circuitos de auto chequeo [42], duplicación con comparación (*DWC - Duplication With Compare*) [43], entre otros.

Figura 7. Taxonomía de las estrategias de confiabilidad en hardware



Fuente: Elaboración propia

4.5.1. PREVENCIÓN DE FALLAS

La prevención de fallas consiste en evitar la presencia de estas a través del diseño y la fabricación del sistema, esta es una estrategia proactiva para identificar todas las potenciales fallas y así eliminar los factores que puedan ocasionarlas, por ejemplo, utilizar componentes de alta confiabilidad para la fabricación de un sistema o implementar reglas de diseño que limite el uso de recursos, lo cual puede disminuir la disipación de energía y con esto, reducir los efectos térmicos que puedan provocar fallas graves. Los errores humanos se pueden minimizar al implementar medidas como: etiquetado, documentación o la producción de componentes que solo se puedan ensamblar de la manera correcta.

No obstante, en la práctica es imposible garantizar que no se presenten fallas durante el diseño, la fabricación o el funcionamiento de un sistema, algún componente eventualmente puede fallar y con esto, hacer que el funcionamiento del sistema sea erróneo, de ahí el nombre de los sistemas intolerantes a fallas. Sin embargo, estos sistemas se caracterizan por la baja o media exigencia en confiabilidad y es posible su reemplazo cuando este sufre algún daño; además, los errores que se puedan presentar no constituyen daños críticos de recursos económicos, medio ambiente o vidas humanas.

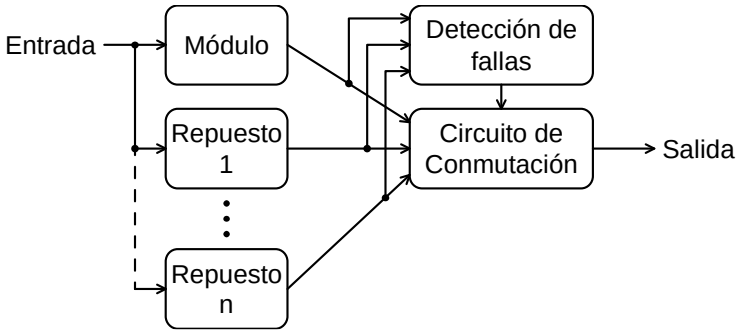
4.5.2. TOLERANCIA A FALLAS

En este enfoque se espera que eventualmente ocurran fallas durante el funcionamiento del sistema, pero con el uso de alguna técnica de confiabilidad lograr su detección para tomar una decisión correctiva o un enmascaramiento para que la respuesta del sistema no se vea afectada [44], [45]. Estos sistemas están orientados a satisfacer las necesidades de confiabilidad en ambientes de operación complejos (p. ej. ambientes con radiación o alta temperatura) y en aplicaciones de seguridad crítica (*SCS - Safety-Critical System*) como sistemas aeroespaciales, vehículos automáticos, transporte ferroviario, aviación, etc. [46]. Donde los posibles efectos de una falla pueden provocar graves daños económicos, ambientales, heridas o pérdida de vidas humanas. A continuación, se exponen algunas técnicas de tolerancia a fallas.

4.5.2.1. REDUNDANCIA DINÁMICA

La redundancia dinámica consiste en aquellos sistemas cuya configuración puede ser cambiada dinámicamente ante la presencia de una falla. Algunas técnicas que cabe mencionar son: duplicación reconfigurable [47], degradación elegante (g*raceful degradation*) [48], reconfiguración, recuperación, entre otras.

Figura 8. Estructura de redundancia dinámica de reconfiguración



Fuente: Adaptado de [44]

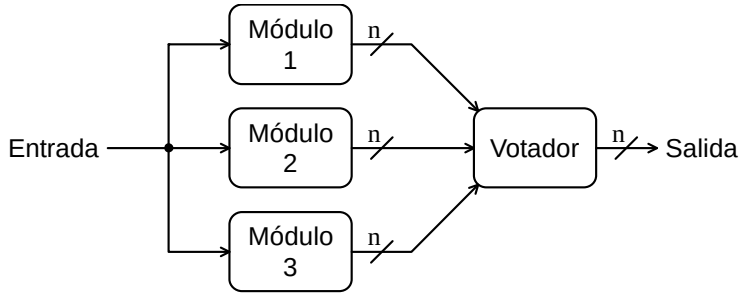
En la Figura 8 se puede ver la estructura de un sistema reconfigurable, donde hay un módulo funcional y n módulos de repuesto, el módulo de detección de fallas detecta la presencia de estas y desconecta el módulo activo defectuoso para activar un módulo en buen estado.

4.5.2.2. REDUNDANCIA ESTÁTICA

La redundancia estática es una técnica pasiva que usa un método de enmascaramiento para cubrir las fallas y se diseña para que no se requiera alguna acción de parte de algún sistema de control [49]. Consiste generalmente en hacer copias del sistema de interés y proporcionar un sistema de votación para decidir el valor correcto de la salida en función de las salidas de las copias, estos sistemas están diseñados de tal manera que, si se presenta una falla esta no interferirá en el funcionamiento correcto del sistema. Este diseño está integrado en la estructura del sistema, es decir, las conexiones entre los elementos permanecen constantes y enmascara los efectos de las fallas. Algunas técnicas que cabe mencionar son: códigos de corrección de errores como Hamming [50], lógica de enmascaramiento [51] y los modelos $M - de - N$.

En la Figura 9 se puede observar la estructura de redundancia estática más común llamada Redundancia Modular Triple, el cual es un caso particular de los modelos $M - de - N$. En estos modelos se necesitan de al menos M de los N módulos del sistema para que este siga funcionando o se enmascare el error. Los sistemas TMR toman los valores $2 - de - 3$, donde se implementan tres copias del módulo principal y el error se enmascara haciendo uso un hardware adicional llamado votador, el cual genera la salida del sistema mediante un voto mayoritario bit a bit donde se obtiene la salida que corresponde al valor de al menos dos de sus tres entradas. Este es un método común empleado hoy en día en sistemas de control aeronáutico, filtros digitales [52], máquinas de estado [53] y sumadores [54], entre otros.

Figura 9. Estructura de redundancia estática TMR

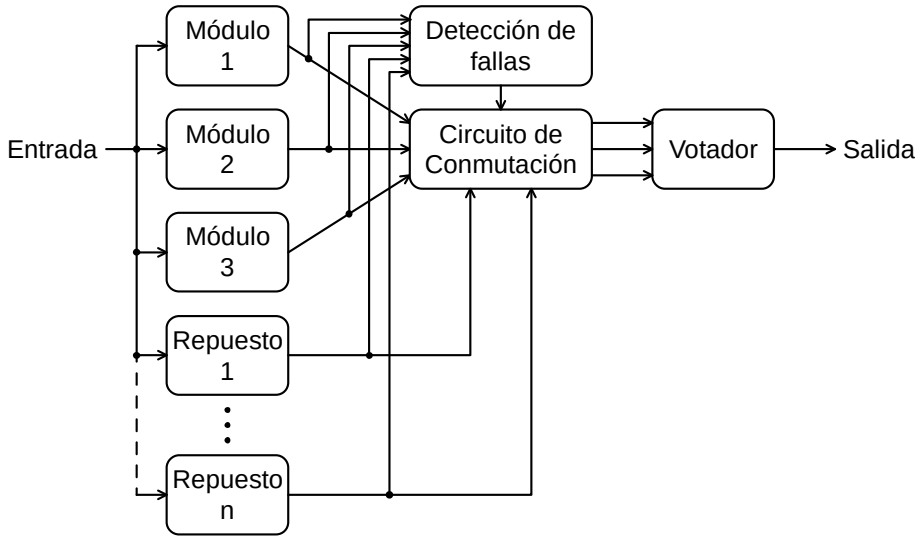


Fuente: Adaptado de [44]

4.5.2.3. REDUNDANCIA HÍBRIDA

La combinación de redundancia dinámica y redundancia estática es llamada redundancia híbrida. Un ejemplo de redundancia hibrida es la técnica de N-redundancia modular reconfigurable [55], donde se puede tener un sistema NMR con votador y además repuestos que pueden reemplazar algunos de los módulos en el sistema NMR. En la Figura 10 se puede ver la estructura de este sistema donde está presente el método TMR (redundancia estática) con el de reconfiguración (redundancia dinámica), en este caso, el detector de fallas detecta cuando uno de los módulos TMR tiene alguna falla para cambiarlo por alguno de los módulos de repuesto, cuando se agoten todos los módulos de repuesto, se tendrá un sistema TMR.

Figura 10. Estructura de redundancia híbrida TMR con reconfiguración



Fuente: Adaptado de [44]

4.6. FORMATO DE NÚMERO

El formato de número o notación usado en el desarrollo del trabajo es el formato de punto flotante de precisión simple, conocido como estándar IEEE-754 [56], donde el número M_x está representado por 32 bits. El MSB es el bit de signo s_x, seguido de un exponente normalizado E_x de 8 bits y los 23 bits restantes conforman el significando o mantisa X. El valor de M_x este dado por la expresión (2), donde $1 \leq X < 2$, con un 1 implícito a la izquierda del punto decimal; como resultado, la precisión efectiva es de 24 bits.

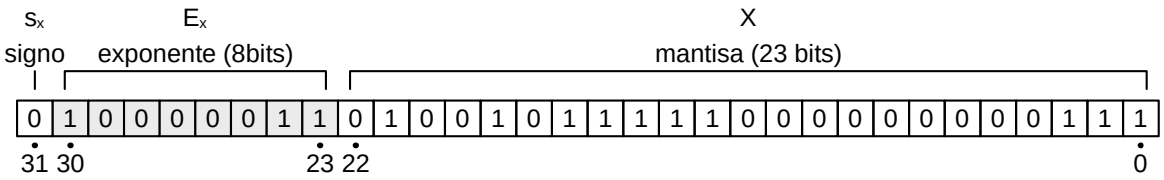
$$M_x = (-1)^{s_x} X 2^{E_x - 127} \quad (2)$$

Para convertir un número decimal en uno de formato IEEE-754 se puede seguir el siguiente procedimiento:

1. Si el número a convertir es negativo asignar $s_x = 1$, en caso contrario $s_x = 0$.
2. Igualar el número de conversión a 2^x.
3. Despejar x, tomar el valor entero inmediatamente inferior como e, sumarle 127 y convertirlo en binario para obtener E_x.
4. Tomar nuevamente el número de conversión e igualarlo a $m2^e$.
5. Despejar m y convertir la parte fraccionaria a binario, la parte entera (que siempre es uno) queda implícita y no se incluye. La conversión a binario se realiza hasta los primeros 24 bits y se aproxima al siguiente número sumándole un 1 si el bit en la posición 24 es uno; por último, el resultado se trunca a 23 bits para obtener la mantisa X.
6. Juntar los resultados de s_x, E_x y X para formar la representación final.

La mantisa se almacena en los bits 0 a 22, el uno implícito con valor de 1 se "coloca" enfrente, luego el bit 22 tiene un valor de $1/2$, el bit 21 vale $1/4$, etc. Como resultado, el valor de la mantisa X se encuentra entre 1 y 2.

Figura 11. Representación de un número decimal en el formato IEEE-754



Fuente: Elaboración propia

Ejemplo 1: Obtener la representación IEEE-754 del número 20.7422.

Siguiendo el proceso descrito anteriormente, se tiene:

$$s_x = 0$$

Luego,

$$20.7422 = 2^x \rightarrow x = \frac{\ln(20.7422)}{\ln(2)} \cong 4.3744 \rightarrow e = 4$$

Como $e = 4$, al sumar 127 se obtiene 131, por lo tanto:

$$E_x = 10000011$$

Tomando nuevamente el número de conversión,

$$20.7422 = m2^e \rightarrow m = \frac{20.7422}{2^4} = 1.2963875$$

Convirtiendo la parte fraccionaria con 24 bits se tiene:

$$0.2963875_{10} = 0.010010111110000000001101_2$$

Como el bit en la posición 24 después de la coma es 1, se aproxima al siguiente número sumándole 1 y se trunca a 23 bits, con lo cual:

$$X = 01001011111000000000111$$

Uniendo los resultados se obtiene la representación mostrada en la Figura 11 o su representación en hexadecimal,

$$41A5F007_{16}$$

La expresión en hexadecimal se obtiene agrupado los 32 bits en 8 conjuntos de 4 bits y convirtiendo cada grupo en el carácter correspondiente.

Ejemplo 2: Obtener la representación IEEE-754 del número -0.078125.

Siguiendo el proceso descrito anteriormente, se tiene:

$$s_x = 1$$

Luego,

$$0.078125 = 2^x \rightarrow x = \frac{\ln(0.078125)}{\ln(2)} = -3.678 \rightarrow e = -4$$

Como $e = -4$, al sumar 127 se obtiene 123, por lo tanto:

$$E_x = 01111011$$

Tomando nuevamente el número de conversión,

$$0.078125 = m2^e \rightarrow m = \frac{0.078125}{2^{-4}} = 1.25$$

Convirtiendo la parte fraccionaria con 24 bits se tiene:

$$0.2963875_{10} = 0.01000000000000000000000_2$$

$$X = 01000000000000000000000$$

Concatenando los resultados se obtiene:

$$3DA000000_{16}$$

La conversión también se puede realizar desde la representación binaria del número a convertir. De este modo, lo que se hace es tomar el valor absoluto del número a convertir y expresarlo en notación científica base 2. Esta representación se obtiene a partir del número en binario, moviendo el punto decimal a la izquierda o a la derecha de forma que solo se tenga un dígito en la parte entera. El exponente de la notación científica será positivo cuando el punto decimal se mueva hacia la izquierda y negativo cuando se mueva a la derecha. Retomando el ejemplo anterior se tendría:

$$|-0.078125_{10}| = 0.078125_{10} = 0.000101_2 = 1.01_2 x 10^{-4}$$

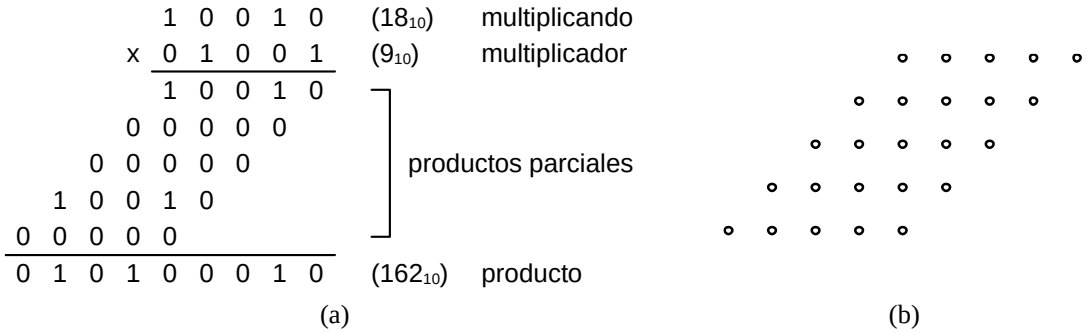
Con esto, el exponente E_x se obtiene sumando 127 al valor del exponente de la notación científica, en este caso se tendría que $E_x = 127 + (-4) = 123$. La mantisa se compone con los primeros 23 bits del resto del número después del punto decimal de la notación científica en base 2 (teniendo en cuenta la aproximación al número siguiente si es necesaria). Para este ejemplo se tendría que $X = 01$, lo cual se completa con ceros a la derecha para tener la longitud de 23 bits y el signo se determina como se indicó inicialmente, por lo que $s_x = 1$. Finalmente, se concatenan las tres partes de la representación IEEE-754, con lo cual se tendría:

$$1\ 01111011\ 01000000000000000000000 = 3DA000000_{16}$$

4.7. MULTIPLICACIÓN RÁPIDA

Un enfoque estándar para realizar una multiplicación de dos números binarios es la de desplazar y sumar o multiplicación normal. En donde, para cada columna del multiplicador se hace un corrimiento del multiplicando de acuerdo con el número de la columna evaluada y se multiplica por el valor de esta para obtener el producto parcial, finalmente se suman los productos parciales para obtener el producto o resultado final (ver Figura 12). Con este método, el número de productos parciales es el mismo que el número de columnas del multiplicador.

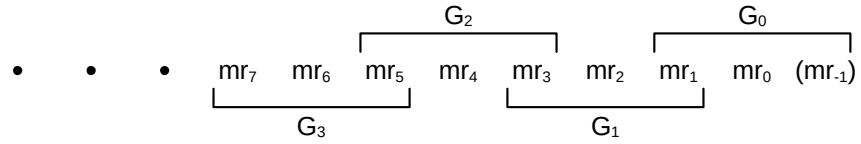
Figura 12. Multiplicación normal: (a) Ejemplo, (b) Matriz de p.p.



Fuente: Adaptado de [57]

La multiplicación normal se puede acelerar reduciendo el número de productos parciales usando diferentes técnicas, una de estas técnicas se conoce como la codificación de Booth, la cual hace uso del algoritmo de Booth en base 4 (*radix-4 Booth Algorithm*), con el cual, el número de productos parciales se reduce de n a $(n+1)/2$, siendo n el numero de bits del multiplicador [57]. Esta codificación considera el multiplicador MR en grupos de 3 bits, donde dos grupos consecutivos tienen un bit en común, empezando con el primer grupo $mr_1 mr_0 mr_{-1}$, el siguiente grupo $mr_3 mr_2 mr_1$, y así sucesivamente como se muestra en la Figura 13.

Figura 13. Agrupación del multiplicador según la codificación de Booth



Fuente: Adaptado de [58]

Las reglas del algoritmo de Booth en base 4 se muestran en la Tabla 1, para todos los valores impares de $i = 1,3,5,7,\cdots$ donde para $i = 1$, $mr_{i-2} = 0$. Los productos parciales negativos se generan tomando el complemento a dos del multiplicando y cuando se opera $\pm 2MO$ se hace un corrimiento a la izquierda de un bit descartando el MSB. Cada producto parcial tiene un tamaño de $u+1$ bits para acomodar los múltiplos de $\pm 2MO$, siendo u la cantidad de bits del multiplicando. El MSB de cada producto parcial representa el signo del mismo ya que se pueden tener productos parciales positivos y negativos, por lo que este se tiene que extender hasta el final de la matriz de multiplicación para su acumulación (ver Figura 15).

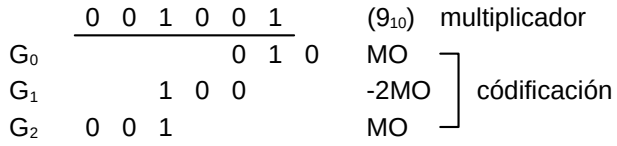
Tabla 1. Codificación Booth en base 4

mr_i	mr_{i-1}	mr_{i-2}	$Operación$
0	0	0	0
0	0	1	MO
0	1	0	MO
0	1	1	$2MO$
1	0	0	$-2MO$
1	0	1	$-MO$
1	1	0	$-MO$
1	1	1	-0

Fuente: Adaptado de [58]

Retomando el ejemplo de la Figura 12, se puede agrupar el multiplicador en tres grupos como se muestra en la Figura 14, en caso de que no se tengan los MSBs para generar el último bloque de 3 bits, se extiende el multiplicador con ceros. Los grupos G_0 y G_2 están conformados por el grupo de bits "010" generan el producto parcial simple MO, que es igual al multiplicando. El grupo G_1 "100" genera el producto parcial $-2MO$, el cual se corre de un bit a la izquierda duplicar su valor y posteriormente se toma el complemento a dos para su representación negativa.

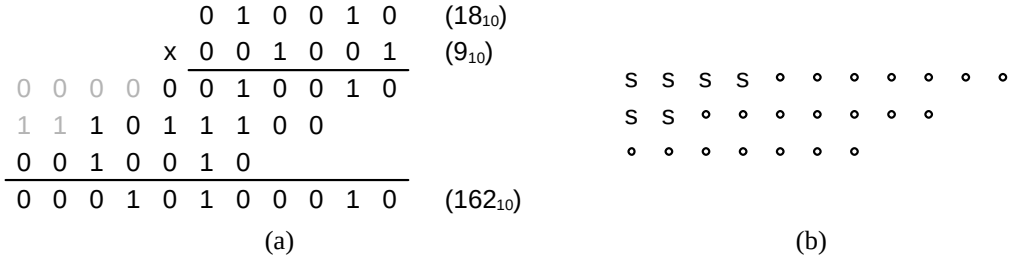
Figura 14. Agrupación del multiplicador en para la codificación Booth



Fuente: Adaptado de [57]

Reescribiendo la multiplicación de la Figura 12 con la codificación de Booth, se obtiene la matriz de multiplicación mostrada en la Figura 15a. Se puede observar la reducción de la cantidad de productos parciales y la extensión del bit de signo necesaria para realizar la acumulación. En la matriz de puntos de los productos parciales en la Figura 15b, s representa la extensión de signo.

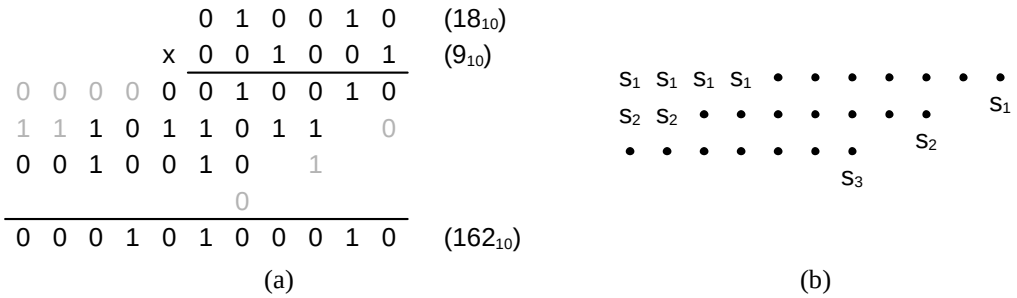
Figura 15. Multiplicación con codificación Booth: (a) Ejemplo, (b) Matriz de p.p.



Fuente: Adaptado de [57]

La representación de números negativos en complemento a dos tiene el inconveniente de que se requiere sumar un uno "1" al LSB, esto implica hardware adicional para generar ese producto parcial directamente en esta representación, para esto, se puede generar el producto parcial en complemento a uno y usar el bit de signo como un bit de corrección que se añade a la acumulación final como se observa en la Figura 16. Los puntos negros en la matriz de productos parciales (Figura 16b) indican la representación en complemento a uno de los números negativos, los bits s_i que están en frente de cada producto parcial son la extensión de signo y los que están debajo de cada producto parcial son el bit de corrección.

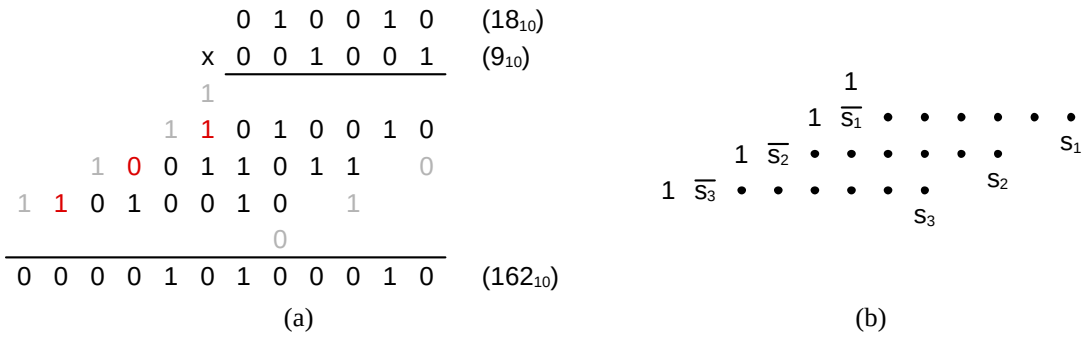
Figura 16. Modificación para p.p. negativos: (a) Ejemplo, (b) Matriz de p.p.



Fuente: Adaptado de [57]

Los bits de extensión de signo de los productos parciales son todos o unos o ceros y se requiere hardware adicional para efectuar su acumulación. Si se agrega un uno a una cadena de unos, la cadena se convierte en ceros más un acarreo que se puede descartar. De este modo, se pueden remplazar las cadenas de la extensión del bit de signo invirtiendo el MSB de cada producto parcial, añadiendo un uno al MSB del primer producto parcial y otro en frente de cada producto parcial para lograr la propagación correcta [57], [58]. Así, la multiplicación de la Figura 16 se puede replantear como se muestra en la Figura 17.

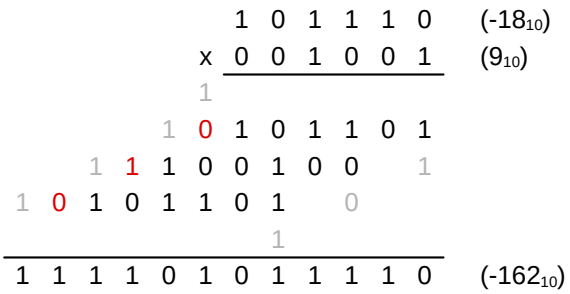
Figura 17. Modificación de extensión de signo: (a) Ejemplo, (b) Matriz de p.p.



Fuente: Adaptado de [57]

Cuando el multiplicando es un número negativo, su signo se opera con el signo del producto parcial que genera la codificación del multiplicador y el resultado de la acumulación se obtiene expresado en complemento a dos. En la Figura 18 se puede ver un ejemplo de aplicación donde se evidencia lo mencionado.

Figura 18. Multiplicación con números negativos con la codificación de Booth

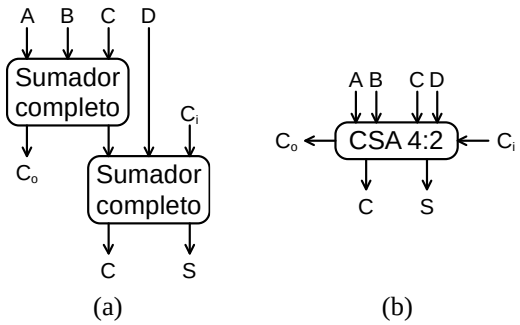


Fuente: Elaboración propia

4.8. SUMADOR CARRY-SAVE 4:2

Una parte importante en la evaluación polinomial que realiza la SFU es la acumulación de los productos parciales y la suma final, para lo cual se hará uso de sumadores rápidos que contribuyen a la reducción del retardo, área y potencia. El sumador carry-save o compresor $4{:}2$ (denominado *CSA 4:2*) es una topología usada frecuentemente para este fin [59], consiste en una estructura para sumar cuatro entradas en simultaneo formada por la combinación de dos sumadores completos (*FA - Full Adders*) conectados en cascada como se observa en la Figura 19. Este circuito tiene cinco bits de entrada y tres de salida. Los 4 bits de entrada A, B, C y D y la salida de suma S tienen el mismo peso. La entrada C_i es la salida de un sumador anterior y la salida C_o se usa como acarreo de entrada del siguiente sumador.

Figura 19. CSA 4:2 versión: (a) Sumadores completos, (b) Diagrama de bloques



Fuente: Adaptado de [60]

El comportamiento del sumador carry-save 4:2 se muestra en la Tabla 2, del cual se obtienen las expresiones mostradas en (3), La ventaja de esta implementación es que presenta un 25% menos de latencia que los dos FAs en cascada [60].

Tabla 2. Tabla de verdad del sumador carry-save 4:2

A	B	C	D	Ci	Co	C	S
0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	1
0	0	0	1	0	0	0	1
0	0	0	1	1	0	1	0
0	0	1	0	0	0	0	1
0	0	1	0	1	0	1	0
0	0	1	1	0	0	1	0
0	0	1	1	1	0	1	1
0	1	0	0	0	0	0	1
0	1	0	0	1	0	1	0
0	1	0	1	0	0	1	0
0	1	0	1	1	0	1	1
0	1	1	0	0	1	0	0
0	1	1	0	1	1	0	1
0	1	1	1	0	1	0	1
0	1	1	1	1	1	1	0

A	B	C	D	Ci	Co	C	S
1	0	0	0	0	0	0	1
1	0	0	0	1	0	1	0
1	0	0	1	0	0	1	0
1	0	0	1	1	0	1	1
1	0	1	0	0	1	0	0
1	0	1	0	1	1	0	1
1	0	1	1	0	1	0	1
1	0	1	1	1	1	1	0
1	1	0	0	0	1	0	0
1	1	0	0	1	1	0	1
1	1	0	1	0	1	0	1
1	1	0	1	1	1	1	0
1	1	1	0	0	1	0	1
1	1	1	0	1	1	1	0
1	1	1	1	0	1	1	0
1	1	1	1	1	1	1	1

Fuente: Elaboración propia

$$\begin{aligned} C_o &= AB + AC + BC \\ S &= A \oplus B \oplus C \oplus D \oplus C_i \\ C &= (A \oplus B \oplus C \oplus D)C_i + (\overline{A \oplus B \oplus C \oplus D})D \end{aligned} \tag{3}$$

5. IMPLEMENTACIÓN DE LA SFU BASADA EN PPA

Las SFUs basadas en aproximaciones polinomiales de grado bajo tienen algunas ventajas sobre otro tipo de métodos de evaluación de funciones elementales. Lo primero es que se encuentran a medio camino entre las búsquedas directas en tablas, las cuales tienen requisitos altos de uso de memoria, y las aproximaciones polinomiales/racionales de grados altos que tienen un alto número de multiplicaciones y sumas que implican un mayor tiempo de retardo en su ejecución. Lo segundo es que son más rápidos que los métodos iterativos que generalmente implican tiempos elevados de ejecución [61]. Las funciones a implementar se consideraron con respecto a las más usadas en estas unidades como: $\sin(x)$, $\cos(x)$, $\log_2(x)$, $1/x$, $\sqrt{x}$, $1/\sqrt{x}$ y 2^x [37], [62], [63]. Se opto por implementar una arquitectura para evaluar un polinomio de grado dos ya que es la mejor opción que compensa entre área y velocidad [61]. El operando de entrada X de n bits se divide en una parte superior X_u y una inferior X_l, con lo cual $X = X_u + X_l$, donde X_u tiene de 6 a 7 bits de longitud tal que $X_u = [.x_1x_2 \dots x_m]$ y $X_l = [.x_{m+1} \dots x_n] \cdot 2^{-m}$. La aproximación polinomial de orden dos para una función $f(X)$ dentro de un rango especifico $(X_u \leq X < X_u + 2^{-m})$ se puede calcular evaluando la expresión:

$$f(X) = C_0 + C_1X_l + C_2X_l^2 \qquad (4)$$

El valor de los coeficientes C_0, C_1 y C_2 son calculados y almacenados en tablas de 2^m direcciones ya que sus valores dependen solamente de X_u, los m MSB de X, que indican el subintervalo a evaluar. Estos coeficientes se obtienen a través de un proceso iterativo en donde se tiene en cuenta el efecto de su redondeo a una longitud de palabra finita para poder realizar su almacenamiento en las tablas y son direccionados por X_u. La evaluación del polinomio se realiza con una unidad de elevación al cuadrado especializada con un tamaño y retardo reducido con respecto a un multiplicador estándar que genera el valor de X_l^2 y un árbol de acumulación fusionado que genera los productos parciales de C_1X_2 y $C_2X_2^2$ y realiza la suma de los mismos junto al coeficiente C_0.

A continuación, en la sección 5.1 se explica el análisis matemático que permite obtener los rangos de evaluación de los polinomios de las diferentes funciones a partir de su representación en el formato IEEE-754 y la reconstrucción del resultado después de su evaluación. Posteriormente en la sección 5.2 se explica el procedimiento matemático para obtener los coeficientes del polinomio una vez se tiene un rango de evaluación definido y se expone el tamaño de las tablas que se obtuvieron para las funciones escogidas que dependen los rangos de evaluación deducidos del análisis anterior. En la sección 5.3 se expone el modelo en software de la SFU y, por último, en la sección 5.4 se presenta su descripción en hardware.

5.1. REDUCCIÓN DE RANGO Y RECONSTRUCCIÓN

La evaluación de una función por medio de una aproximación polinomial se lleva a cabo principalmente en tres pasos: reducción, aproximación y reconstrucción [64]. La reducción consiste en reducir el argumento de entrada a un intervalo determinado. La aproximación consiste en la evaluación polinomial que se realiza sobre el argumento reducido y por último la reconstrucción, que se realiza para obtener el resultado final. La selección del intervalo de entrada para algunas funciones se obtiene naturalmente de la representación en el formato IEEE-754, lo que simplifica los pasos de reducción de rango, sin embargo, para las funciones $sin(x)$, $cos(x)$ y 2^x la selección del rango no es tan intuitiva y se requiere de un pre procesamiento para ajustar el dato al intervalo de aproximación. En este trabajo, se explica en que consiste este pre procesamiento pero se asume que el argumento de entrada a la SFU se encuentra reducido al intervalo apropiado. A continuación, se expone en detalle las consideraciones a tener en cuenta para ajustar el rango del intervalo de entrada de las diferentes funciones y su post procesamiento requerido para normalizar el resultado.

5.1.1. SENO/COSENO

Las funciones $sin(x)$ y $cos(x)$ necesitan varias consideraciones. Primero, se tiene que mapear el ángulo M_x a X haciendo una reducción al primer cuadrante. La reducción se hace considerando las igualdades que se presentan en la Tabla 3 y para valores negativos, se utilizan las identidades $sin(-x) = -sin(x)$ y $cos(-x) = cos(x)$.

Tabla 3. Equivalencias para reducción de cuadrante

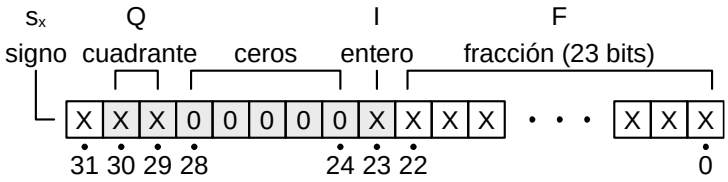
$\boldsymbol{Cuadrante}$	$\boldsymbol{Reducción}$	
1	$sin(x) = sin(x)$	$cos(x) = cos(x)$
2	$sin(\pi/2 + x) = cos(x)$	$cos(\pi/2 + x) = -sin(x)$
3	$sin(\pi + x) = -sin(x)$	$cos(\pi + x) = -cos(x)$
4	$sin(3\pi/2 + x) = -cos(x)$	$cos(3\pi/2 + x) = sin(x)$

Fuente: Elaboración propia

Además de la reducción al primer cuadrante, el formato de representación del dato de entrada debe estar representado en punto fijo $I.F$ donde I denota la parte entera y F la parte fraccionaria. La parte entera solo ocupará un bit, ya que, con la reducción al primer cuadrante se tiene el rango $(0, \pi/2)$ para el ángulo de entrada. La información del cuadrante tiene que estar presente para considerar cuándo se debe computar una función u otra, por esto, se utiliza el formato presentado en la Figura 20 para los datos de entrada a la SFU. Hasta este punto, el dato ingresa a

la SFU con esto en consideración, por lo que en el diseño no se tiene en cuenta este pre procesamiento.

Figura 20. Formato en punto fijo para la función exponencial base 2



Fuente: Elaboración propia

El rango de entrada se puede reducir a $[0,1)$ para el polinomio de aproximación aplicando las propiedades trigonométricas $sin(x) = cos(\pi/2 - x)$ y $cos(x) = sin(\pi/2 - x)$, así, cuando el ángulo sea mayor a 1, se calcula la función opuesta de $\pi/2 - x$. Este pre procesamiento del dato se realiza dentro de la SFU y con esto, las funciones $f(x)$ que evalúa el polinomio de aproximación son: $sin(F)$ y $cos(F)$, dentro del rango $[0,1)$ y no se requiere de post procesamiento, por lo tanto, se tiene:

$$sin(M_x) = sin(F)$$
$$cos(M_x) = cos(F) \quad (5)$$

5.1.2. RECIPROCO DE LA RAÍZ CUADRADA

Expresando esta función con la representación del número de entrada en el formato IEEE-754, se tiene:

$$\frac{1}{\sqrt{M_x}} = \frac{1}{\sqrt{X2^{E_x-127}}}$$

Lo cual se puede reescribir así:

$$\frac{1}{\sqrt{M_x}} = \frac{1}{(X2^{E_x-127})^{\frac{1}{2}}} = \frac{1}{X^{\frac{1}{2}}2^{\frac{E_x-127}{2}}} = \frac{1}{\sqrt{X}}2^{-\frac{E_x-127}{2}}$$

Con esto, la función $f(x)$ a evaluar sería $1/\sqrt{X}$, donde el rango de entrada para X es $(1,2)$ y posteriormente se tendría que hacer un corrimiento de cierta cantidad de bits determinado por la expresión $2^{-(E_x-127)/2}$. Si embargo, esto solo aplica cuando el valor de $(E_x - 127)$ es par, así pues, esto se puede corregir sumando o restando 1 a E_x. El hecho de sumarle o restarle 1 al valor del exponente, duplica o

divide a la mitad el valor de M_x, lo cual se puede compensar dividiendo a la mitad o duplicando el valor de X. En este caso, se optó por la opción de restar 1 a E_x y duplicar X, en consecuencia, la función $f(x)$ a evaluar sería $1/\sqrt{2X}$ cuando E_x es impar. El rango de entrada para X sigue siendo el mismo y con esto se tendría:

$$\frac{1}{\sqrt{M_x}} = \begin{cases} \frac{1}{\sqrt{X}} 2^{-\frac{E_x-127}{2}}, si\ (E_x - 127)\ es\ par \\ \frac{1}{\sqrt{2X}} 2^{-\frac{(E_x-127-1)}{2}}, si\ (E_x - 127)\ es\ impar \end{cases} \quad (6)$$

Se necesitan entonces, dos tablas para esta función, una cuando $(E_x - 127)$ es par y otra cuando $(E_x - 127)$ es impar para el rango de entrada $(1,2)$.

5.1.3. LOGARITMO BASE 2

Expresando el logaritmo en función de la representación del número de entrada en el formato IEEE-754, se tiene:

$$\log_2(M_x) = \log_2(X2^{E_x-127}) = \log_2(X) + \log_2(2^{E_x-127}) = \log_2(X) + E_x - 127$$

Con esto, la función $f(x)$ a evaluar sería $\log_2(X)$ y posteriormente se normalizaría el resultado sumando $(E_x - 127)$, sin embargo, según el trabajo de Piñeiro et al. en [61], existe una pérdida de precisión cuando $(E_x - 127) = 0$. Esta pérdida de precisión se puede minimizar evaluando $f(x)$ igual a $\log_2(X)/(X-1)$ y multiplicando el resultado de esta aproximación por $(X - 1)$. Con esto se tendría:

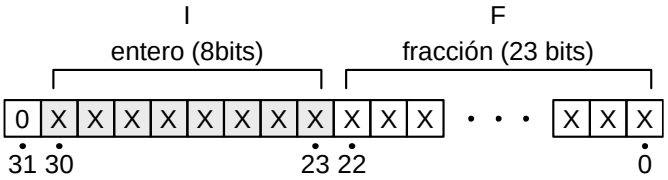
$$\log_2(M_x) = \begin{cases} \log_2(X) + E_x - 127, si\ (E_x - 127) \neq 0 \\ (X-1)\left[\frac{\log_2(X)}{(X-1)}\right], si\ (E_x - 127) = 0 \end{cases} \quad (7)$$

Para esta función se necesitan entonces dos tablas, una cuando $(E_x - 127) \neq 0$ y otro cuando $(E_x - 127) = 0$ para el rango de entrada $(1,2)$.

5.1.4. EXPONENCIAL BASE 2

Para esta función, el pre procesamiento consiste en cambiar la representación del formato en punto flotante a punto fijo con signo $I.F$ donde I denota la parte entera y F la parte fraccionaria como se observa en la Figura 21. Para mantener el tamaño de 32 bits del dato de entrada, el bit de signo es concatenado como 0.

Figura 21. Formato en punto fijo para la función exponencial base 2



Fuente: Elaboración propia

Hay que tener en cuenta que el valor en punto fijo es negativo si el MSB de la parte entera I es 1, Así, el rango de entrada para F es $(0,1)$ y la función $f(x)$ que evalúa el polinomio de aproximación es 2^F; el post procesamiento requerido consiste en multiplicar el resultado de la aproximación por 2^I.

$$2^{M_x} = 2^{I+F} = 2^I 2^F, F \in (0,1) \quad (8)$$

5.1.5. RECIPROCO

Expresando esta función con la representación del número de entrada en el formato IEEE-754, se tiene:

$$\frac{1}{M_x} = \frac{1}{(-1)^{s_x} X 2^{E_x-127}} = (-1)^{s_x} \frac{1}{X} 2^{-(E_x-127)}$$

Con esto, la función $f(x)$ a evaluar sería $1/X$, donde el rango de entrada para X es $(1,2)$ y posteriormente se normaliza el resultado multiplicando por $2^{-(E_x-127)}$. Con lo cual se tiene:

$$\frac{1}{M_x} = (-1)^{s_x} \frac{1}{X} 2^{-(E_x-127)} \quad (9)$$

5.1.6. RAÍZ CUADRADA

Expresando esta función con la representación del número de entrada en el formato IEEE-754, se tiene:

$$\sqrt{M_x} = \sqrt{X 2^{E_x-127}} = (X 2^{E_x-127})^{\frac{1}{2}} = X^{\frac{1}{2}} 2^{\frac{E_x-127}{2}} = \sqrt{X} 2^{\frac{E_x-127}{2}}$$

Con esta expresión, la función $f(x)$ a evaluar sería $\sqrt{X}$, donde el rango de entrada para X es $(1,2)$ y posteriormente se tendría que hacer un corrimiento de cierta

cantidad de bits determinado por la expresión $2^{(E_x-127)/2}$. Si embargo, como se comentó anteriormente, solo aplica cuando $(E_x - 127)$ es par, así pues, restando 1 a E_x y duplicando X se tiene que la función $f(x)$ a evaluar sería $\sqrt{2X}$ cuando E_x es impar. Con lo cual se tendría:

$$\sqrt{M_x} = \begin{cases} \sqrt{\mathrm{X}} \cdot 2^{\frac{E_x-127}{2}}, si\ (E_x - 127)\ es\ par \\ \sqrt{2\mathrm{X}} \cdot 2^{\frac{E_x-127-1}{2}}, si\ (E_x - 127)\ es\ impar \end{cases} \quad (10)$$

Se necesitan entonces, dos tablas para esta función, una cuando $(E_x - 127)$ es par y otra cuando $(E_x - 127)$ es impar para el rango de entrada $(1,2)$.

5.2. CÁLCULO DE LOS COEFICIENTES POLINOMIALES

La generación de los coeficientes polinomiales depende de la función y el intervalo a evaluar, anteriormente se definieron estos parámetros para las funciones a computar por la SFU. En la sección 5.2.1 se expone el procedimiento general para poder obtener los coeficientes de una función cualquiera junto con un ejemplo de aplicación. Una vez se tienen los coeficientes es necesario establecer un tamaño específico para poder almacenarlos en tablas, en la sección 5.2.2 se expone el procedimiento para truncar estos valores y finalmente en la sección 5.2.3 se resumen los resultados obtenidos de la generación de los coeficientes y su truncamiento para cada una de las funciones.

5.2.1. APROXIMACIÓN MINIMAX

Para aproximar una función arbitraría $f(x)$ en un intervalo $[a, b]$ comúnmente se hace uso de un polinomio $p(x)$ de grado k. Las aproximaciones que minimizan el error en el peor de los casos o el error máximo son llamadas aproximaciones minimax [65], donde el error está dado por:

$$E = \|p(x) - f(x)\|_{\infty,[a,b]} = \max_{a \le x \le b} |p(x) - f(x)| \quad (11)$$

Un algoritmo de aproximación minimax popular es el algoritmo de Remez [66], publicado por Evgeny Yakovlevich Remez en 1934, el cual computa el polinomio de minimax de forma iterativa. Este algoritmo es implementado en algunos programas de software como Maple o Mathematica [67]. Para inicializar el algoritmo, se requiere un grupo de $k + 2$ puntos dentro del intervalo $[a, b]$, tal que el polinomio $p(x)$ es:

$$p(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_{k-1} x^{k-1} + a_k x^k$$

Ahora, se hace oscilar de forma alternante el error entre el polinomio $p(x)$ y la función $f(x)$ en los puntos seleccionados [68], para lo cual se tiene el siguiente sistema de ecuaciones:

$$p(x_i) + (-1)^i E = f(x_i) \rightarrow 1 \leq i \leq k+2$$

Las cuales se pueden expresar en forma matricial, así:

$$\begin{pmatrix} 1 & x_0 & x_0^2 & \dots & x_0^k & 1 \\ 1 & x_1 & x_1^2 & \dots & x_1^k & -1 \\ 1 & x_2 & x_2^2 & \dots & x_2^k & 1 \\ \vdots & \vdots & \vdots & \dots & \vdots & \vdots \\ 1 & x_{k+2} & x_{k+2}^2 & \dots & x_{k+2}^k & (-1)^i \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \\ E \end{pmatrix} = \begin{pmatrix} f(x_0) \\ f(x_1) \\ \vdots \\ f(x_{k+2}) \end{pmatrix} \qquad (12)$$

El sistema de ecuaciones se resuelve para encontrar el valor de los coeficientes $a_0, a_1, \dots, a_k$ y E formando el polinomio $p(x)$. Esto no garantiza que el error E tenga el valor de los máximos o mínimos de la función de error, por esto, se halla un nuevo conjunto de puntos calculando las raíces de la función de error $f(x) - p(x)$. Este nuevo conjunto de puntos se utiliza para iterar la representación matricial y así obtener un nuevo polinomio hasta que se cumpla con el criterio de detención.

Al final de cada iteración se tiene un nuevo conjunto de puntos de control, y evaluando el error en estos puntos se tiene $E_m = \text{mín}_i |E_i|$ y $E_M = \text{máx}_i |E_i|$, a medida que se itera, la diferencia entre el antiguo y el nuevo conjunto de puntos se minimiza. Por lo tanto, un criterio de detención es cuando $E_M = \alpha E_m$, siendo α una constante cercana a la unidad, o cuando E_M tiende a $|E|$. Resumiendo, los pasos principales son:

1. Para una función dada $f(x)$ en un intervalo $[a, b]$, se especifica el grado de polinomio $p(x)$.
2. Escoger un conjunto de puntos $X = \{x_1, x_2, \dots, x_{k+2}\}$.
3. Resolver el sistema de ecuaciones (12), para obtener los coeficientes $a_0, a_1, \dots, +a_k$ y el error E.
4. Formar el nuevo polinomio $p(x)$ con los anteriores coeficientes.
5. Computar la función de error $f(x) - p(x)$, para generar el nuevo conjunto de putos.
6. Si se cumple con el criterio de detención, detener la iteración, de lo contrario, utilizar el nuevo conjunto de puntos y continuar desde el paso 3.

Ejemplo 3: Encontrar el polinomio minimax de grado 2, para $f(x) = 2^x$ en el primer subintervalo del rango $[0,1]$ usando $m = 6$.

Sol. Con $m = 6$, se tienen un total de 64 subintervalos ($2^6 = 64$), por lo que el primer subintervalo estaría dado en $\left[0, \frac{1}{64}\right]$. Se escoge inicialmente $X = \left\{0, \frac{1}{192}, \frac{1}{96}, \frac{1}{64}\right\}$, entonces,

$$(a_0 + a_1 x_i + a_2 x_i^2) + (-1)^i E = 2^{x_i}$$

Con lo cual,

$$\begin{pmatrix} 1 & 0 & 0 & 1 \\ 1 & \frac{1}{192} & \frac{1}{192^2} & -1 \\ 1 & \frac{1}{96} & \frac{1}{96^2} & 1 \\ 1 & \frac{1}{64} & \frac{1}{64^2} & -1 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \\ E \end{pmatrix} = \begin{pmatrix} 1 \\ 2^{\frac{1}{192}} \\ 2^{\frac{1}{96}} \\ 2^{\frac{1}{64}} \end{pmatrix}$$

Resolviendo el sistema se tiene:

$$a_0 = 1.000000005913372$$
$$a_1 = 0.693139619632341$$
$$a_2 = 0.241531568950669$$
$$|E| = 5.913372152946231e - 09$$

Luego,

$$p(x) = 1.000000005913372 + 0.693139619632341x + 0.241531568950669x^2$$

Evaluando el error,

$$f(x) - p(x) = 2^x - 1.000000005913372 - 0.693139619632341x - 0.241531568950669x^2$$

Derivando se tiene:

$$\left(f(x) - p(x)\right)' = 2^x ln(2) - \frac{271940370981111x}{562949953421312} - \frac{6243246665383873}{9007199254740992}$$

$$r_1 = 0.003836658717622267057912313666263 6$$
$$r_2 = 0.011792519691389333493557771532309$$

La segunda derivada es,

$$\left(f(x) - p_1(x)\right)'' = 2^x log(2)^2 - \frac{271940370981111}{562949953421312}$$

Calculando los signos para saber si son máximos o mínimos:

$$\big(f(x)-p_1(x)\big)''|_{x=r_1} = -0.001330721601294 \rightarrow máximo$$

$$\big(f(x)-p_1(x)\big)''|_{x=r_2} = 0.001333169975076 \rightarrow mínimo$$

Por lo tanto:

$$E_m = \big(f(x)-p(x)\big)|_{x=r_2} = -7.029488947906445e-09$$

$$E_M = \big(f(x)-p(x)\big)|_{x=r_1} = 7.021582529358575e-09$$

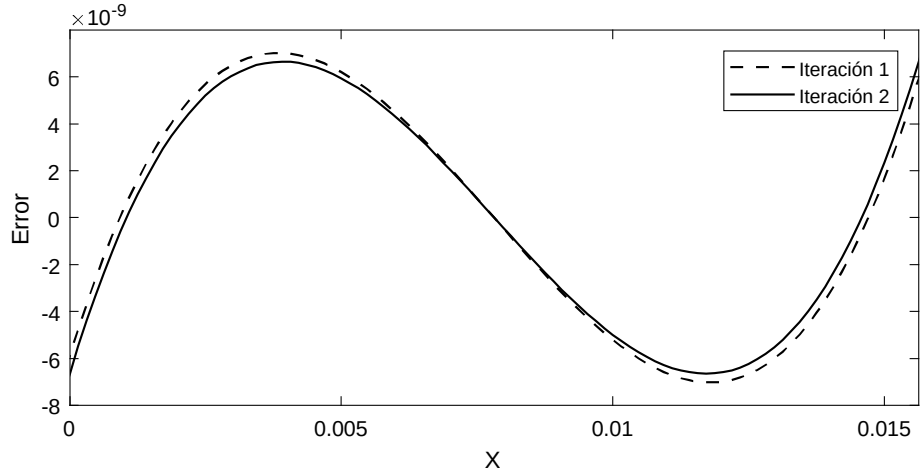
Aún se puede realizar otra iteración para disminuir el valor de E_M. El nuevo conjunto de X es: $\left\{0, r_1, r_2, \frac{1}{64}\right\}$, de donde se obtiene:

$$\begin{aligned} p(x) = {} & 1.000000006650307888975275342165 \\ & + 0.693139524006400218385965810520 87x \\ & + 0.241531652033306183094246649723 68x^2 \end{aligned} \qquad (13)$$

$$|E| = 6.650307888975275342165041277482 2e-09$$

Se encuentra que $E_M = 6.6537181857915e-09$, por lo tanto, se acepta este $p(x)$ como polinomio minimax. En la Figura 22 se puede observar la gráfica del error que se obtiene en cada una de las iteraciones del algoritmo y el efecto de la disminución del error máximo.

Figura 22. Error de 2^x en el rango (0,1/64) con un polinomio de grado dos



Fuente: Elaboración propia

5.2.2. CUANTIFICACIÓN DE LOS COEFICIENTES

El polinomio minimax $p(x)$ de grado dos que se obtiene de la aproximación de una función $f(x)$ tiene sus coeficientes a_0, a_1 y a_2 para los términos de grado cero, uno y dos respectivamente con su longitud total, sin embargo, para poder almacenarlos en tablas, se tienen que redondear a un número de bits específico t, p y q para obtener C_0, C_1 y C_2 respectivamente. Siguiendo el método de Oberman et al. en [62], los coeficientes se pueden cuantificar o truncar de la siguiente manera:

1. Usando la aproximación original minimax con los coeficientes de longitud total, se redondea el coeficiente de grado uno con p bits para obtener C_1.
2. Se computa a_2' usando la expresión (14) con los coeficientes de grado uno a_1 y dos a_2 de longitud total, posteriormente se redondea a_2' a q bits para obtener C_2, compensando parte de los efectos del redondeo de C_1.
3. Se usa la aproximación minimax para computar el coeficiente de grado cero a_0' teniendo en cuenta los valores anteriores de C_1 y C_2 cómo se expresa en (15), y se redondea a t bits para obtener C_0. Este proceso incluye la compensación por haber redondeado C_1 y C_2.

$$a_2' = a_2 + (a_1 - C_1) \cdot 2^m \tag{14}$$

$$f'(x) = f(x_i) - C_1 x_i - C_2 x_i^2 \tag{15}$$

Ejemplo 4: Redondear los coeficientes obtenidos en el Ejemplo 3 usando los valores de $t = 25$, $p = 15$, $q = 11$ y $m = 6$.
Sol. Del análisis realizado anteriormente, expresado en (13) se tiene,

$$a_0 = 1.000000006650307888975275342165$$

$$a_1 = 0.693139524006400218385965810520 87$$

$$a_2 = 0.241531652033306183094246649723 68$$

Siguiendo el primer paso se tiene,

$$C_1 = 2^{-p} \cdot round(a_1 \cdot 2^p)$$
$$= 2^{-15} \cdot round(0.693139524006400218385965810520 87 \cdot 2^{15})$$

$$C_1 = 0.693145751953125_{10} = 0.101100010111001_2$$

Luego,

$$a_2' = a_2 + (a_1 - C_1) \cdot 2^m = 0.24113306344292$$

$$C_2 = 2^{-q} \cdot round(a_2' \cdot 2^q) = 2^{-11} \cdot round(0.24113306344292 \cdot 2^{11})$$

$$C_2 = 0.2412109375_{10} = 0.00111101110_2$$

Para realizar algoritmo minimax se escoge inicialmente $X = \left\{0, \frac{1}{64}\right\}$, luego se tiene:

$$(a_0') + (-1)^i E = 2^{x_i} - C_1 x_i - C_2 x_i^2$$

Con la realización del algoritmo y una primera iteración se obtiene:

$$a_0' = 1.000000001834234602461945971799 9$$

$$|E| = 3.342349314574967315773470900285 9e - 09$$

Por lo tanto,

$$C_0 = 2^{-t} \cdot round(a_0' \cdot 2^t) = 1$$

5.2.3. TAMAÑO DE TABLAS OBTENIDO

En la Tabla 4 se muestra un resumen de los parámetros empleados para aproximar las funciones elementales con el método descrito anteriormente, el valor de m, el número de bits de la cuantificación de los coeficientes t, p y q, y el tamaño correspondiente de la tabla de almacenamiento para cada función.

Tabla 4. Resumen de los parámetros de diseño y el tamaño de las tablas

Función	***rango***	***m***	***t,p,q***	***Tamaño de la tabla***
$sin(F)$	[0,1)	6	27,18,13	$2^6 \cdot (27 + 19 + 12) = 3.625Kb$
$cos(F)$		6	27,18,13	$2^6 \cdot (28 + 18 + 11) = 3.5625Kb$
$1/\sqrt{X}$	(1,2)	7	26,16,10	$2^7 \cdot (25 + 15 + 9) = 6.125Kb$
$1/\sqrt{2X}$		7	26,16,10	$2^7 \cdot (24 + 15 + 9) = 6Kb$
$\log_2(X)$	(1,2)	7	26,15,10	$2^7 \cdot (26 + 16 + 10) = 6.5Kb$
$\log_2(X)/(X-1)$		6	26,15,10	$2^6 \cdot (25 + 15 + 9) = 3.0625Kb$
2^X	(0,1)	6	25,15,11	$2^6 \cdot (25 + 16 + 10) = 3.1875Kb$
$1/X$	(1,2)	7	26,16,10	$2^7 \cdot (25 + 16 + 10) = 6.375Kb$
$\sqrt{X}$	(1,2)	6	25,15,11	$2^6 \cdot (24 + 13 + 8) = 2.8125Kb$
$\sqrt{2X}$		6	25,15,11	$2^6 \cdot (25 + 13 + 9) = 2.9375Kb$
	Total			$44.1875Kb$

Fuente: Elaboración propia

La elección del valor de t, p y q usados en la cuantificación de los coeficientes y la cantidad de subintervalos dados por m se tomaron referenciados de los trabajos [61] y [62]. El número de bits fraccionarios de los coeficientes no coincide en todos los casos con el número de bits almacenados ya que algunos coeficientes permiten eliminar algunos bits iniciales, que quedan como bits implícitos, mientras que, en otros casos, es necesario almacenar el bit de la parte entera, como se observa en la Tabla 5.

Tabla 5. Bits implícitos en los coeficientes polinomiales de las funciones

$\boldsymbol{Función}$	$\boldsymbol{C_0}$	$\boldsymbol{C_1}$	$\boldsymbol{C_2}$
$sin(F)$	$\pm 0.xxx \ldots x$	$+x.xxx \ldots x$	$-0.0xxx \ldots x$
$cos(F)$	$+x.xxx \ldots x$	$-0.xxx \ldots x$	$-0.01xxx \ldots x$
$1/\sqrt{X}$	$+0.1xxx \ldots x$	$-0.0xxx \ldots x$	$+0.0xxx \ldots x$
$1/\sqrt{2X}$	$+0.10xxx \ldots x$	$-0.0xxx \ldots x$	$+0.0xxx \ldots x$
$\log_2(X)$	$+0.xxx \ldots x$	$+x.xxx \ldots x$	$-0.xxx \ldots x$
$\log_2(X)/(X-1)$	$+1.0xxx \ldots x$	$-0.xxx \ldots x$	$+0.0xxx \ldots x$
2^X	$+1.xxx \ldots x$	$+x.xxx \ldots x$	$+0.0xxx \ldots x$
$1/X$	$+0.1xxx \ldots x$	$-0.xxx \ldots x$	$+0.xxx \ldots x$
$\sqrt{X}$	$+1.0xxx \ldots x$	$+0.01xxx \ldots x$	$-0.000xxx \ldots x$
$\sqrt{2X}$	$+1.xxx \ldots x$	$+0.10xxx \ldots x$	$-0.00xxx \ldots x$

Fuente: Elaboración propia

Con los tamaños de las tablas definido, se puede establecer el tamaño del bus para cada uno de los coeficientes, así: 29 bits para C_0 dado por el tamaño de la memoria de $cos(F)$ de 28 bits más el bit de signo; para C_1 es de 20 bits dado por el tamaño de la memoria de $sin(F)$ de 19 bits, más el bit de signo y para C_2 es de 14 bits dado por el tamaño de la memoria de $sin(F)$ con 12 bits, el bit de signo y un bit de alineación para ajustar esta tabla con las tablas de $log_2(X)$ y $1/X$.

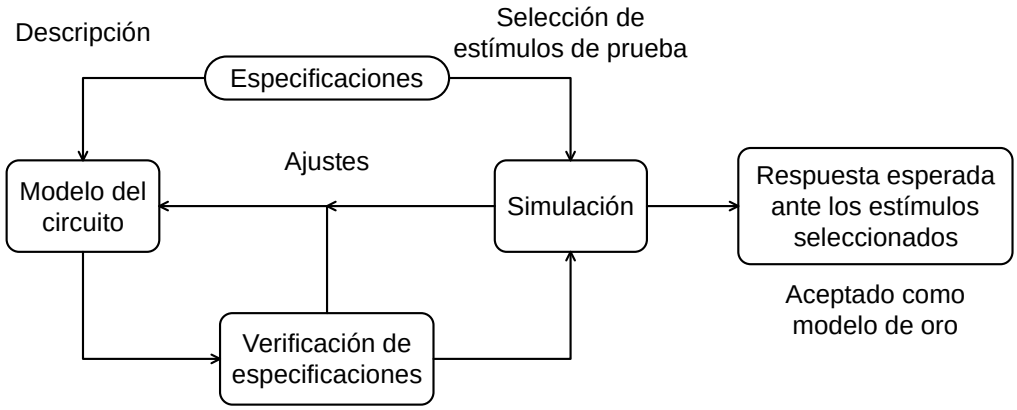
5.3. DESCRIPCIÓN FUNCIONAL EN SOFTWARE

En el proceso de diseño de cualquier sistema digital, la verificación juega un papel importante al evaluar el diseño del hardware final ya que permite comparar los resultados de su comportamiento con las especificaciones originales y encontrar las mejores configuraciones de diseño mediante simulaciones [69]. Además, cada vez es más importante la implementación de procedimientos que garanticen la calidad de los dispositivos puesto que es más fácil hacer corrección de defectos antes que después de su fabricación. Un elemento de hardware que presente fallas en la aplicación para la cual fue diseñado puede implicar el retiro por completo de los dispositivos que lo usen en el mercado provocando pérdidas de tiempo y dinero, y si se usa en aplicaciones críticas, puede implicar heridas o pérdida de vidas humanas.

Existen diversos métodos de verificación, uno de ellos se conoce como verificación funcional. Este método consiste en desarrollar modelos de software que emulen el comportamiento de un circuito o sistema digital y se conocen como modelos de oro (*Golden Models*). Estos modelos emulan el funcionamiento del circuito objetivo, pero no necesariamente sus características arquitectónicas, eléctricas y de tiempo [70]. Son útiles en un entorno de verificación como un marcador o predictor y tienen una gran flexibilidad a la hora de hacer la verificación puesto que su entorno se implementa en software y permiten corroborar, mediante simulación, que el diseño o arquitectura propuesta cumple con las especificaciones exigidas sin tener en cuenta algunos detalles específicos de implementación y ocupando menos tiempo de simulación que los diseños en HDLs (*Hardware Description Languages*) [71]. El prototipo virtual se puede implementar en software que se ejecute en una computadora de uso general y se pueden seguir los siguientes pasos para su desarrollo:

1. Definir las especificaciones del circuito o sistema.
2. Usar las especificaciones como punto de partida para desarrollar un prototipo virtual del modelo del circuito.
3. Depurar el prototipo hasta que esté disponible para una evaluación exhaustiva.
4. Refinar las especificaciones y el prototipo hasta que se esté satisfecho (ver Figura 23).

Figura 23. Proceso para implementar un modelo de oro



Fuente: Adaptado de [70]

Una vez el prototipo virtual se ha evaluado minuciosamente con una cantidad considerable de pruebas, se considera como el modelo de oro, el cual se supone que está libre de errores funcionales para poder seguir con el diseño físico del sistema. En este trabajo se creó un modelo de oro para la SFU basada en aproximaciones polinomiales de grado dos con segmentación uniforme, se eligió Octave como lenguaje y entorno de programación ya que es un software libre, por

lo que no tiene restricciones de distribución académica y permite de forma sencilla monitorear el estado de las variables a nivel de bits, con lo cual se puede depurar de forma detallada la descripción. Como especificaciones iniciales, se establecieron las siguientes:

1. La representación de los datos de entrada y de salida se tiene que hacer con el formato IEEE-754 (ver sección 4.6.)
2. Los valores de los coeficientes de las tablas se obtienen directamente en su representación binaria con los tamaños de los buses para C_0, C_1 y C_2 establecidos en la sección 5.2.3.
3. El valor de X_l^2 se debe truncar en la posición 2^{-27}.
4. Los datos de prueba tienen que estar dentro de los rangos seleccionados en la sección 5.1.

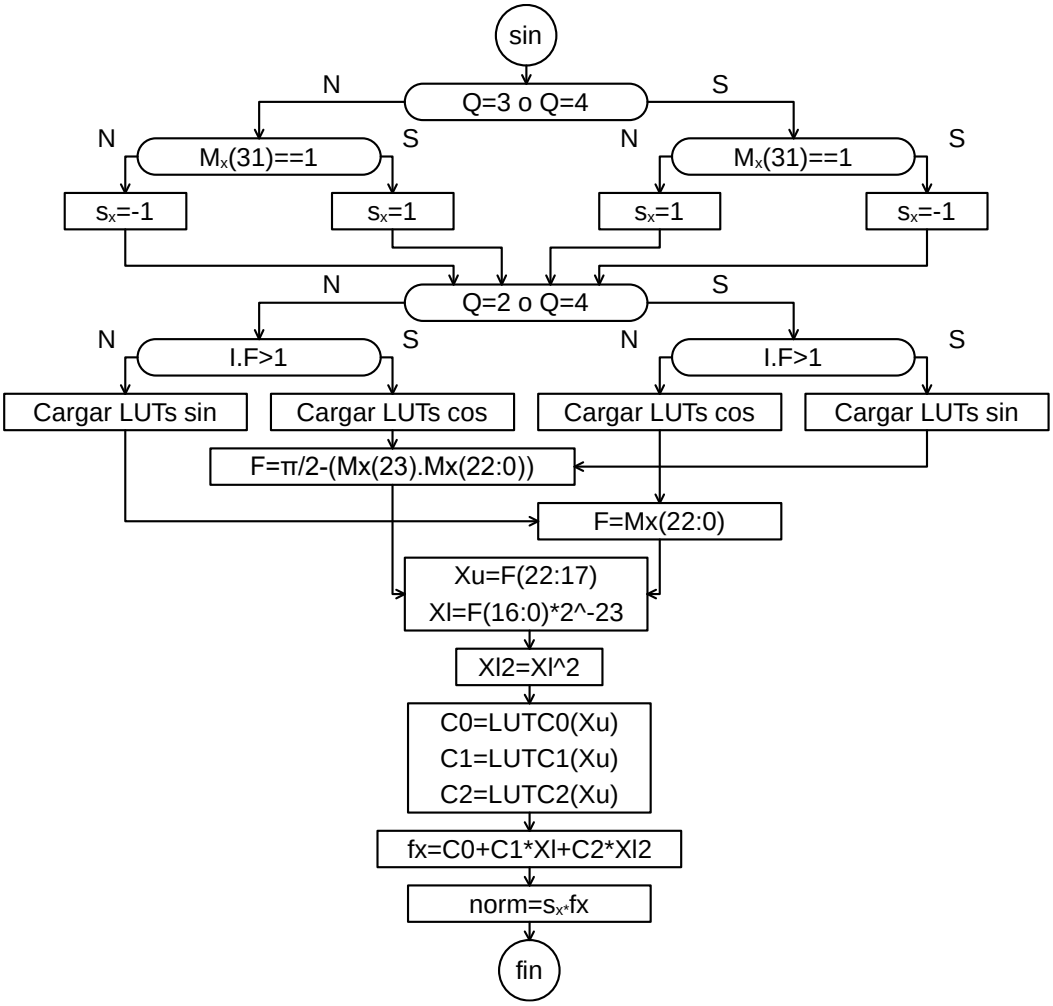
Para las funciones con $m = 6$, X_l tiene m ceros a la izquierda (por lo que su tamaño es de 17 bits) y, por consiguiente, el tamaño de X_l^2 tiene por lo menos $2m$ ceros a la izquierda. El valor de X_l^2 se puede truncar para disminuir su tamaño el cual puede ser menor a los 17 bits de X_l [62], por lo que se escogió para X_l^2 un tamaño de 15 bits, lo que es equivalente a truncar su valor en la posición 2^{-27} y como su valor es pequeño, el error que aporta este truncamiento no es muy grande. Para las funciones con $m = 7$, el tamaño real de X_l^2 es de 13 bits. Después de realizar el proceso descrito en la Figura 23, se obtuvo el modelo de oro de la SFU, aunque su descripción se realizó en un solo código, se puede dividir en sub rutinas dependiendo de la función a computar. A continuación, se presenta la subrutina para cada una de las funciones, explicando las consideraciones que se tuvieron en cuenta a medida que se realizaron los ajustes de la implementación de cada una.

5.3.1. SENO/COSENO

El diagrama de flujo implementado para la función seno se puede observar en la Figura 24, inicialmente, dependiendo del cuadrante y del signo del número de entrada se establece el signo que antecede la evaluación del polinomio de aproximación, esta información se encuentra dentro del dato de entrada a la SFU como se observa en la Figura 20. Posteriormente, dependiendo del cuadrante y la magnitud del dato de entrada se define que tablas de coeficientes se van a cargar y si es necesario hacer el pre procesamiento de restar $\pi/2$ al dato de entrada para mantenerlo dentro del rango $[0,1)$ (el valor de $\pi/2$ se trunca a 23 bits para mantener el tamaño de F). Ahora, se separa la parte fraccionaría del dato F en X_u y en X_l tomando los m MSB que para esta función es 6 (ver Tabla 4), se calcula el valor de X_l^2 (truncado en la posición 2^{-27}), se cargan los valores de las tablas en la

posición de X_u y se evalúa el polinomio de aproximación; finalmente el resultado de la aproximación polinomial se multiplica por el signo considerado inicialmente.

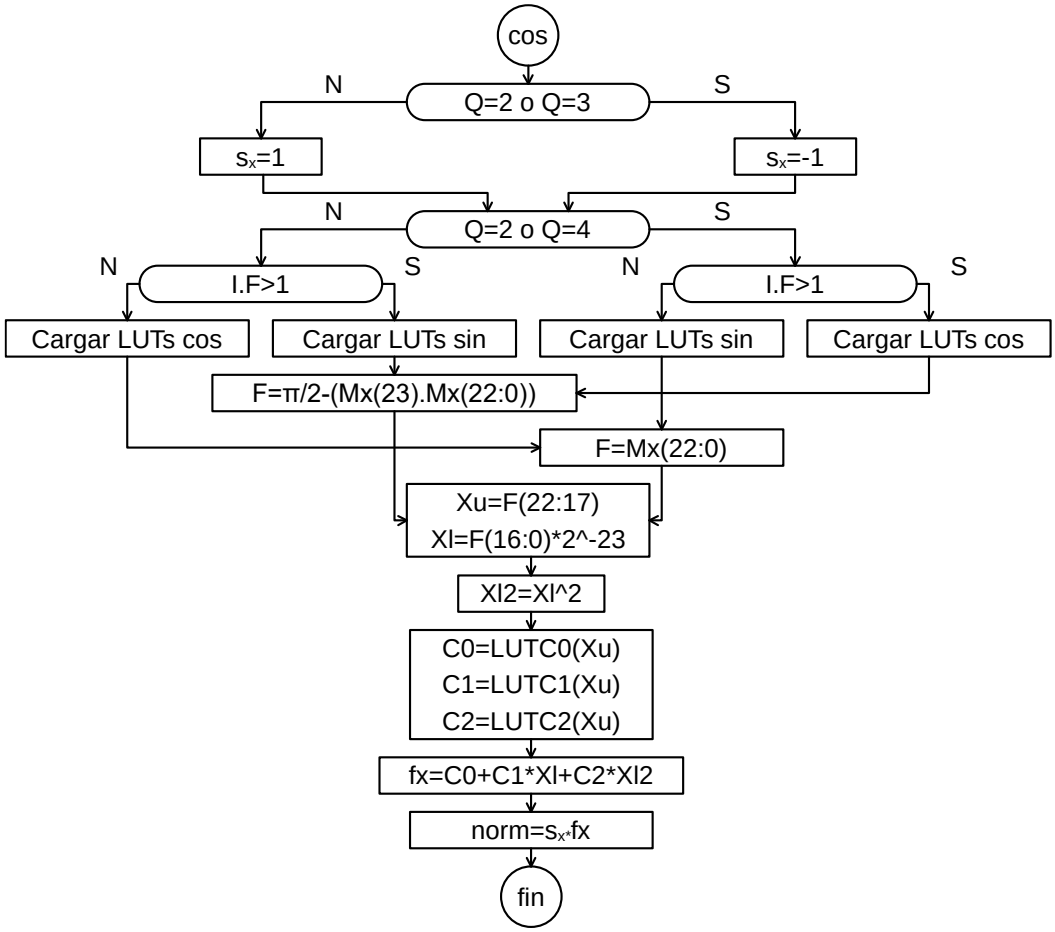
Figura 24. Diagrama de flujo para la función seno



Fuente: Elaboración propia

Para la función coseno se implementó el esquema mostrado en la Figura 25, inicialmente, dependiendo del cuadrante del número de entrada se establece el signo que antecede la evaluación del polinomio de aproximación. Posteriormente, dependiendo del cuadrante y la magnitud del dato de entrada se define que tablas de coeficientes se van a cargar y si es necesario hacer el pre procesamiento de restar $\pi/2$ al dato de entrada para mantenerlo dentro del rango $[0,1)$. El resto del procesamiento es equivalente al proceso usado con la función seno.

Figura 25. Diagrama de flujo para la función coseno

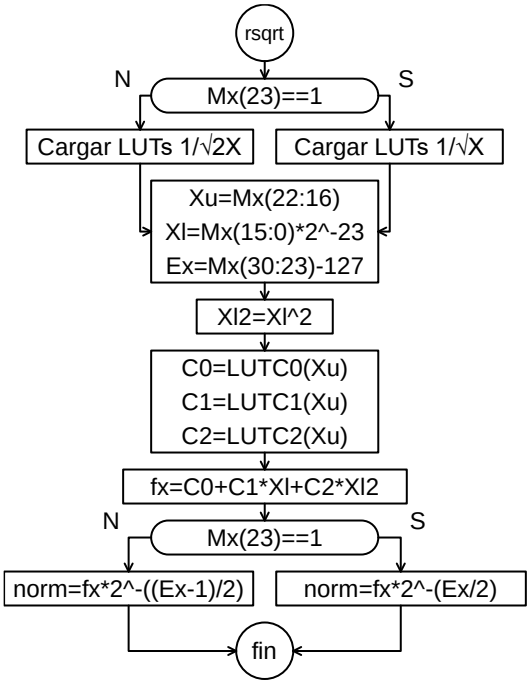


Fuente: Elaboración propia

5.3.2. RECIPROCO DE LA RAÍZ CUADRADA

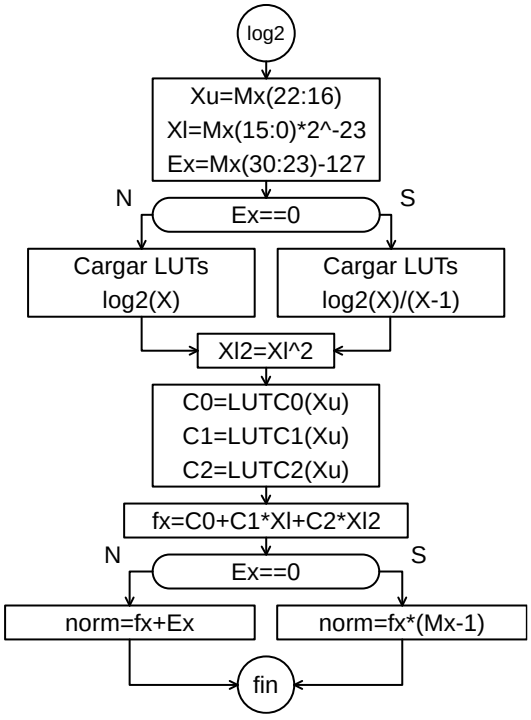
La descripción en software realizada para la función reciproco de la raíz cuadrada se puede expresar con el diagrama mostrado en la Figura 26. Dado un número en su representación binaria, se puede saber si es par o impar verificando el valor de su LSB, si es 0 el número será par, si es 1 será impar. Conforme a la representación en el formato IEEE-754, el bit en la posición 23 del dato de entrada M_x indica si el exponente E_x es par o impar, pero como se encuentra normalizado, se realiza el análisis inverso y el LSB de E_x indica si el número ingresado es par cuando es 1 e impar cuando es 0; teniendo esto en cuenta, se cargan las tablas y se normaliza la evaluación del polinomio de aproximación según como se expresa en (6).

Figura 26. Diagrama de flujo para la función reciproco de la raíz cuadrada



Fuente: Elaboración propia

Figura 27. Diagrama de flujo para la función logaritmo en base 2



Fuente: Elaboración propia

5.3.3. LOGARITMO BASE 2

El esquema del diagrama de flujo para el logaritmo en base 2 mostrado en la Figura 27, muestra inicialmente la descomposición del dato de entrada M_X en X_u, X_l y E_x. El valor de E_x se desnormaliza y se carga una de las dos tablas correspondientes a las dos funciones descritas con anterioridad en la expresión (7), posteriormente se calcula el valor de X_l^2 y se obtienen los coeficientes de las tablas en la posición designada por X_u. Después de computar el polinomio de aproximación se normaliza el resultado dependiendo del valor de E_X.

5.3.4. EXPONENCIAL BASE 2

Como se observa en la Figura 28, para evaluar la función exponencial base 2, lo primero que se hace es cargar las tablas correspondientes a esta función, posteriormente se extrae de M_X el valor de X_u, X_l, se calcula X_l^2 y se obtienen los coeficientes de las tablas en la posición designada por X_u. Hay que tener en cuenta que la representación del dato de entrada está en punto fijo $I.F$ con signo (ver Figura 21), así que después de computar el polinomio de aproximación, se verifica si el número de entrada es negativo con el bit en la posición 30 de M_x (el MSB de I), si el número es negativo, se hace complemento a dos y se multiplica por menos 1 para obtener el valor decimal de I, de lo contrario, únicamente se aísla su valor de M_x y se normaliza como se expresa en (8).

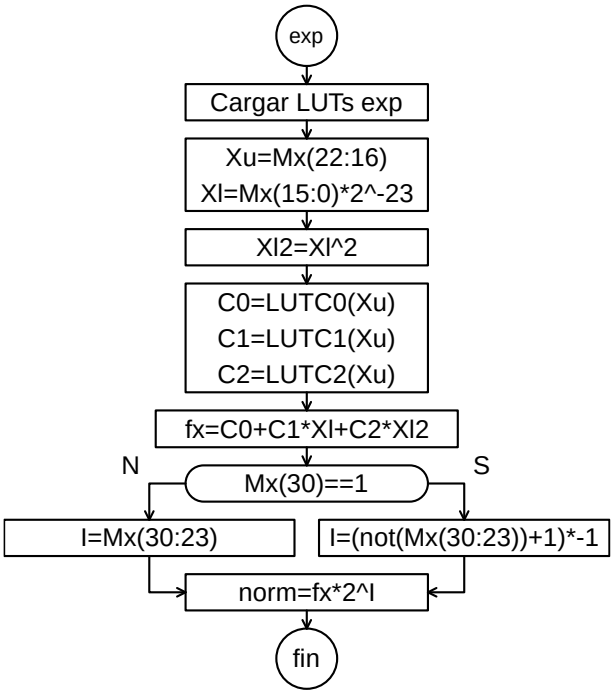
5.3.5. RECIPROCO

El diagrama de flujo de la función reciproco se muestra en la Figura 29, para esta función solo se requiere de una tabla y no hay que hacer ningún tipo de desnormalización ni consideración adicional. Se extrae de M_X el valor de X_u, X_l, s_x, se calcula X_l^2 y se obtienen los coeficientes de las tablas en la posición designada por X_u. Posteriormente se computa el polinomio de aproximación y por último se normaliza el resultado como se expresa en (9).

5.3.6. RAÍZ CUADRADA

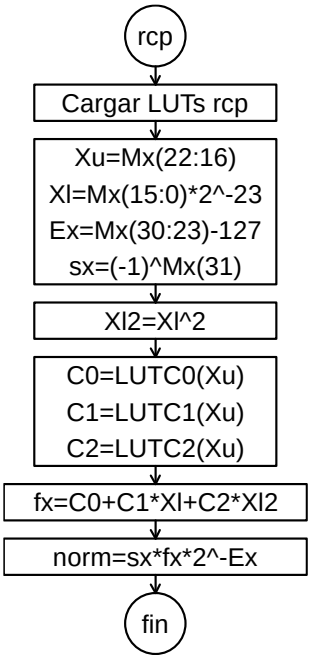
El diagrama de flujo para la raíz cuadrada se muestra en la Figura 30, cuenta con dos tablas, la selección de una u otra al igual que con la función reciproco de la raíz cuadrada depende de si el exponente es par o impar, así que el diagrama de flujo es equivalente cambiando únicamente la normalización del resultado de la evaluación del polinomio de aproximación como se expresa en (10).

Figura 28. Diagrama de flujo para la función exponencial base 2



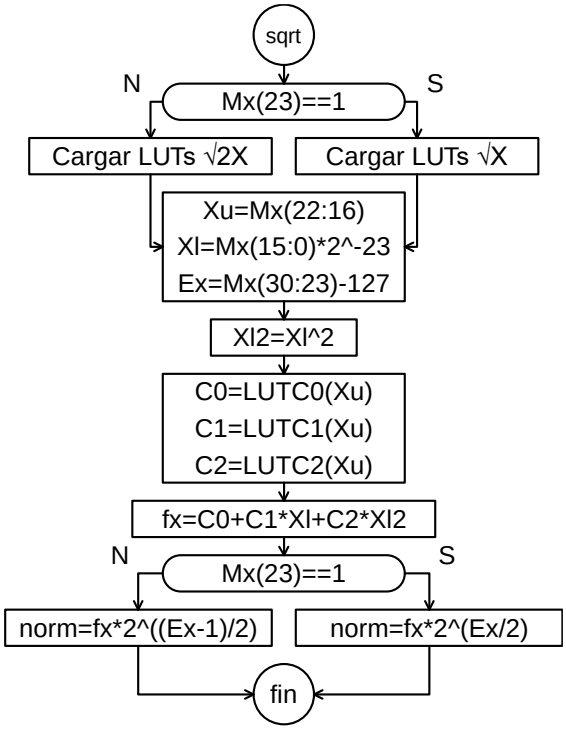
Fuente: Elaboración propia

Figura 29. Diagrama de flujo para la función reciproco



Fuente: Elaboración propia

Figura 30. Diagrama de flujo para la función raíz cuadrada

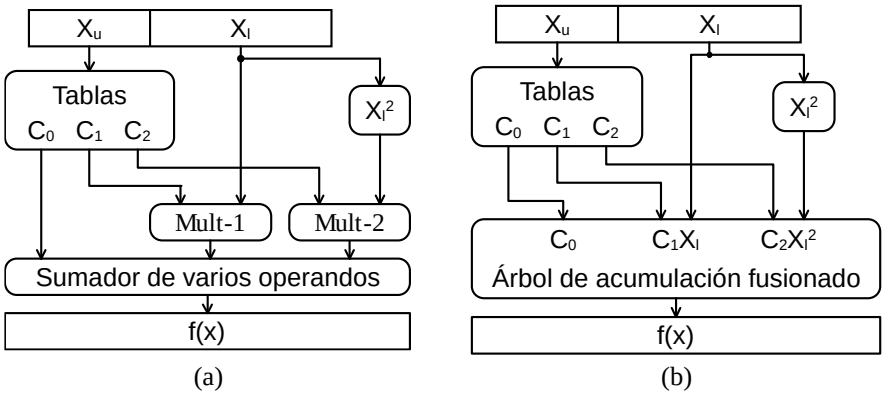


Fuente: Elaboración propia

5.4. MODELADO DE LA MICROARQUITECTURA DE LA SFU

Después de realizar la descripción del modelo Golden y comprobar que el diseño propuesto a nivel de arquitectura corresponde con las especificaciones iniciales, se procede a implementar el circuito con los detalles adicionales que el modelo Golden no considera tales como: buses de interconexión, multiplexación, unidades de control, etc. La estructura general de la arquitectura para evaluar el polinomio de aproximación se puede ver en la Figura 31a, para acelerar la evaluación del polinomio se hace uso de una unidad especializada para generar el valor de X_l^2, una codificación en base 4 con signo (*SD radix-4 recoding*) en los multiplicadores para la generación de los productos parciales de C_1X_l y $C_2X_l^2$, y un sumador multi operando donde se acumulan los productos parciales junto con el coeficiente C_0. La generación de los productos parciales junto con el sumador se une en un solo bloque denominado árbol de acumulación fusionado (ver Figura 31b).

Figura 31. Diagrama PPA de grado dos: a) general, b) optimizado



Fuente: Adaptado de [36]

A continuación, en la sección 5.4.1, se presenta el detalle de la implementación en hardware de las tablas de coeficientes, posteriormente en la sección 5.4.2 se explica en detalle el diseño de la unidad de elevación al cuadrado especializada para el cálculo de X_l^2. En la sección 5.4.3 se presenta el diseño del circuito de codificación de Booth junto con el sumador multi operando y, por último, en la sección 5.4.4 se presenta el diseño del circuito de normalización del resultado de la evaluación del polinomio a el formato IEEE-754.

5.4.1. TABLAS

Para las 7 funciones a implementar en la SFU se tiene la asignación de la Tabla 6, que codifica la selección de los coeficientes dependiendo de la señal sel_op. Sin embargo, como se mostró en la sección 5.1, hay funciones que requieren de dos tablas de coeficientes, donde se selecciona una u otra dependiendo de algunos factores como la paridad del exponente o de si este es igual a cero, por lo que se tiene que adicionar un circuito para poder hacer la diferenciación de las tablas a la hora de seleccionar una función a computar.

Tabla 6. Codificación de la selección de las funciones

$\boldsymbol{Función}$	$\boldsymbol{sel\_op}$
$\sin(X)$	000
$\cos(X)$	001
$1/\sqrt{X}$	010
$\log_2 X$	011
2^X	100
$1/X$	101
$\sqrt{X}$	110

Fuente: Elaboración propia

Para la diferenciación de las tablas de una misma función hay que tener en cuenta cuando el exponente es igual a cero o a 127 por la normalización (ver sección 4.6), para esto, se tiene el bit Z que se genera con un comparador entre el exponente del número de entrada y el valor 127 (ver Figura 32a). Para diferir entre un número par o impar, se toma el LSB del exponente E_0. Con esto, se plantea la codificación mostrada en la Tabla 7, con lo cual, solo hay que agregar una lógica adicional de selección para distinguir entre las diferentes tablas de los coeficientes polinomiales correspondientes a una misma función.

Tabla 7. Codificación de la selección de las tablas de coeficientes

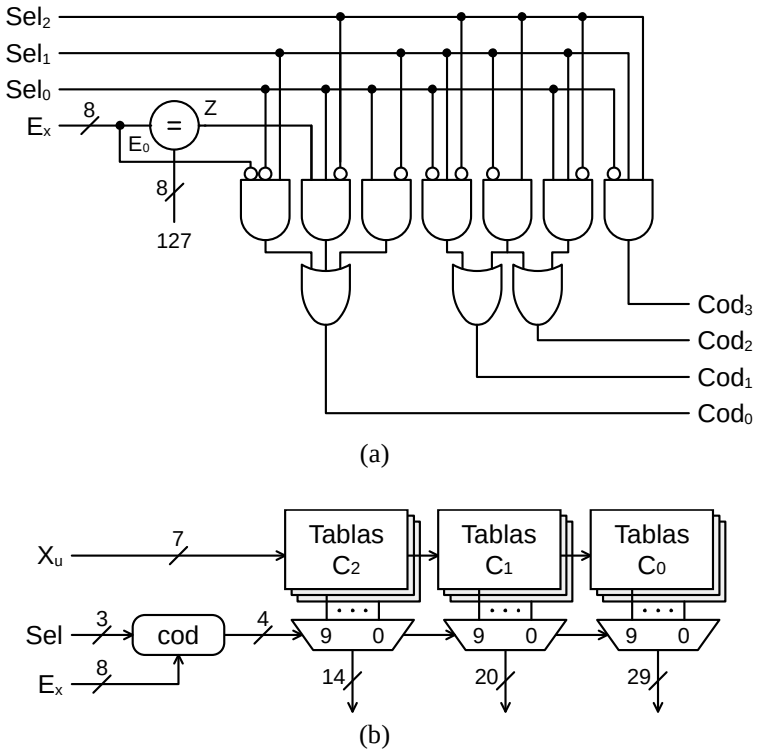
$\boldsymbol{Función}$	$\boldsymbol{sel\_op}$	$\boldsymbol{Z}$	$\boldsymbol{E_0}$	$\boldsymbol{Codificación}$
$\sin(X)$	000	X	X	0000
$\cos(X)$	001	X	X	0001
$1/\sqrt{X}$	010	X	1	0010
$1/\sqrt{2X}$	010	X	0	0011
$\log_2 X$	011	0	X	0100
$\log_2(X)/(X-1)$	011	1	X	0101
2^X	100	X	X	0110
$1/X$	101	X	X	0111
$\sqrt{X}$	110	X	1	1000
$\sqrt{2X}$	110	X	0	1001

Fuente: Elaboración propia

$$
\begin{aligned}
Cod_0 &= \overline{Sel}_1 Sel_0 + Sel_1 \overline{Sel}_0 \bar{E}_0 + \overline{Sel}_2 Sel_0 Z \\
Cod_1 &= \overline{Sel}_2 Sel_1 \overline{Sel}_0 + Sel_2 \overline{Sel}_1 \\
Cod_2 &= \bar{S}el_2 Sel_1 Sel_0 + Sel_2 \overline{Sel}_1 \\
Cod_3 &= Sel_2 Sel_1 \overline{Sel}_0
\end{aligned}
\tag{16}
$$

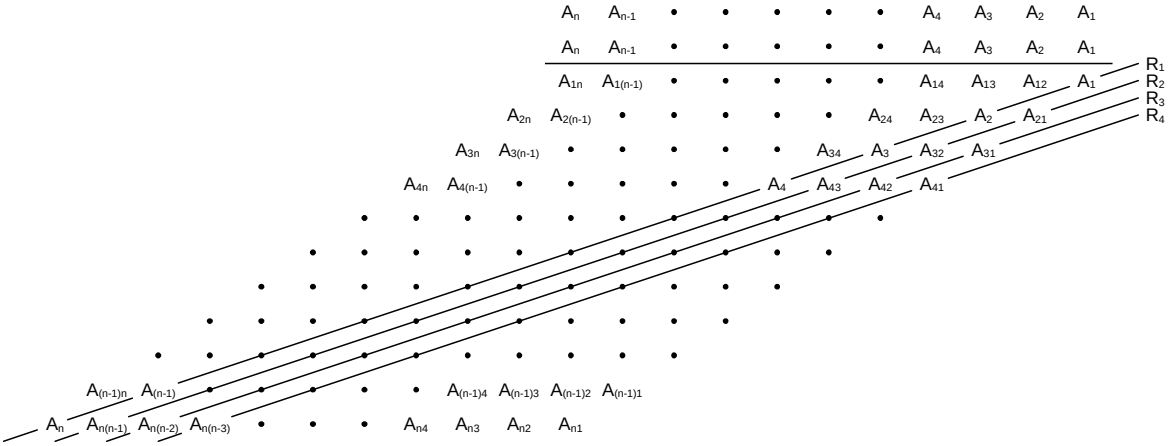
Los coeficientes de las tablas se direccionan por X_u y se multiplexan dependiendo de la codificación asignada en la Tabla 7 (así como la conexión física de las tablas a la entrada de los multiplexores), seleccionando así una de las 10 tablas de coeficientes que se tienen para las diferentes funciones de aproximación. En las tablas calculadas para $m = 6$, el LSB de X_u no se tiene en cuenta para el direccionamiento, pero se tiene en cuenta para X_l. La codificación expresada en (16) se puede ver en la Figura 32a, este circuito combinacional se representa como el bloque "cod" en el esquema general del bloque de tablas mostrado en la Figura 32b.

Figura 32. Selección de coeficientes: (a) Codificación, (b) Esquema general



Fuente: Elaboración propia

Figura 33. Matriz de multiplicación



Fuente: Adaptado de [72]

5.4.2. UNIDAD DE ELEVACIÓN AL CUADRADO ESPECIALIZADA

Para computar X_l^2 se usaron dos técnicas para reducir el área y el retraso: primero, se consideraron los ceros iniciales de X_l (cuando $m = 6$ los 6 primeros bits de X_l son cero), con lo cual, el tamaño del dato de entrada a la unidad queda de 17 bits y segundo, se usó de la técnica de multiplicación de cuarto de cuadrado (*quarter-square multiplying technique*), con base en el trabajo presentado por Totadri y Dhruba en [72], donde se exponen ciertas propiedades sobre la matriz de multiplicación de la unidad de elevación al cuadrado (ver Figura 33) para reducir su tamaño. Para más información acerca de esta técnica se puede referir al trabajo realizado por los autores.

Propiedad 1: La anti diagonal R_1 divide la matriz en dos mitades simétricas y, por lo tanto, la matriz entera puede ser representada por la anti diagonal y todos los elementos de una de las dos mitades de la matriz desplazados una posición a la izquierda. Por esto, los elementos de las subdiagonales R_2, R_3 y R_4 se puede reorganizar como filas R_2', R_3' y R_4' con respecto a R_1 como se observa en la Figura 34.

Propiedad 2: Para la sub matriz mostrada en la Figura 34, los acarreos en las columnas impares comprendidas entre la columna 5 y la $(2 - n3)$ tienen una expresión lógica similar dado que, las columnas son similares. Los elementos de esta sub matriz se pueden resumir para formar el resultado con una misma expresión lógica para todas las columnas impares entre la columna 5 y la $(2 - n3)$. De forma similar se puede hacer esto para las columnas pares entre la columna 6 y la $(2n - 2)$.

Figura 34. Generación de la fila L

Fuente: Adaptado de [72]

Con base en la propiedad dos se genera la columna L, la cual se puede expresar de forma genérica como se expresa en (17). Así, la generación de la columna L solo requiere de compuertas and, or y not. Con esta combinación y aplicando la Propiedad 1, se ordenan los demás elementos de la matriz original respecto a la columna L para formar la matriz reducida que se muestra en la Figura 35, con $k = (n + 6)/2$ y $j = (n - 4)/2$.

$$L_1 = A_1$$
$$L_2 = 0$$
$$L_3 = A_2\bar{A}_1$$
$$L_4 = A_1(A_2 \oplus A_3)$$

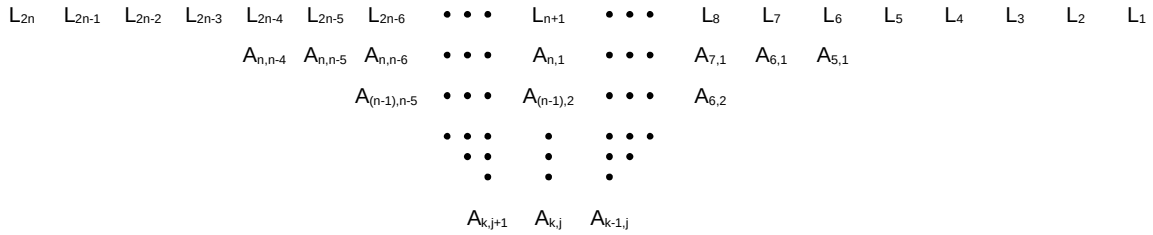
$$L_i = A_{\frac{i+1}{2}}\bar{A}_{\frac{i-1}{2}}\left(\bar{A}_{\frac{i-3}{2}} + \bar{A}_{\frac{i+3}{2}}\right) + A_{\frac{i-3}{2}}\left(A_{\frac{i+1}{2}} \oplus A_{\frac{i+3}{2}}\right), i = 5,7,9,\dots,(2n-3) \quad (17)$$

$$L_{i'} = A_{\frac{i'-2}{2}}\left(A_{\frac{i'}{2}} \oplus A_{\frac{i'+2}{2}}\right) + A_{\frac{i'-4}{2}}A_{\frac{i'}{2}}A_{\frac{i'+2}{2}}, i' = 6,8,10,\dots,(2n-2)$$

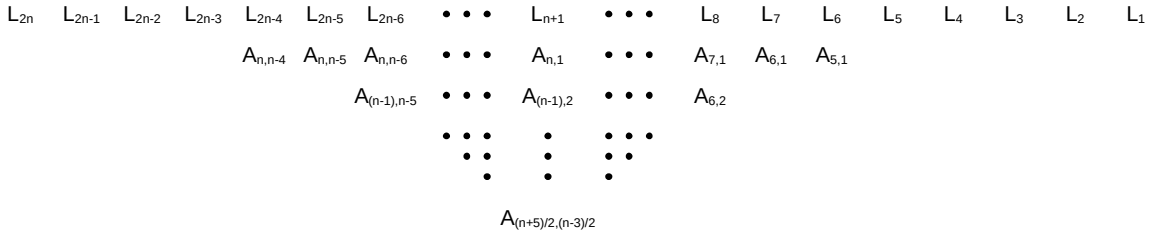
$$L_{2n-1} = A_n(\bar{A}_{n-1} + A_{n-2})$$

$$L_{2n} = A_n A_{n-1}$$

Figura 35. Matriz reducida para: (a) n par, (b) n impar



(a)



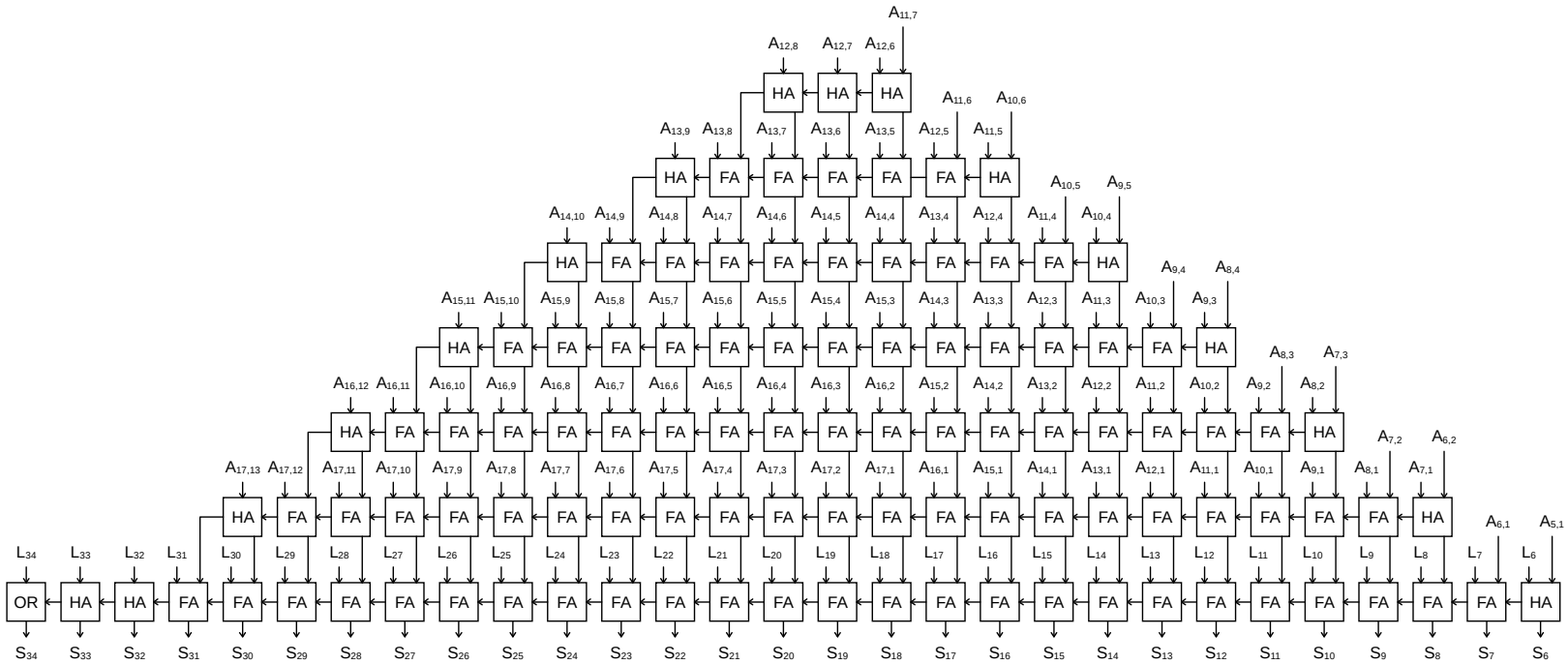
(b)

Fuente: Adaptado de [72]

Al elevar al cuadrado un número de n bits se genera como máximo un número de $2n$ bits, por lo que la columna $2n$ no puede generar acarreo, por esto, la columna del sumador final se puede reemplazar por una compuerta or. Cuando se computa una función con $m = 7$, quedan solo 16 bits de X_l, así que simplemente se concatena un cero a la izquierda. La estructura generada para la unidad de elevación al cuadrado, con el análisis descrito anteriormente, se presenta en la Figura 36.

Figura 36. Unidad de elevación: (a) matriz reducida, b) esquema

(a)



(b)

Fuente: Elaboración propia

5.4.3. ÁRBOL DE ACUMULACIÓN FUSIONADO

Después de la lectura de los coeficientes C_0, C_1 y C_2 de las tablas y del cómputo de X_l^2, la evaluación del polinomio de aproximación (4) implica dos operaciones básicas: la operación de los productos parciales de C_1X_l y $C_2X_l^2$, y la suma de estos junto con el coeficiente C_0. En consecuencia, hay dos formas de acelerar la evaluación del polinomio, reduciendo el número de productos parciales y/o acelerando su acumulación o suma. Usando la codificación de Booth (ver sección 4.7) se puede reducir el número de productos parciales que se tienen que acumular. Los valores de C_1 y C_2 pueden ser tanto positivos como negativos (ver Tabla 5) y al obtenerse en el formato signo magnitud $SM.F$ (el MSB es el bit de signo) se facilita su manipulación para obtener los productos parciales. Tomando los valores de X_l y X_l^2 como los multiplicadores, el bit de signo de los multiplicandos C_1 y C_2 afectan el resultado de los productos parciales ya que si la codificación genera el producto parcial $-2MO$, el signo negativo del multiplicando invierte la lógica de la codificación de Booth de tal manera que se generaría el producto parcial $+2MO$. De este modo, agregando este bit de signo a la codificación se tiene la Tabla 8, donde se generan tres salidas para la codificación: Sim cuando el producto parcial que se genera es simple, es decir, es el mismo valor del multiplicando, Dob cuando se generan los productos parciales dobles $\pm 2MO$ y Neg cuando el producto parcial es negativo.

Tabla 8. Codificación Booth en base 4 con multiplicandos negativos

S	mr_i	mr_{i-1}	mr_{i-2}	$Operación$	Sim	Dob	Neg
0	0	0	0	0	0	0	0
0	0	0	1	MO	1	0	0
0	0	1	0	MO	1	0	0
0	0	1	1	$2MO$	0	1	0
0	1	0	0	$-2MO$	0	1	1
0	1	0	1	$-MO$	1	0	1
0	1	1	0	$-MO$	1	0	1
0	1	1	1	-0	0	0	1
1	0	0	0	-0	0	0	1
1	0	0	1	$-MO$	1	0	1
1	0	1	0	$-MO$	1	0	1
1	0	1	1	$-2MO$	0	1	1
1	1	0	0	$2MO$	0	1	0
1	1	0	1	MO	1	0	0
1	1	1	0	MO	1	0	0
1	1	1	1	0	0	0	0

Fuente: Elaboración propia

Simplificando las expresiones lógicas para cada una de las señales de salida, se obtienen las funciones booleanas expresadas en (18).

$$\begin{gathered} \mathrm{Sim} = mr_{i-1} \oplus mr_{i-2} \\ Dob = \overline{mr_i} mr_{i-1} mr_{i-2} + mr_i \overline{mr_{i-1}}\, \overline{mr_{i-2}} \\ Neg = s \oplus mr_i \end{gathered} \tag{18}$$

Ahora, se tiene que generar un circuito selector con base en la codificación de las señales de la Tabla 8. Se puede tomar las señales Sim y Dob para establecer la generación del producto parcial simple MO o doble MO' y posteriormente, usar la señal Neg junto con una compuerta xor para generar la negación del producto parcial.

Tabla 9. Selector Booth

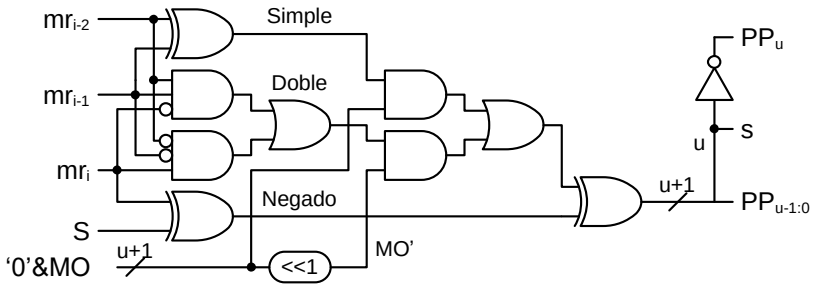
Sim	Dob	mo_i	mo'_i	PP_i
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
1	0	1	1	1
1	1	0	0	X
1	1	0	1	X
1	1	1	0	X
1	1	1	1	X

Fuente: Elaboración propia

$$PP_i = Sim \cdot mo_i + Dob \cdot mo'_i \tag{19}$$

En la Figura 37 se puede ver el circuito generador de productos parciales con base en las expresiones (18) y (19). La generación de los productos parciales negativos en complemento a uno se realiza con la señal de codificación Neg y una compuerta xor, con lo cual, cuando esta vale uno, el dato se niega. El bit de signo s ubicado en la posición u (siendo u la cantidad de bits del multiplicando), se toma para la corrección de los productos parciales negativos (ver Figura 16) y este mismo pero negado completa la generación del producto parcial modificado para evitar la extensión de signo (ver Figura 17).

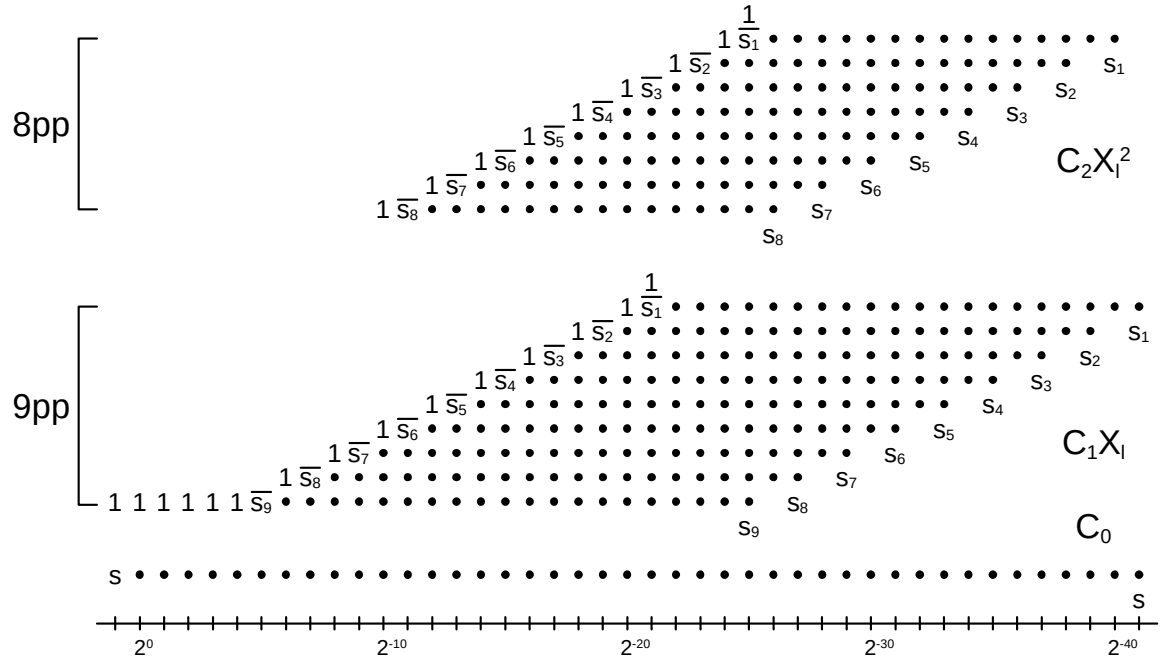
Figura 37. Generador de productos parciales con codificación Booth



Fuente: Elaboración propia

Como se comentó en la sección 5.3, el tamaño de X_l es de 17 bits y el de X_l^2 es de 15 bits, con lo cual, haciendo uso de la codificación de Booth se tienen 9 productos parciales para C_1X_l y 8 para $C_2X_l^2$. En la Figura 38 se puede ver la matriz de los productos parciales, añadiendo a esta, el valor de C_0, con lo cual, se tienen que sumar en total 18 operandos $(8 + 9 + 1)$.

Figura 38. Matriz final de acumulación



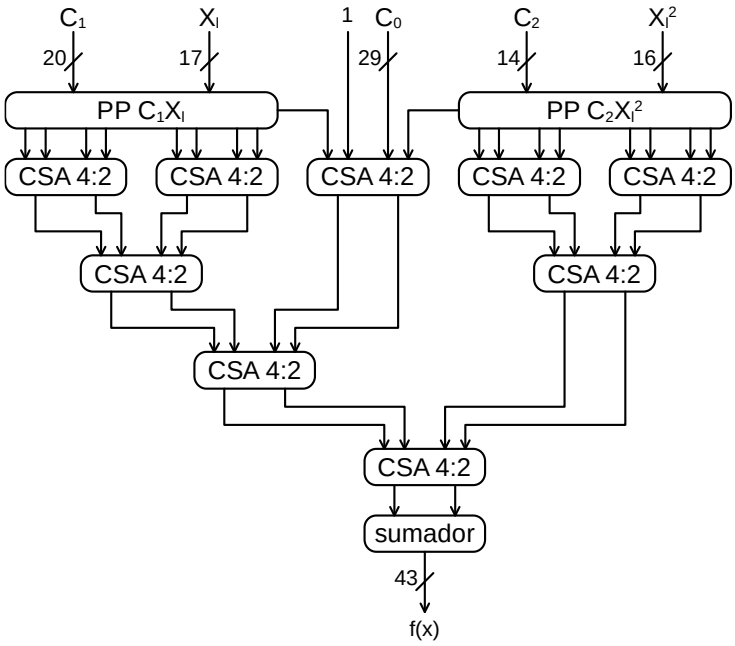
Fuente: Elaboración propia

Para C_0 se agregan 14 ceros a la derecha para ajustar su tamaño a la matriz de acumulación y se genera de forma condicional la representación en complemento a dos (complemento a uno con su ajuste) de los coeficientes negativos. Cuando $m = 6$, X_l tiene por lo menos 6 ceros después del punto, con lo cual, manteniendo el bit de ajuste para los productos parciales dobles, el bit de signo del último

producto parcial de C_1X_l ocupa la posición 5 después de la coma, para X_l^2 se tiene por lo menos 12 bits en cero, con lo cual, el bit de signo del último producto parcial de $C_2X_l^2$ se ubica en la posición 11 después de la coma.

La acumulación de los productos parciales se realizó con un árbol de 4 niveles de sumadores carry-save $4{:}\,2$ (ver sección 4.8), en el primer nivel se tienen dos sumadores para los $8PP$ de $C_2X_l^2$, otros dos para $8PP$ de los $9PP$ de C_1X_l y un tercer sumador para el 9PP de C_1X_l, los ajustes de la representación de números negativos de la generación de los PP de C_1X_l y $C_2X_l^2$ que al estar intercalados, se toman como un solo vector, el coeficiente C_0 y el bit de ajuste de extensión de signo del primer producto parcial de $C_2X_l^2$ que se ubica sobre $\bar{s}_1$ (ver Figura 38), ya que al pre computar los unos de ajuste, este no se podía ubicar en los otros sumadores. Al final se usa un sumador común para obtener el resultado final de la acumulación, como se puede observar en la Figura 39.

Figura 39. Esquema del árbol de acumulación fusionado



Fuente: Elaboración propia

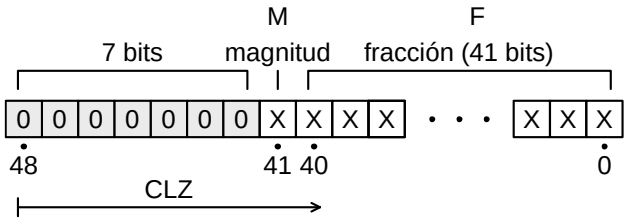
5.4.4. NORMALIZACIÓN DEL RESULTADO

Para la representación final en el formato IEEE-754 del resultado que se genera en el árbol de acumulación fusionado se deben tener en cuenta dos aspectos importantes: el primero es la reconstrucción del resultado expuesto en la sección 5.1 y el segundo, es que el resultado del árbol de acumulación fusionado puede

ser menor a uno, con lo cual hay que ajustar el punto decimal para la correcta representación del formato como se expone en sección 4.6. El ajuste del punto decimal consiste básicamente en contar la cantidad de ceros consecutivos que hay desde el MSB hasta el primer digito distinto de cero para posteriormente correr el punto decimal tal cantidad de posiciones. El corrimiento del punto decimal indica el ajuste que se debe hacer al exponente del formato IEEE-754, al cual se le suma este que también indica la posición desde la cual se toma el valor de la mantisa X.

El tamaño del dato de salida del árbol de acumulación fusionado es de 43 bits y se encuentra en representación de punto fijo usando el formato signo magnitud $SM.F$, 1 bit de signo, 1 bit de la parte entera y 41 bits de la parte fraccionaria. Sin embargo, para la normalización requerida por la función $\log_2(M_x)$, como se expresa en (7), se tiene que sumar el valor del exponente E_x, con lo cual, se tienen que añadir 7 ceros a la izquierda para ajustar el tamaño de bits de la parte entera, por lo que se tendría el formato mostrado en la Figura 40. Para el ajuste del punto decimal se usa un circuito especial que calcula la cantidad de ceros consecutivos que hay desde el MSB hasta el primer digito distinto de cero (*LZC - Leading-Zero Counter*), con lo cual, el ajuste del punto decimal se obtiene realizando la operación $7 - LZC$ y así, cuando el primer digito distinto de cero se encuentre a la izquierda del punto decimal el ajuste será positivo y cuando este a la derecha será negativo.

Figura 40. Formato de ajuste para árbol de acumulación fusionado



Fuente: Elaboración propia

Para las funciones $sin(M_x)$ y $cos(M_x)$, el resultado de la aproximación es directamente la evaluación del polinomio como se demostró en la sección 5.1.1, con lo cual, el exponente de salida depende solamente del ajuste del punto decimal y la normalización del formato IEEE-754 como se muestra en la expresión (20). Para la función $\log_2(M_x)$ se tiene el post procesamiento expresado en (7), el cual afecta al resultado generado en la evaluación del polinomio, pero el exponente desaparece de la expresión, con lo cual, la normalización de esta función también depende solamente del ajuste del punto decimal.

$$sin(M_x)\,/cos(M_x)/\,log_2(M_x) \rightarrow E_{x_o} = 7 - LZC + 127 \qquad (20)$$

Para la función $\sqrt{M_x}$ y $1/\sqrt{M_x}$, la normalización del exponente de salida E_{x_o} se tiene que realizar de dos formas diferentes dependiendo de si el exponente de entrada desnormalizado $(E_{x_i} - 127)$ es par o impar como se explicó en las secciones 5.1.2 y 5.1.6. Posteriormente a esto, se hace el ajuste del punto decimal y la normalización correspondiente al formato IEEE-754 sumando 127. Sin embargo, el ajuste del punto decimal solo se aplica a la función $1/\sqrt{M_x}$, ya que para la función $\sqrt{M_x}$, el coeficiente C_0 tiene parte real y los valores de $C_1 X_l$ y $C_2 X_l^2$ son muy pequeños, con lo cual, el resultado del árbol de acumulación fusionado siempre va a tener la parte entera, por lo cual no hay que ajustar el punto decimal.

$$\frac{1}{\sqrt{M_x}} \rightarrow \begin{cases} E_{x_o} = -\dfrac{E_{x_i} - 127}{2} + 7 - LZC + 127, si\ (E_{x_i} - 127)\ es\ par \\ E_{x_o} = -\dfrac{E_{x_i} - 127 - 1}{2} + 7 - LZC + 127, si\ (E_{x_i} - 127)\ es\ impar \end{cases} \tag{21}$$

$$\sqrt{M_x} \rightarrow \begin{cases} E_{x_o} = \dfrac{E_{x_i} - 127}{2} + 127, si\ (E_{x_i} - 127)\ es\ par \\ E_{x_o} = \dfrac{E_{x_i} - 127 - 1}{2} + 127, si\ (E_{x_i} - 127)\ es\ impar \end{cases} \tag{22}$$

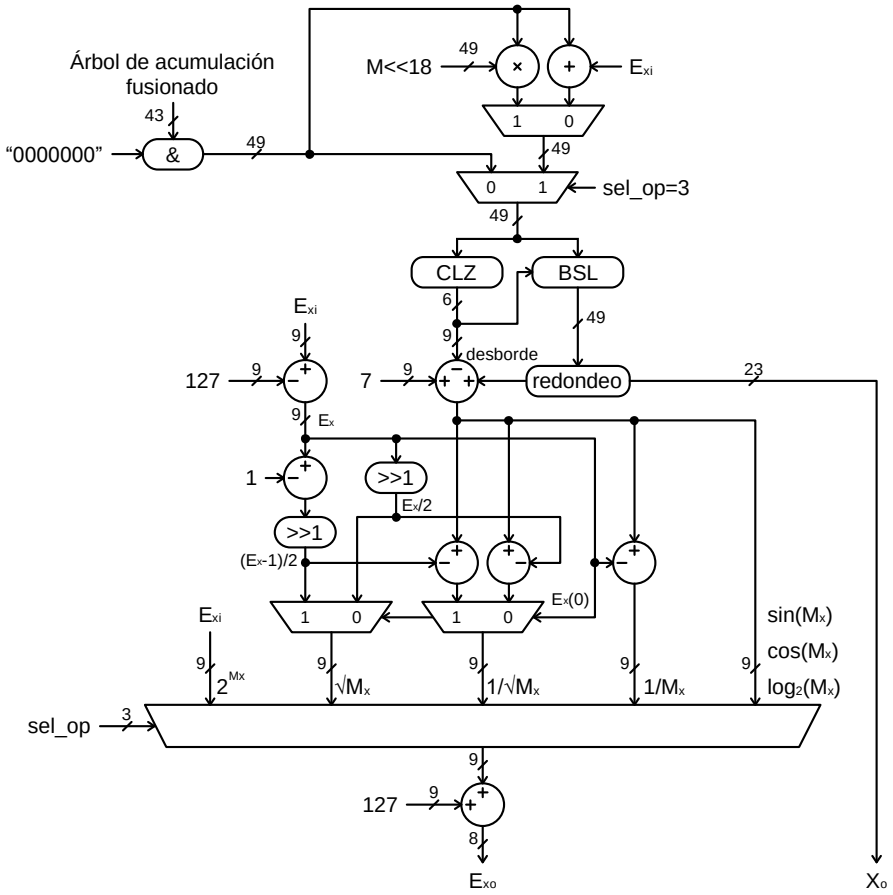
La función 2^{M_x} es un caso especial en el que el formato de entrada se encuentra en punto fijo con signo $I.F$ (ver sección 5.1.4), en donde la parte fraccionaria F se evalúa en el polinomio de aproximación y la parte entera I correspondiente al exponente E_{x_o} se ajusta con la normalización del formato IEEE-754 como se muestra en la expresión (23). Por último, para la función $1/M_x$ solo se tiene el ajuste del punto decimal y la normalización del formato IEEE-754 como se muestra en la expresión (24).

$$2^{M_x} \rightarrow E_{x_o} = I + 127 \tag{23}$$

$$\frac{1}{M_x} \rightarrow E_{x_o} = -(E_{x_i} - 127) + 7 - LZC + 127 \tag{24}$$

La implementación de las anteriores expresiones (20) a (24) se muestra en la Figura 41. El dato de salida del árbol de acumulación fusionado se concatena con 7 ceros en los MSBs según el formato de la Figura 40 y posteriormente se hace el ajuste especial para la función $log_2(M_x)$ según lo expresado en (7). El ajuste del punto decimal para la mantisa X se hace con un circuito de corrimiento a la izquierda (*BSL – Barrel Shifter Left*), el cual desplaza el dato tantas posiciones como indique el contador de ceros CLZ. Posteriormente a esto, se realiza un redondeo del resultado al siguiente par según las normas de redondeo del formato IEEE-754.

Figura 41. Esquema de reconstrucción y normalización del resultado



Fuente: Elaboración propia

El redondeo se hace siguiendo lo expresado en la Tabla 10, siendo el bit 2^{-23} el LSB de la mantisa, y los bits 2^{-24} y 2^{-25} los bits de guarda. Cuando RD es 1, se suma uno al valor de la mantisa y en caso de que ocurra un desborde, este se suma al valor del contador de ceros para ajustar el valor final del exponente.

Tabla 10. Redondeo al par más cercano

2^{-23}	2^{-24}	2^{-25}	RD
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Fuente: Elaboración propia

6. VALIDACIÓN Y EVALUACIÓN FUNCIONAL DE LA SFU

En esta etapa el modelo de hardware descrito en HDL debe ser verificado antes de realizar la síntesis del circuito. La validación consiste en realizar una simulación lógica del modelo de hardware en HDL y comparar los resultados con el modelo Golden teniendo en cuenta las mismas condiciones de operación. Se considera entonces, que el modelo de hardware es correcto si hay coherencia con el modelo Golden. Para esto se generaron 20.000 estímulos aleatorios para cada una de las funciones entre diferentes rangos dependiendo de la función como se muestra en la Tabla 11. La evaluación de los estímulos se realiza con un banco de pruebas (*Test Bench*) realizado en el mismo lenguaje HDL usado en la descripción de la SFU. Las funciones $sin(x)$, $cos(X)$ y 2^x requieren un ajuste especial como se explicó en la sección 5.1, para lo cual se hace uso de un bloque de reducción de rango (*RRO – Range Reduction Operation*) ubicado dentro del banco de pruebas de forma externa a la SFU ya que como se mencionó en esta misma sección, no hace parte del diseño. El banco de pruebas toma como entrada un archivo csv (*Comma Separated Value*) con los 20.000 datos aleatorios y genera un mismo archivo de salida con la evaluación de la SFU. Estos mismos datos se evalúan con el modelo Golden después de que pasan por el bloque RRO y se comparan con los resultados obtenidos en el banco de pruebas para validar su correcto funcionamiento (ver Figura 42a). El reporte indica bajo que estímulos la unidad HDL difiere con el modelo de oro para verificar su funcionamiento, una vez los datos coinciden en su totalidad, se considera que el modelo está finalizado.

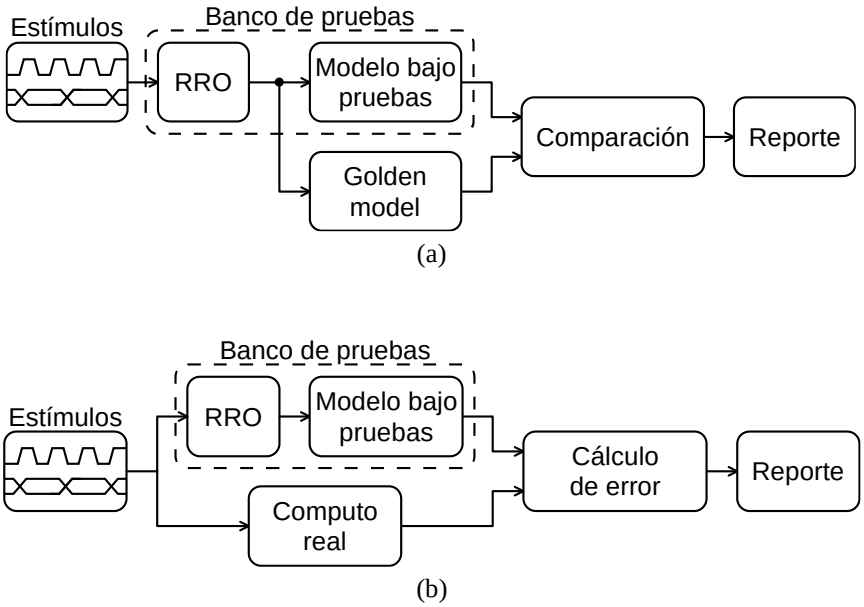
Tabla 11. Rangos de los estímulos de simulación

Función	***Rango***	
	mín	***máx***
$\sin(X)$	-148	148
$\cos(X)$	-148	148
$1/\sqrt{X}$	0	48
$\log_2 X$	0	48
2^X	-48	48
$1/X$	-48	48
$\sqrt{X}$	0	48

Fuente: Elaboración propia

Después de la verificación funcional, se toma el resultado del banco de pruebas para comparar su resultado con un cálculo más aproximado realizado en software, tomado como “computo real” (ver Figura 42b) y se calculan dos tipos de error: el error absoluto e_a que es la diferencia entre el valor real v_r y el valor que se obtuvo en la aproximación v_a., y el error relativo e_r, que es el cociente entre el error absoluto y el valor real. Las expresiones para hallar estos errores están expuestas en las ecuaciones (25) y (26) respectivamente.

Figura 42. Diagrama de bloques: (a) Verificación funcional, (b) Reporte de error



Fuente: Elaboración propia

$$e_a = v_r - v_a \tag{25}$$

$$e_r = \frac{v_r - v_a}{v_r} \tag{26}$$

Los resultados del cálculo del error de las diferentes funciones se pueden observar en la Tabla 12. Se presentan los errores máximos que se presentaron en cada una de las funciones, así como el promedio del error de los 20.000 estímulos usados en cada función para brindar una mejor perspectiva del comportamiento del error de las diferentes funciones.

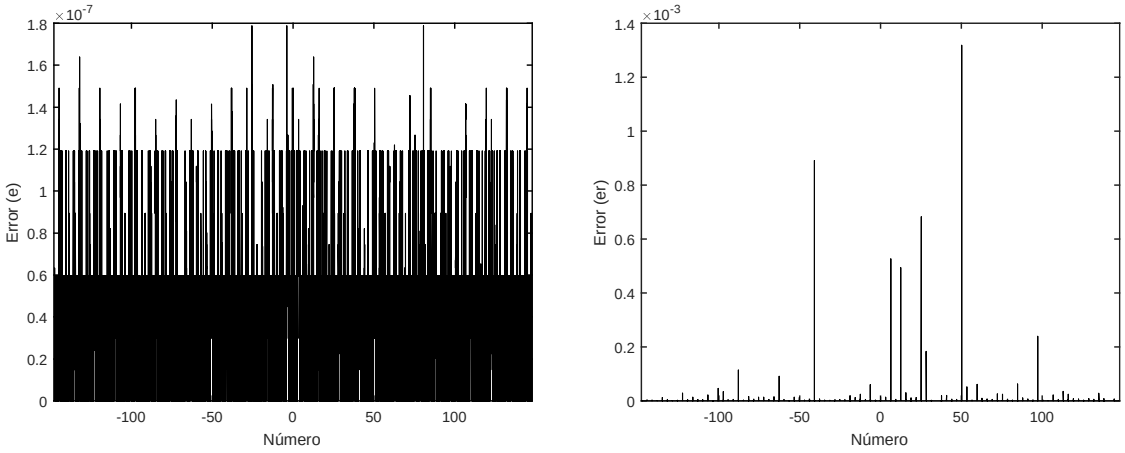
Tabla 12. Resultados del error de la SFU

Función	e_a		e_r	
	$máx$	$promedio$	$máx$	$promedio$
$sin(F)$	1.7881e-07	3.6555292e-08	1.3189e-03	4.85385e-07
$cos(F)$	1.7881e-07	3.6542122e-08	1.0963 e-03	3.6272863e-07
$1/\sqrt{X}$	9.5367e-07	1.00784e-08	2.3653e-07	3.5361765e-08
$\log_2(X)$	4.7684e-07	9.841754e-08	1.6863e-05	2.8862804e-08
2^X	41943040	268132.47	5.8970e-07	1.2264464e-07
$1/X$	1.5259e-05	7.1442687e-09	2.3747e-07	2.7702207e-08
$\sqrt{X}$	1.9073e-06	4.0537586e-07	4.7318e-07	3.5361765e-08

Fuente: Elaboración propia

La función 2^X presenta el error absoluto más grande, lo cual se presenta por la magnitud de los datos calculados ya que su error relativo es incluso más pequeño que el de otras funciones, donde la función $sin(X)$ presenta el error relativo más grande. En los gráficos de la Figura 43 a la Figura 49 se puede observar el comportamiento del error absoluto y relativo de cada una de las funciones en los rangos de prueba que se presentaron en la Tabla 11.

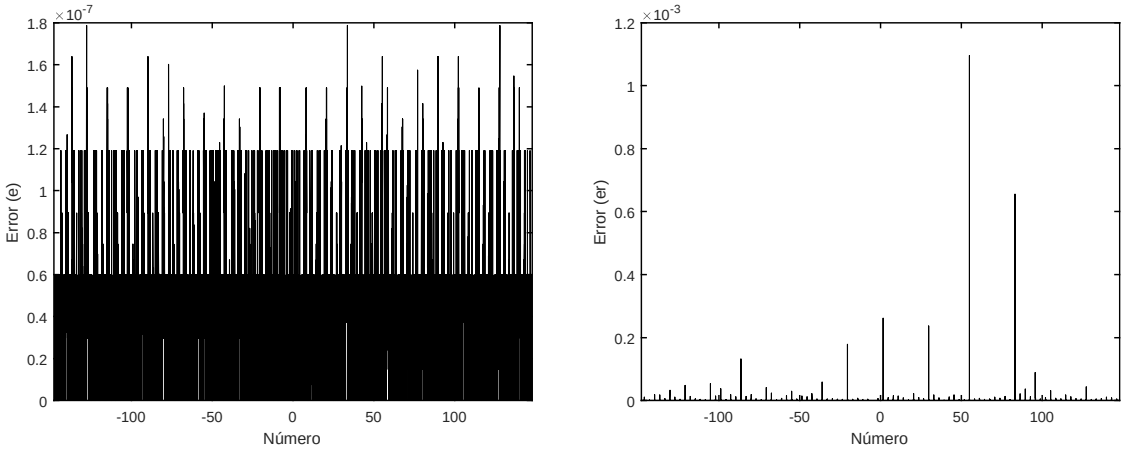
Figura 43. Error: (a), absoluto (b) relativo, de la función $sin(X)$



(a) (b)

Fuente: Elaboración propia

Figura 44. Error: (a), absoluto (b) relativo, de la función $cos(X)$



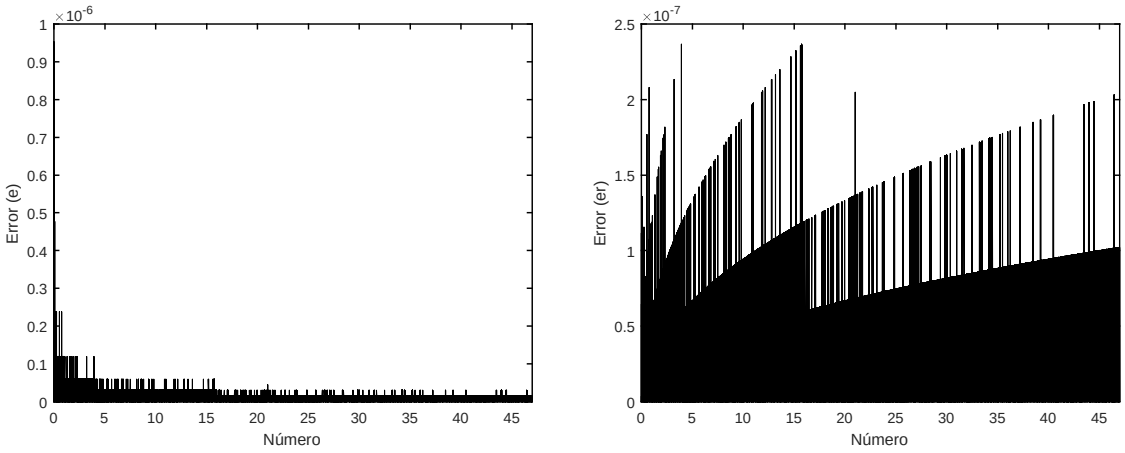
(a) (b)

Fuente: Elaboración propia

El error para las funciones $sin(X)$ y $cos(X)$ (ver Figura 43 y Figura 44) presentan un comportamiento similar, el error absoluto mantiene una distribución uniforme donde no se notan zonas con especial presencia de errores con magnitudes muy

grandes o muy pequeñas como se evidencia en otras funciones. El error relativo se mantiene pequeño con algunos puntos elevados.

Figura 45. Error: (a), absoluto (b) relativo, de la función $1/\sqrt{X}$

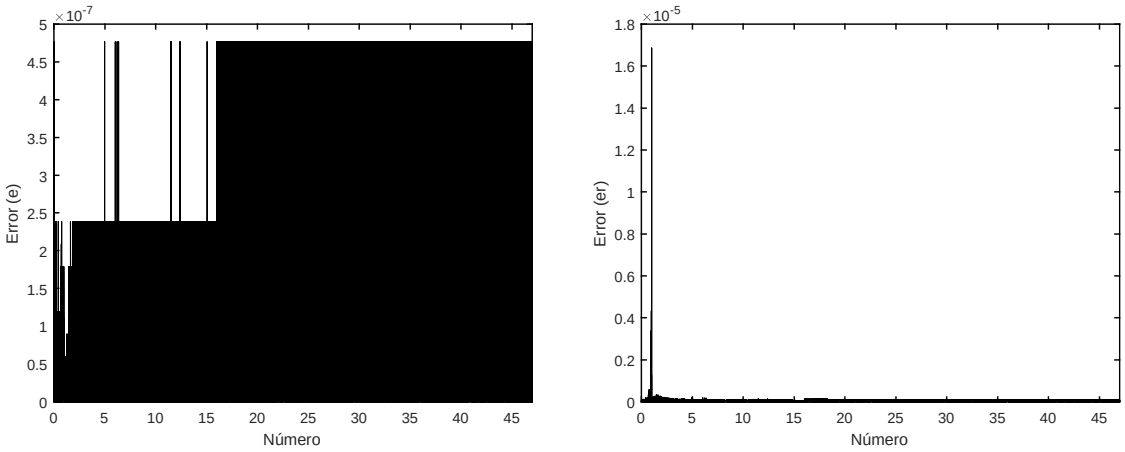


(a) (b)

Fuente: Elaboración propia

El error absoluto en la función $1/\sqrt{X}$ (ver Figura 45) presenta mayor magnitud con los datos cercanos a cero, mientras que el error relativo se mantiene constante en el restante rango de prueba. Para la función $\log_2(X)$ (ver Figura 46) el error absoluto se mantiene constante mientras que el error relativo presenta mayor magnitud con los datos cercanos a cero, pero para el resto de valores presenta una magnitud mucho más pequeña.

Figura 46. Error: (a), absoluto (b) relativo, de la función $\log_2(X)$

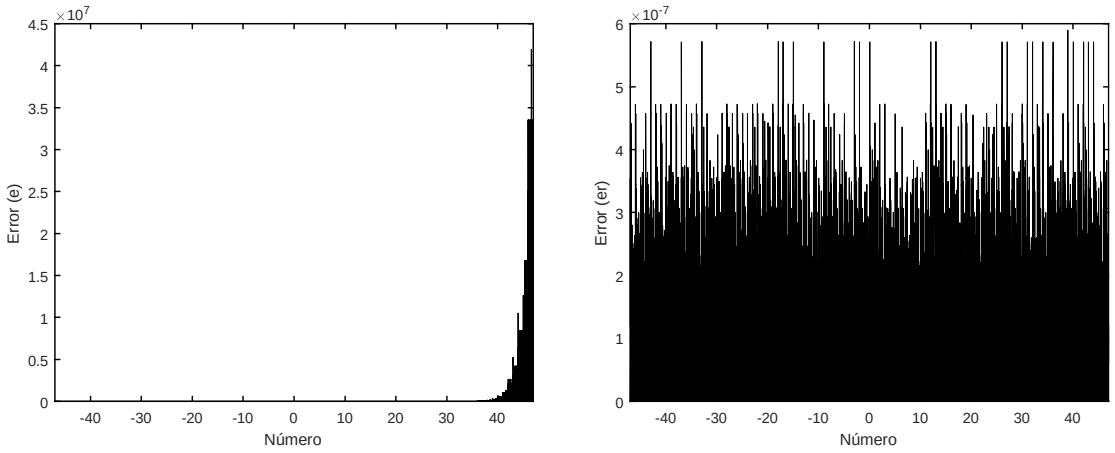


(a) (b)

Fuente: Elaboración propia

Para la función 2^X (ver Figura 47), el error absoluto va en incremento a medida que aumenta el tamaño del número de entrada ya que la magnitud de su resultado es cada vez más grande, sin embargo, el error relativo se mantiene con una distribución uniforme e indica que el porcentaje de error se mantiene constante.

Figura 47. Error: (a), absoluto (b) relativo, de la función 2^X

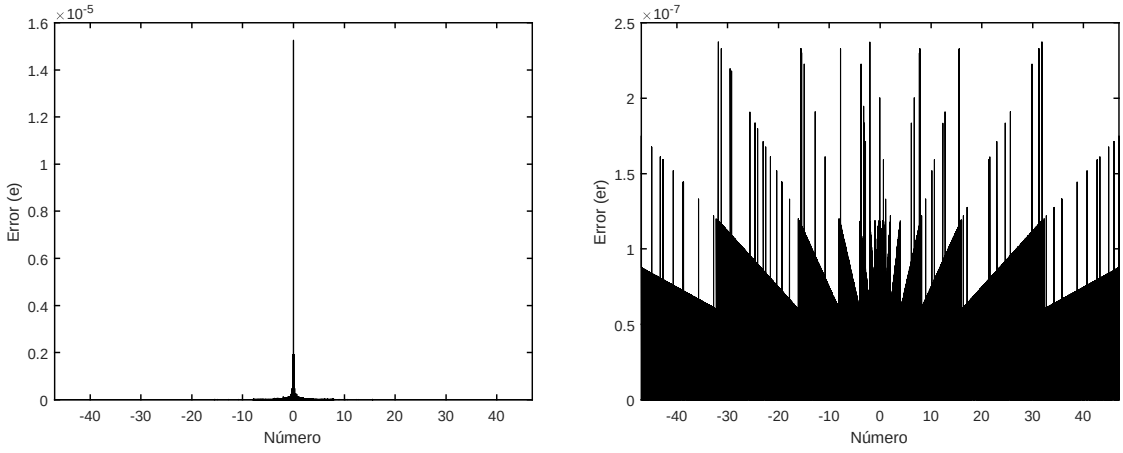


(a) (b)

Fuente: Elaboración propia

En la función $1/X$ (ver Figura 48) se observa que el error absoluto incrementa con los valores cercanos a cero, con un pico grande de error central, mientas que para los demás valores el error se mantiene mucho más pequeño, casi imperceptible en la gráfica. Por otra parte, el error relativo no presenta alguna zona con mayor o menor magnitud de error, sino que se mantiene constante.

Figura 48. Error: (a), absoluto (b) relativo, de la función $1/X$

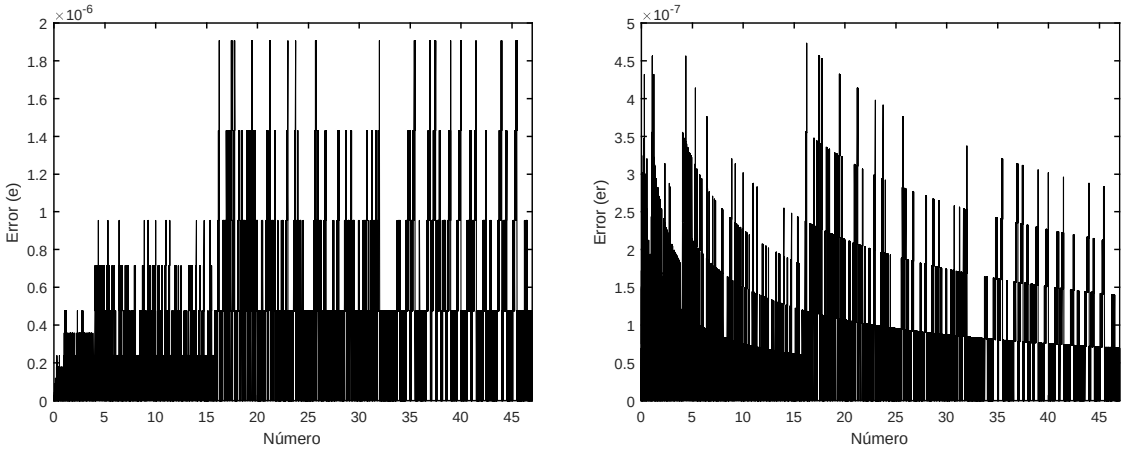


(a) (b)

Fuente: Elaboración propia

La función $\sqrt{X}$ (ver Figura 49) presenta un error absoluto de menor magnitud con los números comprendidos entre el cero y el quince, pero en general presenta una distribución uniforme al igual que el error relativo.

Figura 49. Error: (a), absoluto (b) relativo, de la función $\sqrt{X}$



(a) (b)

Fuente: Elaboración propia

7. DISEÑO Y EVALUACIÓN DE LA TOLERANCIA A FALLOS EN LA SFU

El uso generalizado de sistemas digitales en la mayoría de los aspectos de la vida cotidiana motiva la necesidad de que estos sistemas sean confiables, especialmente aquellos usados en aplicaciones donde las fallas pueden llegar a causar pérdidas financieras o humanas. Aunque el enfoque de diseño más conservador es el uso de materiales y procesos de fabricación más confiables, las fallas durante el ciclo de vida de los sistemas aún pueden ocurrir. La probabilidad general de que el sistema tenga un mal funcionamiento hace inevitable la ocurrencia de alguna falla, con lo cual, la tendencia es diseñar sistemas que puedan tolerar fallas para aumentar su confiabilidad y evitar así que se presenten errores en su funcionamiento.

Para hacer un alisáis teórico de la confiabilidad de un sistema, hay que basarse en la configuración física de los diferentes módulos que lo componen, los cuales pueden estar conectados en serie, paralelo o una combinación de estas, por lo que es conveniente analizar este tipo de modelos de conexión. A continuación, se presentan algunos conceptos básicos acerca de la confiabilidad en los circuitos digitales iniciando con el análisis teórico de las configuraciones básicas de redundancia junto con la configuración puntual de la TMR. Seguidamente en la sección 7.2 se presenta la realización de la TMR sobre la SFU implementada (ver sección 5) y en la sección 7.3 sobre la SFU de comparación.

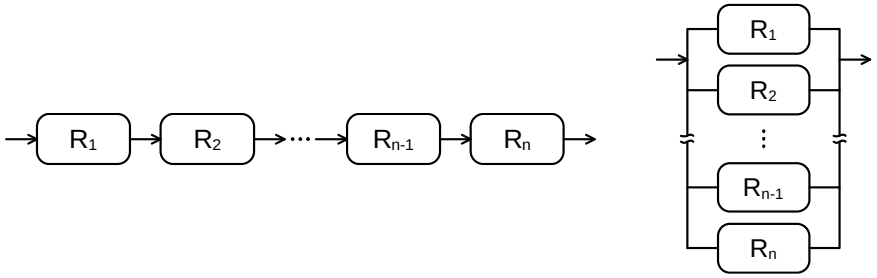
7.1. CONCEPTOS DE CONFIABILIDAD

La confiabilidad de un componente dentro de un sistema es una función de tiempo $R(t)$, que indica la probabilidad de que un sistema o componente funcione de manera correcta desde un tiempo cero hasta un tiempo determinado t. La confiabilidad está dada por la expresión $R(t) = e^{-\lambda t}$, donde el parámetro λ se conoce como la tasa de fallas por hora ($fallas/hora$). Normalmente la tasa de fallas de los circuitos integrados se encuentra entre 10^{-6} y $10^{-8} fallas/h$, por lo que se escogió para las dos unidades una tasa de fallos de $10^{-6} falas/h$, la cual se repartirá entre los componentes de cada SFU dependiendo del porcentaje de área que estos ocupen dentro de la unidad. Un sistema se puede caracterizar por su tiempo de misión, el cual está definido como el tiempo t tal que, $R_{sys}(t) = R_f$, donde R_f es una confiabilidad final predeterminada.

El valor usado para R_f puede depender de la aplicación a la que se quiera someter el circuito en cuestión, pero típicamente suele tomar el valor de 0.95, indicando el tiempo que tarda la confiabilidad de un sistema perfecto en degradarse hasta R_f [73]. La identificación explicita de la dependencia del tiempo se puede omitir en las expresiones de confiabilidad y así, la confiabilidad se puede expresar simplemente

como R y se sobreentiende que la confiabilidad en un tiempo t se puede obtener al sustituir en una expresión cualquiera el valor de $R(t)$ para cada valor de R.

Figura 50. Conexión de n módulos: (a) Serie, (b) Paralelo



(a) (b)

Fuente: Adaptado de [6]

En la conexión serie, todos los elementos deben funcionar para que el sistema general funcione, asumiendo que todos los módulos son independientes, la probabilidad de que el sistema funcione correctamente se puede ver como el producto de la probabilidad de que cada elemento funcione correctamente, con lo cual, la relación de confiabilidad general del sistema es:

$$R_{serie} = \prod_{i=1}^{n} R_i \tag{27}$$

Por otro lado, la probabilidad de que un sistema falle se denota como Q, siendo este el complemento de R, con esto se tiene que:

$$Q_{serie} = 1 - R_{serie} = 1 - \prod_{i=1}^{n} R_i = 1 - \prod_{i=1}^{n} [1 - Q_i] \tag{28}$$

En la conexión en paralelo, todos los elementos tienen que fallar para que el sistema general falle, como los módulos se toman como independientes, la probabilidad de que el sistema falle, se puede ver como el producto de la probabilidad de que cada elemento individual falle, con lo cual se tiene la siguiente relación:

$$Q_{paralelo} = \prod_{i=1}^{n} Q_i \tag{29}$$

Por lo tanto, la confiabilidad de un sistema paralelo es:

$$R_{paralelo} = 1 - Q_{parallel} = 1 - \prod_{i=1}^{n} Q_i = 1 - \prod_{i=1}^{n} [1 - R_i] \quad (30)$$

La confiabilidad de un sistema se puede mejorar con algún tipo de redundancia como se menciona en la sección 4.5.2. Tomando como ideal los circuitos adicionales de conmutación y/o votador, las anteriores ecuaciones (29) y (30) solo se podrían aplicar para los sistemas de redundancia dinámica, ya que son los únicos en los que el sistema general falla cuando todos los módulos fallan. De la configuración en paralelo se derivan los modelos $M - de - N$, los cuales requieren del correcto funcionamiento de M de los N módulos para que el sistema general siga funcionando correctamente. Este modelo sacrifica la ventaja operativa de solo requerir un módulo funcional a cambio de simplificar la complejidad de los circuitos de detección de fallas y conmutador por un circuito más simple llamado votador. A continuación, se explica más en detalle el análisis teórico de este modelo, del cual se deriva el caso particular de tolerancia a fallas usado sobre las SFU.

7.1.1. MODELOS M-DE-N

Un sistema $M - de - N$ es una generalización del modelo paralelo, sin embargo, en lugar de requerir solo uno de los N módulos para que el sistema funcione, se necesitan M módulos. La interpretación de la evaluación de confiabilidad de este sistema se puede realizar de la siguiente manera: la probabilidad de que todos los módulos funcionen correctamente es "$R_1\ R_2 \dots R_{N-1}\ R_N$", si existe la falla de uno de los tres módulos, la probabilidad se podría expresar dependiendo del módulo que falle así: cuando falle R_1 se tendría "$(1 - R_1)R_2 \dots R_{N-1}R_N$", cuando falle R_2 se tendría "$R_1(1 - R_2)R_3R_{N-1}R_N$", hasta cuando falle R_N donde se tendría "$R\_1\ R\_2 \dots R_{N-1}(1 - R_N)$". Si fallan dos módulos, se tendría: cuando falle R_1 y R_2 "$(1 - R_1)(1 - R_2)R_3R_{N-1}R_N$", cuando falle R_1 y R_3 "$(1 - R_1)\ R_2(1 - R_3\ )R_4 \dots R_{N-1}R_N$", hasta cuando falle R_{N-1} y R_N donde se tendría "$R\_1\ R\_2 \dots R_{N-2}(1 - R_{N-1})(1 - R_N)$", y así sucesivamente hasta que fallen $N - M$ de los N módulos, con lo cual se tendría: "$(1 - R_1)(1 - R_2) \dots (1 - R_{N-M}\ )R_{N-M+1} \dots R_{N-1}R_N$" cuando fallen $R_1, R_2, \dots, R_{N-M}$, "$(1 - R_1)R_2(1 - R_3) \dots (1 - R_{N-M+1})R_{N-M+2} \dots R_{N-1}R_N$" cuando fallen $R_1, R_3 \dots, R_{N-M+1}$, hasta "$R_1R_2 \dots R_M(1 - R_{M+1}) \dots (1 - R_{N-1})(1 - R_N)$" cuando fallen $R_{M+1}, \dots, R_N$. La confiabilidad general del sistema sería:

$$
\begin{aligned}
R_{M-de-N} = {} & R_1 R_2 \dots R_{N-1} R_N \\
& + \big((1-R_1)R_2 \dots R_{N-1}R_N + R_1(1-R_2)R_3 \dots R_{N-1}R_N + \cdots \\
& + R_1 R_2 \dots R_{N-1}(1-R_N)\big) \\
& + \big((1-R_1)(1-R_2)R_3 \dots R_{N-1}R_N + (1-R_1)R_2(1-R_3)R_4 \dots R_{N-1}R_N \\
& + \cdots + R_1 R_2 \dots R_{N-2}(1-R_{N-1})(1-R_N)\big) + \cdots \\
& + \big((1-R_1)(1-R_2) \dots (1-R_{N-M})R_{N-M+1} \dots R_{N-1}R_N \\
& + (1-R_1)R_2(1-R_3) \dots (1-R_{N-M+1})R_{N-M+2} \dots R_{N-1}R_N + \cdots \\
& + R_1 R_2 \dots R_M(1-R_{M+1}) \dots (1-R_{N-1})(1-R_N)\big)
\end{aligned}
$$

Asumiendo que todos los módulos son iguales, la expresión se puede reducir a la expresión (31), donde la combinatoria C_r^n representa la cantidad de subconjuntos o combinaciones de r módulos que se pueden realizar con n módulos, esto debido a que se tiene que obtener la cantidad de combinaciones cuando falla un módulo, dos, tres hasta cuando fallan $N-M$ de los módulos del sistema, esta expresión se evalúa como se expresa en (32).

$$
\begin{aligned}
R_{M-de-N} = {} & R^N + C_1^N R^{N-1}(1-R) + C_2^N R^{N-2}(1-R)^2 + \cdots \\
& + C_{N-M}^N R^{N-(N-M)}(1-R)^{N-M}
\end{aligned} \tag{31}
$$

$$
C_r^n = \binom{n}{r} = \frac{n!}{r!\,(n-r)!} \tag{32}
$$

Por ejemplo, la cantidad de combinaciones o situaciones diferentes que se pueden generar cuando fallan dos módulos de un sistema de cinco módulos, evaluando $C_2^5 = 5!/(2!\,(5-2)!)$ son 10. La expresión (31) se puede generalizar así:

$$
R_{M-de-N} = \sum_{i=0}^{N-M} C_i^N R_m^{N-i}(1-R_m)^i \tag{33}
$$

Por lo tanto, la probabilidad de que el sistema falle es:

$$
Q_{M-de-N} = 1 - \sum_{i=0}^{N-M} C_i^N {Q_m}^i (1-Q_m)^{N-i} \tag{34}
$$

7.1.2. TMR

El modelo TMR es un caso particular de los modelos $M-de-N$, donde se requieren $2-de-3$ módulos en correcta operación para que el sistema funcione,

la salida del sistema se obtiene realizando un “voto mayoritario” bit a bit donde se obtiene la salida que corresponde al valor de al menos dos de sus entradas. Este voto se realiza con un hardware extra que se conoce como votador (ver Figura 9). El votador es un elemento adicional el cual también puede fallar y, por tanto, ser analizado dentro del sistema como un elemento más en serie o triplicarse como el módulo que se quiere proteger. Sin embargo, para el análisis del presente trabajo, se asume un comportamiento ideal del votador. Usando la expresión general de los modelos $M - de - N$ expresada en (33), se tiene:

$$R_{TMR} = \sum_{i=0}^{1} C_i^N R_m^{N-i}(1-R_m)^i = C_0^3 R_m^3 (1-R_m)^0 + C_1^3 R_m^2 (1-R_m)^1$$

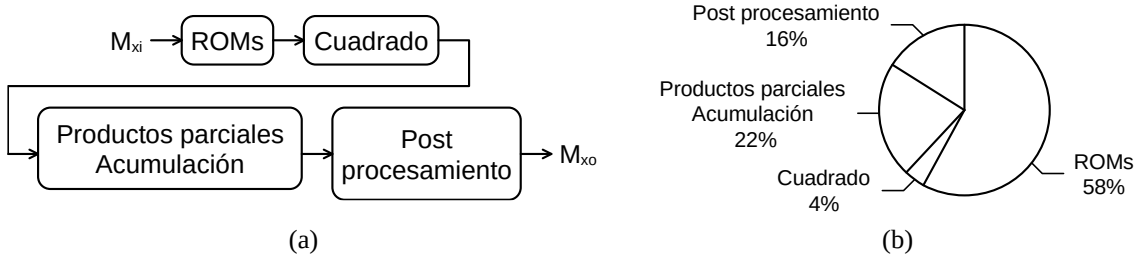
$$R_{TMR} = R_m^3 + 3R_m^2(1-R_m) = R_m^3 + 3R_m^2 - 3R_m^3$$

$$R_{TMR} = 3R_m^2 - 2R_m^3 \tag{35}$$

7.2. TMR EN LA SFU DE PPA

La estructura de la SFU se puede dividir en 4 módulos principales como se presentan en la Figura 51a, la confiabilidad de cada uno de los módulos se establece así: R_{rom} correspondiente a las tablas de los coeficientes junto con la lógica de codificación mostrada en la sección 5.4.1, R_{cu} correspondiente al módulo de la unidad especializada de elevación al cuadrado de la sección 5.4.2, R_{pp} hace referencia al circuito descrito en la sección 5.4.3, que comprende la generación de los productos parciales junto con su acumulación y por último R_{po} que hace referencia al post procesamiento para normalizar el resultado según la sección 5.4.4. La conexión entre las diferentes secciones descritas está en serie, ya que la salida de cada uno de los módulos es procesada por el siguiente y la falla de alguno provocaría una reacción en cadena que haría fallar a toda la unidad generando un resultado erróneo, así, la expresión de confiabilidad de la unidad sin ningún tipo de redundancia se presenta en (36).

Figura 51. SFU de PPA: (a) Esquema de bloques, (b) Área de ocupación



(a) (b)

Fuente: Elaboración propia

$$R_{SFU1} = R_{rom}R_{cu}R_{pp}R_{po} \tag{36}$$

Para aplicar la TMR a la SFU se realizaron tres diferentes combinaciones de los bloques internos de forma arbitraría para buscar incrementar la confiabilidad de la unidad. La primera aplicación de la TMR se realizó a la SFU completa, de donde se obtiene la expresión (37), con el valor de R_{SFU1} compuesto por cada uno de los bloques internos como se expresa en (36).

$$R_{11} = 3R_{SFU1}^2 - 2R_{SFU1}^3 \tag{37}$$

Aplicando la TMR a los bloques internos de las tablas, la unidad de elevación al cuadrado y la evaluación del polinomio de aproximación como un solo bloque se obtiene la expresión (38). La expresión obtenida ya es familiar, ya que al tomar los bloques R_{rom}, R_{cu} y R_{pp} como uno solo, se reemplazan dentro de la expresión regular de la TMR presentada anteriormente en (35) y se multiplica por el bloque R_{po} que queda en serie.

$$R_{12} = \left(3\left(R_{rom}R_{cu}R_{pp}\right)^2 - 2\left(R_{rom}R_{cu}R_{pp}\right)^3\right)R_{po} \tag{38}$$

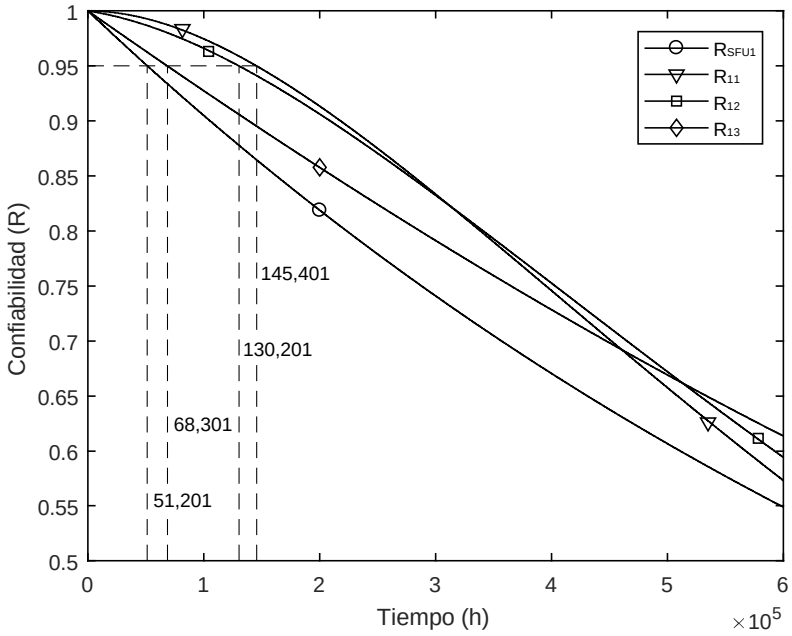
Para la tercera configuración, se seleccionó el bloque de la unidad de elevación al cuadrado y al bloque de la evaluación del polinomio, que comprende la generación de los productos parciales y su acumulación, lo cual corresponde a los circuitos descritos en las secciones 5.4.2 y 5.4.3 respectivamente, obteniendo así la expresión (39).

$$R_{13} = R_{rom}(3R_{cu}^2 - 2R_{cu}^3)\left(3R_{pp}^2 - 2R_{pp}^3\right)R_{po} \tag{39}$$

Para poder representar la confiabilidad de la SFU en términos del tiempo, se remplazan los valores de R por su equivalente dependiente del tiempo $R(t) = e^{-\lambda t}$ en las expresiones (36), (37), (38) y (39). El valor de λ para cada uno de los bloques se obtiene dependiendo del área que ocupa partiendo del valor de $\lambda = 10^{-6} fallas/h$. Por lo tanto, se tiene: $\lambda_{rom} = 5.8x10^{-7}$, $\lambda_{cu} = 4x10^{-8}$, $\lambda_{pp} = 2.2x10^{-7}$ y $\lambda_{po} = 1.6x10^{-7}$. En la gráfica de la Figura 52 se presenta el comportamiento de la confiabilidad de las diferentes configuraciones de redundancia para una confiabilidad $R_f = 0.95$. Además del análisis de confiabilidad, se recopilaron los datos de la cantidad de recursos usados por cada configuración, el tiempo de retardo o frecuencia a la que podría trabajar la unidad y el consumo de energía en términos de la disipación total de potencia térmica (*TTPD - Total Thermal Power Disipation*).

Los datos se recopilaron usando el programa Quartus II Edición Web con la FPGA Cyclone IV EP4CE115F29 de la tarjeta DE2-115, la cual cuenta con $114{,}480$ elementos lógicos. Las pruebas para el consumo de energía se realizaron con los 20.000 estímulos aleatorios usados en la sección 6.

Figura 52. Confiabilidad TMR de la SFU PPA



Fuente: Elaboración propia, la respuesta en el tiempo de R_{SFU1}, R_{11}, R_{12} y R_{13} corresponden a las configuraciones de redundancia expresadas en (36), (37), (38) y (39) respectivamente

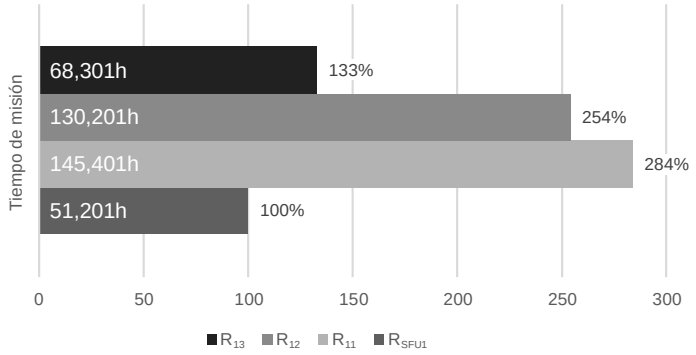
Tabla 13. Resultados de la TMR en la SFU de PPA

	Elementos	Utilización (%)	Tiempo de misión (h)	TTPD (mW)	Frecuencia (MHz)
R_{SFU1}	6,754	5.899	51,201	160.65	10.3
R_{11}	11,996	10.478	145,401	169.49	8.7
R_{12}	10,273	8.973	130,201	166.96	9.7
R_{13}	7,245	6.328	68,301	163.57	10.1

Fuente: Elaboración propia

El tiempo de misión de la SFU sin redundancia R_{SFU1} es de $51{,}201h$, la unidad de redundancia completa R_{11} tiene un tiempo de misión de $11{,}996h$ lo cual representa un aumento de 184%. Con la segunda configuración de redundancia R_{12} se tiene un tiempo de misión de $10{,}273h$ que equivale a un aumento del 154% y, por último, la unidad R_{13} presenta un tiempo de misión de $7{,}245h$ que equivale a un aumento de 33%. En la Figura 53 se puede observar el tiempo de misión de las diferentes configuraciones de redundancia de forma gráfica.

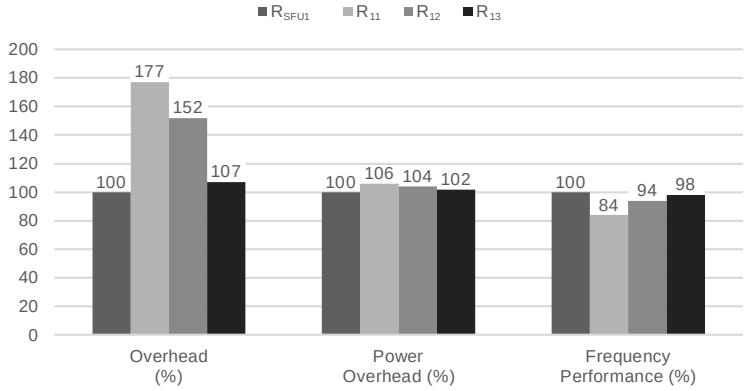
Figura 53. Tiempos de misión en la SFU de PPA



Fuente: Elaboración propia, los tiempos de R_{SFU1}, R_{11}, R_{12} y R_{13} corresponden a las configuraciones de redundancia expresadas en (36), (37), (38) y (39) respectivamente

La unidad sin redundancia R_{SFU1} utiliza $6{,}754$ elementos lógicos que equivalen a una utilización total de la FPGA de 5.8%, tiene una disipación de potencia 160.65mW y tiene una frecuencia de operación de $10.3MHz$. Aplicando la técnica de tolerancia a fallos a la unidad completa R_{11}, la cantidad de elementos lógicos usados aumenta en un 77% (*overhead*), la disipación de potencia aumenta un 6% (*power overhead*) y presenta un rendimiento de frecuencia de -16% ya que su frecuencia de operación es menor (*frequency performance*). Con la segunda configuración de redundancia R_{12}, el uso de elementos lógicos aumenta un 52%, la potencia disipada aumenta en un 4% y el rendimiento en frecuencia es de -6%. Por último, la última configuración R_{13} tiene un aumento de la cantidad de elementos lógicos requeridos de 7%, un aumento de potencia de 2% y un rendimiento de frecuencia de -2%. En la Figura 54 se puede observar de forma gráfica el comportamiento de los aspectos anteriormente mencionados.

Figura 54. Sobrecostos de la TMR en la SFU de PPA



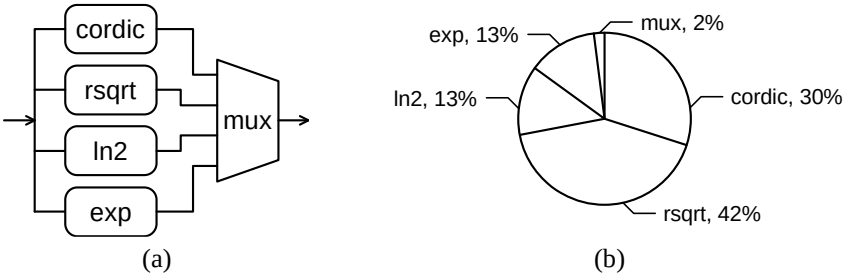
Fuente: Elaboración propia, las configuraciones de R_{SFU1}, R_{11}, R_{12} y R_{13} corresponden a los sistemas expresados en (36), (37), (38) y (39) respectivamente

Para que una solución de tolerancia a fallas se pueda tener en cuenta en una aplicación real el overhead debería ser menor o igual al 10%, en la Figura 54 se puede observar que la única configuración que cumple con este requerimiento es la configuración R_{13} expresada en (39), donde se realiza la TMR a los bloques de la unidad de elevación al cuadrado y al bloque de la evaluación del polinomio, con un overhead del 7% y un aumento en el tiempo de misión de 33%.

7.3. TMR EN LA SFU DE COMPARACIÓN

Al momento de buscar modelos de SFUs disponibles para evaluar la técnica de tolerancia a fallas seleccionada y poder hacer su comparación con la SFU implementada en este trabajo no se encontraron muchas opciones salvo por el trabajo realizado por Rodríguez Condia et al. en [23], el cual se tomará como modelo de comparación ya que cuenta con una descripción HDL de acceso libre. La arquitectura simplificada de esta SFU se puede observar en la Figura 55. La SFU cuenta con cuatro módulos principales: el módulo $cordic$ que usa un algoritmo del mismo nombre para calcular las funciones $sin(x)/cos(x)$, los módulos de $ln2$ y exp, los cuales implementan una aproximación lineal por partes para calcular las funciones $log_2(x)$ y 2^x. Por último, el módulo $rsqrt$, el cual hace uso del algoritmo rápido inverso de la raíz cuadrada (*FISR - Fast Inverse Square Root*) para el cálculo de la función $1/\sqrt{x}$. Para saber más detalles acerca de la implementación de esta SFU se puede remitir al trabajo realizado por los autores.

Figura 55. SFU de comparación: (a) Esquema de bloques, (b) Área de ocupación



Fuente: Adaptado de [23]

De forma general, los módulos $cordic$, $rsqrt$, $ln2$, y exp se encuentran en paralelo entre sí y su conjunto en serie con un multiplexor el cual se usa para seleccionar el resultado de la operación deseada. La expresión de confiabilidad de la SFU sin ningún tipo de redundancia se muestra en (40).

$$R_{SFU2} = \Big(1 - \Big((1 - R_{cordic})\big(1 - R_{rsqrt}\big)(1 - R_{ln2})\big(1 - R_{exp}\big)\Big)\Big) R_{mux} \qquad (40)$$

Al igual que con la SFU de PPA se realizaron tres combinaciones diferentes de los módulos internos para aplicar la TMR. La primera aplicación de la TMR se realizó a la SFU completa, con lo cual se tiene la expresión (41).

$$R_{21} = 3R_{SFU2}^2 - 2R_{SFU2}^3 \tag{41}$$

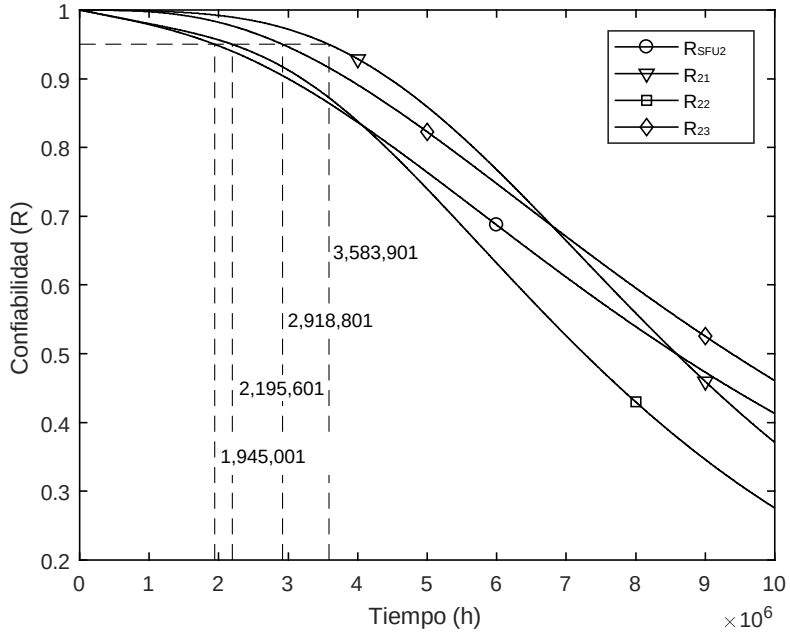
Aplicando sobre los módulos $cordic$, $rsqrt$, $ln2$, y exp se obtiene la expresión (42). Esta expresión de obtiene de forma directa al reemplazar la expresión regular de la TMR mostrada en (35) sobre la expresión de la SFU sin redundancia mostrada en (40).

$$R_{22} = \left(1 - \left(\left(1 - (3R_{cordic}^2 - 2R_{cordic}^3)\right)\left(1 - \left(3R_{rsqrt}^2 - 2R_{rsqrt}^3\right)\right)\left(1 - (3R_{ln2}^2 - 2R_{ln2}^3)\right)\left(1 - \left(3R_{exp}^2 - 2R_{exp}^3\right)\right)\right)\right) R_{mux} \tag{42}$$

Por último, se aplicó la TMR al multiplexor de salida, con lo cual se obtiene la expresión (43).

$$R_{23} = \left(1 - \left((1 - R_{cordic})(1 - R_{rsqrt})(1 - R_{ln2})(1 - R_{exp})\right)\right)(3R_{mux}^2 - 2R_{mux}^3) \tag{43}$$

Figura 56. Confiabilidad TMR de la SFU de comparación



Fuente: Elaboración propia, la respuesta en el tiempo de R_{SFU2}, R_{21}, R_{22} y R_{23} corresponden a las configuraciones de redundancia expresadas (40), (41), (42) y (43) respectivamente.

Para representar las expresiones de confiabilidad (40), (36), (41), (42) y (43) en términos del tiempo, se efectúa el mismo procedimiento que se realizó anteriormente reemplazando los valores de R por su expresión dependiente del tiempo $R(t) = e^{-\lambda t}$. El valor de λ para cada uno de los módulos se obtiene dependiendo del área que ocupa partiendo del valor de $\lambda = 10^{-6} fallas/h$. Por lo tanto, se tiene: $\lambda_{mux} = 2x10^{-8}$, $\lambda_{cordic} = 3x10^{-7}$, $\lambda_{rsqrt} = 4.2x10^{-7}$ y $\lambda_{ln2} = \lambda_{exp} = 1.3x10^{-7}$. En la gráfica de la Figura 56 se puede observar el comportamiento de la confiabilidad de las diferentes configuraciones de redundancia para $R_f = 0.95$. Con esta unidad también se hizo el análisis de consumo de energía usando la misma cantidad de estímulos que con la SFU de PPA para hacer una comparación equitativa y el análisis de la disipación de potencia.

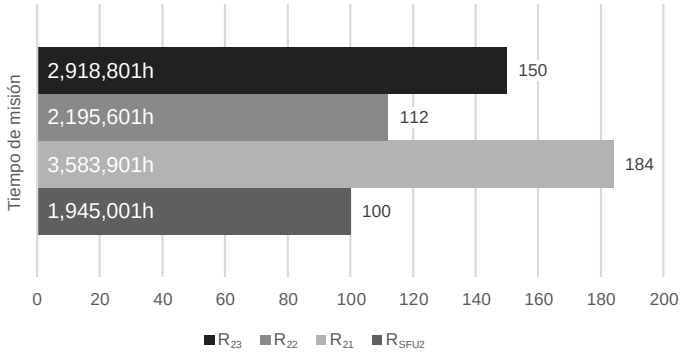
Tabla 14. Resultados del TMR de la SFU de comparación

	Elementos	Utilización (%)	Tiempo de misión (h)	TTPW (mW)	Frecuencia (MHz)
R_{SFU2}	8,639	7.546	1,945,001	158.61	9.9 – 0.52
R_{21}	15,243	13.314	3,583,901	168.18	9.1 – 0.48
R_{22}	15,124	13.211	2,195,601	164.09	8.8 – 0.46
R_{23}	9,075	7.927	2,918,801	159.88	9.7 – 0.51

Fuente: Elaboración propia

El tiempo de misión de la SFU de comparación sin redundancia R_{SFU2}, para una confiabilidad de 0.95 es de $1{,}945{,}001h$, la primera configuración con la técnica de tolerancia a fallos R_{21} presenta un tiempo de misión de $3{,}583{,}901h$ equivalente a un aumento de 84%, la segunda configuración R_{22} tiene un tiempo de misión de $2{,}195{,}601h$ que representa un aumento de 12% y, por último, la tercera configuración R_{23} presenta un aumento de 50% con un tiempo de misión de $2{,}918{,}801h$.

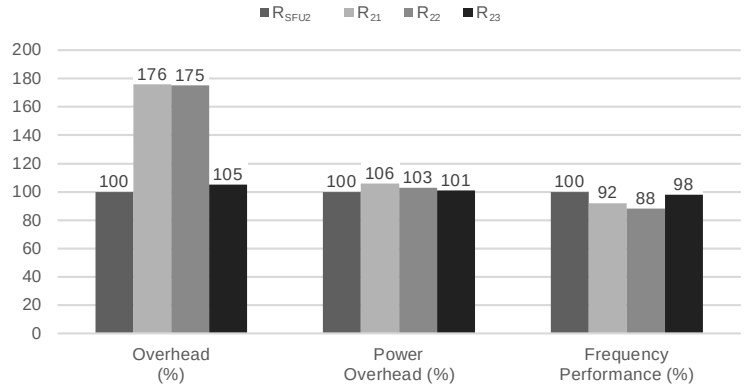
Figura 57. Tiempos de misión en la SFU de comparación



Fuente: Elaboración propia, los tiempos de R_{SFU2}, R_{21}, R_{22} y R_{23} corresponden a las configuraciones de redundancia expresadas en (40), (41), (42) y (43) respectivamente

La unidad sin redundancia R_{SFU2} hace uso de $8{,}639$ elementos lógicos que representa una utilización del 7.5% de la FPGA, presenta una disipación de potencia de $158.61mW$ y una frecuencia de operación de $9.9MHz$ para las funciones $rsqrt$, $ln2$ y exp que utilizan técnicas no cíclicas y $0.52MHz$ para las funciones $sin(x)$ y $cos(x)$ que necesitan de 19 ciclos de reloj para computar el resultado. La primera configuración con la técnica de tolerancia de fallas R_{21} presenta un *overhead* de 76%, un aumento en la disipación de potencia de 6% y un rendimiento de frecuencia de -8%. La segunda configuración R_{22} tiene un *overhead* de 75%, una *power overhead* de 3% y un rendimiento de -12%. Por último, la tercera configuración R_{23} necesita 5% más de elementos lógicos, presenta un aumento de disipación de potencia de 1% y un rendimiento de frecuencia de -2%.

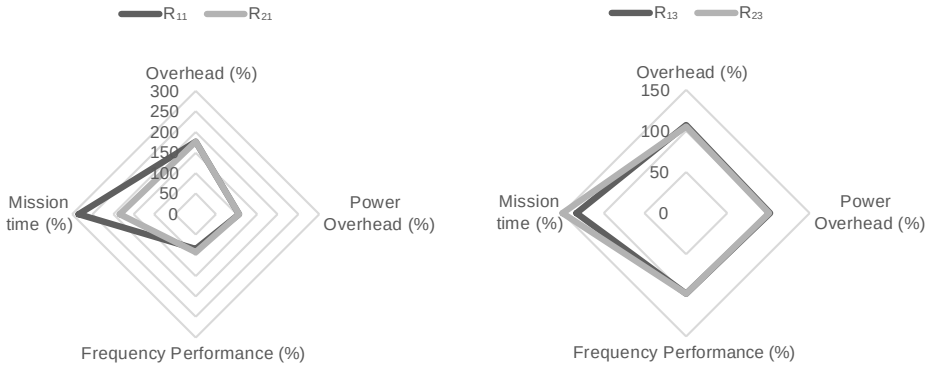
Figura 58. Sobrecostos de la TMR en la SFU de comparación



Fuente: Elaboración propia, las configuraciones de R_{SFU2}, R_{21}, R_{22} y R_{23} corresponden a los sistemas expresados en (40), (41), (42) y (43) respectivamente

Contrastando los resultados de la SFU de comparación con los de la SFU de PPA, se puede observar que la confiabilidad sin la TMR es mayor en la SFU de comparación ya que su tiempo de misión es mayor. Esto se puede deber a que la arquitectura de la SFU de comparación tiene una arquitectura en paralelo, con lo cual, la falla de alguno de sus módulos internos no inhabilita el funcionamiento total de la SFU mientras que la SFU de PPA tiene una arquitectura en serie, implicando que una falla en alguno de sus bloques interno afecte el funcionamiento general de la unidad. Al igual que con la SFU de PPA, la SFU de comparación tiene una única configuración que cumple con el requisito de no superar el 10% de *overhead* y es la configuración R_{23} expresada en (43), donde se aplica redundancia al multiplexor, que al ser este el único módulo en serie y por tanto el más crítico, logra incrementar en un 50% el tiempo de misión con solo un 5% de *overhead*, logrando así un mejor rendimiento que la SFU de PPA.

Figura 59. Resultados TMR: (a) Mejor rendimiento, (b) Implementable



(a) (b)

Fuente: Elaboración propia, las configuraciones de R_{11}, R_{21}, R_{13} y R_{23} corresponden a los sistemas expresados en (37)(40), (41), (39) y (43) respectivamente

En la Figura 59 se puede observar una gráfica de comparación de dos implementaciones de cada una de las SFU con los datos representados en formato porcentual. En la parte izquierda se presentan los resultados de la configuración que presento un mejor rendimiento de cada una de las unidades, se puede observar que el modelo R_{11} correspondiente a la SFU implementada, tiene un mayor aumento en el tiempo de misión mientras que los demás parámetros de comparación se mantienen equivalentes, sin embargo, ambas unidades superan el 10% de *overhead*. En la parte derecha de la imagen, se observa las dos únicas implementaciones que no superan este 10% de *overhead*, en esta comparación la configuración de la SFU de comparación presenta un mejor rendimiento ya que porcentualmente su tiempo de misión aumenta en mayor medida que en la SFU implementada y en general, los demás parámetros son equivalentes.

8. DISCUSIÓN

En la literatura presente hoy en día, no se encontraron estudios previos en los que se use alguna técnica de endurecimiento en las unidades de funciones especiales, por esto, el principal aporte que se realiza en este trabajo es exponer los resultados de la implementación de la TMR sobre dos unidades de funciones especiales con diferentes arquitecturas, en esencia, una arquitectura serie y otra paralela. Con esto, se tiene una base de comparación para otros trabajos que se puedan realizar sobre este tipo de unidades, además de que se presenta una implementación viable para cada una de las arquitecturas que aumentan su confiabilidad, lo cual es importante para los sistemas donde la confiabilidad es un factor clave para la integridad de su correcto funcionamiento.

La culminación de este trabajo deja como resultado una descripción a nivel de microarquitectura de acceso libre la cual puede ayudar a agilizar el proceso de investigación e implementación de otras técnicas de endurecimiento sobre las unidades de funciones especiales que puedan llegar a ser más optimas y eficientes, ya que la poca o nula disponibilidad de descripciones a nivel de microarquitectura de las SFU y sus pocos detalles técnicos de implementación impiden la exploración, evaluación e implementación de diferentes técnicas de tolerancia a fallas.

Los resultados de la implementación de la SFU muestran que la unidad presenta buenas características de aproximación con un error relativo pequeño pero un rendimiento no muy bueno con respecto a la frecuencia de operación. Haciendo una corta comparación con el trabajo realizado por Qoutb et al [37], el orden de magnitud del error relativo que se obtuvo en ese trabajo es de -4, mientras que el error relativo que se obtuvo con la SFU implementada en este trabajo llega a ser 10.000 veces más pequeño, con un orden de magnitud de -8 para la mayoría de las funciones implementadas. Sin embargo, el rendimiento obtenido en la SFU es menor ya que mientras que en el trabajo anteriormente mencionado se obtuvo una frecuencia de operación de $36MHz$, en la unidad implementada esta frecuencia es de $10.3MHz$, por lo que es unas 3.5 veces más lenta.

Se puede evidenciar que el uso de la TMR presenta resultados diferidos en las dos unidades debido a la arquitectura de cada unidad. La unidad de comparación sin ningún tipo de redundancia presenta un tiempo de misión mayor al de la unidad implementada gracias a su arquitectura paralela, por lo tanto, es más confiable. Sin embargo, se puede rescatar que el uso de esta técnica de redundancia tiene un mayor impacto en la unidad implementada que en la unidad de comparación, ya que la unidad implementada presenta una arquitectura serie, por lo que el tiempo de misión tuvo un mayor aumento porcentual con las diferentes propuestas de redundancia que en la unidad de comparación.

9. CONCLUSIONES Y TRABAJOS FUTUROS

9.1. CONCLUSIONES

Finalmente, la evaluación de rendimiento y confiabilidad de la SFU implementada basada en Aproximación Polinomial por Partes deja como resultado una unidad con bajo porcentaje de error, una velocidad de operación relativamente baja y una configuración factible de tolerancia a fallas que permite aumentar su confiabilidad.

El uso de una segunda unidad de comparación permite ratificar que los métodos no iterativos son más rápidos que los iterativos, pero a diferencia de lo mencionado en la literatura, el método no iterativo que fue el usado en la unidad implementada presento un menor uso de recursos lógicos que la unidad de comparación que usa una técnica iterativa, sin embargo, esto se puede deber a que la unidad no utiliza métodos iterativos en todas las funciones que computa.

El modelo Golden permitió abordar el problema de diseño desde una perspectiva menos detallada que si se hubiera abordado directamente desde la implementación en hardware, lo que dio la posibilidad de evaluar y validar diferentes aspectos de la arquitectura de una manera más efectiva que si se hubieran tenido que probar y modificar sobre el diseño HDL, también permitió supervisar, verificar y recalcular de forma rápida y puntual el valor de los coeficientes de las tablas.

La validación funcional de la unidad permite tener la certeza de que su diseño es correcto y que su funcionamiento es óptimo, con lo cual, la unidad está disponible para su libre uso donde se puedan estudiar otras técnicas de tolerancia a fallas y comparar sus resultados con los expuestos en el presente trabajo o usar la unidad existente como punto de comparación para otras implementaciones. La descripción HDL se encuentra alojada en el sitio OpenCores, por favor, siéntase libre se revisarla y usarla para realizar nuevos aportes.

El uso de la técnica de tolerancia a fallas TMR permite observar que su uso sobre una arquitectura serie es más efectiva que sobre una arquitectura paralela ya que el fallo de un módulo de una arquitectura serie presenta una reacción en cadena haciendo que los demás módulos posteriores al del fallo presenten un funcionamiento erróneo, mientras que en una arquitectura paralela, se pueden tomar medidas para omitir el uso del módulo defectuoso y con esto, no perder la funcionalidad completa del sistema.

La estimación de la confiabilidad de los modelos de las SFUs con las diferentes configuraciones de redundancia aplicando la TMR muestra que todas las configuraciones ayudan a mejorar el tiempo de misión de las unidades, pero

debido a las restricciones recomendadas de no superar el 10% de overhead, solo se tiene una configuración viable en para cada unidad, R_{13} para la arquitectura implementada y R_{23} en la arquitectura de comparación, con las que se obtiene un aumento en el tiempo de misión de 33% y un 50% respectivamente para un confiabilidad de 0.95.

9.2. TRABAJOS FUTUROS

En el diseño de las funciones $sin(X)$ y $cos(X)$ se realizó la reducción de rango al intervalo $[0,1)$ por comodidad en el diseño ya que, como se mencionó en la sección 5.1.1, cuando el ángulo es mayor a 1, se calcula la función opuesta de $\pi/2 - X$, por las propiedades $sin(X) = cos(\pi/2 - X)$ y $cos(X) = sin(\pi/2 - X)$. Esto permite que la selección de una función u otra se realice solo con el bit de la parte entera del número, sin necesidad de un comparador, sin embargo, el intervalo se podría reducir aún más a $[0, \pi/4)$ con el objetivo de comprimir la distancia de la segmentación y con esto, reducir el error de aproximación a costo del hardware adicional que implicaría el comparador.

Realizar un análisis del error de aproximación de la SFU de forma más detallada teniendo en cuenta el error que pueda aportar cada una de las partes del circuito para tener una visión más clara de cuanto error aporta cada decisión de diseño y con esto, poder proponer diferentes soluciones que puedan reducir el error final.

El componente más crítico en un sistema tolerante a fallas es el votador ya que el resultado final pasa por este componente, por lo tanto, es importante tener en cuenta que también puede fallar, con lo cual, se plantea la implementación de un sistema en el cual, el votador no se asuma con un comportamiento ideal, sino que también se implemente con la misma o alguna otra técnica de tolerancia a fallas.

Optimizar la implementación de la unidad de elevación al cuadrado especializada usando otro tipo de sumadores más eficientes en lugar de sumadores completos.

BIBLIOGRAFÍA

[1] S. Yoon, “Future shocks: 17 technology predictions for 2025 | World Economic Forum,” *World Economic Forum*, 2020. https://www.weforum.org/agenda/2020/06/17-predictions-for-our-world-in-2025/ (accessed Jun. 12, 2021).

[2] C. G. Spinola, J. Canero, G. Moreno-Aranda, J. M. Bonelo, and M. Martin-Vazquez, “Real-time image processing for edge inspection and defect detection in stainless steel production lines,” *2011 IEEE Int. Conf. Imaging Syst. Tech. IST 2011 - Proc.*, vol. 1, pp. 170–175, 2011, doi: 10.1109/IST.2011.5962196.

[3] F. L. Blasco and C. Tang, “Implementation of a multi-user detector for satellite return links on a GPU platform,” *2014 7th Adv. Satell. Multimed. Syst. Conf. 13th Signal Process. Sp. Commun. Work. ASMS/SPSC 2014*, vol. 2014-Janua, pp. 66–72, 2014, doi: 10.1109/ASMS-SPSC.2014.6934525.

[4] Z. L. Liu and C. F. Wang, “Efficient analysis of GPS antenna onboard self-driving car in practical environments with GPU accelerated MoM-PO method,” *2016 IEEE/ACES Int. Conf. Wirel. Inf. Technol. ICWITS 2016 Syst. Appl. Comput. Electromagn. ACES 2016 - Proc.*, pp. 6–7, 2016, doi: 10.1109/ROPACES.2016.7465397.

[5] C. López-Ongil, M. García-Valderas, M. Portela-García, and L. Entrena-Arrontes, “Techniques for fast transient fault grading based on autonomous emulation,” *Proc. -Design, Autom. Test Eur. DATE ’05*, vol. I, pp. 308–309, 2005, doi: 10.1109/DATE.2005.302.

[6] R. S. Swarz and D. P. Siewiorek, “Reliable Computer Systems,” in *Reliable Computer Systems*, 2nd ed., 1992.

[7] R. V. Kshirsagar and R. M. Patrikar, “A novel fault tolerant design and an algorithm for tolerating faults in digital circuits,” *Proc. - 2008 3rd Int. Des. Test Work. IDT 2008*, pp. 148–153, 2008, doi: 10.1109/IDT.2008.4802486.

[8] S. Furber, “Glosary,” in *ARM System-On-Chip Architecture*, 2nd ed., Addison-Wesley, 2000, p. 408.

[9] T. Risset, “SoC (System on Chip),” in *Encyclopedia of Parallel Computing*, D. Padua, Ed. Boston, MA: Springer US, 2011, pp. 1837–1842. doi: 10.1007/978-0-387-09766-4_5.

[10] G. Martin and H. Chang, “Tutorial 2 System-on-Chip Design,” pp. 12–17, 2001, doi: 10.1109/ICASIC.2001.982487.

[11] IEEE SOC Conference, “System on Chip (SoC) devices,” *34th IEEE INTERNATIONAL SYSTEM-ON-CHIP CONFERENCE*, 2020. https://www.ieee-socc.org/ (accessed Jul. 30, 2021).

[12] D. J. Greaves, “System on Chip Design and Modelling,” in *Notas de Lectura*, vol. II, no. January, 2011, p. 3. [Online]. Available: https://www.cl.cam.ac.uk/teaching/1011/SysOnChip/socdam-notes1011.pdf

[13] B. Falsafi *et al.*, “NVIDIA GPU,” *Encycl. Parallel Comput.*, pp. 1339–1344, 2011, doi: 10.1007/978-0-387-09766-4_194.

[14] J. Nickolls and W. J. Dally, “The GPU computing era,” *IEEE Micro*, vol. 30, no. 2, pp. 56–69, 2010, doi: 10.1109/MM.2010.41.
[15] J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone, and J. C. Phillips, “GPU computing,” *Proc. IEEE*, vol. 96, no. 5, pp. 879–899, 2008, doi: 10.1109/JPROC.2008.917757.
[16] A. Cano, “A survey on graphic processing unit computing for large-scale data mining,” *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.*, vol. 8, no. 1, 2018, doi: 10.1002/widm.1232.
[17] L. Deligiannidis and H. R. Arabnia, “Security surveillance applications utilizing parallel video-processing techniques in the spatial domain,” in *Emerging Trends in Image Processing, Computer Vision and Pattern Recognition*, Elsevier Inc., 2015, pp. 117–130. doi: 10.1016/B978-0-12-802045-6.00008-9.
[18] P. N. Glaskowsky, “NVIDIA ’ s Fermi : The First Complete GPU Computing Architecture,” no. September, 2009.
[19] J. M. P. Cardoso, J. G. F. Coutinho, and P. C. Diniz, “GPU ACCELERATORS,” in *Embedded Computing for High Performance*, Jonathan Simpson, 2017, pp. 31–33. doi: 10.1016/c2015-0-00283-0.
[20] D. Berjón, G. Gallego, C. Cuevas, F. Morán, and N. García, “Optimal piecewise linear function approximation for gpu-based applications,” *IEEE Trans. Cybern.*, vol. 46, no. 10, pp. 2584–2595, 2015, doi: 10.1109/TCYB.2015.2482365.
[21] E. Lindholm, J. Nickolls, S. Oberman, and J. Montrym, “Nvidia Tesla: a Unified Graphics and Computing Architecture To Enable Flexible, Programmable Graphics and High-Performance Computing,” *IEEE Micro*, pp. 39–55, 2008.
[22] A. Li, S. L. Song, M. Wijtvliet, A. Kumar, and H. Corporaal, “SFU-driven transparent approximation acceleration on GPUs,” *Proc. Int. Conf. Supercomput.*, vol. 01-03-June, 2016, doi: 10.1145/2925426.2926255.
[23] J. E. R. Condia, J. D. Guerrero-Balaguera, C. F. Moreno-Manrique, and M. S. Reorda, “Design and Verification of an open-source SFU model for GPGPUs,” *Proc. Bienn. Balt. Electron. Conf. BEC*, vol. 2020-Octob, 2020, doi: 10.1109/BEC49624.2020.9276748.
[24] J.-M. Muller, “The Classical Theory of Polynomial or Rational Approximations,” in *Elementary Functions Algorithms and Implementation*, 3rd ed., 2016, pp. 39–65. doi: 10.1007/978-1-4899-7983-4.
[25] M. D. Ercegovac and J. M. Muller, “Digit-recurrence algorithms for division and square root with limited precision primitives,” *Conf. Rec. Asilomar Conf. Signals, Syst. Comput.*, vol. 2, no. 5, pp. 1440–1444, 2003, doi: 10.1109/acssc.2003.1292224.
[26] E. Antelo, P. Montuschi, T. Lang, and A. Nannarelli, “Low latency digit-recurrence reciprocal and square-root reciprocal algorithm and architecture,” *Proc. - Symp. Comput. Arith.*, pp. 147–154, 2005, doi: 10.1109/arith.2005.29.
[27] L. Deng and J. An, “A low latency high-throughput elementary function generator based on enhanced double rotation CORDIC,” *Proc. - 2014 Symp.*

Comput. Appl. Commun. SCAC 2014, pp. 125–130, 2014, doi: 10.1109/SCAC.2014.33.

[28] S. F. Hsiao, C. S. Wen, and M. Y. Tsai, “Low-cost design of reciprocal function units using shared multipliers and adders for polynomial approximation and Newton Raphson interpolation,” *2010 Int. Symp. Next-Generation Electron. ISNE 2010 - Conf. Progr.*, no. xm, pp. 40–43, 2010, doi: 10.1109/ISNE.2010.5669204.

[29] R. Bhoyar, P. Palsodkar, and S. Kakde, “Design and implementation of goldschmidts algorithm for floating point division and square root,” *2015 Int. Conf. Commun. Signal Process. ICCSP 2015*, pp. 1588–1592, 2015, doi: 10.1109/ICCSP.2015.7322785.

[30] F. De Dinechin and A. Tisserand, “Multipartite table methods,” *Comput. Arith. Vol. III*, vol. 54, no. 3, pp. 381–392, 2015, doi: 10.1142/9789814651141.

[31] S. F. Hsiao, C. S. Wen, and P. H. Wu, “Compression of lookup table for piecewise polynomial function evaluation,” *Proc. - 2014 17th Euromicro Conf. Digit. Syst. Des. DSD 2014*, pp. 279–284, 2014, doi: 10.1109/DSD.2014.53.

[32] I. Koren and O. Zinaty, “Evaluating Elementary Functions in a Numerical Coprocessor Based on Rational Approximations,” *IEEE Trans. Comput.*, vol. 39, no. 8, pp. 1030–1037, 1990, doi: 10.1109/12.57042.

[33] J. E. Stine and M. J. Schulte, “Symmetric table addition method for accurate function approximation,” *J. VLSI Signal Process. Syst. Signal Image. Video Technol.*, vol. 21, no. 2, pp. 167–177, 1999, doi: 10.1023/A:1008004523235.

[34] D. M. Lewis, “Interleaved Memory Function Interpolators with Application to an Accurate LNS Arithmetic Unit,” *IEEE Trans. Comput.*, vol. 43, no. 8, pp. 974–982, 1994, doi: 10.1109/12.295859.

[35] D. U. Lee, R. C. C. Cheung, W. Luk, and J. D. Villasenor, “Hardware implementation trade-offs of polynomial approximations and interpolations,” *IEEE Trans. Comput.*, vol. 57, no. 5, pp. 686–701, 2008, doi: 10.1109/TC.2007.70847.

[36] M. J. Schulte and E. E. Swartzlander, “Hardware Designs for Exactly Rounded Elementary Functions,” *IEEE Trans. Comput.*, vol. 43, no. 8, pp. 964–973, 1994, doi: 10.1109/12.295858.

[37] A. E. G. Qoutb, A. M. El-Gunidy, M. F. Tolba, and M. A. El-Moursy, “High speed special function unit for graphics processing unit,” *Proc. 2014 9th Int. Des. Test Symp. IDT 2014*, pp. 24–29, 2015, doi: 10.1109/IDT.2014.7038581.

[38] D. De Caro, N. Petra, and A. G. M. Strollo, “High-performance special function unit for programmable 3-D graphics processors,” *IEEE Trans. Circuits Syst. I Regul. Pap.*, vol. 56, no. 9, pp. 1968–1978, 2009, doi: 10.1109/TCSI.2008.2010150.

[39] S. F. Hsiao, H. J. Ko, Y. L. Tseng, W. L. Huang, S. H. Lin, and C. S. Wen, “Design of hardware function evaluators using low-overhead nonuniform segmentation with address remapping,” *IEEE Trans. Very Large Scale Integr. Syst.*, vol. 21, no. 5, pp. 875–886, 2013, doi: 10.1109/TVLSI.2012.2202295.

[40] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital Memory & Mixed-Signal VLSI Circuits*. Springer, 2000.
[41] J. F. Ziegler and W. A. Lanford, “Effect of cosmic rays on computer memories,” *Science (80-.).*, vol. 206, no. 4420, pp. 776–788, 1979, doi: 10.1126/science.206.4420.776.
[42] N. G. Padharpurkar and V. Ravi, “Implementation of BIST using self-checking circuits for multipliers,” *2015 Int. Conf. Comput. Commun. Secur. ICCCS 2015*, 2016, doi: 10.1109/CCCS.2015.7374183.
[43] J. Johnson *et al.*, “Using duplication with compare for on-line error detection in FPGA-based designs,” *IEEE Aerosp. Conf. Proc.*, 2008, doi: 10.1109/AERO.2008.4526470.
[44] M. Hafezparast, “FAULT-TOLERANT HARDWARE DESIGNS AND THEIR RELIABILITY ANALYSES,” 1990.
[45] K. a Al-kofahi, “Reliability Analysis of Triple Modular Redundancy System with Spare,” no. December 1993, 1993.
[46] T. Zhang and J. Wang, “A Spatial-Temporal Model for Software Fault Tolerance in Safety-Critical Applications,” *Proc. - 2017 IEEE Int. Conf. Softw. Qual. Reliab. Secur. Companion, QRS-C 2017*, pp. 575–576, 2017, doi: 10.1109/QRS-C.2017.100.
[47] I. Informatics and N. Demokritos, “TOTALLY SELF CHECKING RECONFIGURABLE DUPLICATION SYSTEM,” 1995.
[48] M. A. Alves, “Graceful Degradation : Journey of a concept, from fault-tolerance to information loss,” in *2021 16th Iberian Conference on Information Systems and Technologies (CISTI)*, 2021, pp. 1–4. doi: 10.23919/CISTI52073.2021.9476345.
[49] K. Matsumoto, M. Uehara, and H. Mori, “Evaluating the fault tolerance of Stateful TMR,” *Proc. - 13th Int. Conf. Network-Based Inf. Syst. NBiS 2010*, pp. 332–336, 2010, doi: 10.1109/NBiS.2010.86.
[50] Y. Cui, M. Lou, J. Xiao, X. Zhang, S. Shi, and P. Lu, “Research and implementation of SEC-DED Hamming code algorithm,” no. 2011, pp. 0–4, 2013.
[51] F. J. O. Dias, “Fault Masking in Combinational Logic Circuits,” *IEEE Trans. Comput.*, vol. C–24, no. 5, pp. 476–482, 1975, doi: 10.1109/T-C.1975.224249.
[52] M. Prasanth, U. Har Narayan, G. Rajkumar, and V. Elmaran, “Fault-Tolerant Digital Filters in FPGA using Hardware Redundancy Techniques,” *Angew. Chemie Int. Ed.*, pp. 256–259, 2017.
[53] S. Choi, J. Park, and H. Yoo, “Area-efficient fault tolerant design for finite state machines,” *2020 Int. Conf. Electron. Information, Commun. ICEIC 2020*, pp. 4–5, 2020, doi: 10.1109/ICEIC49074.2020.9051122.
[54] S. Radhakrishnan, T. Nirmalraj, S. Ashwin, V. Elamaran, and R. K. Karn, “Fault Tolerant Carry Save Adders - A NMR Configuration Approach,” *2018 Int. Conf. Control. Power, Commun. Comput. Technol. ICCPCCT 2018*, vol. 3, no. 1, pp. 210–215, 2018, doi: 10.1109/ICCPCCT.2018.8574227.
[55] S. Aithal and S. Krishna Kumar, “Reconfigurable Triple Modular Redundant

and N-Modular Redundant systems with variable Reliability in multi-processor environment," *2012 18th Annu. Int. Conf. Adv. Comput. Commun. ADCOM 2012*, no. 4, pp. 33–38, 2012, doi: 10.1109/ADCOM.2012.6563581.
[56] American National Standards Institute, "IEEE Standard for Binary Floating-Point Arithmetic," 1985.
[57] N. E. H. Weste and D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, vol. 53, no. 9. 2013.
[58] I. Koren, "Computer Arithmetic Algorithms," in *Computer Arithmetic Algorithms*, 2nd ed., 2018, pp. 141–180. doi: 10.1201/9781315275567.
[59] R. Dornelles, G. Paim, B. Silveira, M. Fonseca, E. Costa, and S. Bampi, "A power-efficient 4-2 Adder Compressor topology," *Proc. - 2017 IEEE 15th Int. New Circuits Syst. Conf. NEWCAS 2017*, no. 1, pp. 281–284, 2017, doi: 10.1109/NEWCAS.2017.8010160.
[60] A. Pishvaie, G. Jaberipur, and A. Jahanian, "Improved CMOS (4; 2) compressor designs for parallel multipliers," *Comput. Electr. Eng.*, vol. 38, no. 6, pp. 1703–1716, 2012, doi: 10.1016/j.compeleceng.2012.07.015.
[61] J. A. Piñeiro, S. F. Oberman, J. M. Muller, and J. D. Bruguera, "High-speed function approximation using a minimax quadratic interpolator," *IEEE Trans. Comput.*, vol. 54, no. 3, pp. 304–318, 2005, doi: 10.1109/TC.2005.52.
[62] S. F. Oberman and M. Y. Siu, "A high-performance area-efficient multifunction interpolator," *Proc. - Symp. Comput. Arith.*, pp. 272–279, 2005, doi: 10.1109/arith.2005.7.
[63] J. Cao, B. W. Y. Wei, and J. Cheng, "High-performance architectures for elementary function generation," *Proc. - Symp. Comput. Arith.*, pp. 136–144, 2001, doi: 10.1109/arith.2001.930113.
[64] P. T. P. Tang, "Table-lookup algorithms for elementary functions and their error analysis," *Proc. - Symp. Comput. Arith.*, pp. 232–236, 1991, doi: 10.1109/arith.1991.145565.
[65] J. Muller, *Elementary functions: algorithms and implementation*, vol. 35, no. 05. 1998. doi: 10.5860/choice.35-2759.
[66] A. Tasissa, "Function approximation and the Remez algorithm," pp. 1–12, 2019.
[67] J. M. Muller, "'Partially rounded' small-order approximations for accurate, hardware-oriented, table-based methods," *Proc. - Symp. Comput. Arith.*, vol. 00, no. C, pp. 114–121, 2003, doi: 10.1109/arith.2003.1207668.
[68] W. Fraser, "A Survey of Methods of Computing Minimax and Near-Minimax Polynomial Approximations for Functions of a Single Independent Variable," *J. ACM*, vol. 12, no. 3, pp. 295–314, 1965, doi: 10.1145/321281.321282.
[69] G. Falcao *et al.*, "Shortening design time through multiplatform simulations with a portable OpenCL golden-model: The LDPC decoder case," *Proc. 2012 IEEE 20th Int. Symp. Field-Programmable Cust. Comput. Mach. FCCM 2012*, pp. 224–231, 2012, doi: 10.1109/FCCM.2012.46.
[70] H. Kaeslin, "Functional Verification," in *Top-Down Digital VLSI Design*, 2014, pp. 301–355. doi: 10.1016/b978-0-12-800730-3.00005-8.
[71] H. A. Technique and D. S. Design, "Diseño e implementación de un sistema

de verificación funcional utilizando la técnica de aceleración por hardware para optimizar el diseño de sistemas electrónicos digitales.," *INGENIUM*, vol. 13, no. 26, pp. 126–145, 2012, doi: 10.21500/01247492.1288.

[72] T. Jayashree and D. Basu, "On Binary Multiplication Using the Quarter Square Algorithm," *IEEE Trans. Comput.*, vol. C–25, no. 9, pp. 957–960, 1976, doi: 10.1109/TC.1976.1674723.

[73] J. F. Wakerly, "Microcomputer Reliability Improvement Using Triple-Modular Redundancy," no. 6, 1976.