

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica



Desarrollo de una herramienta para la generación de circuitos aritméticos aproximados

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

Deykel Hernández Araya

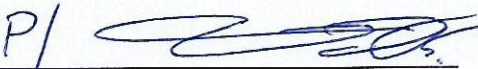
20 de junio de 2017

INSTITUTO TECNOLÓGICO DE COSTA RICA
ESCUELA DE INGENIERÍA ELECTRÓNICA
PROYECTO DE GRADUACIÓN
ACTA DE APROBACIÓN

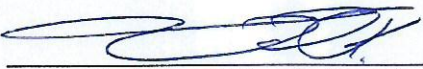
Defensa de Proyecto de Graduación
Requisito para optar por el título de Ingeniero en Electrónica
Grado Académico de Licenciatura
Instituto Tecnológico de Costa Rica

El Tribunal Evaluador aprueba la defensa del proyecto de graduación denominado Desarrollo de una herramienta para la generación de circuitos aritméticos aproximados, realizado por el señor Deykel Israel Hernández Araya y, hace constar que cumple con las normas establecidas por la Escuela de Ingeniería Electrónica del Instituto Tecnológico de Costa Rica.

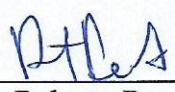
Miembros del Tribunal Evaluador


M.Sc. Jorge Castro Godínez

Profesor lector


M.Sc. Miguel Hernández Rivera

Profesor lector

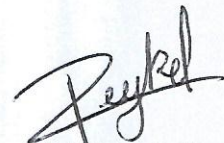

Dr. Roberto Pereira Arroyo

Profesor asesor

Cartago, 12 de junio de 2017

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

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de tesis realizado y por el contenido del presente documento.



Deykel Hernández Araya

Cartago. 5 de junio de 2017

Céd.: 7 0226 0295

A ti mi querida madre

Agradecimientos

Primeramente agradezco a Dios por darme vida, salud y sabiduría para superar todos los retos que se me han presentado a lo largo de este camino permitiéndome llegar a este punto, el inicio de mi carrera profesional.

A mi familia, que han sido mis mayores cómplices en muchos de los proyectos e ideas que se me han ocurrido, en los que nunca he encontrado un no como respuesta y con los que he aprendido los valores importantes que rigen mi vida convirtiéndome en la persona que soy hoy.

A mi madre en especial, por ser la persona que se esfuerza y sacrifica cada día por brindarnos lo mejor tanto a mi hermano como a mí, por ser mi mayor ejemplo de que se debe trabajar duro para alcanzar metas y cumplir sueños.

A la familia Castro-Solano, por hacerme sentir a gusto y brindarme el calor de una segunda familia durante el desarrollo de este proyecto en un país completamente diferente, también por el apoyo, ejemplos de vida y consejos brindados durante mi estancia.

Al profesor Jorge Castro Godínez por darme la oportunidad y responsabilidad de asumir este proyecto, por los consejos, confianza y paciencia brindada. Y por permitirme vivir una experiencia con la cual a parte de abrir mi mente en el campo profesional, también se da una apertura de ella hacia el intercambio cultural y social por el cual el mundo atraviesa.

A mis profesores lectores y mi profesor asesor por la formación académica y apoyo brindado no solo en este punto de mi carrera sino a lo largo de ella.

En fin, a las personas que siempre estuvieron apoyándome directa o indirectamente, a las que siempre han esperado lo mejor de mí y a las que aún esperan más.

Deykel Hernández Araya

Cartago, 20 de junio de 2017

Resumen

Computación aproximada se presenta como un nuevo paradigma que entre muchas capas de aplicación permite el diseño de circuitos energéticamente eficientes, esto aprovechando que muchas aplicaciones tienen cierta tolerancia a pérdidas en la precisión de sus cálculos, lo que admite variar las estructuras y a su vez el comportamiento lógico de las unidades aritméticas que realizan estos cálculos.

Este proyecto desarrolla una herramienta que es capaz de generar descripciones en hardware de unidades aritméticas aproximadas propuestas en la comunidad, además de ejecutar diversos procesos en herramientas externas de simulación y síntesis que brindan información relevante para el diseño de un circuito donde se desee introducir aproximaciones.

Es entonces como este documento proporciona primeramente las características de estructura y comportamiento lógico de las unidades aritméticas desarrolladas, seguido del proceso de diseño de la herramienta que incluye las consideraciones y restricciones tomadas para la descripción en hardware, escritura de scripts hacia herramientas externas y estructura de programación. Por último se presentan varios de los casos para los que la herramienta puede ser usada comprobando a su vez el comportamiento correcto de las unidades descritas.

Palabras clave: Circuitos aritméticos aproximados, Computación aproximada, Eficiencia energética, Sumadores aproximados, Multiplicadores aproximados, Divisores aproximados

Abstract

Approximate computing is presented as a new paradigm that among many application layers, allows the design of energy efficient circuits. This taking advantage of that many applications have some tolerance to losses in the precision of their calculations, which allows to vary the structures and in turn the logical behavior of the arithmetic units that perform these computations.

This project develops a tool that is capable of generating approximate arithmetic units hardware descriptions proposed in the community, in addition to execute various processes in external simulation and synthesis tools that provide relevant information for the design of a circuit in which is desired to introduce approximations.

It is then as this document first provides the characteristics of structure and logical behavior of the developed arithmetic units, followed by the process of design of the tool that includes the considerations and restrictions taken for the description in hardware, script writing towards external tools and structure programming. Finally, several of the cases are presented for which the tool can be used while checking the correct behavior of the described units.

Keywords: Approximate Arithmetic Circuits, Approximate Computing, Energy Efficiency, Approximate Adders, Approximate Multipliers, Approximate Dividers.

Índice general

Índice de figuras	v
Índice de tablas	vii
Abreviaciones	ix
1. Introducción	1
1.1. Computación Aproximada	2
1.2. Unidades aritméticas aproximadas	3
1.3. Generación de unidades aritméticas aproximadas	4
1.4. Contribución del proyecto	4
2. Fundamentos teóricos y estado del arte	7
2.1. Disipación de potencia en circuitos	7
2.2. Sumadores	8
2.3. Sumadores aproximados	9
2.3.1. Sumadores de baja Potencia	9
2.3.1.1. Sumador Aproximado Espejo (AMA, por sus siglas en inglés)	10
2.3.1.2. Sumador de parte baja OR (LOA, por sus siglas en inglés)	11
2.3.1.3. Approximate XOR/XNOR-based Adder (AXA)	11
2.3.1.4. Sumador aproximado basado en compuertas de transmisión (TGA, por sus siglas en inglés)	13
2.3.1.5. Sumador aproximado inexacto (InXA, por sus siglas en inglés)	14
2.3.1.6. Sumador de acarreo libre (CFA, por sus siglas en inglés)	15
2.3.2. Sumadores de alto rendimiento	16
2.3.2.1. Sumador de error tolerante II (ETAII, por sus siglas en inglés)	16
2.3.2.2. Sumador casi correcto (ACAI, por sus siglas en inglés)	16
2.3.2.3. Sumador de precisión configurable (ACAII, por sus siglas en inglés)	17
2.3.2.4. Sumador de precisión graciosamente-degradable (GDA)	17
2.3.2.5. Sumador Genérico de precisión configurable (GeAr, por sus siglas en inglés)	18
2.4. Multiplicadores	19
2.5. Multiplicadores Aproximados	19

2.5.1.	Multiplicador imparcial de rango dinámico para aplicaciones aproximadas (DRUM, por sus siglas en inglés)	19
2.5.2.	Multiplicador aproximado redondeado a base (RoBa, por sus siglas en inglés)	21
2.6.	Divisores	22
2.7.	Divisores Aproximados	22
2.7.1.	Divisor aproximado sin restauración (AXDnr, por sus siglas en inglés)	22
2.7.2.	Divisor aproximado con base redondeada de alta velocidad y eficiencia energética (SEERAD, por sus siglas en inglés)	24
2.7.3.	Divisor dinámico de baja potencia para aplicaciones aproximadas (DAC, por sus siglas en inglés)	26
2.8.	Métricas	27
2.8.1.	Producto retardo potencia (PDP, por sus siglas en inglés)	27
2.8.2.	Distancia de error (ED, por sus siglas en inglés)	28
2.8.3.	Distancia media de error (MED, por sus siglas en inglés)	28
2.8.4.	Tasa de error (ER, por sus siglas en inglés)	28
2.9.	Bibliotecas de circuitos aritméticos aproximados	28
2.9.1.	EvoApprox8b: Biblioteca de sumadores y multiplicadores aproximados para diseño de circuito y evaluación comparativa de métodos de aproximación	28
2.9.2.	lpaclib: Biblioteca de multiplicadores y sumadores aproximados	29
3.	Diseño	31
3.1.	Estructura de la herramienta	31
3.1.1.	Generalidades	31
3.1.2.	Archivos base	31
3.1.3.	Organización de directorios	32
3.1.4.	Tipo de archivos	33
3.2.	Descripción de hardware	33
3.2.1.	Sumadores	34
3.2.1.1.	Sumadores de baja Potencia	34
3.2.1.2.	Sumadores de alto rendimiento	34
3.2.1.3.	LOA	35
3.2.2.	Multiplicadores	35
3.2.2.1.	DRUM	36
3.2.2.2.	RoBa	36
3.2.3.	Divisores	37
3.2.3.1.	AXDnr	37
3.2.3.2.	DAC	37
3.2.3.3.	SEERAD	38
3.3.	Archivos de Prueba	40
3.4.	Vinculación con herramientas externas	40
3.4.1.	Script para compilación y simulación	40
3.4.2.	Script para Síntesis	41
3.4.3.	Script para simulación post-síntesis	43

3.4.4.	Script para estimación de potencia post-síntesis	43
3.5.	Núcleo de la herramienta	43
3.5.1.	Utilidades	44
3.5.2.	Unidades aproximadas	44
3.5.3.	Funciones de Generación	44
3.5.4.	Funciones de ejecución	45
3.5.5.	Generador de unidades	45
3.5.6.	AAUG	46
3.6.	Flujo de la herramienta	46
3.6.1.	Generación	47
3.6.2.	Simulación pre-síntesis	47
3.6.3.	Validación	47
3.6.4.	Síntesis	47
3.6.5.	Simulación Post-Síntesis	47
4.	Resultados	49
4.1.	Generación de datos para comparación	49
4.1.1.	Ejecución de herramienta	49
4.1.2.	Comparación entre procesos	52
4.1.3.	Múltiples comparaciones	54
4.1.4.	Comparaciones derivadas	55
4.2.	Introducción de optimizaciones	56
4.3.	Diferencias en Descripción	57
4.4.	Cambios en tecnología	59
4.5.	Limitaciones	61
4.6.	Variedad de unidades	64
5.	Conclusiones	69
	Bibliografía	71
A.	Manual de uso	75
A.1.	Requerimientos	75
A.2.	Configuración previa	75
A.3.	Ejecución	76

Índice de figuras

1.1. Predicción de la tendencia de las mayores fuentes de disipación de potencia	2
2.1. <i>m-bit RCA adder</i>	8
2.2. Implementación en compuertas lógicas de un FA, en rojo la ruta de retardo crítica	8
2.3. Estructura de un Sumador de baja potencia	9
2.4. Estructura para la parte baja en un LOA	11
2.5. Diagrama lógico de InXA1	14
2.6. Diagrama lógico de InXA2	14
2.7. Diagrama lógico de InXA3	15
2.8. Diagrama de bloques ETAII	16
2.9. Ejemplo numérico de dos números enteros de 20 bits	17
2.10. Implementación general del ACAII, $k=\frac{l}{2}$	17
2.11. Estructura de sumador GDA, (segmentos= $A_i, B_i; i = 0,3$)	18
2.12. Implementación general del GeAr: orden de derecha a izquierda	19
2.13. Diagrama de bloques de la implementación de DRUM	20
2.14. Ejemplo numérico en el multiplicador DRUM. ($N=16, k=6$) (a) Números de entrada (b) Entradas aproximadas (c) Resultado antes de corrimiento (d) Resultado Aproximado (e) Resultado Preciso	21
2.15. Diagrama de bloques de la implementación de RoBa	22
2.16. Implementación de un Divisor exacto sin restauración EXDnr: 8 bits dividendo, 4 bits divisor	23
2.17. Posibles configuraciones para el divisor AXDnr: en rojo celdas exactas reemplazadas por celdas aproximadas. Sustitución (A) Vertical (B) Horizontal (C) Cuadrada (D) Triangular	24
2.18. Diagrama de bloques de la implementación de SEERAD	26
2.19. Diagrama de bloques de la implementación de DAC	27
3.1. Estructura de carpeta general de la herramienta	32
3.2. Estructura de carpeta de unidades generadas	32
3.3. Flujo de simulación	41
3.4. Flujo de Síntesis	42
3.5. Estructura central de la herramienta	44
3.6. Flujo de herramienta	48
4.1. Script para generación de sumadores de baja potencia	50
4.2. Distancia media del error en sumadores LPA de 10 bits	50

4.3. Tasa de error para sumadores LPA de 10 bits	51
4.4. Potencia dinámica de sumadores de 10 bits tipo LPA sobre valor de referencia del sumador RCA $32.789 \mu W$	52
4.5. Potencia dinámica obtenida en síntesis contra valor medio de error	53
4.6. Potencia dinámica obtenida en post-síntesis contra valor medio de error	53
4.7. Ruta crítica o retardo contra el error medio de distancia	54
4.8. Ruta Crítica contra Potencia	55
4.9. PDP para sumadores LPA de 10 bits, valor de sumador RCA 0.206 pJ	55
4.10. Línea modificada en plantilla de Síntesis de sumadores	56
4.11. Potencia dinámica de sumadores de 10 bits tipo LPA sobre valor de referencia del sumador RCA $32.789 \mu W$ con optimización	56
4.12. Área en el GeAr	57
4.13. Potencia GeAr 90nm	58
4.14. Potencia GeAr2 90nm	58
4.15. Potencia GeAr 65nm	59
4.16. MED para distintas combinaciones de R y P en el GeAr	60
4.17. ER para distintas combinaciones de R y P en el GeAr	61
4.18. MED para multiplicadores aproximados	62
4.19. Datos Multiplicadores	63
4.20. Datos de divisor SEERAD	64
4.21. Comparación divisores	65
4.22. Comparación para divisor AXDNR con celda AXSC1	66
4.23. Comparación para divisor AXDNR con celda AXSC2	67
4.24. Comparación para divisor AXDNR con celda AXSC3	68

Índice de tablas

2.1. Tabla de verdad para sumador preciso	9
2.2. Tabla de verdad para sumador preciso	10
2.3. Tabla de verdad de las propuestas AMA ante las entradas de la tabla 2.2	11
2.4. Tabla de verdad de las propuestas AXA ante las entradas de la tabla 2.2	12
2.5. Tabla de verdad para VAXA	13
2.6. Tabla de verdad para TGA	14
2.7. Tabla de verdad para InXA	15
2.8. Funciones de una celdas exactas para suma y resta	22
2.9. Funciones de celdas de resta aproximadas	23
2.10. Valores en cada grupo y su correspondiente parámetro D y L	25
3.1. Referencia a archivos generados	33
3.2. Biblioteca de Sumadores	35
3.3. Biblioteca de Multiplicadores	36
3.4. Limitaciones de precisión en SEERAD	38
3.5. Biblioteca de divisores	39
A.1. Requerimientos de la herramienta	75
A.2. Opciones de la herramienta	76
A.3. Referencia a parámetros de unidades	77

Capítulo 1

Introducción

La eficiencia energética es actualmente un parámetro relevante en el diseño de sistemas computacionales [1]. Encontrar un equilibrio entre este parámetro y la capacidad o velocidad de procesamiento de información siempre será un desafío.

Hoy en día se ha dado un crecimiento acelerado de la cantidad de información que requiere ser procesada [2], por ejemplo el procesamiento de imágenes de muchos píxeles donde se encuentra una característica como la profundidad de bits (puede variar entre 1 bit, escala de blanco y negro, a 24 bits múltiples tonos) para cada píxel, lleva a que la información que se debe procesar aumente dependiendo de las dimensiones de la imagen, lo cual incide en el consumo de energía de los circuitos.

Aunado a esto, el fin práctico de la teoría de escalado de Dennard ha ocasionado un aumento significativo en la densidad de potencia de los chips actuales, dado que se ha incursionado rápidamente en tecnologías de fabricación de semiconductores muy pequeñas.

El nivel de integración que se ha logrado alcanzar gracias a esta reducción en tamaño trae consigo beneficios y perjuicios, por ejemplo en un procesador con más cantidad de transistores permite mejorar rendimiento pero a su vez implica mayor consumo de potencia tanto dinámica como estática [3].

La predicción de la tendencia en disipación de potencia que tendrían los circuitos basados en tecnología CMOS se presentó hace varios años en la *International Technology Roadmap for Semiconductors* (ITRS), que es un grupo formado por expertos en el área de semiconductores [4]. Esta tendencia es creciente conforme se reduzca el tamaño en la tecnología de fabricación.

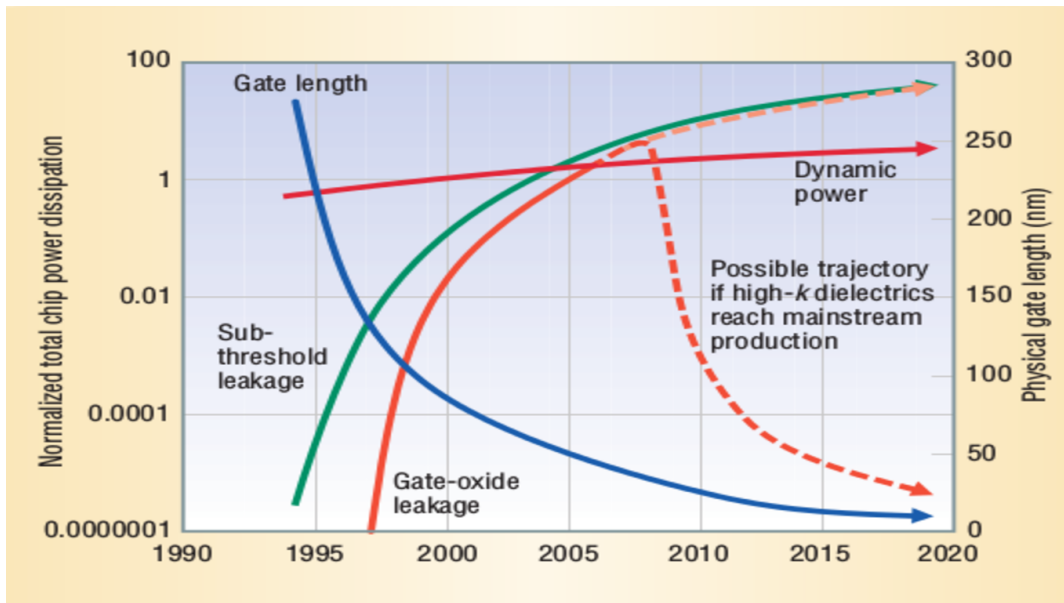


Figura 1.1: Predicción de la tendencia de las mayores fuentes de disipación de potencia

Al encontrarse la tecnología de semiconductores dentro del régimen nanométrico, el uso de cálculos precisos o libres de fallos se considera energéticamente ineficiente [5].

1.1. Computación Aproximada

Recientemente se ha planteado un nuevo paradigma de diseño energéticamente eficiente [1, 5, 6, 7], debido a que aprovecha que un conjunto significativo de aplicaciones son capaces de tolerar ciertas pérdidas de precisión en los resultados de sus cálculos [6]. Existen diversas fuentes que permiten tener esta característica de tolerancia, entre ellas se pueden mencionar: limitaciones perceptuales humanas, datos de entrada redundantes y señales de entrada ruidosas [1]. Por ejemplo, debido a las limitaciones del ojo humano, en el procesamiento de una imagen, un cambio pequeño en la resolución o nitidez no afectará la percepción de la persona, de esta manera se permite un resultado aceptable en lugar de uno preciso.

Diversos trabajos alrededor de AC han propuesto múltiples técnicas a diferentes niveles de abstracción, desde el software hasta el hardware. Por ejemplo, a nivel de hardware se puede intencionalmente reducir la tensión de alimentación en algunos componentes para una compensación entre energía y precisión, a su vez en la capa del software se podrían ignorar selectivamente ciertos cálculos que no son críticos y que permiten que el sistema alcance un mismo objetivo, a lo largo de estas capas otras técnicas que se pueden encontrar son [2]:

- Programación: Loop perforation, code perforation, fusión de hilos, núcleos sintonizables, reducción de patrones.
- Arquitectura: almacenaje aproximado, extensiones ISA, aceleradores aproximados.
- Circuito: lógica imprecisa, voltage overscaling, cálculos analógicos, escalado de precisión.

1.2. Unidades aritméticas aproximadas

AC propone una solución para diseñar e implementar ciertas aplicaciones y sistemas de una manera energéticamente más eficiente, los circuitos aritméticos no han escapado de este nuevo paradigma de diseño, entre ellos el sumador presenta relevante importancia, pues, por ejemplo, en el procesamiento digital de señales los sumadores son fundamentales en la determinación de la velocidad y el consumo de potencia del sistema [8].

En la actualidad se encuentran disponibles muchas propuestas de sumadores aproximados, entre las que destacan diseños que basan su operación en especulación, segmentación y aproximación de un sumador completo. Además se encuentra una variación en los sumadores de segmentación donde para cada sub-sumador la entrada de su acarreo es seleccionada diferente. Una clasificación siguiendo esta línea de sumadores aproximados es propuesta en [8].

Para efectos de este proyecto los sumadores se dividen en dos grupos. El primer grupo cuenta con sumadores conocidos como sumadores aproximados de alto rendimiento (*High-performance approximate adders, HPA*) están compuestos por sub-sumadores, unidades de predicción de acarreo y/o unidades de corrección de errores [9]. El segundo grupo está compuesto por los sumadores cuyo comportamiento se basa en aproximar al sumador completo y se conocerán a partir de aquí como sumadores aproximados de baja potencia (*Low-power approximate Adders*).

Como otra unidad básica en los circuitos aritméticos se encuentran los multiplicadores. Aunque han recibido menor atención que los sumadores, se pueden encontrar algunos diseños propuestos en la comunidad [5]. En [10] se mencionan tres metodologías aplicables para los multiplicadores aproximados: aproximación en la generación de productos parciales, aproximación (incluyendo truncado) en el árbol de producto parcial y usar diseños de sumadores (se señala como poco eficiente en la compensación entre precisión y energía [5]), contadores o compresores aproximados para acumular los errores parciales.

En el caso de los divisores, la atención es mucho menor debido posiblemente al bajo uso que se le da a estos en comparación con los sumadores o multiplicadores. En [11] se propone un divisor de baja potencia y bajo error, con un análisis teórico y una implementación, esta propuesta señala que puede alcanzar hasta un 70% de ahorro de energía con la introducción de un error absoluto promedio de solo 3.08%. Otra propuesta de divisor aproximado es SEERAD [12], donde se menciona que alcanza a ser de 14 a 300 veces menor en cuanto a consumo de energía en comparación con otros divisores.

Para algunos de estos diseños se han propuesto métricas que permiten evaluar su desempeño, entre ellas se puede mencionar la distancia de error (*Error Distance, ED*) y distancia media de error (*Mean Error Distance, MED*) propuestas en [13] y utilizadas tanto para sumadores como multiplicadores en [8] [10].

1.3. Generación de unidades aritméticas aproximadas

Cuando se requiera para algún propósito específico alguna de estas unidades aritméticas, se podrían implementar de manera manual. *Register-Transfer Level* (RTL) es un nivel de abstracción de diseño donde se concentra en la descripción del flujo de las señales a través de registros y de operaciones lógicas sobre estas señales, permitiendo por ejemplo que el comportamiento de sumadores, multiplicadores así como divisores sea fácilmente especificado y que lenguajes de descripción de hardware (*Hardware Description Language*, HDL) que utilizan este nivel de abstracción puedan crear una representación del circuito descrito.

Parámetros como errores en cálculos, consumo de potencia, área o retardo, se pueden obtener de este modelo RTL siguiendo diversos procesos en herramientas especializadas en simulación y síntesis, estas herramientas a su vez requieren cierto tipo de archivos para funcionar correctamente, los cuales se deben de crear de manera manual. También una validación del correcto comportamiento de la unidad se podría realizar haciendo una comparación ante un modelo en un nivel de abstracción más alto por ejemplo C++, este modelo se realizaría de la misma manera.

En un proceso de diseño, cuando se requiera una de estas unidades aritméticas aproximadas, la etapa de elección de alguna de ellas implicaría su implementación, simulación y síntesis de manera manual para luego realizar una comparación con la que se pueda elegir la unidad que presenta mayores beneficios para la aplicación en la cual se desea utilizar, lo cual requiere bastante tiempo.

Una manera de ahorrarse este tiempo es que el proceso se realice de manera automática. Sin embargo en la actualidad no se cuenta con una herramienta que permita la exploración, elección y proporcione la respectiva implementación a nivel de RTL de unidades aritméticas aproximadas.

1.4. Contribución del proyecto

Por lo anteriormente mencionado este proyecto desarrolla una herramienta capaz de generar componentes aritméticos aproximados, así como los archivos necesarios para su simulación, verificación y síntesis. La herramienta cuenta con una descripción en verilog de los componentes básicos necesarios para construir las unidades aritméticas aproximadas deseadas, así como los scripts que permiten a herramientas externas como Modelsim o Synopsys generar simulación y estimación de consumo (área, potencia) respectivamente.

Con la finalidad de que la herramienta propuesta sea de utilidad para una herramienta de *High-Level Synthesis* además de brindar a la comunidad un fácil acceso a las unidades aritméticas aproximadas, este proyecto debe de alcanzar lo siguiente:

- Análisis e implementación de un conjunto de componentes aritméticos aproximados existentes en la comunidad.

- Comparación de rendimiento de los componentes aritméticos aproximados según especificaciones dadas de diseño.
- Creación a nivel RTL del componente aritmético aproximado deseado, así como sus respectivos modelos de simulación, verificación y síntesis.

Capítulo 2

Fundamentos teóricos y estado del arte

En este capítulo se muestran los circuitos aritméticos aproximados que se implementan en la herramienta, se detallan las características de su comportamiento y estructura. Además se introducen dos diferentes tipos de métricas sencillas que son utilizadas para determinar valores de error en las unidades aritméticas aproximadas. Por otra parte se presentan dos bibliotecas que buscan facilitar la selección de componentes que se ajusten a ciertas características al igual que la herramienta que en este trabajo se propone.

2.1. Disipación de potencia en circuitos

El consumo de potencia en los circuitos CMOS proviene de dos fuentes, estático y dinámico. El primero se debe a corrientes de fuga que existen en los transistores aún así cuando el circuito esta inactivo. El consumo dinámico por su parte se debe a la carga y descarga de capacitancias de los transistores y conexiones, se da cuando hay transiciones de estado es decir cuando se da la conmutación [3].

$$P_{total} = P_{din} + P_{est}; \quad (2.1)$$

$$P_{din} = P_{cambio} + P_{corto} + P_{compuerta}; \quad (2.2)$$

En la ecuación 2.2 P_{cambio} es debido al esfuerzo de realizar un cambio en la salida de una compuerta en la presencia de cargas de otras compuertas en el circuito, P_{corto} es la corriente de corto circuito que fluye durante las transiciones y $P_{compuerta}$ es la corriente de túnel compuerta-óxido que fluye durante el periodo de transición. Una estimación de P_{cambio} se puede realizar asumiendo que el promedio de carga capacitiva por compuerta es C_L y calculando la potencia necesaria para cargar y descargar con una frecuencia de reloj de f_c [14], α representa una estimación de la actividad de cambio (*switching activity*) del circuito .

$$P_{cambio} = \alpha \cdot V_{DD}^2 \cdot C_L \cdot f_c; \quad (2.3)$$

La parte estática es la suma acumulativa de todos los mecanismos de fuga [14].

2.2. Sumadores

Los sumadores son ampliamente utilizados para realizar sumas binarias de dos números, un sumador comunmente usado es el *ripple-carry adder* (RCA), el cual basa su operación en m 1-bit sumadores completos (*full adder*, FA) colocados en cascada, el acarreo se propaga a través de cada FA por lo que el retardo de propagación crece en proporción a m [8].

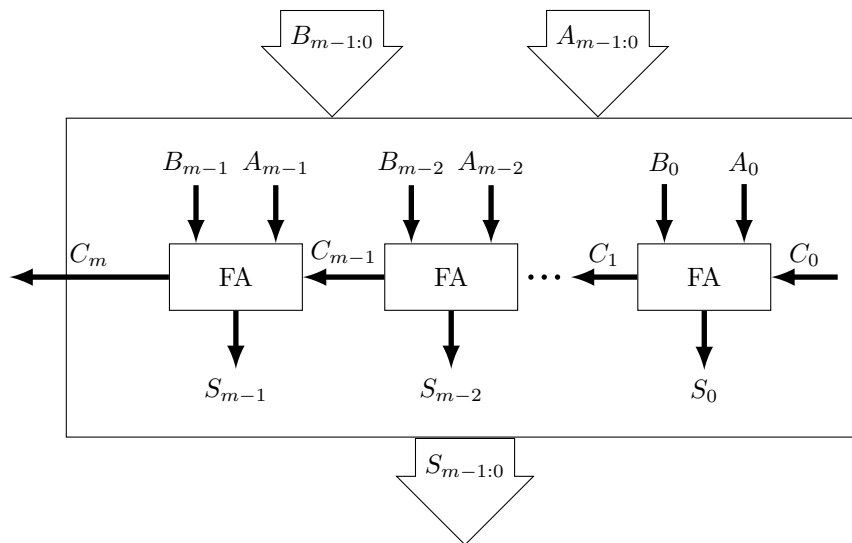


Figura 2.1: m -bit RCA adder

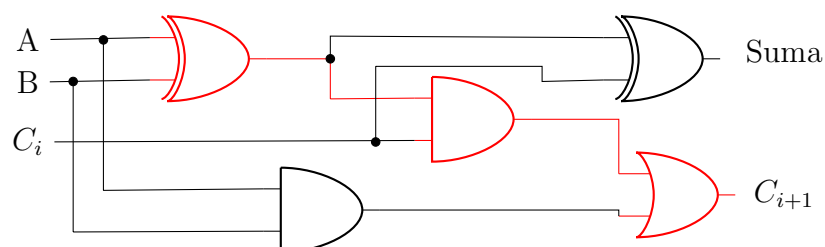


Figura 2.2: Implementación en compuertas lógicas de un FA, en rojo la ruta de retardo crítica

Tabla 2.1: Tabla de verdad para sumador preciso

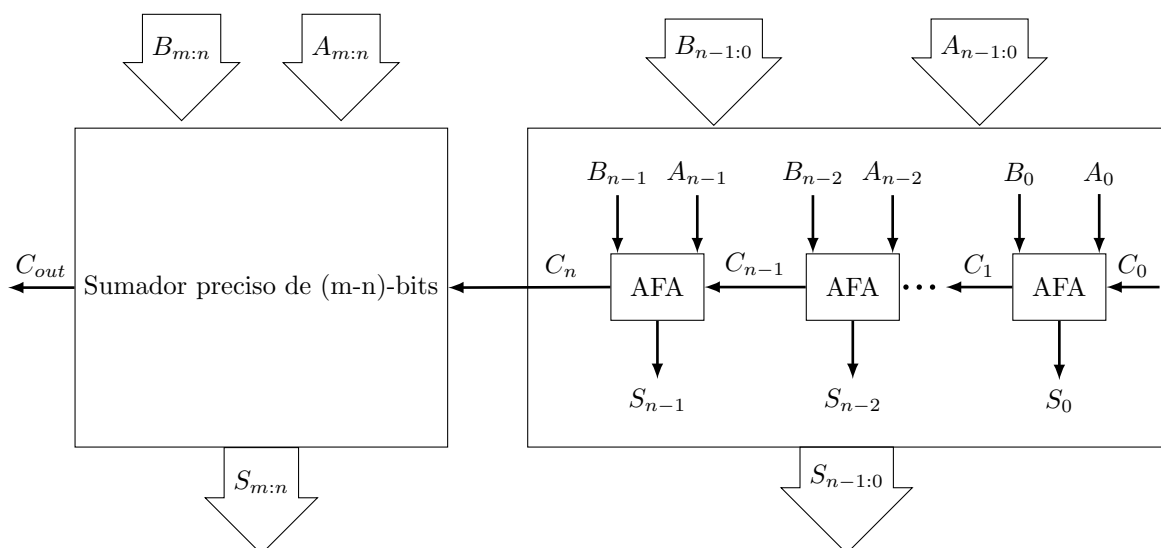
Entradas			Salidas	
A	B	C_{in}	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

2.3. Sumadores aproximados

Se propone dos clasificaciones para los sumadores, *Low Power Adder* (LPA) para los que presentan mejor eficiencia energética y *High Performance Adder* (HPA) para los que muestran mayor rapidez en cálculos.

2.3.1. Sumadores de baja Potencia

En este tipo de sumadores se busca aumentar la eficiencia energética mediante la reducción de área y en algunos casos de la ruta de retardo crítica como la mostrada en la figura 2.2, están compuestos por módulos que basan su operación en aproximar al sumador completo (*approximate full adders, AFA*). Para la implementación de estos sumadores se parte de un sumador común como el RCA de m -bits, la aproximación se da en los n -bits menos significativos, se modifica la estructura como la mostrada en la figura 2.1 cambiando n módulos FA por módulos AFA, mientras que los restantes $(m-n)$ -bits más significativos se suman en FAs.

**Figura 2.3:** Estructura de un Sumador de baja potencia

Algunas propuestas que se pueden encontrar en la literatura son:

2.3.1.1. Sumador Aproximado Espejo (AMA, por sus siglas en inglés)

Una manera convencional de implementar un FA es por medio de un sumador espejo (MA, por sus siglas en inglés) lo que lleva a que los AMA se generen mediante la modificación de la estructura básica de los MA removiendo algunos transistores. Al retirar estos transistores hay mejoras tales como el incremento de la carga y descarga de las capacitancias, menor disipación de energía y reducción de área [15]. Es importante destacar que no se pueden remover algunos transistores sin tener algún criterio, en [15] se toma en cuenta la introducción mínima de errores en la tabla de verdad del FA, asegurandose también de no generar circuitos abiertos o cortos. A continuación se muestra la tabla de verdad

Tabla 2.2: Tabla de verdad para sumador preciso

Entradas			Salidas	
A	B	C_{in}	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Mediante los criterios anteriormente mencionados, [15] propone cinco diferentes configuraciones de AMA las cuales se muestran a continuación:

- **AMA1:** Este configuración cuenta con ocho transistores menos que un MA convencional y en la tabla de verdad se introduce un error en el acarreo de salida (C_{out} , por sus siglas en inglés) y dos errores en la suma.
- **AMA2:** Se hace la observación de que en la tabla de verdad del sumador completo (tabla 2.2) que $sum=C_{out}$ para seis de ocho casos, por lo que se establece esta relación introduciendo dos errores en la suma mientras que C_{out} es correcto en todos los casos.
- **AMA3:** Se combinan AMA1 y AMA2 con lo que se introduce un error en C_{out} y tres errores en la suma.
- **AMA4:** Se observa en la tabla 2.2 que $C_{out}=A$ y $C_{out}=B$ para seis de ocho casos, entonces se fija la condición $C_{out}=A$, de esta manera se generan tres errores en la suma y dos en C_{out} , así también se rompe el retardo de propagación.

- **AMA5:** Se hace una extensión de AMA4 introduciendo un error más en la suma, el resultado de la suma se fija independiente del acarreo de entrada (C_{in} , por sus siglas en inglés) estableciendo $sum=B$ y $C_{out}=A$.

Tabla 2.3: Tabla de verdad de las propuestas AMA ante las entradas de la tabla 2.2

Preciso		AMA_1		AMA_2		AMA_3		AMA_4		AMA_5	
Sum	C_{out}	Sum	C_{out}	Sum	C_{out}	Sum	C_{out}	Sum	C_{out}	Sum	C_{out}
0	0	0	0	1	0	1	0	0	0	0	0
1	0	1	0	1	0	1	0	1	0	0	0
1	0	0	1	1	0	0	1	0	0	1	0
0	1	0	1	0	1	0	1	1	0	1	0
1	0	0	0	1	0	1	0	0	1	0	1
0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	1	0	1	0	1	0	1	1	1
1	1	1	1	0	1	0	1	1	1	1	1

2.3.1.2. Sumador de parte baja OR (LOA, por sus siglas en inglés)

Los AFA en este caso son compuertas OR con lo que se corta la propagación del acarreo, una compuerta AND extra se encarga de generar el bit de acarreo para la parte superior lo que hace que disminuya la imprecisión del sumador aproximado [16].

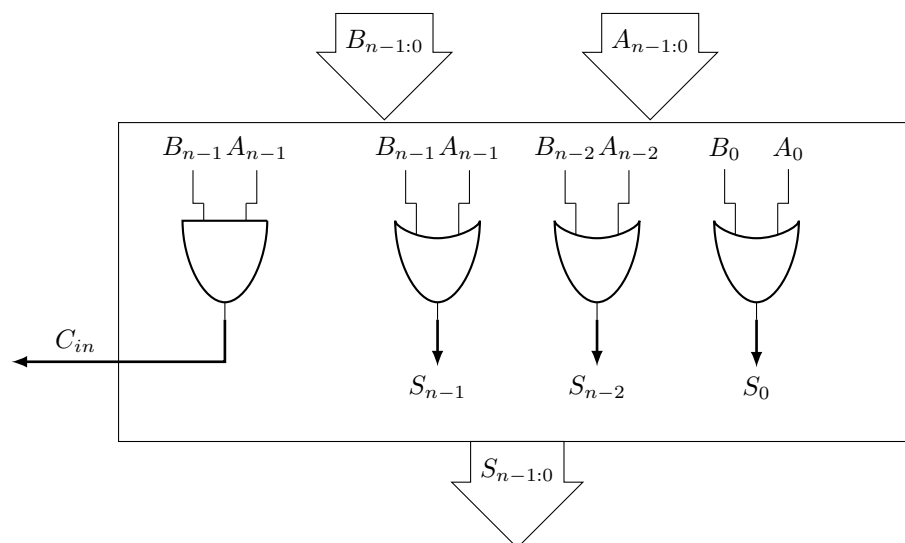


Figura 2.4: Estructura para la parte baja en un LOA

2.3.1.3. Approximate XOR/XNOR-based Adder (AXA)

Este tipo de sumadores se basan en compuertas XOR o NOR con multiplexores implementados por transistores de paso, tres propuestas son presentadas en [17] y se resumen a continuación:

- **AXA1:** Esta estructura se basa en la operación XOR y se compone por ocho transistores e introduce 4 errores tanto en C_{out} como en la suma, las ecuaciones que representan su comportamiento son las siguientes:

$$Sum = C_{in} \quad (2.4)$$

$$C_{out} = \overline{(A \oplus B) C_{in} + \overline{A} \cdot \overline{B}} \quad (2.5)$$

- **AXA2:** Cuenta con seis transistores, que consisten en una compuerta XNOR de cuatro transistores y un bloque de transistores de paso, sus ecuaciones son las siguientes:

$$Sum = \overline{(A \oplus B)} \quad (2.6)$$

$$C_{out} = (A \oplus B) C_{in} + AB \quad (2.7)$$

- **AXA3:** Es una extensión de AXA2 donde se utilizan dos transistores más en una configuración de transistor de paso con lo que se tiene una mejor precisión en la suma, sus ecuaciones se presentan a continuación:

$$Sum = \overline{(A \oplus B) C_{in}} \quad (2.8)$$

$$C_{out} = (A \oplus B) C_{in} + AB \quad (2.9)$$

Tabla 2.4: Tabla de verdad de las propuestas AXA ante las entradas de la tabla 2.2

Preciso		AXA ₁		AXA ₂		AXA ₃	
Sum	C_{out}	Sum	C_{out}	Sum	C_{out}	Sum	C_{out}
0	0	0	0	1	0	0	0
1	0	1	0	1	0	1	0
1	0	0	1	0	0	0	0
0	1	1	0	0	1	0	1
1	0	0	1	0	0	0	0
0	1	1	0	0	1	0	1
0	1	0	1	1	1	0	1
1	1	1	1	1	1	1	1

Una propuesta basada en AXA es presentada en [18] bajo el nombre Vdd-connected 1-bit AXA-based adder (VAXA), lo que busca es generar resultados correctos cuando C_{in} es '0', su tabla de verdad se muestra a continuación.

Tabla 2.5: Tabla de verdad para VAXA

Preciso		VAXA	
Sum	C_{out}	Sum	C_{out}
0	0	0	0
1	0	1	0
1	0	1	0
0	1	1	1
1	0	1	0
0	1	1	1
0	1	0	1
1	1	1	1

2.3.1.4. Sumador aproximado basado en compuertas de transmision (TGA, por sus siglas en inglés)

Al igual que en algunas de las propuestas mencionadas anteriormente, las dos propuestas de TGA presentadas en [19] buscan reducir la complejidad de circuito en cuanto al número de transistores removiendo algunas compuertas del sumador completo.

- **TGA1:** En este caso se establece $C_{out}=B$ mediante la observación que seis de los ocho casos presentes en la tabla 2.2 presentan esta condición, así la propagación del acarreo es reducida, la suma también es modificada resultando en dos errores de ocho casos posibles, a continuación las ecuaciones para el TGA1:

$$Sum = (A\bar{\oplus}B)C_{in} + A\bar{B} \quad (2.10)$$

$$C_{out} = B \quad (2.11)$$

- **TGA2:** La observación en este caso es que existen cuatro casos de seis donde si A o B son '1' C_{out} también es '1', de esta manera para calcular el C_{out} se utiliza una sola compuerta OR. Al igual que en el TGA1 no hay propagación de acarreo debido a que el C_{out} depende únicamente de las entradas A y B, sus ecuaciones son las siguientes:

$$Sum = (A\bar{\oplus}B)C_{in} \quad (2.12)$$

$$C_{out} = A + B \quad (2.13)$$

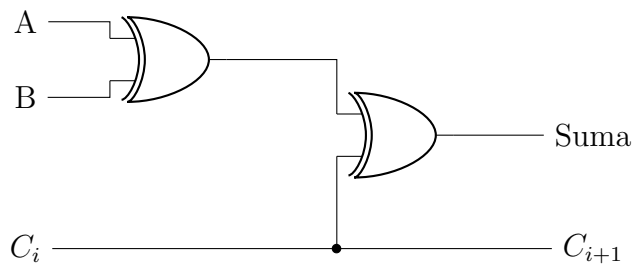
Tabla 2.6: Tabla de verdad para TGA

Preciso		TGA1		TGA2	
Sum	C_{out}	Sum	C_{out}	Sum	C_{out}
0	0	0	0	0	0
1	0	1	0	1	0
1	0	0	1	0	1
0	1	0	1	0	1
1	0	1	0	0	1
0	1	1	0	0	1
0	1	0	1	0	1
1	1	1	1	1	1

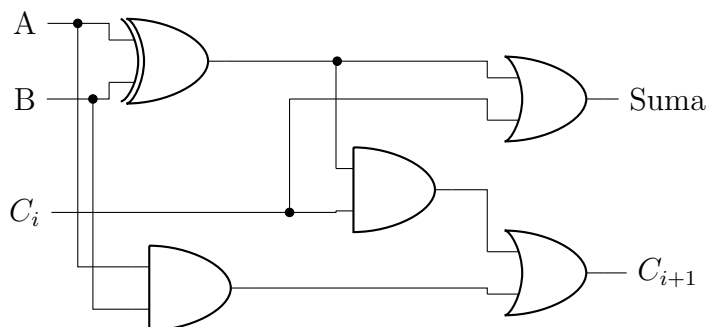
2.3.1.5. Sumador aproximado inexacto (InXA, por sus siglas en inglés)

Se presentan en [20] tres propuestas de sumadores aproximados que como características se mencionan un número menor de transistores que los diseños de AXA y AMA así como menos resultados incorrectos tanto en la suma como en el C_{out} .

- InXA1:** La aproximación se da en el acarreo mientras se mantiene la suma de manera precisa, el diagrama en compuertas lógicas de la estructura del InXA1 se muestra a continuación:

**Figura 2.5:** Diagrama lógico de InXA1

- InXA2:** En este caso se mantiene el acarreo preciso mientras que se aproxima la suma, su diagrama es el siguiente:

**Figura 2.6:** Diagrama lógico de InXA2

- **InXA3**: Al igual que el InXA2 esta configuración cuenta con un acarreo preciso y una suma aproximada, los errores son introducidos en diferentes resultados.

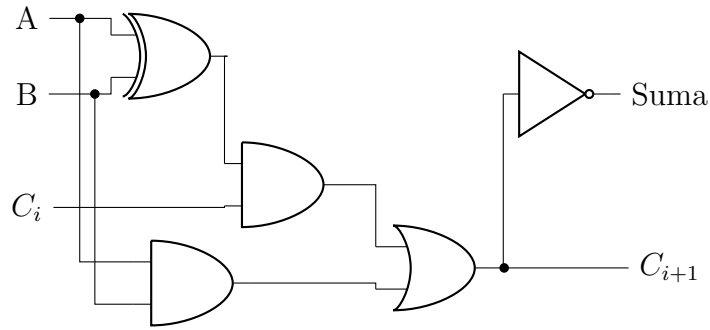


Figura 2.7: Diagrama lógico de InXA3

Tabla 2.7: Tabla de verdad para InXA

Preciso		InXA1		InXA2		InXA3	
Sum	C_{out}	Sum	C_{out}	Sum	C_{out}	Sum	C_{out}
0	0	0	0	0	0	1	0
1	0	1	1	1	0	1	0
1	0	1	0	1	0	1	0
0	1	0	1	1	1	0	1
1	0	1	0	1	0	1	0
0	1	0	1	1	1	0	1
0	1	0	0	0	1	0	1
1	1	1	1	1	1	0	1

2.3.1.6. Sumador de acarreo libre (CFA, por sus siglas en inglés)

En [21] se propone un sumador aproximado con el nombre de AFA, debido a la generalidad de este nombre se le asigna CFA según una de las características con las que cuenta esta propuesta. La observación que se hace en este caso es que al minimizar el número de literales se reduce la complejidad del circuito en cuanto a ruta crítica, conmutación de capacitancia y área [21], se puede minimizar este número mediante complemento de minterminos y/o maxterminos. Siguiendo estos enfoques se llega a la propuesta presentada mediante las siguientes ecuaciones:

$$Sum_i = A_i \oplus B_i \oplus A_{i-1} \quad (2.14)$$

$$C_{out} = A_i \quad (2.15)$$

De la ecuación 2.15 se extrae el nombre, ya que el C_{in_i} de cada módulo no depende del acarreo anterior $C_{out_{i-1}}$.

2.3.2. Sumadores de alto rendimiento

Este tipo de sumadores le apuestan a reducir la ruta de retardo crítico mediante la segmentación de la suma completa en sub-sumadores, cada uno de estos ejecuta la suma de manera precisa en el segmento de bits correspondientes, lo que hace que la ruta de retardo se reduzca al tamaño de cada sub-sumador. La contra parte en esta reducción en el retardo es que estos sumadores suelen tener una mayor área.

2.3.2.1. Sumador de error tolerante II (ETAII, por sus siglas en inglés)

Una primera propuesta de este tipo de sumador es Error-Tolerant Adder I (ETA I) el cual tenía problemas calculando la suma para números pequeños, ETAII se propone en [22] con el fin de resolver este problema. Para obtener la estructura del ETAII, se divide un sumador de m bits en k sub-sumadores ($k \geq 2$), cada uno de estos bloques contiene m/k bits, por lo que m tiene que ser divisible entre k . Cada sub-sumador se compone de un generador de acarreo y un generador de suma, características importantes a mencionar de esta estructura es que el generador de acarreo no toma en cuenta el acarreo proveniente del sub-sumador anterior, mas sin embargo el generador de suma si lo considera. De esta forma la propagación del acarreo solo se da entre dos bloques consecutivos.

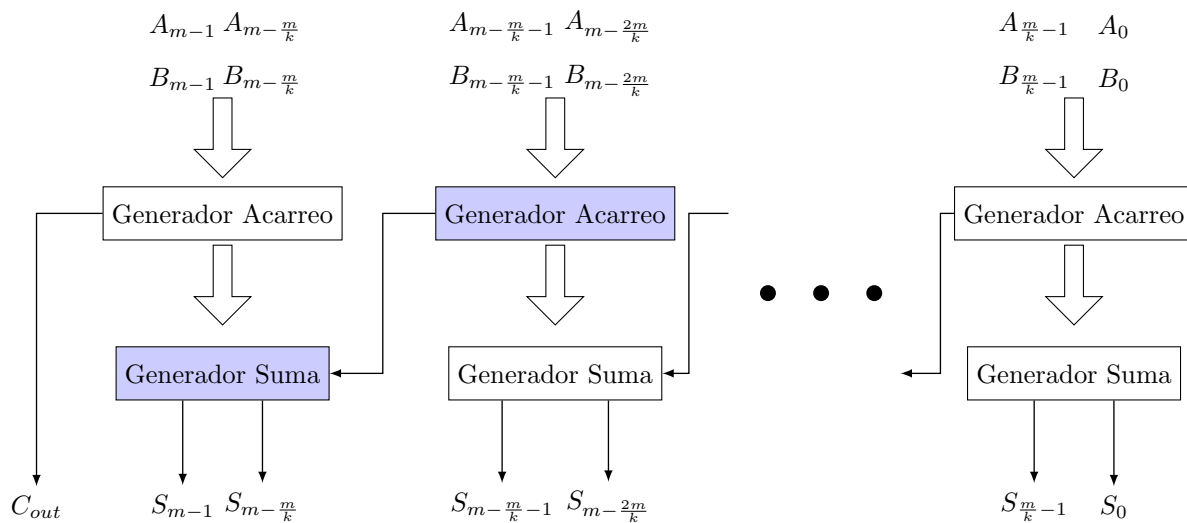


Figura 2.8: Diagrama de bloques ETAII

2.3.2.2. Sumador casi correcto (ACAI, por sus siglas en inglés)

En este caso los sub-sumadores están compuestos por l bits de los cuales se utilizan $l-1$ bits para producir el bit de suma en la en la posición l de cada sub-sumador, en otras palabras a partir del bit en la posición l de un sumador de m bits, se toman los $l-1$ bits anteriores para generar el acarreo necesario, lo que reduce la propagación del acarreo a la misma que tendría un sumador de l bits [23], en este tipo de sumador existe un traslape entre los sub-sumadores.

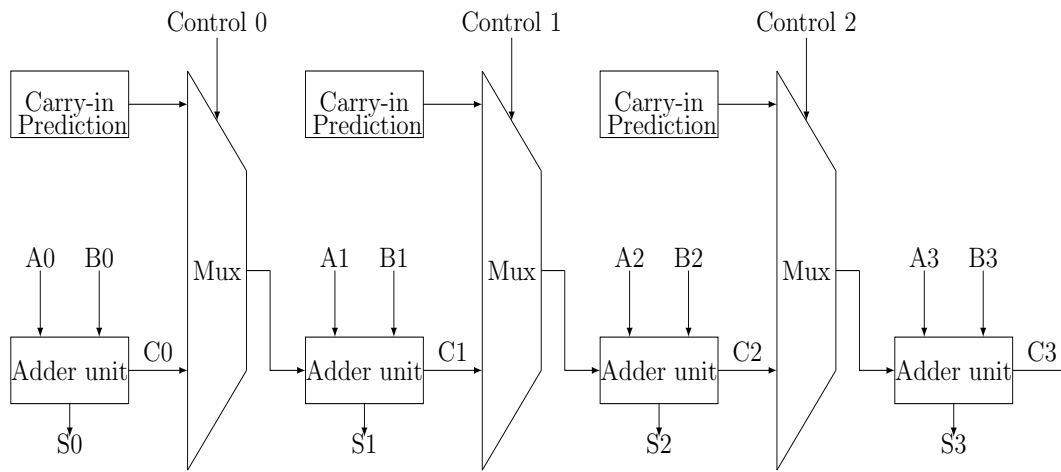


Figura 2.11: Estructura de sumador GDA, (segmentos= $A_i, B_i; i = 0..3$)

2.3.2.5. Sumador Genérico de precisión configurable (GeAr, por sus siglas en inglés)

En [9] se propone un sumador genérico con precisión configurable (GeAr, por sus siglas en inglés), el mismo segmenta como los anteriores una suma de m bits en sumas de l bits realizadas por k sub-sumadores, además se define R como bits resultantes y P como bits previos. R son los bits que contribuyen con la suma final y P son utilizados para predecir el acarreo en cada sub-sumador. Solo el primer sub-sumador presenta una excepción aportando l bits a la suma final, relaciones importantes entre estos parámetros son presentadas a continuación.

$$l = R + P \quad (2.16)$$

$$k = ((m - l)/R) + 1 \quad (2.17)$$

Para el primer sub-sumador se cumple:

$$Sum[l - 1 : 0] = A[l - 1 : 0] + B[l - 1 : 0] \quad (2.18)$$

El resultado de los siguientes sub-sumadores se pueden obtener mediante:

$$Sum[(R * i) + P - 1 : R * (i - 1) + P] = A[R * i) + P - 1 : R * (i - 1)] + B[R * i) + P - 1 : R * (i - 1)] \quad (2.19)$$

donde $1 < i \leq k$.

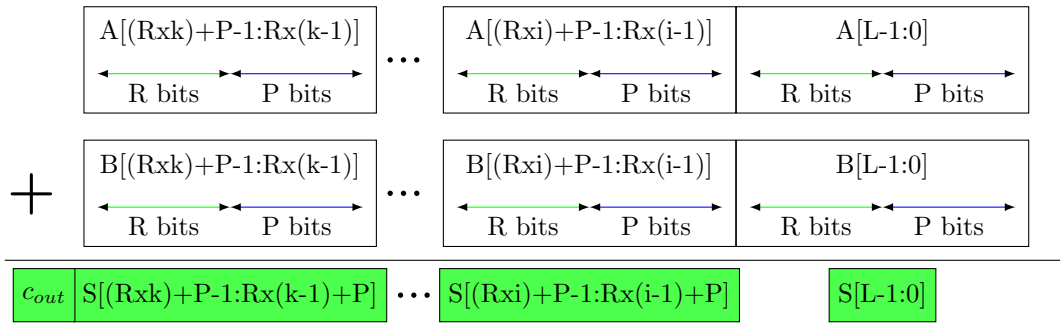


Figura 2.12: Implementación general del GeAr: orden de derecha a izquierda

2.4. Multiplicadores

Esta operación suele ser más compleja que la suma, en su proceso incluye tres elementos: multiplicando, multiplicador y producto. El procedimiento general es recorrer el multiplicador desde el bit menos significativo al más significativo. Se generan productos parciales los cuales, para cada bit del multiplicador, es el multiplicando si el bit del multiplicador es 1 y 0s si el bit es 0. Luego cada producto parcial es desplazado hacia la izquierda un espacio relativo a su previo producto parcial. Cuando se obtienen todos los productos parciales los mismos se suman. Para una multiplicación donde el multiplicando y multiplicador esten compuestos por N bits el producto será $2N$ bits. Existen varios métodos para realizar la multiplicación entre ellos: algoritmo de Booth, secuencia de suma-corrimiento y multiplicador en arreglo.

2.5. Multiplicadores Aproximados

2.5.1. Multiplicador imparcial de rango dinámico para aplicaciones aproximadas (DRUM, por sus siglas en inglés)

La premisa con la que trabaja esta propuesta en [26] es que no todos los bits de un número son igualmente importantes, por lo que se limita la cantidad de bits que se aplican a un multiplicador central preciso seleccionando cuidadosamente un rango para cada uno de los operandos.

El modo de operación que sigue DRUM es el siguiente, asumiendo que ambos operandos son de N bits, primero se obtiene la posición del bit 1 más significativo mediante módulos que detectan este bit llamados detector de uno conductor (LOD, por sus siglas en inglés), luego a partir de la posición del bit más significativo se seleccionan los $k-2$ bits consecutivos basados en la precisión requerida, donde k es definido en etapa de diseño y especifica el ancho de bits que utilizará el multiplicador central.

Para reducir el error se aproximan los bits menos significativos restantes, para esto sea t la posición donde se encuentra el bit 1 más significativo con $0 \leq t \leq N-1$, así el valor de los $t-k+2$ bit menos significativos restantes esta entre 0 y $2^{t-k+2} - 1$, si

se asume una distribución uniforme para los valores de los operandos, el valor esperado es casi igual a 2^{t-k+1} , de esta manera se aproximan los $t - k + 2$ bits colocando un 1 en la posición $t - k + 1$ y 0 desde la posición $t - k$ a 0, luego estos ceros son truncados y se opera en el multiplicador central los operandos de k bits formados [26].

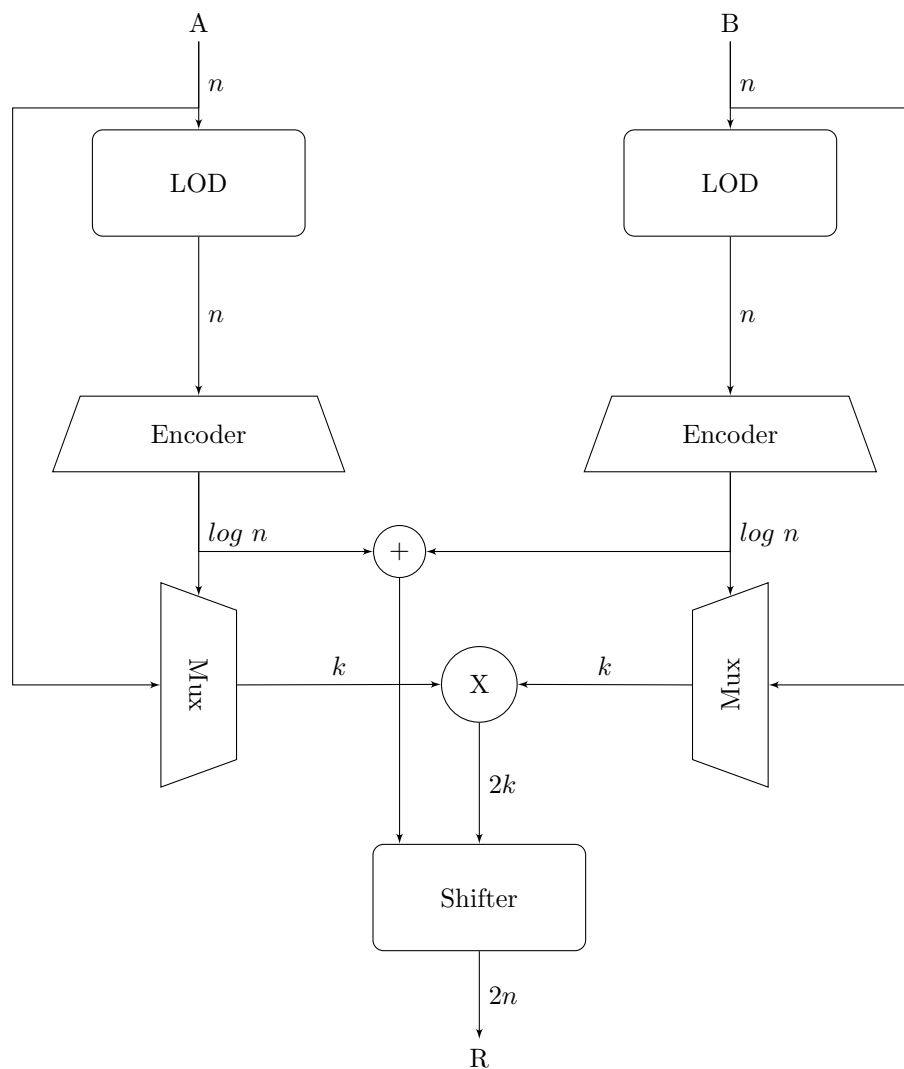


Figura 2.13: Diagrama de bloques de la implementación de DRUM

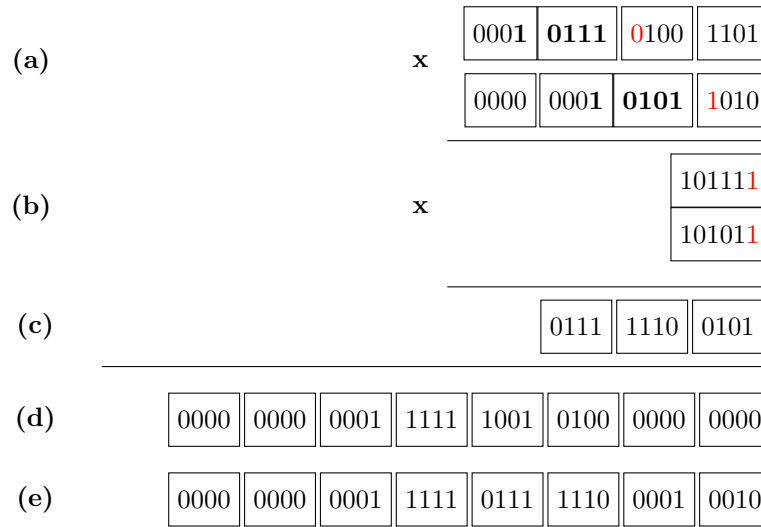


Figura 2.14: Ejemplo numérico en el multiplicador DRUM. ($N=16$, $k=6$) (a) Números de entrada (b) Entradas aproximadas (c) Resultado antes de corrimiento (d) Resultado Aproximado (e) Resultado Preciso

2.5.2. Multiplicador aproximado redondeado a base (RoBa, por sus siglas en inglés)

El algoritmo propuesto en [27] para este multiplicador aprovecha que al tener números redondeados en potencia de 2 las multiplicaciones se pueden realizar mediante corrimientos. Sea A y B multiplicando y multiplicador respectivamente, sus representaciones redondeadas serán A_r y B_r . La operación de multiplicación puede ser escrita como:

$$A \cdot B = (A_r - A) \cdot (B_r - B) + A_r \cdot B + B_r \cdot A - A_r \cdot B_r \quad (2.20)$$

De esta expresión es la que se observa que hay tres términos que se pueden realizar mediante corrimiento, $A_r \cdot B$, $B_r \cdot A$, $A_r \cdot B_r$. El término restante $(A_r - A) \cdot (B_r - B)$ es omitido con la observación de que su implementación en hardware es compleja y que el peso en el resultado final suele ser pequeño, de esta forma la aproximación propuesta de la ecuación 2.20 es :

$$A \cdot B \approx A_r \cdot B + B_r \cdot A - A_r \cdot B_r \quad (2.21)$$

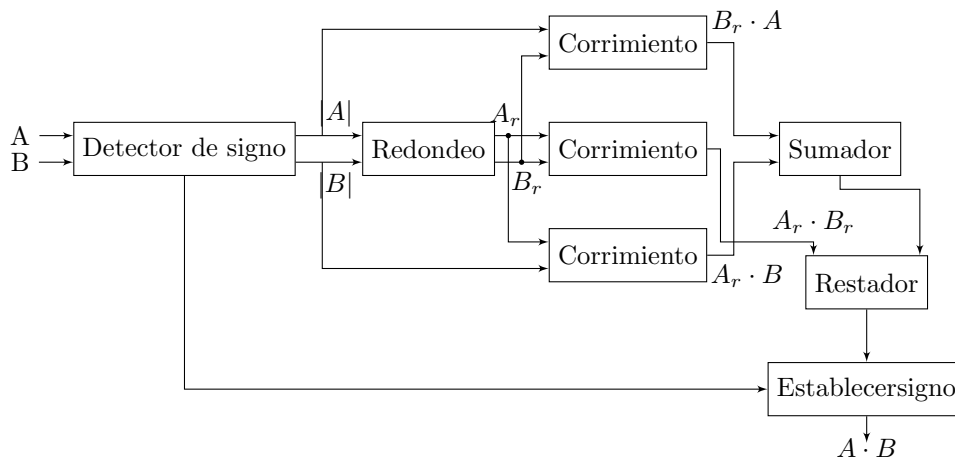


Figura 2.15: Diagrama de bloques de la implementación de RoBa

2.6. Divisores

Proceso más lento que el de la multiplicación, incluye cuatro elementos: dividendo, divisor, cociente y residuo. Existen varias técnicas para realizar la división entre ellas: secuencia de corrimiento-sumar/restar, método SRT, división multiplicativa y división en arreglo.

2.7. Divisores Aproximados

A pesar de que cuentan con un menor uso, en la literatura se encuentran propuestas interesantes, de las cuales se presentan a continuación tres, SEERAD, DAC y AXDNR, estos últimos dos presentan características similares en cuanto a la estructura de sus resultados mas no de su operación.

2.7.1. Divisor aproximado sin restauración (AXDnr, por sus siglas en inglés)

El diseño de este divisor toma como ventaja el hecho que la operación básica de la división es la resta. Una celda exacta de resta es muy similar a una celda exacta de suma como se puede apreciar en la siguiente tabla tomada de [28] que muestra sus respectivos comportamientos lógicos.

Tabla 2.8: Funciones de una celdas exactas para suma y resta

	Suma o Diferencia	C_{out} o B_{out}
Suma	$S = X \oplus Y \oplus C_{in}$	$C_{out} = X \oplus Y \cdot C_{in} + XY$
Resta	$D = X \oplus Y \oplus B_{in}$	$B_{out} = \overline{X \oplus Y} \cdot B_{in} + \overline{XY}$

En las relaciones anteriores B_{in} y B_{out} se refieren a los bit que se toman prestados de

la parte baja y parte alta respectivamente. Al igual que para los sumadores un restador de N bits se puede formar colocando en cascada N celdas de resta exactas.

Para generar un divisor exacto sin restauración (EXDnr, por sus siglas en inglés) [28] presenta la celda básica con la que se compone EXDCnr, la misma esta compuesta por una compuerta XOR y una celda de resta exacta (EXSC, por sus siglas en inglés). Una señal de control Q es introducida, esta permite que la celda se pueda configurar como un restador exacto o un sumador exacto $Q=0$ y $Q=1$ respectivamente, las relaciones de su comportamiento se muestran a continuación.

$$\begin{aligned}
 R &= X \oplus (Y \oplus Q) \oplus B_{in} \\
 B_{out} &= \overline{X \oplus Y \oplus Q} \cdot B_{in} + \overline{X}(Y \oplus Q)
 \end{aligned}
 \tag{2.22}$$

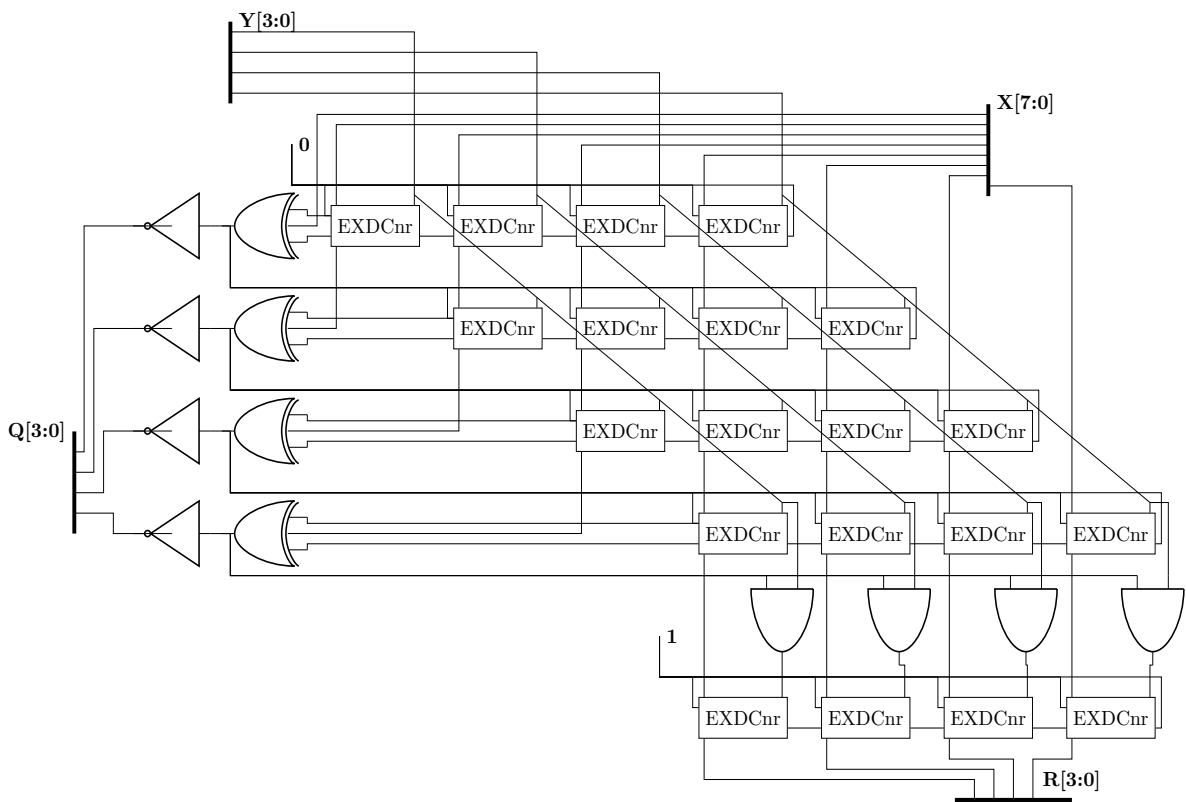


Figura 2.16: Implementación de un Divisor exacto sin restauración EXDnr: 8 bits dividendo, 4 bits divisor

Tres celdas de restadores aproximados (AXSC, por sus siglas en inglés) son propuestos en [28], las funciones lógicas que representan su comportamiento se muestran en la siguiente tabla.

Tabla 2.9: Funciones de celdas de resta aproximadas

	Diferencia	B_{out}
AXSC1	$X \oplus Y + B_{in}$	$\overline{X \oplus Y} \cdot B_{in} + \overline{X}Y$
AXSC2	$X \oplus Y \oplus B_{in}$	B_{in} o D
AXSC3	B_{out}	$\overline{X \oplus Y} \cdot B_{in} + \overline{X}Y$

Teniendo en cuentas las celdas anteriormente presentadas se puede hacer una modificación en la celda EXDCnr para obtener una versión aproximada AXDCnr mediante la sustitución de la celda de resta exacta EXSC por una de las celdas de resta aproximadas AXSC.

Sustitución similar se hace en el divisor exacto sin restauración para obtener la versión aproximada AXDnr, sustituyendo la celda exacta EXDCnr por la aproximación AXDCnr.

Cuatro configuraciones pueden ser obtenidas mediante la sustitución de estas celdas en el divisor exacto, sustitución vertical, horizontal, en cuadrado y triangular.

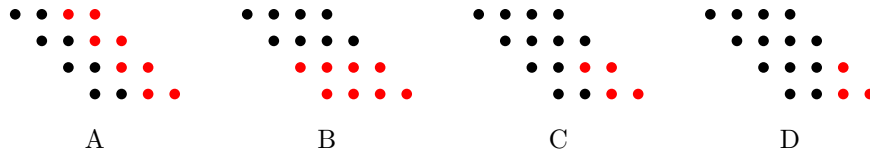


Figura 2.17: Posibles configuraciones para el divisor AXDnr: en rojo celdas exactas reemplazadas por celdas aproximadas. Sustitución (A) Vertical (B) Horizontal (C) Cuadrada (D) Triangular

2.7.2. Divisor aproximado con base redondeada de alta velocidad y eficiencia energética (SEERAD, por sus siglas en inglés)

La propuesta de [29] es un divisor de alta velocidad y aún así energéticamente eficiente. A su vez la precisión de SEERAD es configurable en etapa de diseño, esta característica hace que el área de hardware requerido y el retardo del componente puedan aumentar o disminuir según la precisión con la que se elija trabajar. La síntesis del algoritmo que implementa este divisor es cambiar el divisor a su valor redondeando más cercano y operar con este nuevo número.

Sea **A** el dividendo y **B** el divisor, el número más cercano se expresa de la siguiente manera $B_r = \frac{2^{K+L}}{D}$ donde 2^K es la base del logaritmo de B que se representará como B_f , de esta manera se aproxima la división mediante la siguiente ecuación.

$$\frac{A}{B} \approx \frac{A}{B_r} = \frac{A}{\frac{2^{K+L}}{D}} = \frac{D \cdot A}{2^{K+L}} = \frac{D \cdot A}{2^L \cdot B_f} \quad (2.23)$$

En la ecuación 2.23 se muestran dos parámetros importantes en la determinación de la precisión con la que se puede contar para este divisor, D y L (Tupla). Para determinar cual es la mejor tupla para la cantidad de bits con las que se quiere trabajar se debe realizar una búsqueda exhaustiva, la cual consiste en cambiar el valor de A desde 0 hasta el máximo y luego el error de dividir A con B_f es comparado al hacer la división con B, así la tupla con el menor error relativo será la elegida.

Si se utiliza la tupla encontrada para todos los anchos de bits, se obtiene una implementación con menor retardo pero a un costo de imprecisión mayor. Para mejorar la precisión se pueden considerar diferentes tuplas para cierta cantidad de bits dada, para

lograrlo se forman diferentes grupos dependiendo de la combinación de los bits más significativos de B, esto incrementa el retardo.

En [29] bajo la búsqueda exhaustiva se muestran cuatro niveles de precisión dados por la relación $\log_2|Group| + 1$, cabe destacar que al aumentar la cantidad de grupos se incrementa la precisión pero a su vez sobrecalentamiento [29]. En estos grupos propuestos el valor de D es seleccionado tal que el error relativo sea mínimo, a su vez para poder alcanzar una implementación en hardware más simple y con menor retardo se selecciona un valor de L para todos los grupos en cada nivel de precisión. Toda esta clasificación de grupos y niveles se muestran en la siguiente tabla.

Tabla 2.10: Valores en cada grupo y su correspondiente parámetro D y L

Grupo	Nivel de precisión	Índice	B	D	L
1	1	0	0...01X.....X	5	3
2	2	0	0...010X.....X	12	4
		1	0...011X.....X	9	
4	3	0	0...0100X....X	28	5
		1	0...0101X....X	24	
		2	0...0110X....X	20	
		3	0...0111X....X	17	
8	4	0	0...01000X....X	120	7
		1	0...01001X....X	108	
		2	0...01010X....X	97	
		3	0...01011X....X	88	
		4	0...01100X....X	82	
		5	0...01101X....X	76	
		6	0...01110X....X	70	
		7	0...01111X....X	66	

El procedimiento generado por esta estructura es como sigue, sea A y B números de N bits, se determina el signo tanto del divisor como del dividendo (se asume que ambos números se encuentren en complemento a dos), en el bloque de redondeo se determina la base de B en la forma de $2^K = B_f$, luego B_f es usado para determinar los valores D y L según el nivel de precisión elegido. Una vez que se tenga el valor de D se procede a realizar la multiplicación, para esta se utiliza unidades de corrimiento y un sumador.

Importante característica de este divisor es el tamaño de bits con los se realizan sus operaciones, la multiplicación generará un resultado de $2N$ bits, la unidad de corrimiento se encarga de correr la entrada $K+L$ bits hacia la derecha, al realizar este corrimiento no se desechan bits por lo que la salida de este será de $2N+L$, donde N son los bits de la parte entera y $N+L$ representan la parte fraccionaria del número, por último se determina el signo del resultado [29].

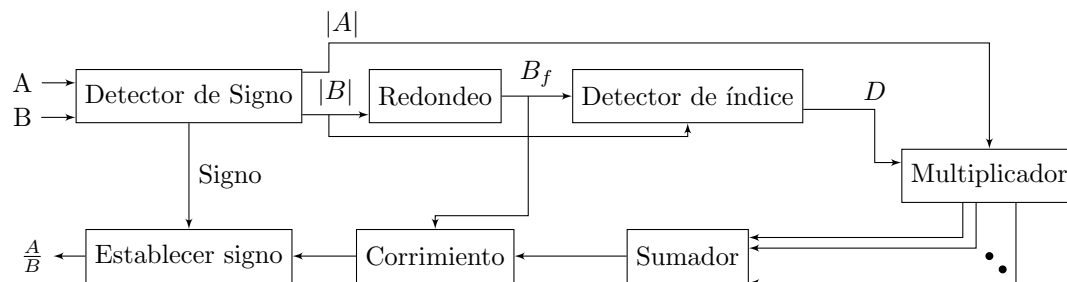


Figura 2.18: Diagrama de bloques de la implementación de SEERAD

2.7.3. Divisor dinámico de baja potencia para aplicaciones aproximadas (DAC, por sus siglas en inglés)

Similar a como trabaja el DRUM, toma como ventaja la importancia de algunos bits sobre otros en una representación numérica, con estos bits que cuentan con mayor peso se hace la operación de división en un divisor exacto central y utiliza algunos módulos extra para determinar cuales son estos bits más significativos. Entre estos elementos adicionales se encuentran detectores de uno conductor, multiplexadores y decodificadores al igual que los utilizados para el DRUM.

El primer paso del algoritmo que este propuesta utiliza es identificar el uno conductor en cada operando, seguido se define el parámetro k como el rango del divisor exacto, con este se pueden tener diversos intercambios entre precisión y otras métricas de diseño. A partir del uno conductor se eligen los siguientes $k-1$ bits, este conjunto es el que se dirige hacia el divisor exacto, los bits restantes son truncados [30].

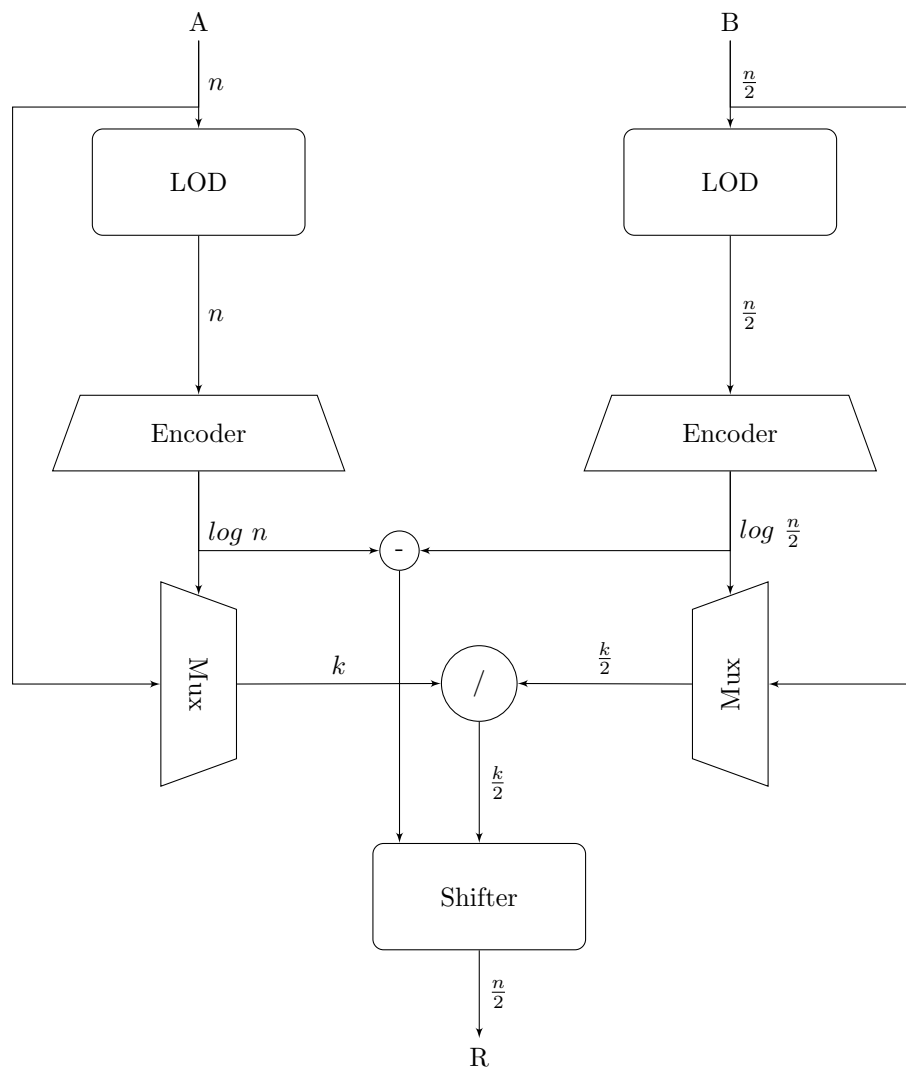


Figura 2.19: Diagrama de bloques de la implementación de DAC

Cabe destacar ciertas características, DAC esta diseñado para mantener un radio de 2/1 entre el dividendo y el divisor por lo cual ante dos operandos de n bits y de $n/2$ respectivamente la salida será de $n/2$ bits. Si el uno conductor se encuentra entre los k bits no se necesita truncado y este se traslada directamente al divisor exacto central [30].

2.8. Métricas

2.8.1. Producto retardo potencia (PDP, por sus siglas en inglés)

Al igual que la energía la unidad de medida es el Julio, se considera cuando se quiere optimizar tanto rendimiento como consumo de potencia. Se define por:

$$PDP = P_{promedio} \cdot T_{cp}; \quad (2.24)$$

Donde T_{cp} es el retardo dado por la ruta de retardo crítica y $P_{promedio}$ es la potencia promedio disipada por un circuito durante un período [14].

2.8.2. Distancia de error (ED, por sus siglas en inglés)

Se define básicamente por la distancia aritmética entre dos números es decir el valor absoluto de la diferencia en este caso del resultado aproximado y el resultado preciso de la suma [13].

$$ED = |a - b| = \left| \sum_i a[i] * 2^i - \sum_j b[j] * 2^j \right| \quad (2.25)$$

Donde a denota el resultado del sumador preciso y b el del sumador aproximado.

2.8.3. Distancia media de error (MED, por sus siglas en inglés)

Se define distancia media de error como el promedio de ED para un conjunto dado de vectores de entrada siempre y cuando se trabaje con entradas de distribución uniforme [31].

$$MED = E[ED] = \sum_i ED_i * P(ED_i) \quad (2.26)$$

donde $P(ED_i)$ es la probabilidad de ED_i .

2.8.4. Tasa de error (ER, por sus siglas en inglés)

Porcentaje de todas las salidas erróneas ante todas las salidas [31].

$$ER = \sum_i P(ED_i), \text{ si } ED_i \neq 0 \quad (2.27)$$

2.9. Bibliotecas de circuitos aritméticos aproximados

2.9.1. EvoApprox8b: Biblioteca de sumadores y multiplicadores aproximados para diseño de circuito y evaluación comparativa de métodos de aproximación

En [32] una biblioteca con componentes aritméticos aproximados es propuesta, se compone por 430 sumadores y 471 multiplicadores aproximados de 8 bits, incluye modelos en verilog, Matlab y C para cada componente.

Pámetros como área, potencia y retardo son estimados con bibliotecas de TSMC de

180 y 45 nm. Algunas métricas relativas al comportamiento del error también son generadas para cada componente, con todos estos parámetros una búsqueda bajo ciertas restricciones de diseño se puede realizar permitiendo obtener rápidamente el modelo del componente que mejor se ajuste. Esta biblioteca se encuentra disponible en la web señalada en [32].

2.9.2. Ipaclib: Biblioteca de multiplicadores y sumadores aproximados

Una metodología para generar y explorar el espacio estructural de un multiplicador aproximado usando variantes de módulos elementales como multiplicadores o sumadores tanto aproximados como precisos es presentada en [33]. Se generan varios diseños los cuales también se sintetizan y se verifican utilizando herramientas como Synopsys Design Compiler y Modelsim respectivamente.

Varios sumadores y multiplicadores componen esta biblioteca que se puede encontrar en la web señalada en [33], en esta se pueden encontrar tanto la descripción en VHDL así como un modelo de comportamiento en C.

Capítulo 3

Diseño

En este capítulo se presenta la metodología para el diseño de la estructura de la herramienta y de los archivos que la componen. Estos archivos incluyen código fuente, archivo de configuración y bibliotecas de plantillas. También se muestra las consideraciones para la descripción de las unidades aritméticas, para la elaboración de archivos de prueba y para la creación de scripts, así como sus debidas restricciones.

3.1. Estructura de la herramienta

3.1.1. Generalidades

Es importante conocer el ambiente donde se desarrolla la herramienta. El sistema operativo utilizado es la distribución Ubuntu 14.04 de LINUX, se utiliza el lenguaje de programación C++ para el desarrollo del código fuente, este código es editado mediante el entorno de desarrollo integrado (IDE, por sus siglas en inglés) Eclipse el cual es de fuente abierta.

3.1.2. Archivos base

Se crean dos archivos necesarios para el correcto funcionamiento, el primero es un archivo “makefile” con el cual se puede compilar o recompilar (dado el caso de modificación o inserción de código) el código fuente y generar un ejecutable. El segundo documento es un archivo de configuración el cual posteriormente será leído por la herramienta para conocer las rutas de las herramientas externas así como de las bibliotecas de celdas necesarias para distintos procesos. Este archivo de configuración es el que el usuario modificaría para establecer lo anteriormente mencionado.

3.1.3. Organización de directorios

La carpeta principal de la herramienta contiene dos subcarpetas. La primera es donde se almacena el código fuente y modelos en C++ de los circuitos aritméticos desarrollados. La segunda contiene plantillas de los archivos que son modificables por la herramienta estos incluyen modelos en verilog, scripts y archivos de pruebas (testbenchs).

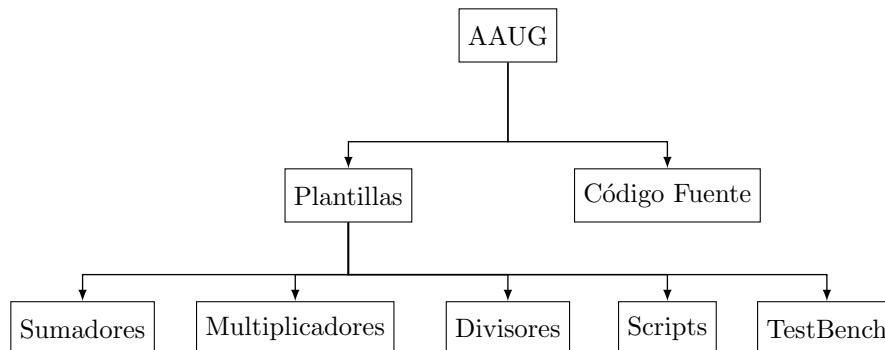


Figura 3.1: Estructura de carpeta general de la herramienta

Una carpeta extra es generada cuando se ejecuta la herramienta, esta incluye las unidades aritméticas solicitadas, reportes de área, potencia y retardo, así como sus respectivos archivos para simulación, síntesis y simulación post-síntesis. La ruta de esta carpeta por defecto es en la carpeta principal pero esto también es modificable en el archivo de configuración. Esta carpeta presenta una estructura jerárquica que facilita el intercambio y búsqueda de archivos para los procesos que se deseen realizar.

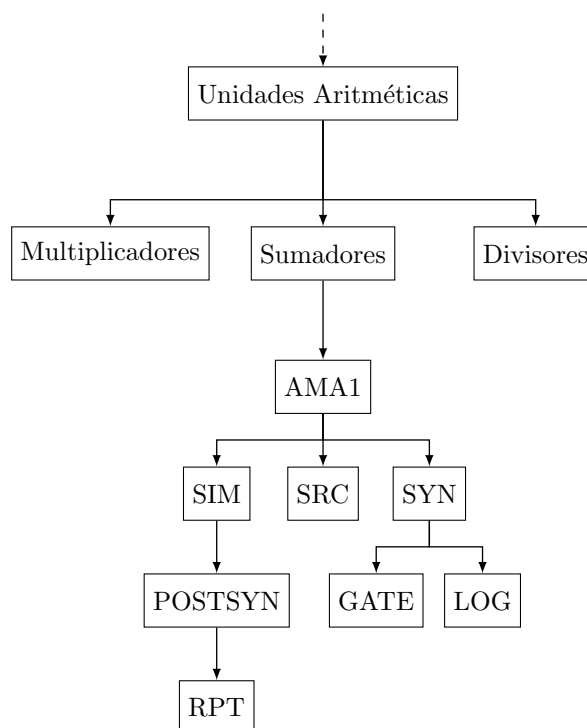


Figura 3.2: Estructura de carpeta de unidades generadas

3.1.4. Tipo de archivos

Debido a los varios procesos que la herramienta puede ejecutar se generan múltiples archivos necesarios o intermedios para cada uno, estos archivos se almacenan en la carpeta respectiva de cada proceso, es importante conocerlos con el fin de entender la organización de la herramienta.

Tabla 3.1: Referencia a archivos generados

Tipo	Extensión	Descripción
Achivo fuente Verilog	*.v	Archivos con las descripciones de las unidades aritméticas
Value Change Dump format	*.vcd	Guarda información de traza de transición de señal
Switching Activity Interchange Format	*.saif	Guarda información de traza de transición de señal
Tool command language	*.tcl	Almacenan scripts necesarios para ejecutar herramientas externas o procesos
Standard Delay Format	*.sdf	Almacena información de temporizado
Biblioteca de tecnología	*.db	Formato de Synopsys, contiene información de celdas de determinada biblioteca de algún fabricante
Reportes	*.rpt *.log	Guardan información de los procesos generados
Plantillas	*.txt	Archivos con las plantillas de scripts y archivos de prueba

3.2. Descripción de hardware

Los circuitos aritméticos presentados en el capítulo 2 se describen en el lenguaje de descripción de hardware Verilog, debido a su sencillez para detallar el comportamiento y estructura de las unidades, además de que se cuenta con experiencia previa en este lenguaje.

3.2.1. Sumadores

Se presenta a continuación las características de diseño de los sumadores que se implementaron, así como las restricciones de su diseño. Para las restricciones se debe recordar los parámetros m , n , l , k presentados en el capítulo 2 para cada sumador.

3.2.1.1. Sumadores de baja Potencia

Se comienza con los sumadores de baja potencia, como ya se mostró en el capítulo anterior se componen por sumadores de 1 bit, los cuales pueden ser aproximados o precisos, de estos sumadores es necesario conocer las ecuaciones que describen el comportamiento lógico de los bits de suma y de acarreo para poder ser trasladadas a Verilog .

Las expresiones necesarias para los sumadores aproximados AXA1, AXA2, AXA3, TGA1, TGA2 y CFA ya se presentaron en el capítulo 2.

En el caso de los otros sumadores se realiza un paso previo para obtener las ecuaciones, el cual consiste en utilizar el software libre Logic Friday que permite introducir tablas de verdad y/o circuitos para obtener sus respectivas expresiones booleanas.

La instanciación de estos módulos en el sumador de baja potencia se hace mediante bloques generadores incluidos en Verilog-2001, estos bloques son utilizados en la mayoría de los circuitos aritméticos desarrollados en este trabajo.

Restricción: $n \leq m$

3.2.1.2. Sumadores de alto rendimiento

En el caso de los sumadores de alto rendimiento todos tienen una característica en común y es que su comportamiento lógico puede ser representado por el GeAr bajo ciertas condiciones de R y P. Por lo que la única descripción que se realiza es la de este sumador, el cual sigue la estructura presentada en la figura 2.12. A continuación las combinaciones de R y P para los demás sumadores [9]:

- ACAI: $R=1$, $P=L-1$
- ACAII: $R=L/2$, $P=L/2$
- ETAII: $R=L/2$, $P=L/2$
- GDA: P es igual para todos los subsumadores;

Restricción ACAI: $l < m$

Restricción ETAII y ACAII: $l < m$, $l \% 2 == 0$

Restricción GeAR y GDA: $l < m$, $(m - l) \% R == 0$

3.2.1.3. LOA

El sumador LOA tiene características intermedias. Al utilizar compuertas OR para aproximar su parte baja, simplifica el hardware necesario para generar el bit de suma así como a su vez corta la propagación del acarreo, características de los sumadores de baja potencia y sumadores de alto rendimiento respectivamente. Por lo que su descripción no es tan modular como los sumadores anteriormente mencionados.

Restricción: $n \leq m$

Una síntesis de los archivos de sumadores desarrollados y sus respectivas dependencias se muestran en la siguiente tabla.

Tabla 3.2: Biblioteca de Sumadores

Módulos	Dependencias	Descripción
AMA, AXA, CFA, InXA, TGA, VAXA, FA	-	Módulos AFA de sumadores aproximados, sus respectivas variaciones y un sumador completo
RCA	FA	Sumador Exacto de m bits
LOA	FA	Sumador características intermedias
LPA	RCA AFA	Sumador de baja potencia
GeAr	RCA	Sumador configurable de alto rendimiento
GeAr2	-	Sumador configurable de alto rendimiento segunda versión

3.2.2. Multiplicadores

Cabe mencionar que la mayoría de descripciones de hardware están parametrizados por lo que las modificaciones que hace la herramienta son básicamente en los parámetros de cada módulo. En el caso de los multiplicadores presentados, estos cuentan con módulos que no son tan flexibles como decodificadores y multiplexadores los cuales deben de ser específicamente diseñados para la cantidad de bits totales y aproximados con las que se desean trabajar. En este caso la herramienta no solo modifica parámetros sino que estructura interna de estos módulos rígidos.

3.2.2.1. DRUM

El DRUM por ejemplo cuenta con módulos como decodificadores, multiplexadores, unidad de corrimiento y detectores de uno conductor (LOD), de los que los dos primeros son básicamente construidos durante la ejecución de la herramienta y los otros dos son solo modificados. Todos estos módulos descritos en Verilog se pueden encontrar en un repositorio de la Universidad de Brown (Brown University Scale Lab, <https://github.com/scalelab>). Se toman estos módulos respetando la licencia de distribución y se parametrizan para que sea más sencilla su modificación por la herramienta.

Restricción: $2 < k < N$, $N < 32bits(Modelsim)$

3.2.2.2. RoBa

Para realizar la estructura de la figura 2.15 se reciclan algunos módulos del DRUM, por ejemplo para la unidad de redondeo se utiliza el módulo detector de uno conductor y el codificador, las unidades de corrimiento también son las mismas.

Las operaciones de suma y resta se hacen mediante los operadores respectivos permitidos en Verilog. En cuanto a las operaciones sobre el signo, estas se omiten asumiendo que se trabajará con números positivos.

Tabla 3.3: Biblioteca de Multiplicadores

Módulo	Dependencias	Descripción
RAM	RCA	Multiplicador exacto en arreglo
LOD	-	Detector de uno conductor
Mux	-	Multiplexador
Encoder	-	Codificador
Barrel_Shifter	-	Unidad de corrimiento
DRUM	LOD Mux Encoder Barrel_Shifter	Multiplicador aproximado
RoBa	LOD Encoder Barrel_Shifter	Multiplicador aproximado

3.2.3. Divisores

Al igual a como se ha trabajado los divisores incorporan elementos reciclados de los diseños anteriores, especialmente módulos usados en el DRUM.

3.2.3.1. AXDnr

Varios módulos se describen para lograr el comportamiento de este divisor. Primero se parte de los módulos base que son los que realizan la resta de 1 bit, estos son AXSC1, AXSC2, AXSC3 los cuales cuentan con las expresiones booleanas necesarias para ser trasladados a Verilog. Con estos módulos y siguiendo la lógica de los sumadores de baja potencia se realiza un restador de baja potencia (LPS).

El módulo básico para realizar una operación de división en arreglo es el EXDCnr, el cual se traslada a verilog de manera que por medio de parámetros se pueda cambiar de ser un módulo exacto a uno aproximado AXDCnr.

Se realiza una modificación del LPS llamada MLPS en la cual se instancia según parámetros un módulo EXDCnr o AXDCnr, siguiendo la estructura de la figura 2.16 este módulo se puede observar como las filas.

Un módulo auxiliar RCC es realizado para realizar la corrección del residuo, se compone por las compuertas AND y módulos EXDCnr. Todos estos módulos anteriormente mencionados se instancian en el módulo del divisor AXDNR donde se termina de describir el comportamiento lógico.

La manera de construcción de este divisor permite mediante parámetros tres de las cuatro configuraciones propuestas las cuales son sustitución vertical, horizontal y en cuadrado. Estos parámetros son el número de columnas (R) y el número de filas (P).

Restricción: $N \% 2 == 0$, $N \geq 6$, $R \leq N/2$, $P \leq N/2$

3.2.3.2. DAC

Esta propuesta cuenta con una estructura básicamente exacta a la del DRUM, la única diferencia es como se implementa la operación precisa reducida. En el caso del DRUM se utiliza el operador aritmético permitido en Verilog de multiplicación para hacer dicha operación, dejando así a la herramienta de síntesis la tarea de traducir esto a hardware.

Este paso en el DAC si se especifica y se utiliza una instanciación de un divisor exacto en arreglo como el de la figura 2.16. Un divisor exacto de este tipo cuenta con dos salidas el cociente y el residuo, para que los resultados puedan representarse en la cantidad de bits de salida del cociente se tiene que cumplir que el divisor sea mayor a la mitad más alta del dividendo.

Esta característica hace que los errores aumenten al compararlos con modelos exac-

tos de división. Por lo que hay que tomar en cuenta que esta característica aumenta la cantidad de errores ya introducidos por la lógica de aproximación del DAC.

Restricción: $n \% 2 == 0$, $k \% 2 == 0$, $4 \leq k < n$

3.2.3.3. SEERAD

La estructura presentada en la figura 2.18 se modifica un poco para permitir la reutilización de los módulos ya desarrollados sin perder la lógica de aproximación propuesta.

Para realizar la multiplicación en [29] se propone utilizar módulos de corrimiento y luego sumar los productos parciales, en lugar de este proceso se utiliza un multiplicador en arreglo exacto de la cantidad de bits necesaria para representar el parámetro D de la tabla 2.10. Con esto queda limitado el nivel de precisión a la cantidad de bits del divisor a como se muestra en la siguiente tabla.

Tabla 3.4: Limitaciones de precisión en SEERAD

Cantidad de bits	Precisión
3	1
4	≤ 2
5	≤ 3
6	≤ 3
≥ 7	≤ 4

Esta implementación se realiza para números positivos por lo que se omite el detector y corrección de signo de la estructura original.

Restricción: $n > 2 == 0$, $Precisión \leq 4$

Tabla 3.5: Biblioteca de divisores

Módulo	Dependencias	Descripción
AXSC1,AXSC2,AXSC3 EXSC	-	Restadores aproximados y restador preciso
EXDCnr	AXSC EXSC	Módulo configurable para suma o resta precisa o aproximada
MLPS	EXDCnr	Modificación a restador de baja potencia
RCC	EXDCnr	Circuito Corrector de residuo
Xor3	-	Simple compuerta xor de 3 entradas
AXDNR	Xor3 RCC MLPS	Divisor
Barrel_shifter_Right	-	Circuito de corrimiento a la derecha
SEERAD	Encoder LOD Mux RCA_ArrayMultiplier Barrel_shifter_Right	Divisor
Simpledivider	AXDNR	Conjunto de módulos necesarios para un divisor en arreglo exacto, módulo superior CEXD
DAC	Encoder LOD Mux Barrel_Shifter CEXD	Divisor

3.3. Archivos de Prueba

Los archivos de prueba (TestBench) son importantes para los procesos que la herramienta realiza, cuatro de estos son generados de manera que se ajusten a características similares entre los componentes. La cantidad supera en uno a los tipos de unidades aritméticas que se pueden generar, esto es debido a que la estructura del resultado del divisor SEERAD es muy diferente a los otros dos divisores por lo que se optó por generar un archivo de prueba único para este divisor.

```

Definir escala de tiempo;
Módulo : TESTBENCH
Definir Parámetros;
Definir entradas y salidas;
Definir arreglo;
Instanciar módulo superior;
Inicialización: cargar datos de prueba en arreglo;
for i ← 0 to tamaño arreglo do
    for j ← 0 to tamaño arreglo do
        Cargar en entrada 1 valor de arreglo en posición i;
        Cargar en entrada 2 valor de arreglo en posición j;
        Esperar;
        Escribir en archivo de salida;
    end
end
end

```

Algoritmo 1: TestBench General

3.4. Vinculación con herramientas externas

La herramienta propuesta hace uso de otras para realizar estimaciones de potencia, área y retardo. La ejecución de estas herramientas se hace mediante el uso de scripts, los cuales se diseñaron de la forma más general posible para que puedan ser sencillamente modificados por la herramienta propuesta durante su ejecución.

3.4.1. Script para compilación y simulación

Se crean tres scripts para ser ejecutados en la herramienta Modelsim, que es la encargada de realizar este paso, cada uno de estos corresponde a los tres diferentes tipos de unidades aritméticas que se pueden generar. Todos estos scripts siguen el flujo que se muestra en la siguiente imagen.

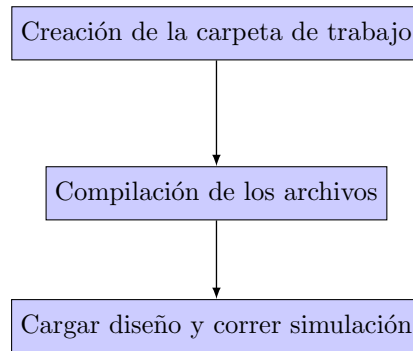


Figura 3.3: Flujo de simulación

La carpeta de trabajo es donde se almacenan los archivos compilados por ModelSim, por defecto su nombre es “*Work*”. Posteriormente son compilados los archivos, en este caso al utilizar Verilog como lenguaje de descripción la opción para compilar es *vlog*.

Luego de estos pasos se llama al comando de simulación *vsim* usando el nombre del módulo superior, en este caso para todos los archivos de prueba generados el nombre utilizado es “TESTBENCH”. Después se produce un archivo de cambios de valor (VCD) y se ejecuta la simulación mediante el comando *run*.

Como se esta trabajando mediante línea de comandos, a este script hay que agregarle el comando *quit* para que salga de ModelSim cuando se termine de ejecutar este proceso.

3.4.2. Script para Síntesis

Para realizar este proceso se utiliza la herramienta *Design Compiler (DC)* de Synopsys, para esto se crean cuatro archivos de los cuales 1 es general para cada tipo de unidad aritmética aproximada y los 3 restantes son uno para cada uno.

Se sigue la estrategia de compilación de arriba a abajo (Top-Down) que se encuentra en la guía para el usuario de DC, esta estrategia compila juntos el módulo superior y sus submódulos. El flujo básico para realizar la síntesis se muestra a continuación

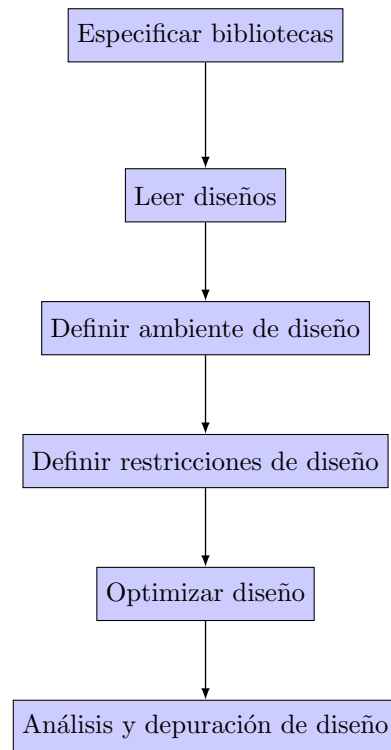


Figura 3.4: Flujo de Síntesis

Antes de especificar las bibliotecas, en el script se definen ciertas variables que son las que la herramienta propuesta modificará, estas variables son los nombres de las unidades, las rutas donde se encuentran y las rutas para los archivos de salida que se van a generar. La especificación de las bibliotecas se realiza con el archivo general anteriormente mencionado.

La biblioteca principal es la que se conoce como biblioteca objetivo (*target.library*) que es utilizada para generar las listas de red (netlist) y definir condiciones operativas del diseño. Otra biblioteca que se especifica en el script es la de enlace (*link.library*), esta es la que DC utiliza para resolver referencias de celdas. A las dos bibliotecas se les asigna el mismo valor.

A la hora de leer diseños DC permite dos maneras mediante el comando *read* y la combinación *analyze-elaborate*, para sintetizar se utiliza la última opción, analizando primero cada submódulo que compone el módulo superior incluyéndolo y luego elaborando este además de definirlo como el diseño actual con el comando *currentdesign*.

Para el objetivo de esta síntesis no se define el ambiente del diseño ni restricciones de optimización. Esto porque primero el ambiente queda a criterio del usuario que utilice la herramienta propuesta y en cuanto a la optimización, lo que se desea es que el proceso respete lo más posible las estructuras de los circuitos propuestos, igualmente queda a criterio del usuario la inclusión de estas restricciones. Las únicas restricciones que puede presentarse al ejecutarse este script son las de reglas, que están implícitas en la biblioteca de celdas que se utilice.

La múltiple instanciación de un mismo diseño es algo que se debe resolver antes de compilar, para esto se utiliza el comando *uniquify* que lo que hace es realizar una copia y renombrar un diseño para cada instancia. La mayoría de los circuitos aritméticos

aproximados realizados utiliza múltiple instanciación por lo que esta opción es tomada en cuenta para el desarrollo del script.

Luego se procede a llamar al algoritmo de síntesis por medio del comando *compile* el cual también tiene muchas opciones de optimización, queda de nuevo a criterio del usuario modificar esta línea en la plantilla para lograr los objetivos que se deseen.

Por último se generan las listas de red a nivel de compuertas (gate-level netlist) y un archivo *.sdf que contiene la información sobre retardo, además se generan otros reportes como área, tiempo y potencia.

3.4.3. Script para simulación post-síntesis

La simulación Post-Síntesis utiliza ModelSim por lo que su flujo difiere en muy poco con respecto al mostrado en la figura 3.3. Las diferencias destacan en sobre que se aplica la simulación, primero se debe de contar con la biblioteca de celdas utilizadas para la síntesis descritas en Verilog y estas se deben de acceder para la simulación. La compilación se hace en este caso sobre la biblioteca anteriormente mencionada, la red a nivel de compuertas y el archivo de pruebas. Por último a la hora de ejecutar el simulador se escribe con la opción de cargar el archivo que contiene la información de retardo generado por el script de Síntesis.

3.4.4. Script para estimación de potencia post-síntesis

Se utiliza Prime Time de Synopsys par realizar esta estimación, el script sigue un flujo similar al de la síntesis, en este caso se utiliza el comando *read* para leer la red a nivel de compuertas. Prime Time permite dos modos de análisis el promedio y el basado en el tiempo, el primero calcula la potencia promedio basado en tasas de conmutación y el segundo usa simulación de actividad a nivel de compuerta para estimar la potencia pico y promedio. Ambos modos son seleccionables en la plantilla que se desarrolló.

3.5. Núcleo de la herramienta

A como se ha mencionado anteriormete la herramienta se desarrolla en el lenguaje de programación C++, se hace uso de funciones para realizar los diversos procesos necesarios para su funcionamiento. Los archivos se desarrollan de manera jerárquica para facilitar el llamado a funciones y la modificación.

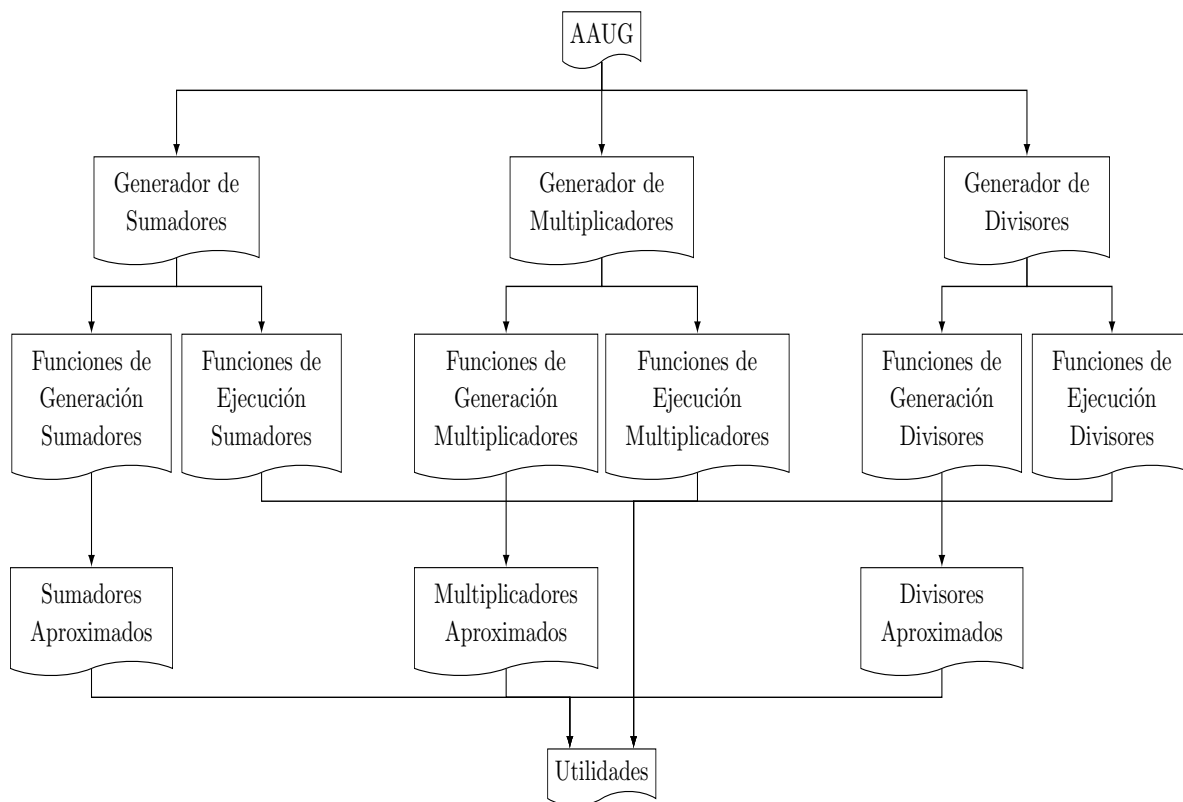


Figura 3.5: Estructura central de la herramienta

Se puede observar en la figura 3.5 que la herramienta tiene tres ramas similares, a continuación se explica cada uno de los elementos que las componen.

3.5.1. Utilidades

Se realiza un conjunto de funciones generales que efectúan procedimientos simples pero necesarios para otras etapas de la herramienta. Dentro de este documento se pueden encontrar funciones de conversión entre tipos de datos, creación de directorios y resumen de resultados entre otras. Es utilizado en la mayoría de los procedimientos o funciones que se encuentran sobre el en la figura 3.5.

3.5.2. Unidades aproximadas

En estos archivos se definen los modelos de referencia de las unidades aritméticas aproximadas, cada una se escribe como una función para que pueda ser llamada donde se requiera. Cada modelo se realiza en un archivo con extensión .cpp y luego todos se recopilan en un archivo cabecera .h así se facilita la inserción de nuevos modelos posteriormente.

3.5.3. Funciones de Generación

A como su nombre lo dice son las encargadas de producir diferentes tipos de archivos, entre ellos están las unidades aritméticas descritas en Verilog, las métricas, el archivo

de estímulo y TestBench para simulación, así como el que se necesita para la simulación Post-Síntesis. El algoritmo que sigue la mayoría de estas funciones es el siguiente:

```

function GENERAR(parámetros)
  Abrir archivo de salida;
  Abrir plantilla;
  while no línea final de plantilla do
    if línea leída == línea llave then
      | Bandera de modificar activa;
    else
      if Bandera de modificar then
        | Modificar línea y escribir en archivo de salida;
        | Bandera de modificar desactivada;
      else
        | Escribir línea en archivo de salida
      end
    end
  end
end

return

```

Algoritmo 2: Funciones de Generación

Pequeñas diferencias se dan en el caso de los archivos de estímulo y métricas. En lugar de abrir una plantilla y recorrerla, lo que se hace es escribir los resultados directamente al archivo de salida en el formato que sea necesario para que concuerden con lo esperado por el testbench. Una característica importante a mencionar para los divisores es que a excepción del divisor SEERAD, las representaciones binarias de los números de prueba que se colocan en el archivo de estímulo son una concatenación del divisor y dividendo debido a la característica de que el divisor debe tener la mitad de cantidad de bits que el dividendo, esto es tomado en cuenta en los archivos de pruebas desarrollados anteriormente.

3.5.4. Funciones de ejecución

Se encargan de generar los scripts para la ejecución de las herramientas Modelsim, DesignCompiler y PrimeTime, además de dar la orden de ejecución. Para la creación y modificación de los script de plantilla siguen el mismo algoritmo que las funciones de generación. Por otra parte se le agrega al algoritmo, las instrucciones necesarias para hacer llamados a la línea de comandos y ejecutar los scripts, así como ejecutar herramientas secundarias como el conversor *vc2saif* de Synopsys que permite crear el archivo .saif necesario para las estimaciones de potencia.

3.5.5. Generador de unidades

Son los encargados de reunir tanto las funciones de generación y ejecución, hacen el llamado a la función respectiva dependiendo del proceso deseado. El algoritmo que

siguen es el siguiente:

```

function GENERARUNIDADES(parámetros)
for  $i \leftarrow 0$  to proceso do
    switch  $i$  do
        case Generar
            Crear directorios;
            Crear la unidad aritmética aproximada;
        end
        case Simular
            Crear Testbench;
            Crear archivo de estímulo;
            Crear y ejecutar script de simulación pre-síntesis Modelsim;
        end
        case Validar
            Comparar respuesta de simulación en Modelsim con respuesta de
            modelos de referencia;
        end
        case Sintetizar
            Crear y ejecutar script DesignCompiler;
        end
        case Simulación Post-Síntesis
            Crear nuevo TestBench;
            Crear y ejecutar script de simulación post-síntesis Modelsim;
            Crear y ejecutar script de PrimeTime;
        end
    endsw
end
return

```

Algoritmo 3: Generadores de unidades

3.5.6. AAUG

Este es el archivo que reúne todas las funciones anteriormente descritas, en él se encuentra la función encargada de recibir los parámetros por línea de comandos, la misma verifica si estos cumplen las condiciones y/o restricciones para la unidad aritmética aproximada deseada, si es así, hace el llamado y traslada los parámetros a la función generadora de unidades respectiva.

3.6. Flujo de la herramienta

Como se ha podido deducir la herramienta trabaja con cinco procesos, Generación, Simulación, Validación, Síntesis y Simulación Post-Síntesis, cada uno de estos depende de su predecesor por lo que es una herramienta con un flujo lineal hasta el proceso que

se desee.

3.6.1. Generación

En este proceso se generaran las descripciones en Verilog de la unidad aritmética que se desea, además de los directorios necesarios para el proceso al cual se desea llegar.

3.6.2. Simulación pre-síntesis

Este proceso necesita las descripciones de las unidades generadas en el proceso anterior, además del archivo de estímulo, la respuesta de los modelos de referencia y el testbench que la herramienta genera si se indica este proceso. Los archivos importantes que se generan aquí son la respuesta del circuito al estímulo que se guarda en un archivo de texto plano extensión .txt y el archivo de cambios de valor .vcd.

3.6.3. Validación

Este proceso es muy sencillo, solo carga la respuesta de los modelos de referencia y la compara con la respuesta del circuito descrito en Verilog, indica en pantalla si obtienen el mismo resultado o no.

3.6.4. Síntesis

El primero paso que realiza es convertir el archivo de cambios de valor al formato .saif. Luego se carga este archivo junto con las descripciones de hardware y la biblioteca de celdas. Ejecuta la síntesis obtiene la lista de red a nivel de compuerta (gate-level netlist) .v y el archivo .sdf que guarda información sobre temporizado necesaria posteriormente, además se obtienen reportes de área, potencia y retardo.

3.6.5. Simulación Post-Síntesis

En este caso se ejecuta la simulación y luego se hace una estimación de potencia sobre los resultados. Al igual que en la simulación, se necesita el archivo de estímulo y un testbench un poco modificado que genera la herramienta si se desea este proceso, además esta simulación se hace sobre la lista de red y se toma en cuenta las características de temporizado que se encuentran indicadas en el archivo .sdf respectivo. Por último es necesario la biblioteca de celdas descrita en Verilog. La salida de este paso es un archivo de cambio de valor .vcd que se transforma nuevamente al formato .saif, esto para poder realizar los dos modos de estimación de potencia que permite PrimeTime, junto con estos dos archivos y la lista de red se obtiene un reporte de potencia.

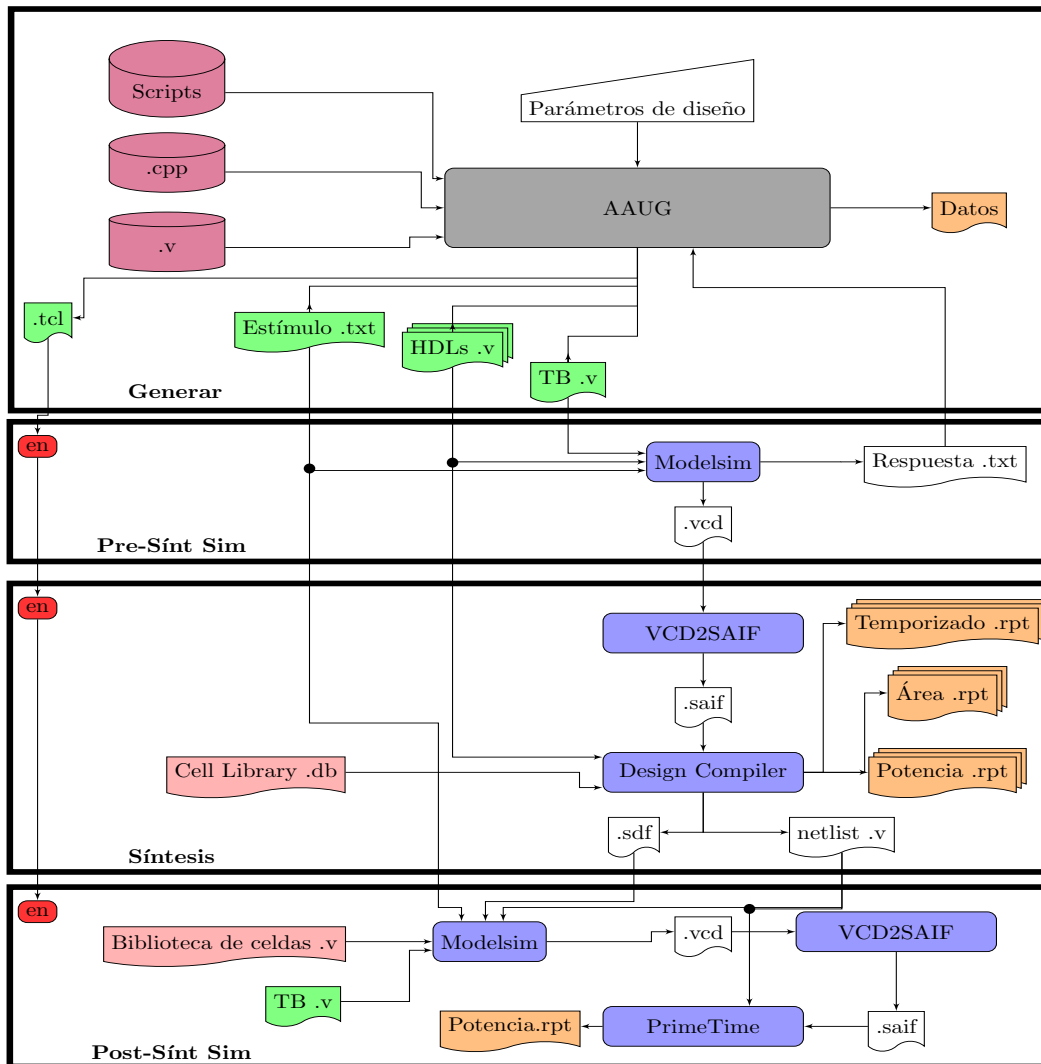


Figura 3.6: Flujo de herramienta

Capítulo 4

Resultados

Se presenta a continuación las pruebas de funcionamiento realizadas a la herramienta bajo diversos casos para los cuales se puede emplear, además de análisis sobre los resultados de las unidades aritméticas generadas en cada caso.

4.1. Generación de datos para comparación

Una de las situaciones que facilita el proceso de elección de algún componente en etapa de diseño, es contar con datos varios que permitan una comparación entre posibles unidades. El caso a continuación presenta la comparación de diversas métricas para los sumadores de baja potencia, en este caso los parámetros de diseño son un tamaño de sumador de 10 bits con aproximaciones de 1 a 8 bits.

4.1.1. Ejecución de herramienta

Para esto se crea un script que permita la llamada recursiva a la herramienta, variando los parámetros necesarios para generar todos los sumadores de baja potencia. Todos estos parámetros y la estructura necesaria para hacer una llamada a la herramienta están debidamente explicados en el manual de usuario en el apéndice [A](#).

```

GenerateData.tcl x
1 bw="10"
2 aprox="1"
3 sed -i '3c\\FilePath=Adders_Blocks/LPA/Adders_10_1' config.cfg
4 ./AAUG -a AMA1 -bw $bw -l $aprox -psy -full -ALL
5 ./AAUG -a AMA2 -bw $bw -l $aprox -psy -full -ALL
6 ./AAUG -a AMA3 -bw $bw -l $aprox -psy -full -ALL
7 ./AAUG -a AMA4 -bw $bw -l $aprox -psy -full -ALL
8 ./AAUG -a AMA5 -bw $bw -l $aprox -psy -full -ALL
9 ./AAUG -a AXA1 -bw $bw -l $aprox -psy -full -ALL
10 ./AAUG -a AXA2 -bw $bw -l $aprox -psy -full -ALL
11 ./AAUG -a AXA3 -bw $bw -l $aprox -psy -full -ALL
12 ./AAUG -a VAXA -bw $bw -l $aprox -psy -full -ALL
13 ./AAUG -a LOA -bw $bw -l $aprox -psy -full -ALL
14 ./AAUG -a TGA1 -bw $bw -l $aprox -psy -full -ALL
15 ./AAUG -a TGA2 -bw $bw -l $aprox -psy -full -ALL
16 ./AAUG -a InXA1 -bw $bw -l $aprox -psy -full -ALL
17 ./AAUG -a InXA2 -bw $bw -l $aprox -psy -full -ALL
18 ./AAUG -a InXA3 -bw $bw -l $aprox -psy -full -ALL
19 ./AAUG -a CFA -bw $bw -l $aprox -psy -full -ALL

```

Figura 4.1: Script para generación de sumadores de baja potencia

Como se puede observar en la figura 4.1 el proceso final al que se desea llegar es Post-síntesis, para este proceso y para la síntesis se utilizó la biblioteca genérica de Synopsys de 90nm disponible en la web para la comunidad universitaria. También cabe destacar que se le añade al script en la línea 3 la modificación del archivo de configuración de la ruta donde se desean colocar los resultados. Es en esta ruta donde se construye el conjunto de carpetas mostrado en la figura 3.2 y es en la carpeta superior donde la herramienta provee dos archivos de resumen de métricas, uno contiene tanto el ER como el MED, el otro contiene datos sobre potencia, área y ruta crítica.

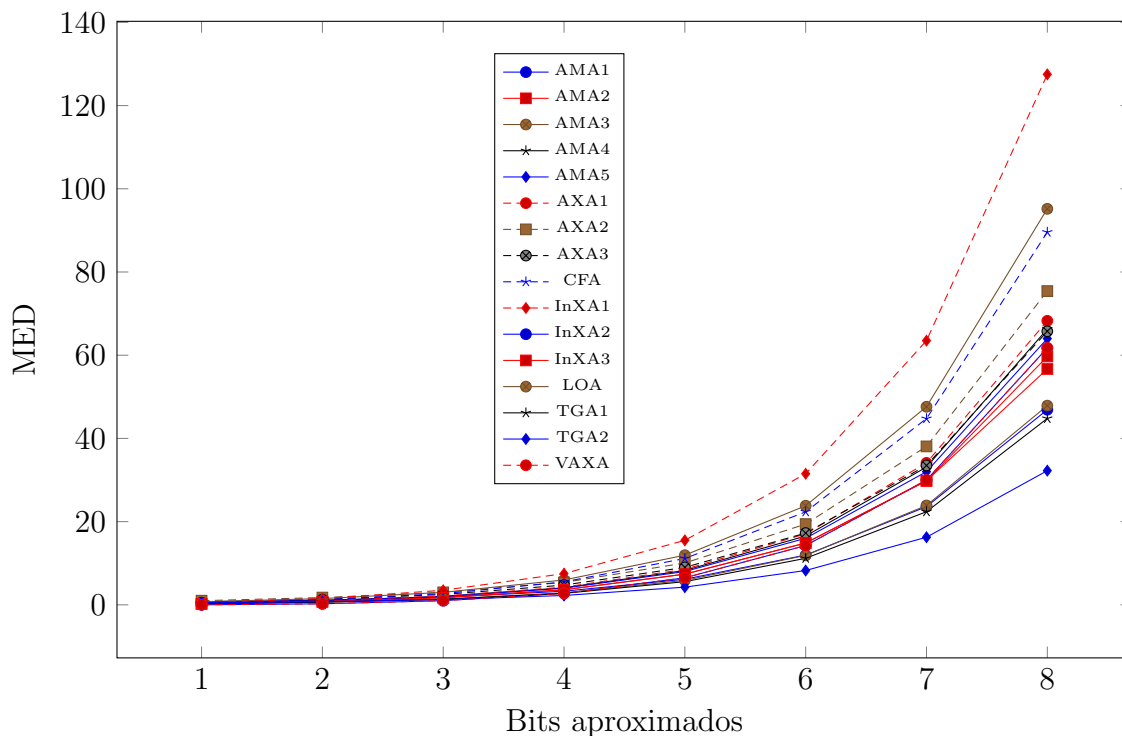


Figura 4.2: Distancia media del error en sumadores LPA de 10 bits

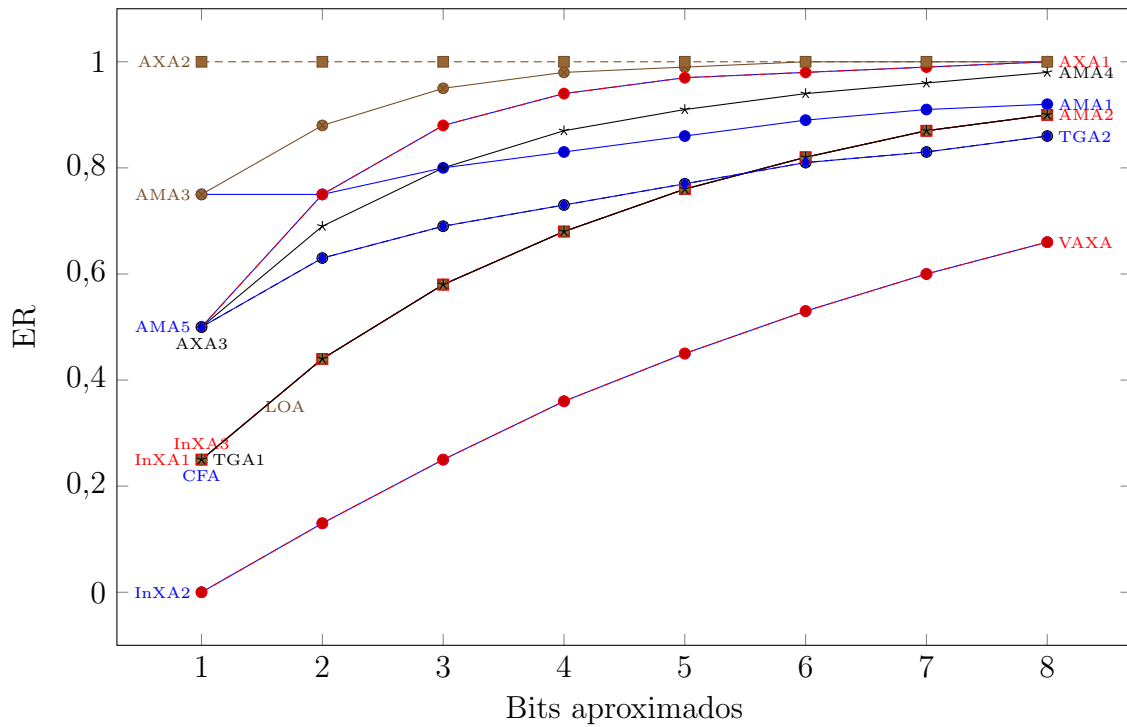


Figura 4.3: Tasa de error para sumadores LPA de 10 bits

Luego de correr el script para todos los sumadores, de los datos obtenidos se pueden formar gráficos como los de las figuras 4.2 y 4.3, que nos muestra la distancia media del error y la tasa de error. Como es de esperarse dada la introducción de una lógica diferente para la suma binaria ambas métricas son crecientes cuando se aumenta la cantidad de bits aproximados.

Una particularidad que se puede observar en la figura 4.3 es la relación entre los sumadores InXA2 y VAXA los cuales presentan la misma curva de ER, hay que recordar que las descripciones de hardware realizadas en este trabajo son sobre el comportamiento lógico de las unidades aritméticas aproximadas propuestas y no sobre su estructura que es en lo que difieren algunos diseños, por lo que se presentan casos de comportamiento lógico igual además de características de área, retardo y potencia debido a la interpretación que por ejemplo realiza la herramienta de síntesis. Este no es el único caso, también los sumadores InXA3 y AMA2 presentan comportamiento lógico similar.

En cuanto a las métricas que tienen relación con energía, los reportes presentan la potencia dinámica, potencia de fuga, la ruta crítica, área y con estos datos se pueden hacer comparaciones bastante interesantes, la más simple es observar como se comporta la potencia con respecto a los bits aproximados.

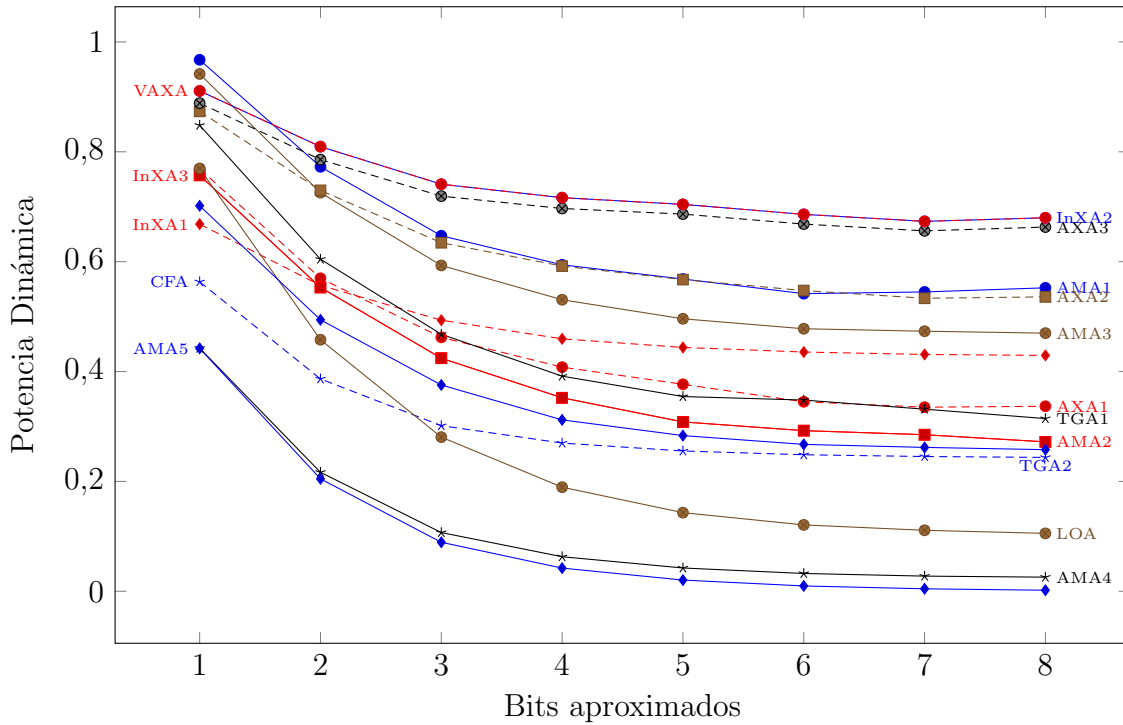


Figura 4.4: Potencia dinámica de sumadores de 10 bits tipo LPA sobre valor de referencia del sumador RCA 32.789 μW

Se puede observar en la figura 4.4 que la potencia dinámica disminuye bastante conforme se van aproximado más bits, hay que recordar que este tipo de potencia depende de entre otras cosas de las capacitancias en los nodos que conmutan, por lo que tiene sentido que para sumadores como el LOA o AMA5, que reducen su estructura bastante, esta potencia esté por debajo del 20% del valor de referencia.

4.1.2. Comparación entre procesos

Otro caso interesante que se puede lograr con la obtención de datos anterior es verificar datos de potencia entre la síntesis y la post-síntesis, esto se puede hacer para tener una mayor seguridad del verdadero consumo de la unidad que se está creando. Las diferencias son pocas pero a veces es importante tener estas estimaciones y dado el caso algunas que vayan más allá de lo que por ejemplo la herramienta propuesta puede lograr.

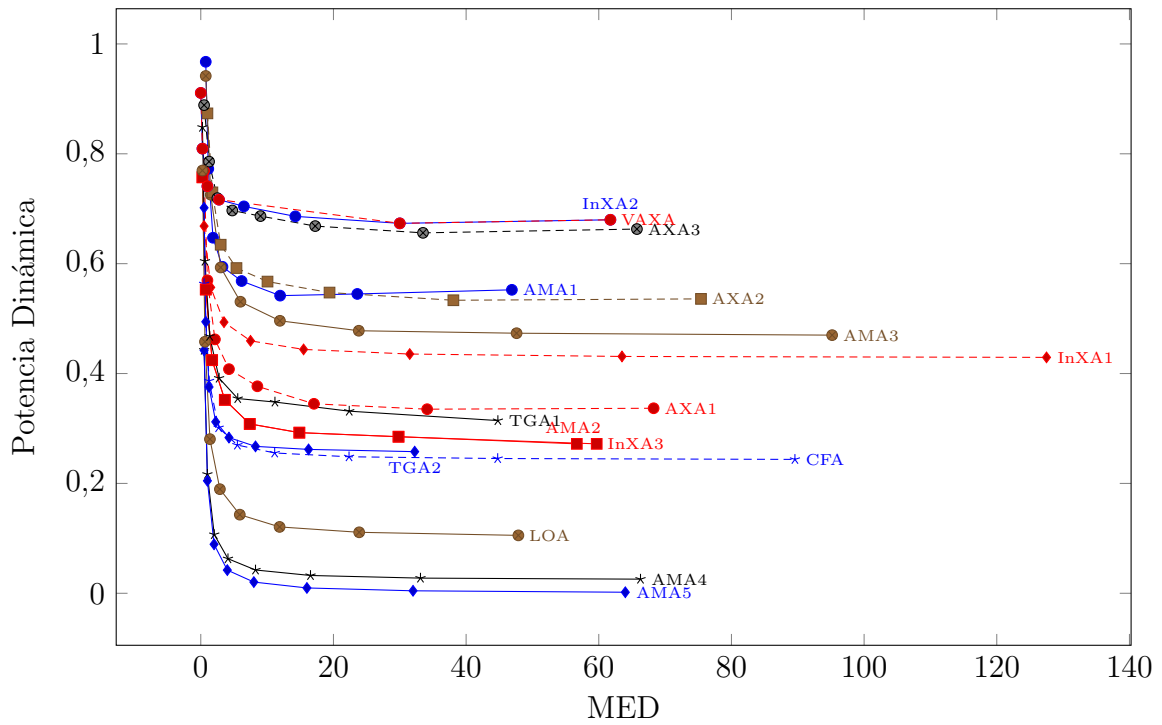


Figura 4.5: Potencia dinámica obtenida en síntesis contra valor medio de error

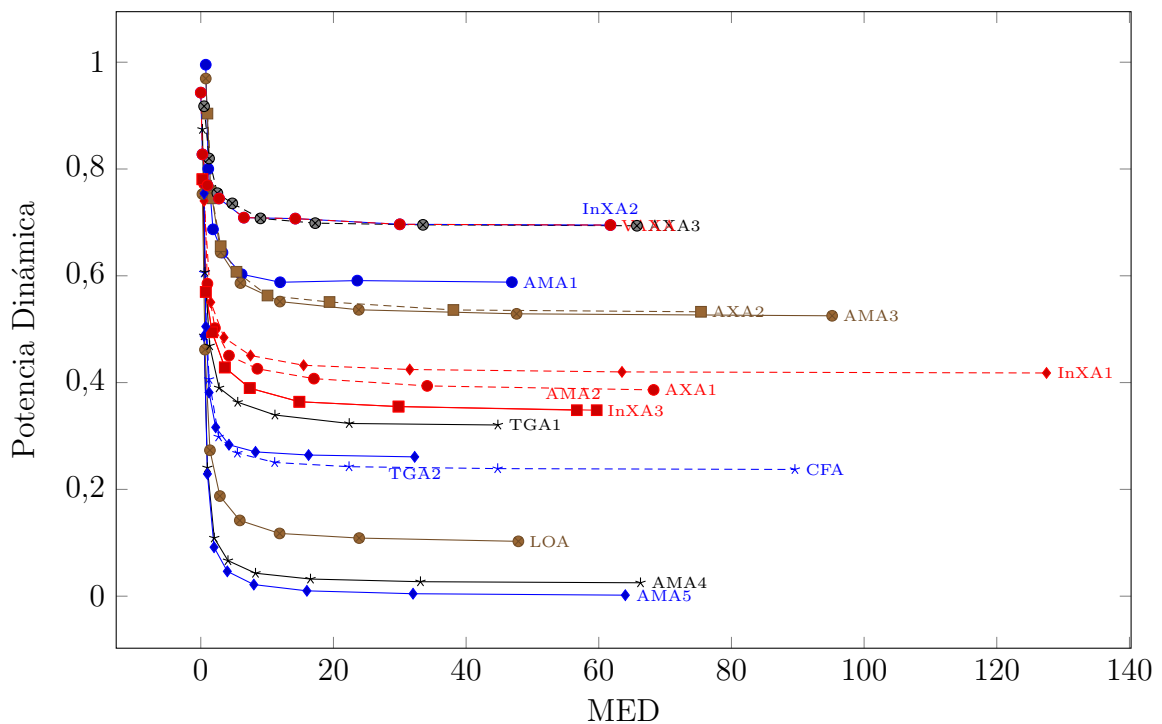


Figura 4.6: Potencia dinámica obtenida en post-síntesis contra valor medio de error

A como se mencionó los gráficos anteriores no muestran diferencia muy marcada, pero estas si se pueden dar, por ejemplo se observa el aumento de la potencia en el sumador AXA3. El comportamiento que se muestra en estas figuras es que a menor distancia media de error es mayor la potencia, puesto que eso implica que tiene mayor cantidad de unidades exactas de suma.

4.1.3. Múltiples comparaciones

Al tener la posibilidad de obtener diferentes datos, las combinaciones que se pueden tener para comparar son varias y depende de los objetivos para el que se desee las unidades generadas.

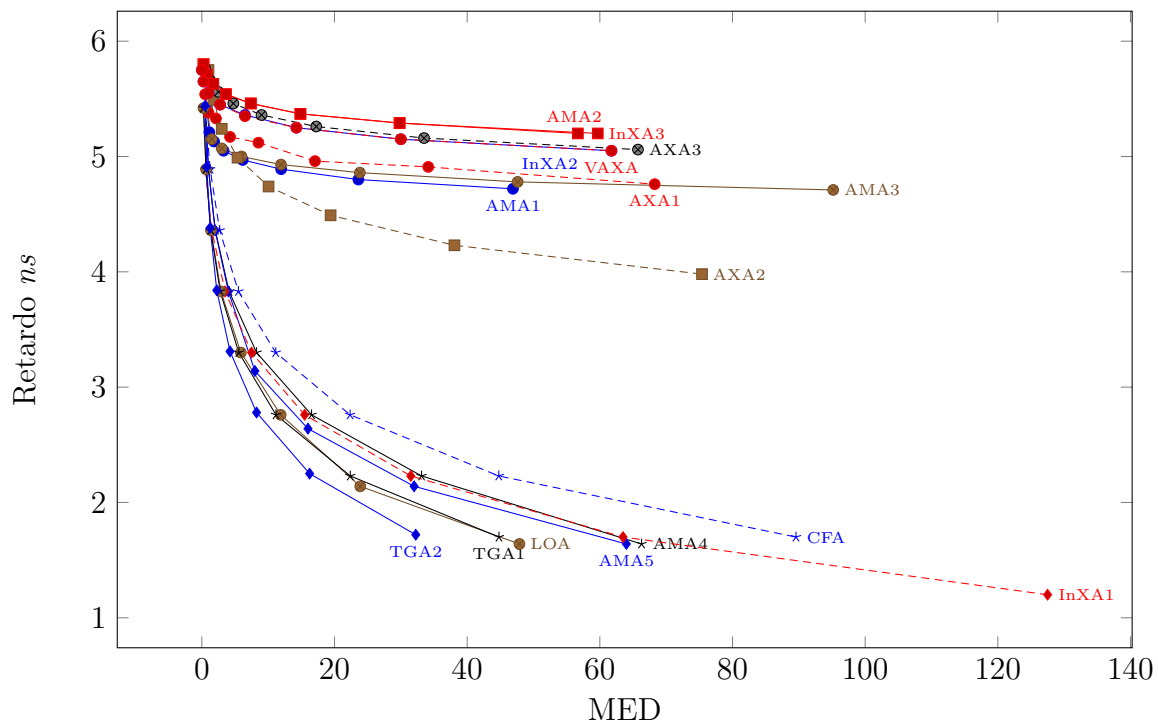


Figura 4.7: Ruta crítica o retardo contra el error medio de distancia

En la figura 4.7 se realiza la comparación entre el retardo y el error medio de distancia, el comportamiento es similar al del gráfico de potencia contra error, pues tal como se ha planteado entre menos bits se aproximen el comportamiento del LPA será más parecido al de un RCA, por eso es que el retardo es máximo cuando el error es mínimo.

Por último se muestra un gráfico que compara potencia y retardo, en este caso la cantidad de bits aproximados se leen de derecha a izquierda. Teniendo esta variedad de datos se demuestra la facilidad para la que un diseñador pueda buscar los que mejor se ajusten a sus restricciones.

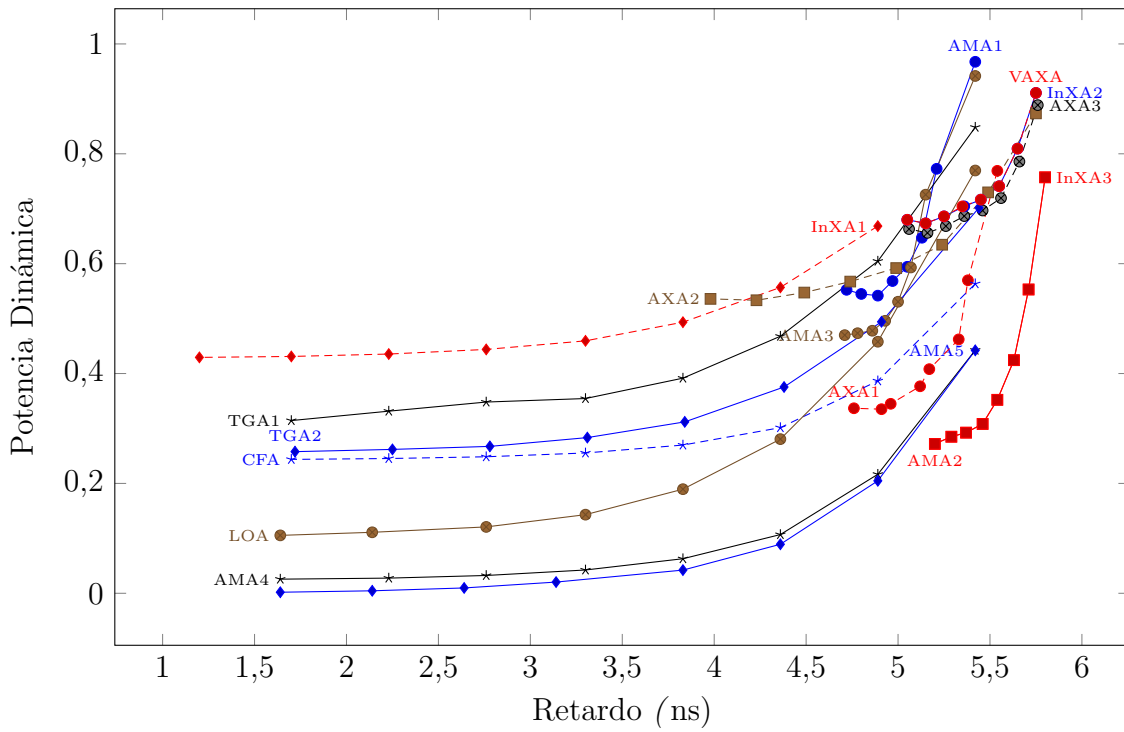


Figura 4.8: Ruta Crítica contra Potencia

4.1.4. Comparaciones derivadas

A partir de los datos que se pueden obtener directamente, se pueden derivar otros y obtener métricas como la del producto potencia retardo que nos da una visión del consumo de energía del componente aritmético generado.

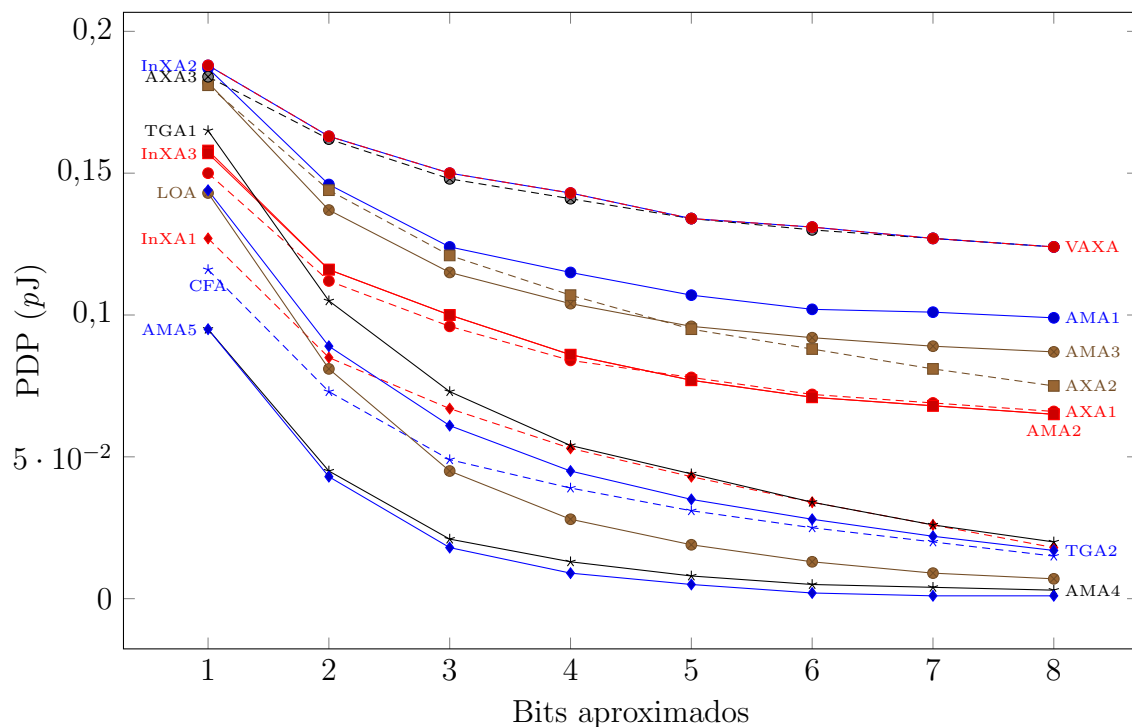


Figura 4.9: PDP para sumadores LPA de 10 bits, valor de sumador RCA 0.206 pJ

4.2. Introducción de optimizaciones

Si los resultados de una unidad no son convincentes siempre se puede buscar una manera de optimizarlos. Gracias a que la herramienta propuesta genera scripts personalizados para cada unidad, la optimización se puede hacer de manera sencilla. La optimización de la síntesis se puede realizar de dos maneras, la primera es modificar la plantilla del tipo de unidad aritmética aproximada con el que se está trabajando y volver a ejecutar la herramienta propuesta. La segunda es modificar el script personalizado dentro de la carpeta correspondiente a la unidad ya generada y ejecutarlo en la herramienta que corresponda.

```
#optimization in boundary
uniquify
compile -boundary optimization -map_effort medium
change_names -rule verilog
check_design > $LOG_PATH/$STOPLEVEL-check_design.log
```

Figura 4.10: Línea modificada en plantilla de Síntesis de sumadores

Para ejemplificar esto se hace la modificación en la plantilla general de síntesis para los sumadores en la línea que muestra la figura 4.10 y se vuelve a ejecutar el script de la figura 4.1. La optimización introducida hace que DesignCompiler propague constantes y pins desconectados, esto ayuda a reducir área.

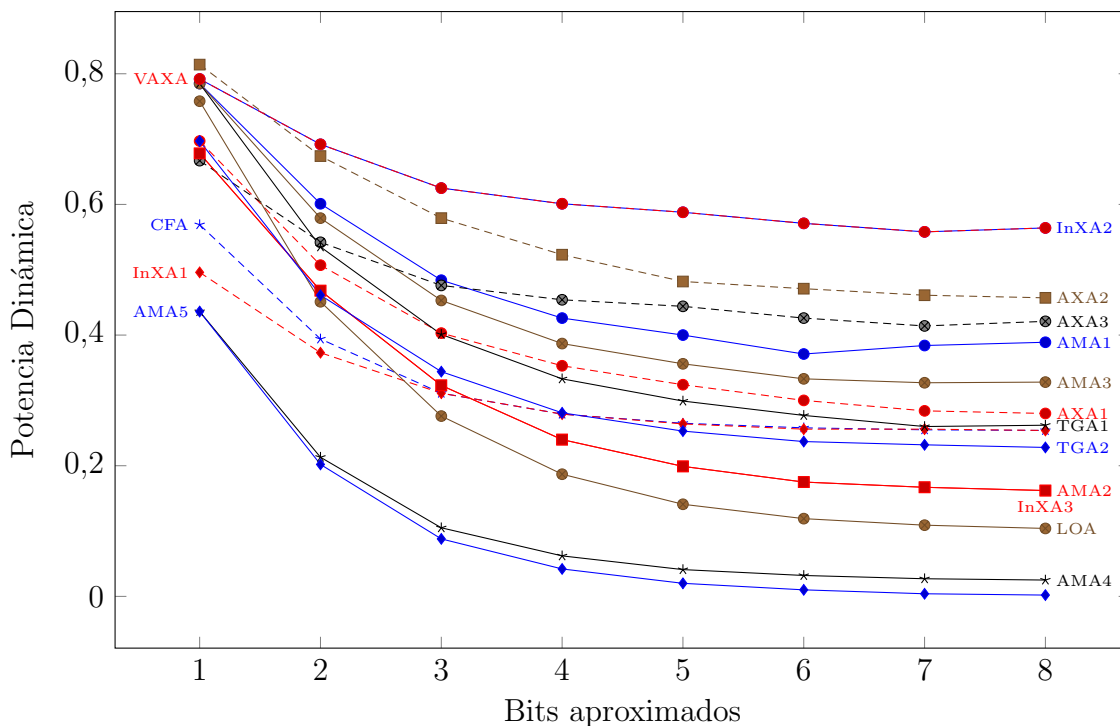


Figura 4.11: Potencia dinámica de sumadores de 10 bits tipo LPA sobre valor de referencia del sumador RCA $32.789 \mu W$ con optimización

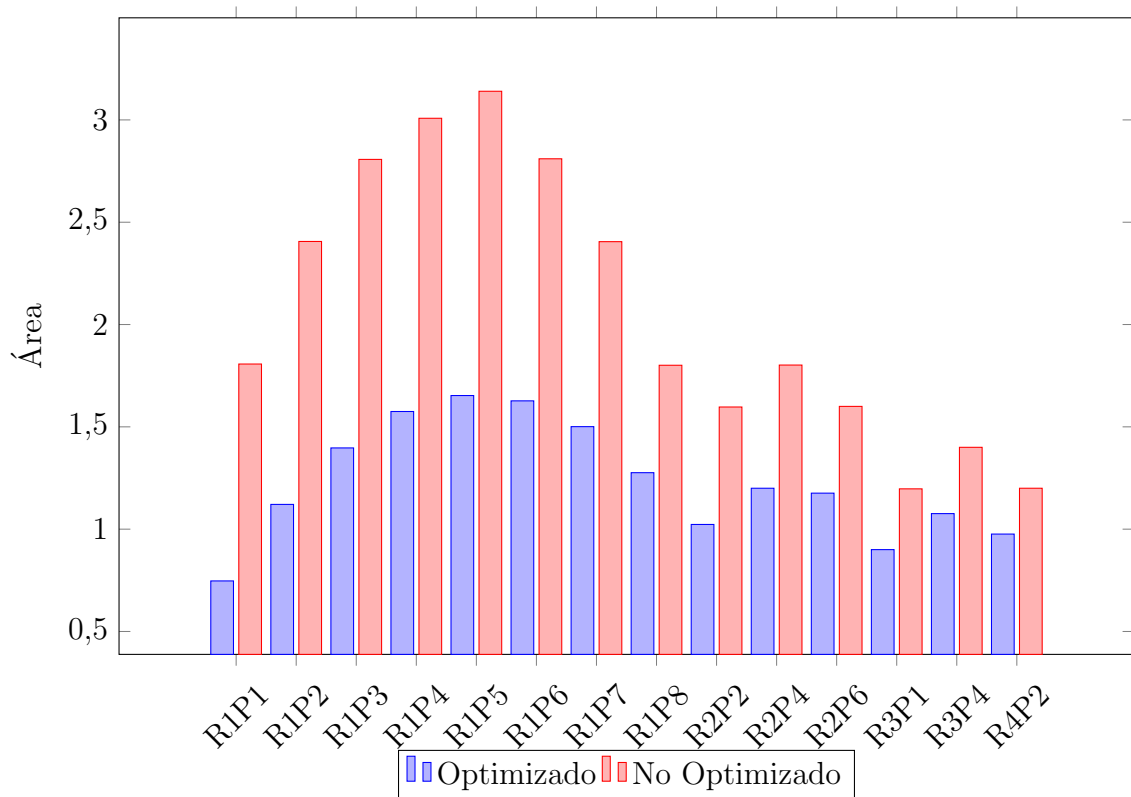


Figura 4.12: Área en el GeAr

En la figura 4.11 se observa como con la optimización realizada hay una reducción en la potencia con respecto a la mostrada en la figura 4.4. Quizás sea más notorio lo que esta optimización realiza si se observa los datos de área mostrados en el gráfico de la figura 4.12, cabe destacar que estos datos estan referenciados al valor de área correspondiente para un sumador RCA de 10 bits con optimización y sin ella a como corresponda. Se puede recordar que en el sumador GeAr se corta la propagación del acarreo en cada subsumador por lo que estos pines quedan desconectados en la descripción de hardware realizada, además el acarreo de entrada de cada subsumador se le coloca un 0 lógico. Con estas características es sencillo interpretar el comportamiento de los datos de área optimizados y no optimizados para el GeAr.

4.3. Diferencias en Descripción

Se realizan dos modelos del GeAr, uno utilizando una descripción modular esto es instanciando varios submódulos al módulo superior y la otra es una descripción más rígida que define el comportamiento lógico del sumador en el módulo superior. A las dos descripciones se les hace una corrida en la herramienta llegando hasta el proceso de síntesis, los resultados obtenidos están referenciados a los respectivos valores de un sumador RCA de 10 bits.

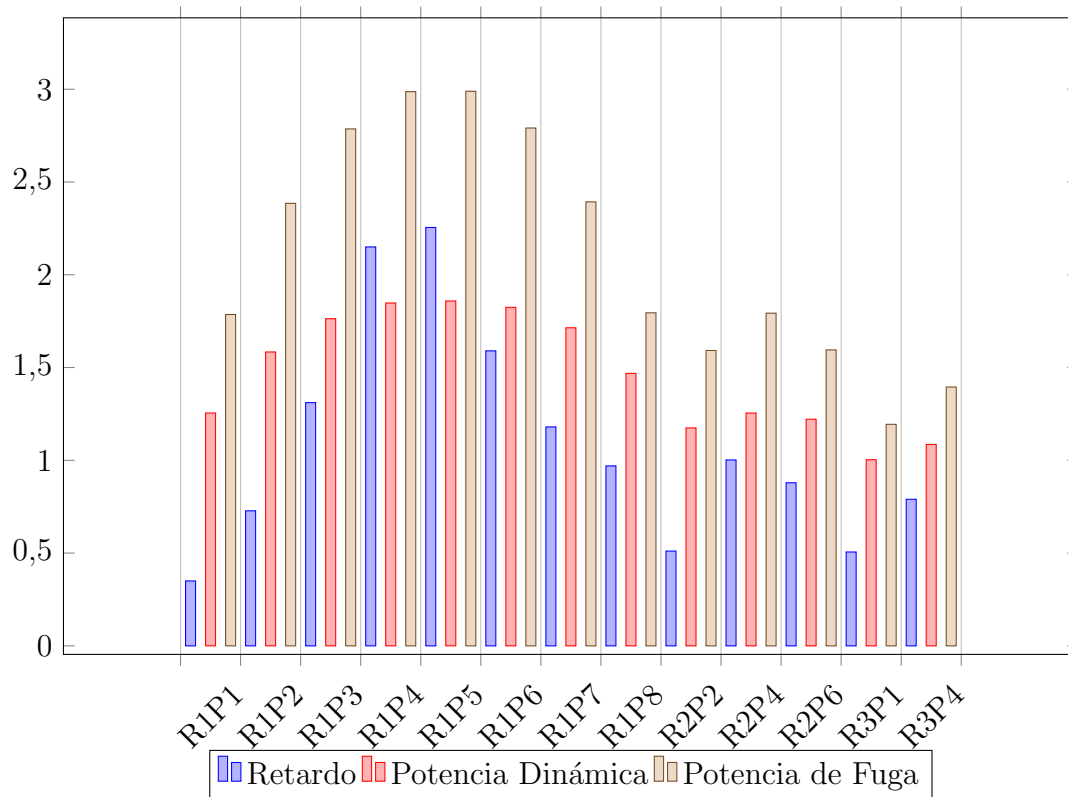


Figura 4.13: Potencia GeAr 90nm

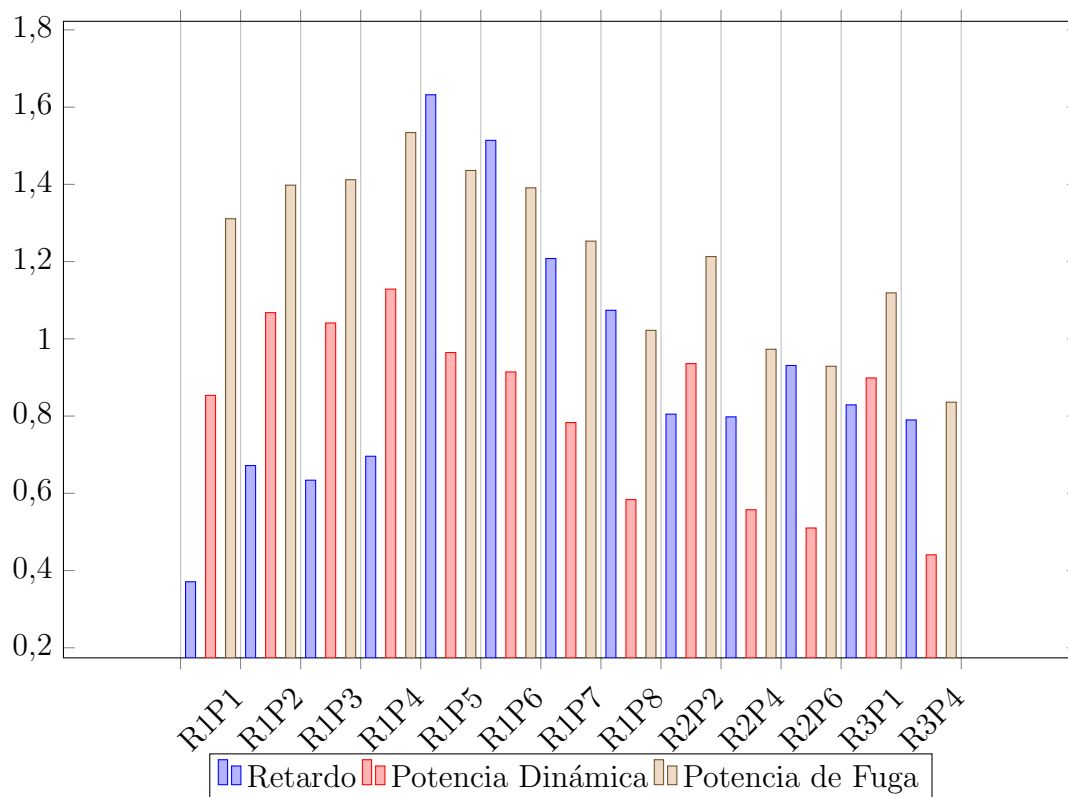


Figura 4.14: Potencia GeAr2 90nm

Como se puede observar comparando las figuras anteriores, la descripción modular presenta mayores valores en todas las áreas, cabe mencionar que estos valores no presentan

optimización por lo que hay diversos elementos que no son necesarios como los mencionados en la sección de optimización que pueden estar siendo tomados en cuenta para hacer las estimaciones.

También es extraño observar que el comportamiento del retardo no aplique exactamente a lo que la teoría menciona sobre el alto rendimiento que este sumador debería de presentar, dado que varios de los valores de retardo superen a su referencia, gracias a los reportes de tiempo que la herramienta genera se podría seguir la ruta que presenta el mayor retardo y conocer más porque o de que forma se está interpretando el comportamiento lógico del GeAr descrito.

4.4. Cambios en tecnología

Como se ha mostrado diferencias pueden ocurrir en los resultados mediante la inclusión de optimizaciones o diferentes descripciones de hardware. Otro escenario que se puede probar fácilmente con la herramienta propuesta es utilizar diferentes bibliotecas para sintetizar y observar diferencias entre los modelos en una u otra tecnología.

Para esto se modifica el archivo de configuración de la herramienta en la línea que se debe indicar la ruta de la biblioteca para síntesis y su nombre. En los siguientes resultados se utilizó una biblioteca de 65nm de TSMC, no se realizó optimización y como se ha trabajado los valores estan referenciados a un sumador RCA de 10 bits también sintetizado en esta tecnología.

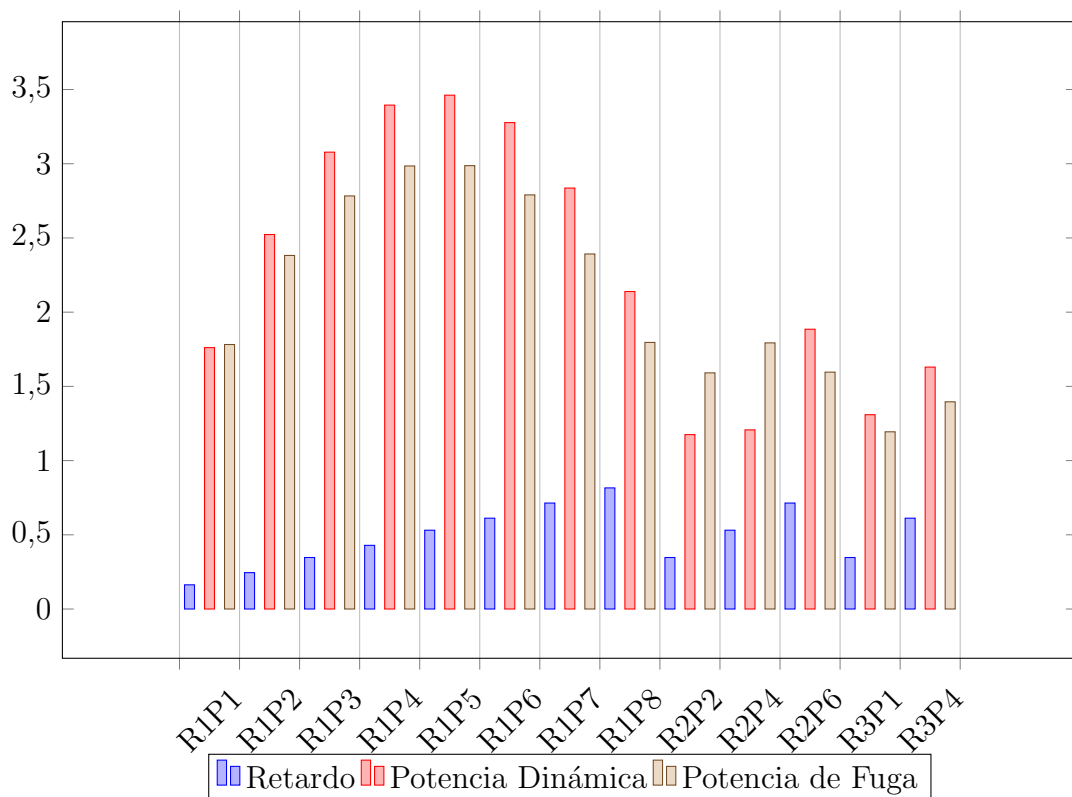


Figura 4.15: Potencia GeAr 65nm

Lo primero que se puede notar con este cambio es que el comportamiento del Gear ahora si concuerda con la teoría puesto que el retardo para todas las configuraciones esta por debajo del valor de referencia.

Se puede recordar que el retardo se reduce conforme el tamaño de cada subsumador, asimismo este depende de los valores de R y P por lo que es congruente que el menor retardo sea para la combinación 1-1 y el mayor para la combinación 1-8.

Ahora que se puede constatar que los modelos del GeAr concuerdan con lo esperado, se puede mostrar como es su comportamiento en cuanto a la introducción de errores para esto se muestra la tasa de error y la distancia media del error.

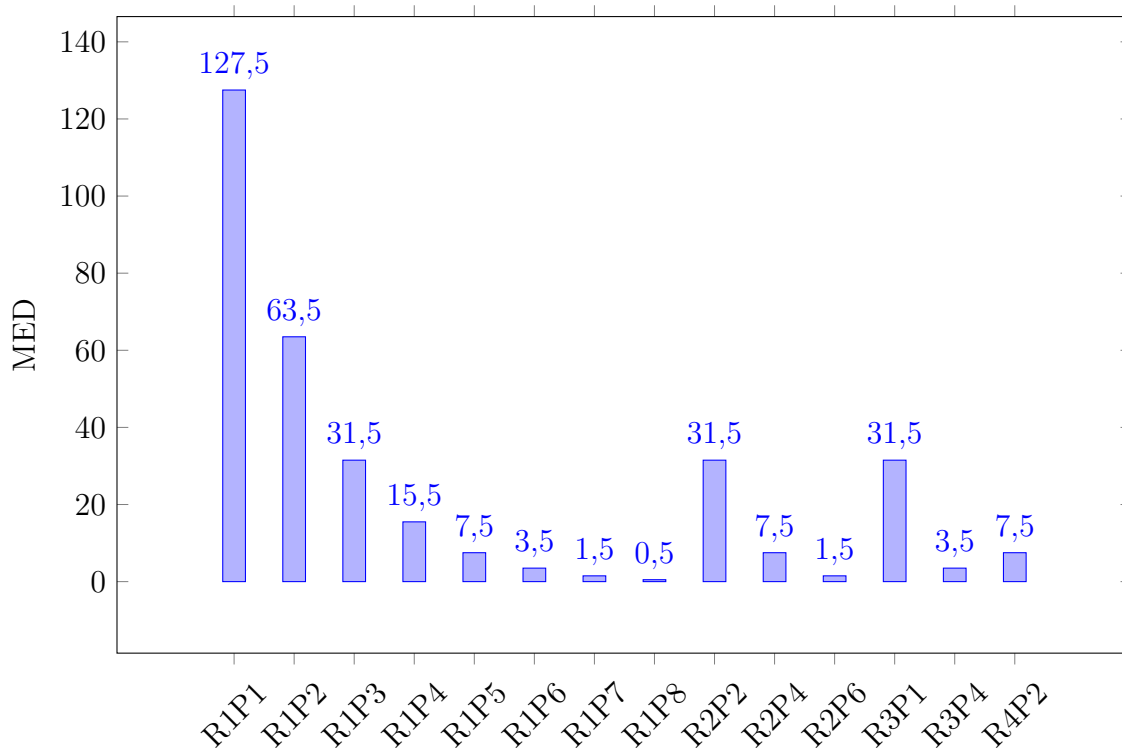


Figura 4.16: MED para distintas combinaciones de R y P en el GeAr

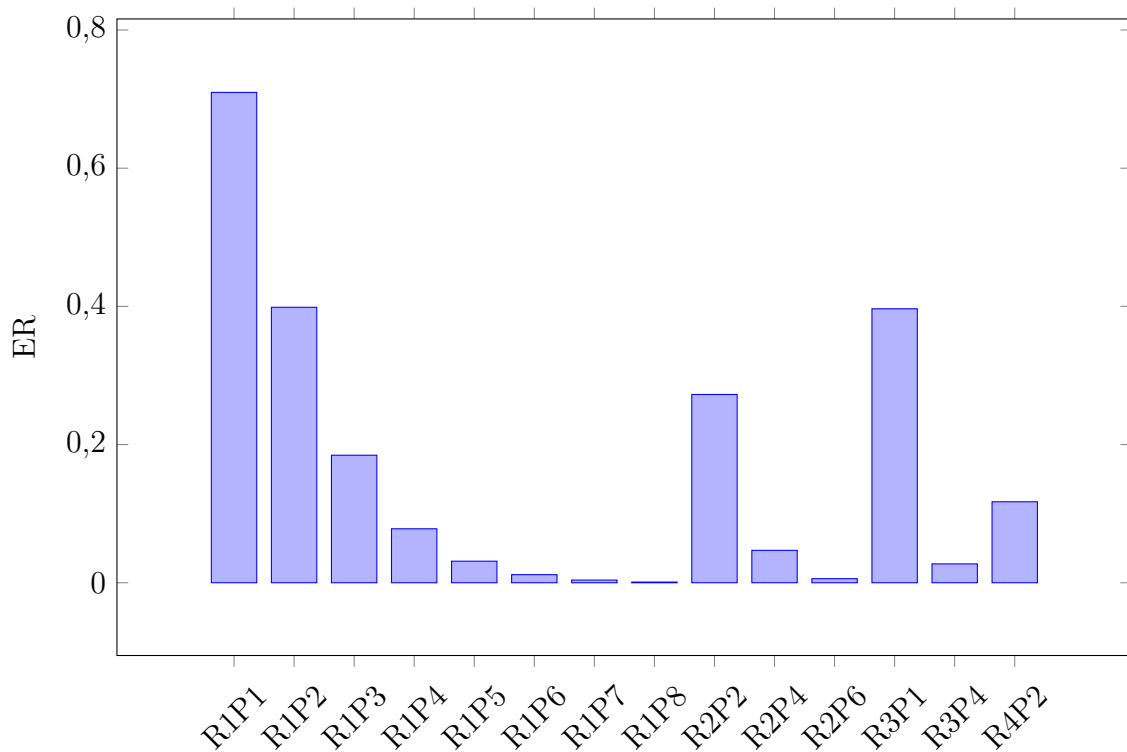


Figura 4.17: ER para distintas combinaciones de R y P en el GeAr

Como se observa en la gráfica del MED se da una repetición del valor en las configuraciones que presentan mismo tamaño de subsumador esto es debido a que los errores en el GeAr suelen darse en el punto donde se corta la cadena de acarreo y al tener la misma cantidad de subsumadores para estas combinaciones los errores ocurren en el mismo punto ante los mismos valores de entrada. Los errores se generan en el GeAr cuando un subsumador genera un acarreo y este no se propaga. Siendo esto así, la tasa de error debería de ser baja para los sumadores que cuenten con menor cantidad de subsumadores y viceversa, esto se puede observar en la figura 4.17.

4.5. Limitaciones

Las múltiples simulaciones que se realizaron arrojaron algunas limitaciones extra a las que cada unidad por su estructura o lógica ya tenían contempladas. Una de ellas se integro a la herramienta y es la cantidad de valores de prueba, que depende del ancho de bits con el que se desea trabajar.

Se limitó para la opción de realizar una verificación completa, esto es utilizar todas las posibles combinaciones, se permite si el ancho de bits es menor o igual a 12, esto debido a lo extenso que es el proceso para números de mayor representación binaria. Cabe mencionar que este es un parámetro fácilmente modificable en el código de la herramienta, por lo que también queda a decisión del usuario el valor del mismo según los requerimientos que tenga de diseño.

La otra limitación surge en la herramienta externa ModelSim, la cual cuenta con una

capacidad de 32bits para representar valores numéricos. Representaciones decimales son utilizadas por ejemplo en los encoders del multiplicador DRUM, estas por lo tanto no pueden superar el valor 2^{32} .

A continuación se muestran los resultados para los multiplicadores implementados, los parámetros utilizados son un ancho de bits de 16 y debido a la limitación anteriormente mencionada se optó por utilizar 512 números aleatorios lo que da un total de $2^{(9-2)}$ resultados, el valor de la semilla para estos valores es 123, se realizó el flujo hasta síntesis utilizando la biblioteca de 65nm anteriormente mencionada.

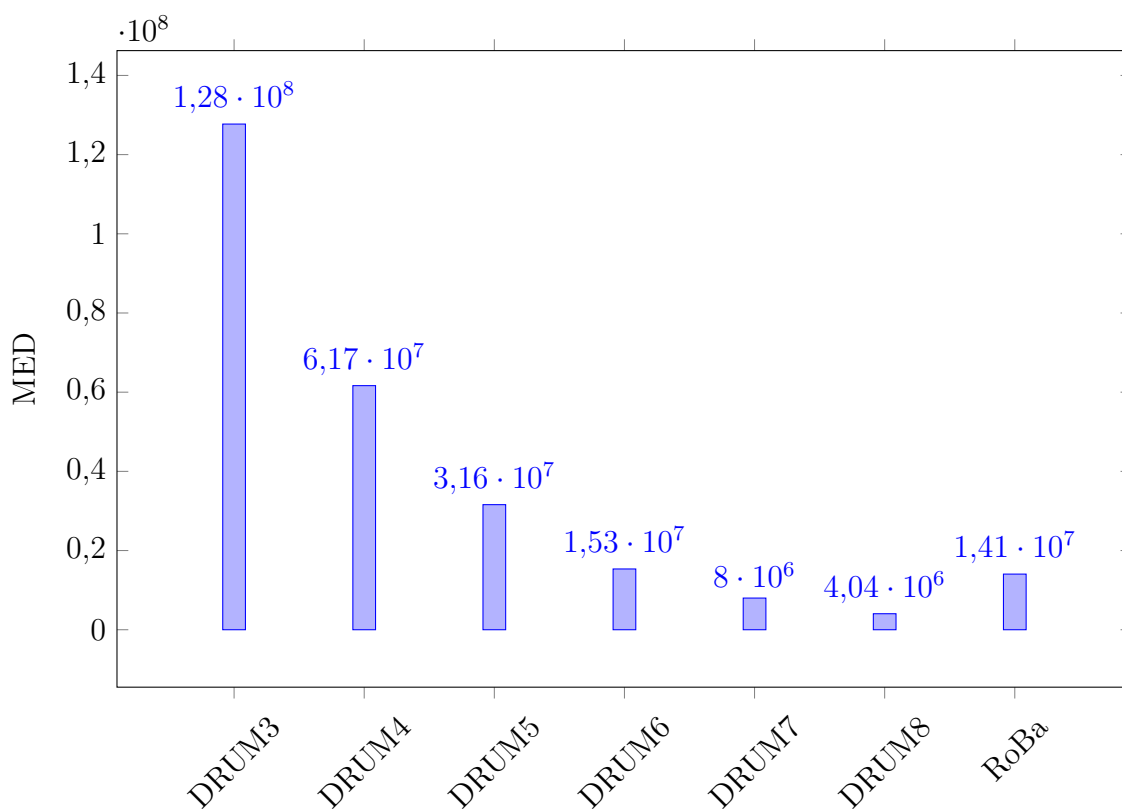


Figura 4.18: MED para multiplicadores aproximados

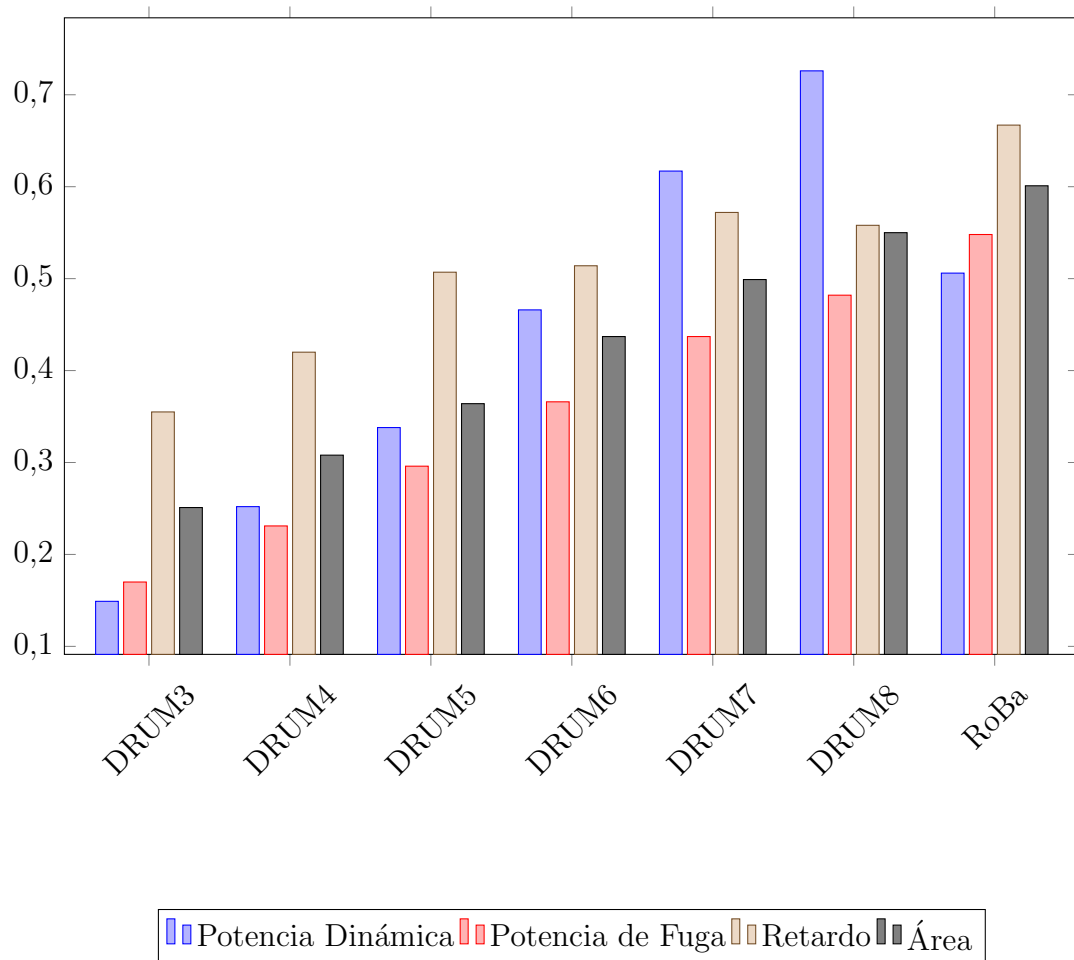


Figura 4.19: Datos Multiplicadores

A simple vista los valores del MED pueden desconcertar un poco, pero es válido recordar que los resultados que se esperan al hacer una multiplicación de 16 bits son números que pueden alcanzar valores de hasta $2^{32} - 1$ y al no tener una aplicación específica con la que comparar que tan grande es permisible este valor lo que se puede analizar es el comportamiento que presenta.

Se puede observar en la figura 4.18 que el MED para el DRUM se reduce conforme se aumenta cantidad de k bits, esto es porque estos bits son los que se utilizan en el multiplicador exacto central de su estructura por lo que a mayor tamaño de este menor serán los errores que se produzcan, esta capacidad de configurabilidad no la presenta el multiplicador RoBa por lo que cuenta con un único valor. Los resultados de potencia, retardo y área están referenciados a un multiplicador exacto en arreglo, los valores son 1.3706 mW , $74.4529 \mu\text{W}$, 2.76 ns , 2823.479998 .

La figura 4.19 muestra que todas las unidades generadas presentan un ahorro en potencia, retardo y área, siendo las que tienen mayor MED las que presentan más beneficios energéticos. Los gráficos de tasa de error no se presentan puesto que el valor para la mayoría de estas implementaciones es de 1, por lo que no es un parámetro que brinde mucha información comparativa.

4.6. Variedad de unidades

La variedad en estructuras y comportamiento lógico que presentan las unidades desarrolladas se pueden comprobar con los divisores, los cuales gracias a su configurabilidad pueden alcanzar una gran gama de opciones para una misma cantidad de bits.

A continuación se muestran los resultados de los divisores desarrollados. El sumador exacto que sirve como núcleo en el DAC y AXDNR es un divisor en arreglo, el cual cuenta con restricciones ya mencionadas en el capítulo 3 para poder obtener resultados exactos.

Estas restricciones son tomadas en cuenta al generar los datos de prueba por parte de la herramienta y se intenta al máximo que los números que se produzcan cumplan las condiciones pero esto depende de la semilla que se elija para los números aleatorios.

Aclarado esto los demás parámetros son un ancho de bits de 8, 512 combinaciones, semilla 123 y flujo hasta síntesis de todas las posibles combinaciones de los divisores.

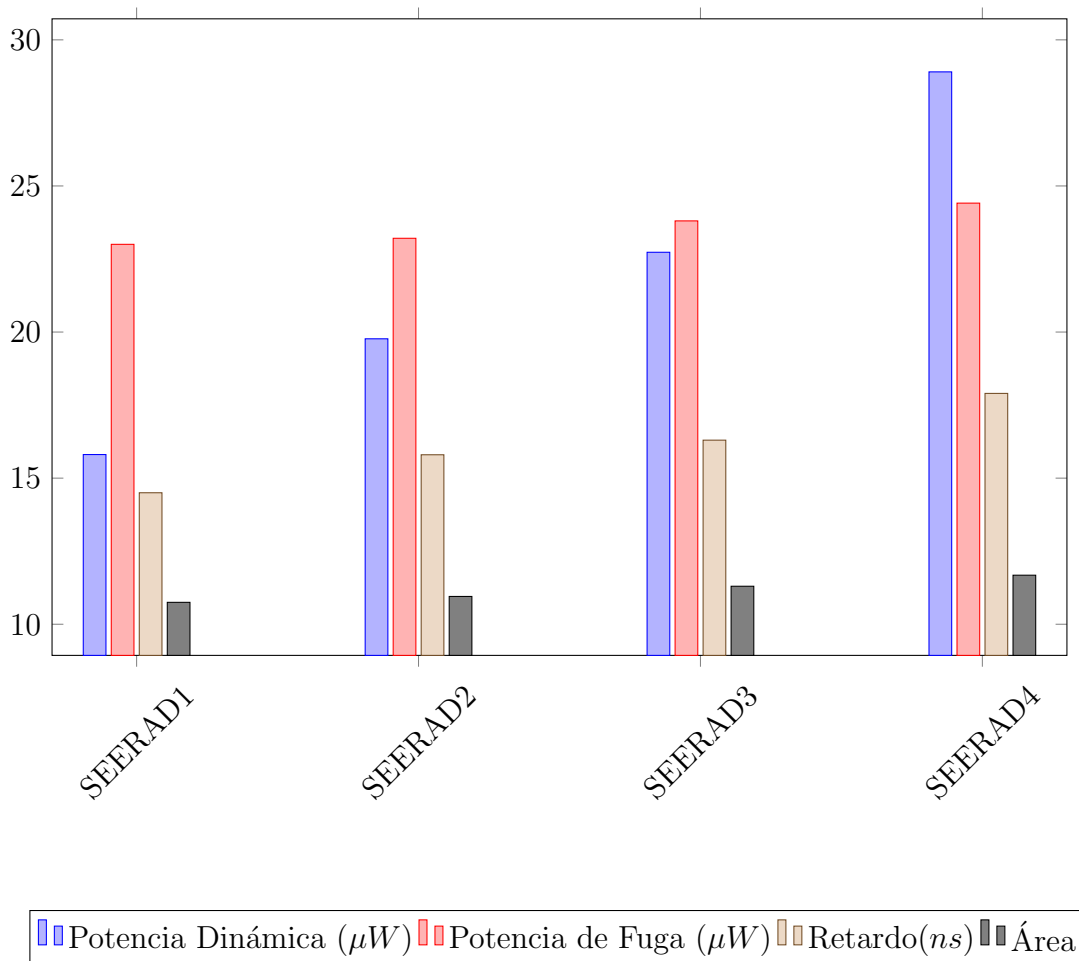


Figura 4.20: Datos de divisor SEERAD

Debido a que el SEERAD presenta una estructura de salida diferente a la del resto de divisores inclusive al del exacto no se plantea una comparación con las demás unidades de este tipo, mas si con sus posibles niveles de precisión. El valor de área se divide en un factor de 100, el de potencia dinámica en 10 y el de retardo se multiplica por 10 para que se puedan representar mejor en la gráfica.

Los valores de la siguiente figura si se referencian a un divisor exacto en arreglo con los siguientes valores respectivamente, 136.0377 , 9.6278 , 1.69 ns, 355.3199 .

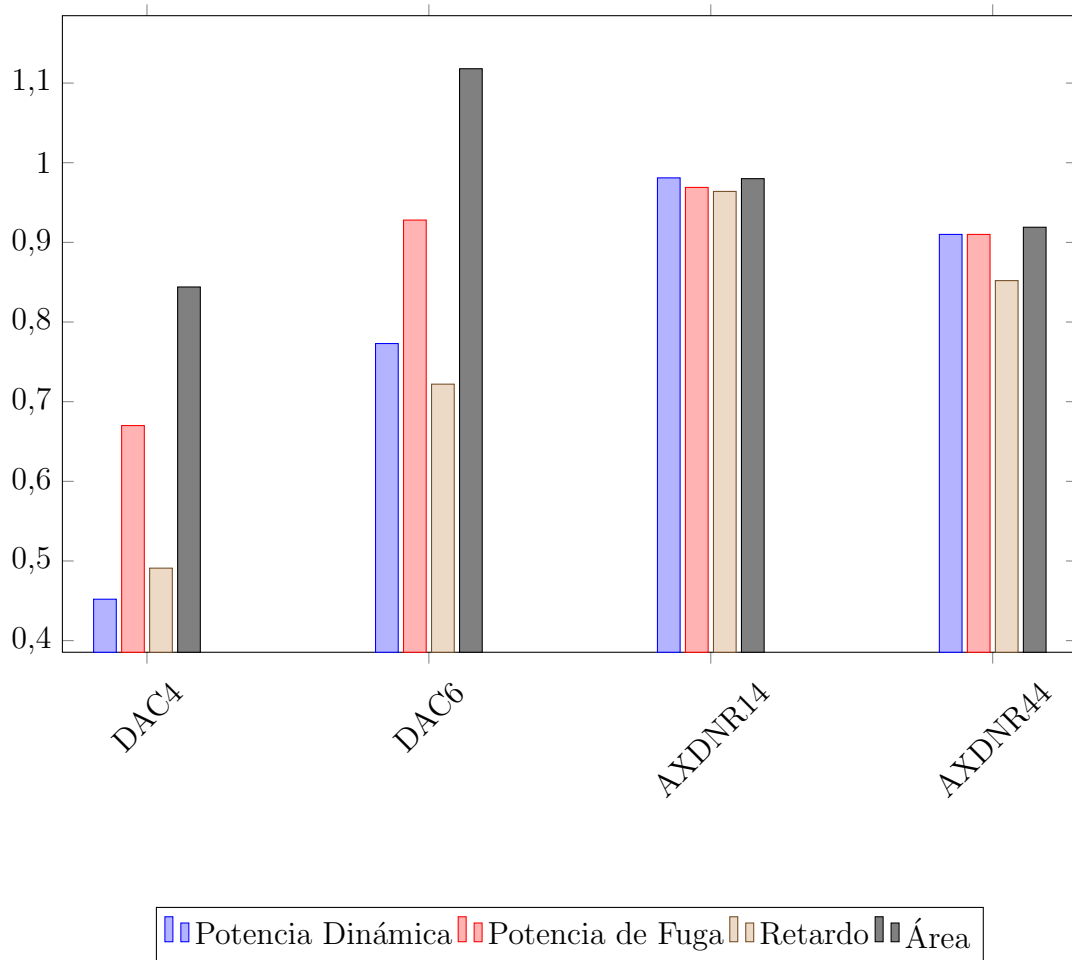


Figura 4.21: Comparación divisores

En el gráfico de la figura 4.21 se muestran las dos posibles combinaciones del divisor DAC para un tamaño de 8 bits y se presentan el caso que genera menos y más cantidad de errores en el cociente de la operación del divisor AXDNR, cabe mencionar que la celda aproximada de resta usada en este caso es la AXSC1, el número de combinaciones así como la semilla permanecen igual a como se ha trabajado.

La combinación fila-columna 1-4 es la que introduce menor cantidad de errores esto debido a que quiere decir que se aproxima por completo el residuo de la operación y el bit menos significativo del cociente, se puede obtener una mejor visión de esto al referirse a la figura 2.16, así como se puede deducir que la combinación 4-4 es la que introduce mayor error.

El DAC presenta mejores resultados en los parámetros mostradas a excepción del área del DAC6 donde posiblemente la lógica que antecede al divisor exacto central lleve a al aumento que se muestra a pesar de que su potencia se encuentra por debajo del valor de referencia.

Los siguientes gráficos muestran las posibles combinaciones del divisor AXDNR y sus resultados del flujo hasta síntesis.

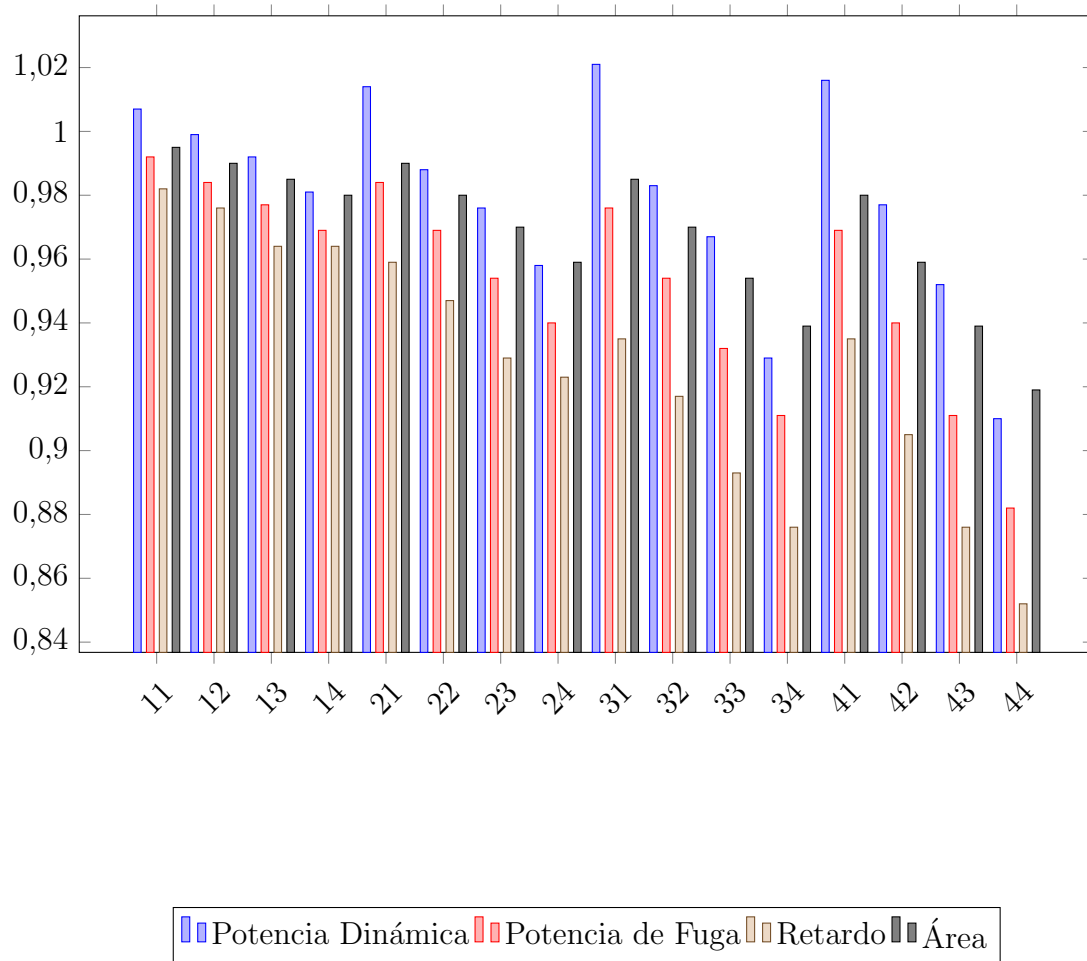


Figura 4.22: Comparación para divisor AXDNR con celda AXSC1

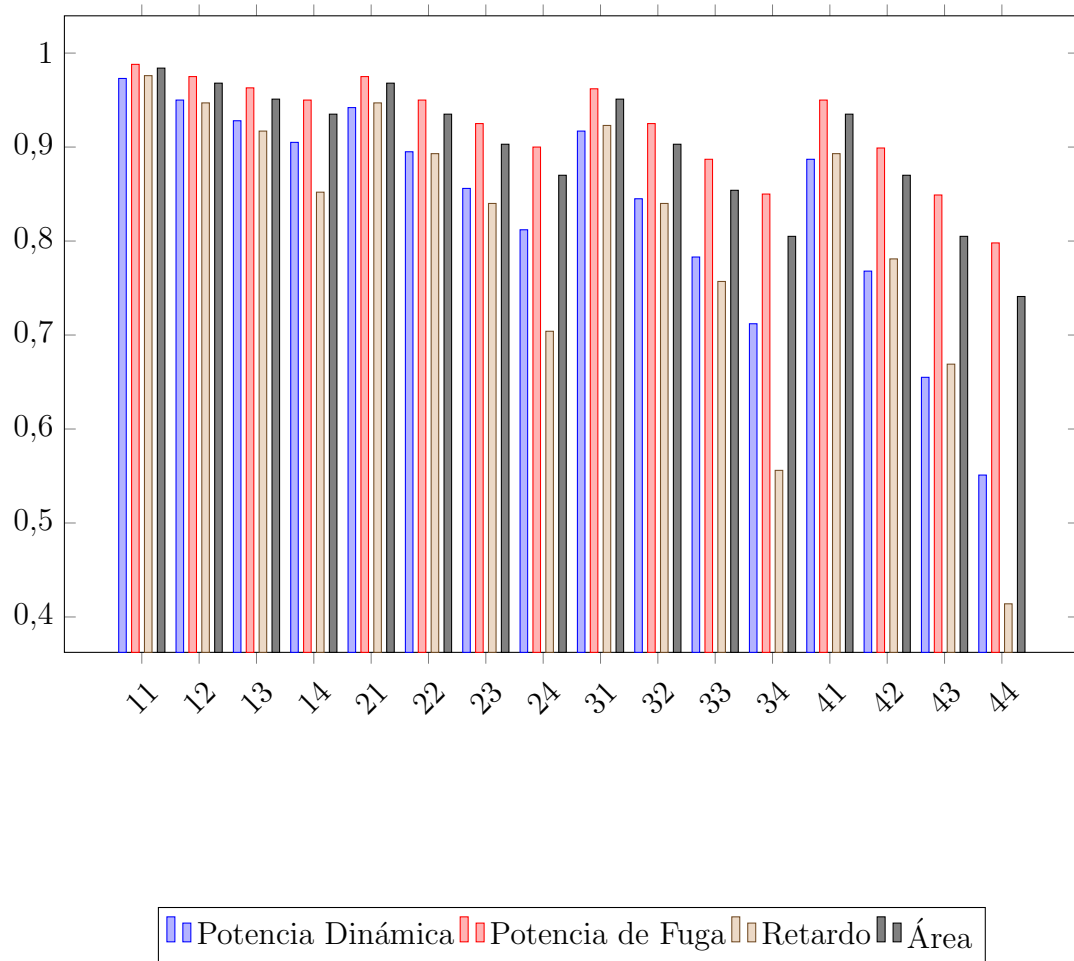


Figura 4.23: Comparación para divisor AXDNR con celda AXSC2

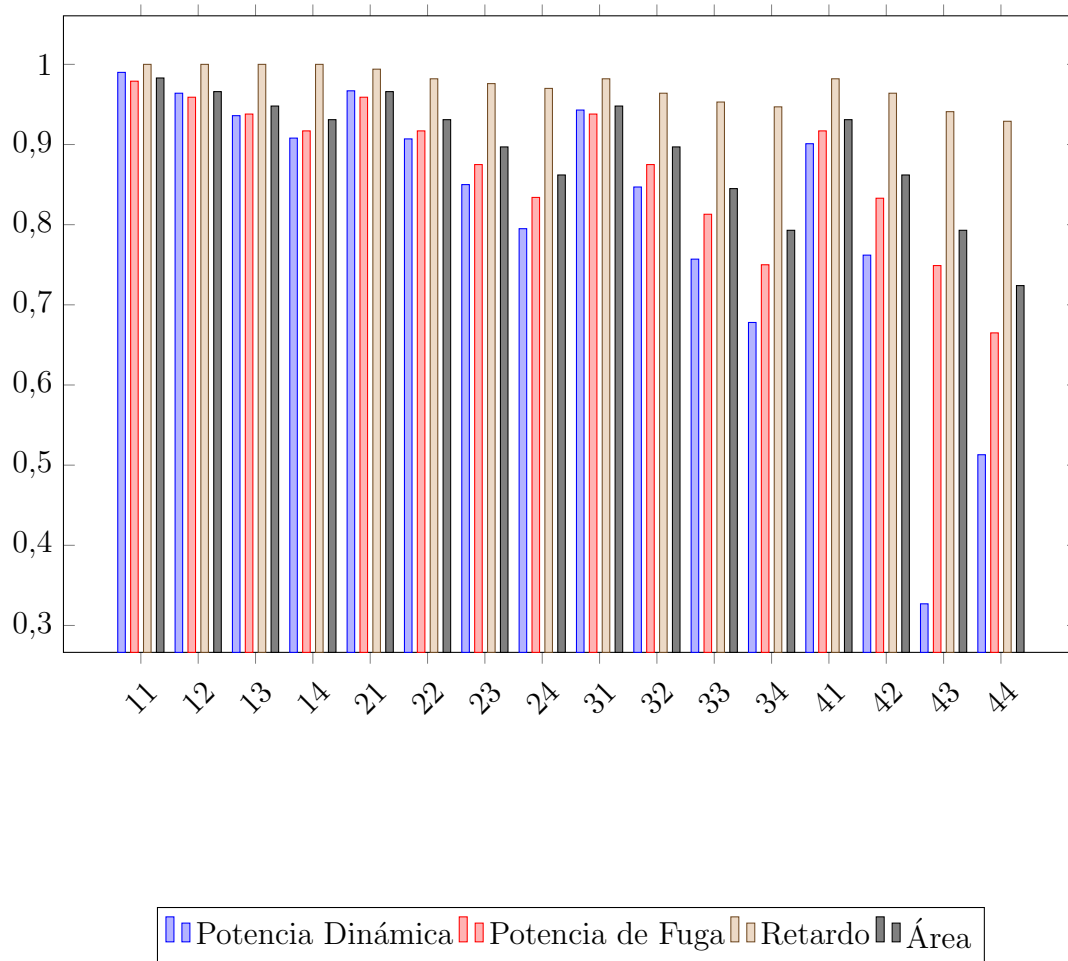


Figura 4.24: Comparación para divisor AXDNR con celda AXSC3

Capítulo 5

Conclusiones

Al finalizar el desarrollo de la herramienta una de las conclusiones más agradables es poder afirmar que la computación aproximada es un verdadero paradigma de diseño energéticamente eficiente al ser comprobado que la introducción de errores mediante la modificación de estructura y comportamiento lógico de unidades aritméticas exactas llevan a que parámetros que influyen en el consumo de energía como potencia se reduzca hasta en un 80 % para sumadores de baja potencia así como el retardo se reduce hasta en un 60 % para sumadores de alto rendimiento. Estos parámetros a su vez se ven influenciados por diversas variables, entre ellas la manera de describir el hardware, las condiciones ambientales y/o optimizaciones a nivel de síntesis, así como la biblioteca de celdas utilizadas ya que en ella se encuentran características propias de cada fabricante, lo que puede llevar a resultados diversos, por ende se deben tomar en cuenta estas variables para entender y verificar el comportamiento presentado por la unidad aritmética generada.

A pesar de que los resultados pueden variar bastante dependiendo de las variables ya mencionadas tener una noción de lo que se espera sobre estos es importante, esto se logra un poco con la descripción modular utilizada para la mayoría de circuitos desarrollados ya que de esta manera se obliga un poco más a la herramienta de síntesis a cumplir cierta estructura o comportamiento lógico deseado.

Al solo describir comportamiento lógico en ciertas unidades y no tener control sobre la estructura física a nivel de transistores que la propuesta de ellas implica lleva a que se dupliquen algunas, principalmente en los sumadores LPA, pese a esto los beneficios que se esperan son congruentes con las características de cada unidad.

Las diferencias estructurales que las unidades aritméticas aproximadas desarrolladas presentan, conllevan una variedad de características que pueden ser aprovechadas para cumplir diferentes parámetros de diseño. Para la comparación de unidades mediante las métricas de error se debe de tomar en cuenta características propias de escenario de simulación y características del tipo de unidad, debido a que se pueden malinterpretar los resultados obtenidos si esto no se considera.

En cuanto a la herramienta como software, su rapidez para ejecutar los procesos depende de varias variables, entre las cuales se puede mencionar, la máquina donde se

ejecute, la cantidad de procesos deseados y la cantidad de datos a evaluar que se deseen.

El uso de la metodología de arriba-abajo no solo para compilación sino para orden estructural además de también una programación modular en este caso por medio de funciones que llaman otras, presentan beneficios en el manejo, modificación y búsqueda de archivos a lo largo de la estructura de la herramienta, lo que a su vez facilita la inclusión de nuevos modelos y ampliación de la herramienta. La ejecución de la herramienta por línea de comandos facilita el desarrollo de scripts recursivos que permiten generar gran cantidad de unidades ya se para uso o para comparación.

Bibliografía

- [1] J. Han and M. Orshansky, “Approximate computing: An emerging paradigm for energy-efficient design,” pp. 1–6, May 2013. [1](#), [2](#)
- [2] Q. Xu, T. Mytkowicz, and N. S. Kim, “Approximate computing: A survey,” *IEEE Design Test*, vol. 33, pp. 8–22, Feb 2016. [1](#), [2](#)
- [3] G. M. Roperó, *Reducción del consumo de potencia en unidades funcionales mediante cotejo de códigos de operación*. PhD thesis, Universidad Complutense de Madrid, 2009. [1](#), [7](#)
- [4] N. S. Kim, T. Austin, D. Baauw, T. Mudge, K. Flautner, J. S. Hu, M. J. Irwin, M. Kandemir, and V. Narayanan, “Leakage current: Moore’s law meets static power,” *Computer*, vol. 36, pp. 68–75, Dec 2003. [1](#)
- [5] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel, “Invited: Cross-layer approximate computing: From logic to architectures,” in *2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, June 2016. [2](#), [3](#)
- [6] J. Han, “Introduction to approximate computing,” in *2016 IEEE 34th VLSI Test Symposium (VTS)*, pp. 1–1, April 2016. [2](#)
- [7] L. Sekanina, “Introduction to approximate computing: Embedded tutorial,” in *2016 IEEE 19th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, pp. 1–6, April 2016. [2](#)
- [8] H. Jiang, J. Han, and F. Lombardi, “A comparative review and evaluation of approximate adders,” in *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI, GLSVLSI ’15*, (New York, NY, USA), pp. 343–348, ACM, 2015. [3](#), [8](#)
- [9] M. Shafique, W. Ahmad, R. Hafiz, and J. Henkel, “A low latency generic accuracy configurable adder,” in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, June 2015. [3](#), [18](#), [34](#)
- [10] H. Jiang, C. Liu, N. Maheshwari, F. Lombardi, and J. Han, “A comparative evaluation of approximate multipliers,” in *2016 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH)*, pp. 191–196, July 2016. [3](#)
- [11] S. Hashemi, R. I. Bahar, and S. Reda, “A low-power dynamic divider for approximate applications,” in *2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, June 2016. [3](#)

- [12] R. Zendegani, M. Kamal, A. Fayyazi, A. Afzali-Kusha, S. Safari, and M. Pedram, “Seerad: A high speed yet energy-efficient rounding-based approximate divider,” in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1481–1484, March 2016. [3](#)
- [13] J. Liang, J. Han, and F. Lombardi, “New metrics for the reliability of approximate and probabilistic adders,” *IEEE Transactions on Computers*, vol. 62, pp. 1760–1771, Sept 2013. [3](#), [28](#)
- [14] S. Mohanty, *Low-Power High-Level Synthesis for Nanoscale CMOS circuits*. Springer, 2008. [7](#), [8](#), [28](#)
- [15] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, “Low-power digital signal processing using approximate adders,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 32, pp. 124–137, Jan 2013. [10](#)
- [16] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, “Bio-inspired imprecise computational blocks for efficient vlsi implementation of soft-computing applications,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, pp. 850–862, April 2010. [11](#)
- [17] Z. Yang, A. Jain, J. Liang, J. Han, and F. Lombardi, “Approximate xor/xnor-based adders for inexact computing,” in *2013 13th IEEE International Conference on Nanotechnology (IEEE-NANO 2013)*, pp. 690–693, Aug 2013. [11](#)
- [18] S. Kim and Y. Kim, “Energy-efficient hybrid adder design by using inexact lower bits adder,” in *2016 IEEE Asia Pacific Conference on Circuits and Systems (APCCAS)*, pp. 355–357, Oct 2016. [12](#)
- [19] Z. Yang, J. Han, and F. Lombardi, “Transmission gate-based approximate adders for inexact computing,” in *Proceedings of the 2015 IEEE/ACM International Symposium on Nanoscale Architectures (NANOARCH₁₅)*, pp. 145–150, July 2015. [13](#)
- [20] H. A. F. Almurib, T. N. Kumar, and F. Lombardi, “Inexact designs for approximate low power addition by cell replacement,” in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 660–665, March 2016. [14](#)
- [21] S. Dutt, H. Patel, S. Nandi, and G. Trivedi, “Exploring approximate computing for yield improvement via re-design of adders for error-resilient applications,” in *2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID)*, pp. 134–139, Jan 2016. [15](#)
- [22] N. Zhu, W. L. Goh, and K. S. Yeo, “An enhanced low-power high-speed adder for error-tolerant application,” in *Proceedings of the 2009 12th International Symposium on Integrated Circuits*, pp. 69–72, Dec 2009. [16](#)
- [23] A. K. Verma, P. Brisk, and P. Ienne, “Variable latency speculative addition: A new paradigm for arithmetic circuit design,” in *2008 Design, Automation and Test in Europe*, pp. 1250–1255, March 2008. [16](#)

- [24] A. B. Kahng and S. Kang, “Accuracy-configurable adder for approximate arithmetic designs,” in *DAC Design Automation Conference 2012*, pp. 820–825, June 2012. 17
- [25] R. Ye, T. Wang, F. Yuan, R. Kumar, and Q. Xu, “On reconfiguration-oriented approximate adder design and its application,” in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 48–54, Nov 2013. 17
- [26] S. Hashemi, R. I. Bahar, and S. Reda, “Drum: A dynamic range unbiased multiplier for approximate applications,” in *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 418–425, Nov 2015. 19, 20
- [27] R. Zendegani, M. Kamal, M. Bahadori, A. Afzali-Kusha, and M. Pedram, “Roba multiplier: A rounding-based approximate multiplier for high-speed yet energy-efficient digital signal processing,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, pp. 393–401, Feb 2017. 21
- [28] L. Chen, J. Han, W. Liu, and F. Lombardi, “Design of approximate unsigned integer non-restoring divider for inexact computing,” in *Proceedings of the 25th Edition on Great Lakes Symposium on VLSI, GLSVLSI ’15*, (New York, NY, USA), pp. 51–56, ACM, 2015. 22, 23
- [29] R. Zendegani, M. Kamal, A. Fayyazi, A. Afzali-Kusha, S. Safari, and M. Pedram, “Seerad: A high speed yet energy-efficient rounding-based approximate divider,” in *2016 Design, Automation Test in Europe Conference Exhibition (DATE)*, pp. 1481–1484, March 2016. 24, 25, 38
- [30] S. Hashemi, R. I. Bahar, and S. Reda, “A low-power dynamic divider for approximate applications,” in *2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, June 2016. 26, 27
- [31] C. Liu, J. Han, and F. Lombardi, “An analytical framework for evaluating the error characteristics of approximate adders,” *IEEE Transactions on Computers*, vol. 64, pp. 1268–1281, May 2015. 28
- [32] V. Mrázek, R. Hrbáček, Z. Vašíček, and LukášSekanina, “Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods,” in *Proc. of the 2017 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pp. 258–261, European Design and Automation Association, 2017. 28, 29
- [33] S. Rehman, W. El-Harouni, M. Shafique, A. Kumar, and J. Henkel, “Architectural-space exploration of approximate multipliers,” in *Proceedings of the 35th International Conference on Computer-Aided Design, ICCAD ’16*, (New York, NY, USA), pp. 80:1–80:8, ACM, 2016. 29

Apéndice A

Manual de uso

A.1. Requerimientos

Los requerimientos varían de acuerdo al proceso que se quiera realizar a como sigue en la siguiente tabla.

Tabla A.1: Requerimientos de la herramienta

	Requerimiento	Proceso
g++	Compilador para C++	Compilación código
Modelsim	Agregado al \$PATH	Simulación Pre y Post Síntesis
Synopsys (DC compiler)	Agregado al \$PATH	Estimación de área potencia y retardo
Synopsys (PrimeTime)	Agregado al \$PATH	Estimación de potencia a nivel de compuertas

A.2. Configuración previa

Se debe establecer en el archivo de configuración *config.cfg* las rutas donde se encuentra la biblioteca de celdas que se desean usar, la ruta de los archivos de salida y la escala de tiempo usada para simulación, todo esto está debidamente indicado en el archivo.

A.3. Ejecución

./AAUG [-a | -m | -d] <Nombre de unidad> -bw <Número de bits> -l <Número de bits aproximados> -r <Número de bits R> -p <Número de bits P> [-gen | -sim | -val | -syn | -psy] [-rand | -full] -c <Número de combinaciones> -seed <semilla> [-MED | -ER | -ALL]

Tabla A.2: Opciones de la herramienta

Opción	Parámetro requerido	Descripción
-a	<Nombre de unidad>	Selección de sumadores
-m	<Nombre de unidad>	Selección de multiplicadores
-d	<Nombre de unidad>	Selección de divisores
-bw	<Número de bits>	Ancho de bits para la unidad deseada
-l	<Número de bits aproximados>	Bits a aproximar o tamaño de subsumador
-r	<Número de bits R>	Bits R de subsumador o filas
-p	<Número de bits P>	Bits P de subsumador o columnas
-gen	-	Selecciona proceso hasta generación
-sim	-	Selecciona proceso hasta simulación
-val	-	Selecciona proceso hasta validación
-syn	-	Selecciona proceso hasta síntesis
-psy	-	Selecciona proceso hasta post síntesis
-rand	-	Selecciona valores de prueba aleatorios
-full	<Número de combinaciones >	Selecciona todas las posible combinaciones de valores
-seed	<semilla >	Selecciona la semilla si los números son aleatorios
-MED	-	Selecciona métrica de distancia de error promedio
-ER	-	Selecciona métrica de tasa de error
-ALL	-	Selecciona todas las métricas posibles

Tabla A.3: Referencia a parámetros de unidades

Unidad	Parámetros	Referencia
LPA	m	bw
	n	l
ETAII	m	bw
	l	l
ACAI	m	bw
	l	l
ACAII	N	bw
	l	l
GDA	m	bw
	R	r
	P	p
GeAr	m	bw
	R	r
	P	p
DRUM	N	bw
	k	l
RoBa	N	bw
	N	bw
AXDnr	EXCS	l
	Filas	r
	Columnas	p
SEERAD	N	bw
	Precisión	l
DAC	N	bw
	k	l

