Una aproximación al diseño de un Sumador de Punto Flotante de Doble Precisión basado en el Estándar IEEE 754

Sergio D. Nemirovsky - Eduardo A. Daniello
Miguel A. Sagreras - Alberto E. Dams

Facultad de Ingeniería
Universidad de Buenos Aires
http://www.fi.uba.ar

snemiro@fi.uba.ar, edaniel@fi.uba.ar,
msagre@cactus.fi.uba.ar, adams@cactus.fi.uba.ar

The Floating Point addition operation has the singularity to need a deep analysis of the operands to verify all the particular issues, without requiring a high demand of integrated logic. This article proposes a IEEE 754 compliant hardware design which solves the double precision addition operation. This design is part of a Double Precision Floating Point Unit developement project. This solution is FPGA (Field Programmable Gate Array) technology oriented, but a future ASIC (Application Specific Integrated Circuit) implementation is not discarded.

La operación de suma en aritmética de Punto Flotante reviste la particularidad de, sin demandar volúmenes de lógica elevados, necesitar un tratamiento intensivo de los operandos a fin de cubrir tanto los casos particulares como cuestiones de rango entre los números a sumar. En este trabajo se propone un diseño de hardware que, cumpliendo con el Estándar IEEE 754, tiende a resolver la operación de suma en Doble Precisión. El diseño es parte de un proyecto de desarrollo de una Unidad de Punto Flotante de Doble precisión según el estándar antes mencionado. El desarrollo está orientado a la implementación en tecnología FPGA, (Field Programmable Gate Array), aunque no se descarta una posterior adecuación del mismo a implementaciones ASIC, (Application Specific Integrated Circuit).

Una aproximación al diseño de un Sumador de Punto Flotante de Doble Precisión basado en el Estándar IEEE 754

Sergio D. Nemirovsky - Eduardo A. Daniello
Miguel A. Sagreras - Alberto Dams

Facultad de Ingeniería
Universidad de Buenos Aires
http://www.fi.uba.ar

snemiro@fi.uba.ar, edaniel@fi.uba.ar,
msagre@cactus.fi.uba.ar, adams@cactus.fi.uba.ar

Resumen

La operación de suma en aritmética de Punto Flotante reviste la particularidad de, sin demandar volúmenes de lógica elevados, necesitar un tratamiento intensivo de los operandos a fin de cubrir tanto los casos particulares como cuestiones de rango entre los números a sumar. En este trabajo se propone un diseño de hardware que, cumpliendo con el Estándar IEEE 754, tiende a resolver la operación de suma en Doble Precisión. El diseño es parte de un proyecto de desarrollo de una Unidad de Punto Flotante de Doble precisión según el estándar antes mencionado. El desarrollo está orientado a la implementación en tecnología FPGA, (Field Programmable Gate Array), aunque no se descarta una posterior adecuación del mismo a implementaciones ASIC, (Application Specific Integrated Circuit).

1. Antecedentes Temáticos

El total del contenido de la presente publicación surge como corte de un trabajo de investigación cuyo objeto es lograr una Unidad de Punto Flotante (FPU) de doble precisión compatible con el estándar IEEE 754. El equipo de investigación está conformado por los autores del presente trabajo siendo éste el primero relativo al objeto de estudio del mismo. Por esa razón, se inició una búsqueda bibliográfica a fin de recabar información sobre el estado del arte del desarrollo de FPUs tanto en el campo comercial como en el académico. Los proyectos comerciales divulgan sólo la arquitectura global de sus diseños, lo que no permite conocer detalles de la implementación. Por otra parte, las arquitecturas desarrolladas por entidades académicas suelen ser abiertas a la comunidad y hasta se pueden obtener sus códigos fuente en lenguajes descriptores (VHDL, Verilog, etc.).

De entre los trabajos diponibles y publicados en diversos medios, sólo dos correspondientes a los denominados “grupos de desarrollo abiertos” dan una idea acabada del objeto de estudio que investigan. Son los trabajos del “Free IP Cores Projects” [6] de Open Cores y el “Digital IC Project 2001” [7] correspondiente al procesador LEON de European Space Agency o ESA.

El primero de estos trabajos se caracteriza por implementar una FPU completa de Simple Precisión (32 bits) que realiza las cuatro operaciones básicas de aritmética y conversiones entre enteros y números de punto flotante en ambos sentidos. El sucinto informe redactado por Rudolf Usselmann expone un sencillo diagrama de bloques de la arquitectura, las señales de entrada y salida y los modos de redondeo soportados por la FPU.

En simultáneo con el desarrollo del trabajo principal del cual surge este documento, se concluyó un proyecto de ESA cuyo informe fue redactado por Martin Kasprzyk. Dicho informe explica la implementación de una FPU para el procesador LEON, basado en arquitectura SPARC V8.

Ninguno de los proyectos anuncia si la implementación final está orientada a ASIC, FPGA u otra tecnología, así como no revelan detalles sobre la implementación interna de cada una de estas etapas.

En todos los casos se dispone del código fuente de los trabajos. Se intentaron sintetizar los mismos con resultados negativos debido a incompatibilidad en las herramientas de síntesis.

2. Descripción General del Algoritmo

Para dar un marco al presente trabajo, se comienza por presentar la secuencia de pasos que impone ejecutar la operación de suma de dos operandos, los que se resumen en las siguientes cuatro etapas:

1. **Reconocimiento y Desempaquetado de los operandos:** Reconocer los operandos implica su identificación para separar las sumas triviales y aquellas que conducen a excepciones de las sumas de operandos regulares.
2. **Operación de Suma propiamente dicha:** Es la ejecución de la suma para cada uno de los casos apuntados en el ítem anterior.
3. **Post-Normalización del Resultado Intermedio:** De ser necesario hacerlo, se aplica el criterio de corrección de la mantisa, (y por consecuencia al exponente), para respetar el formato de mantisa impuesto por el estándar.
4. **Redondeo y Empaquetado del Resultado Definitivo:** Basado en el modo de redondeo elegido externamente, se procede a redondear el resultado y a encuadrarlo en el formato del estándar.

3. Arquitectura

En búsqueda de una solución de altas prestaciones, se requiere desde un principio reducir al mínimo la latencia. Por lo tanto, se ha propuesto una estructura de Pipeline para la arquitectura del sumador. Esta elección se sustenta en que permite el tratamiento de múltiples instrucciones con mínima latencia, (la de una etapa de Pipeline), siempre que no haya dependencias de datos.

Se desechan opciones tales como las soluciones por cálculo iterativo así como la ejecución “Flowthru”.

El diseño basado en arquitecturas iterativas posee la ventaja de requerir menores recursos de hardware, por lo que impacta en forma positiva en el tamaño de la lógica FPGA a implementar. Su principal desventaja y la razón por la cual se desecha el uso de este tipo de arquitecturas es la generación de latencia, dado que no permite la ejecución de la instrucción *n + 1* hasta tanto el bucle iterativo no haya concluído la ejecución de la instrucción *n*.

La ejecución ”Flowthru”es combinacional pura, y fuerza el tratamiento de la operación compuesta por las etapas antes descriptas en un único bloque. Un circuito combinacional puro que cumpla con todas las tareas necesarias sólo permitiría tratar las instrucciones de a una por vez y la latencia estaría directamente ligada a la demora del circuito en estabilizar la totalidad de sus valores.

4. Clasificación de la Suma según su Tratamiento

Dentro de las operaciones que se deben resolver están aquellas que utilizan dos números regulares o no singulares, así como aquellas que conducen a resultados triviales o a excepciones. Sumar cero o infinito a un número regular, o resolver operaciones tales como $(\infty - \infty)$ conducen siempre a resultados rápidamente calculables, mientras que la suma de dos números regulares impone seguir los cuatro pasos básicos antes mencionados.

Las sumas “triviales” o que conducen a resultados del tipo de excepción, generan resultados que son:

- Un operando no modificado.
- Un valor singular conocido.

Por lo tanto, es conveniente tratar a dichas operaciones matemáticas en forma separada, dando así origen a una línea de cálculo de atención de excepciones, en este trabajo denominada **Pipeline de Excepciones**. Estas operaciones deben pasar el resultado en forma directa, no requiriendo el desempaquetado del operando, ni su posterior post-normalización y redondeo.

Por otra parte, toda suma de dos números regulares, es atendida por el sumador propiamente dicho, y su línea de cálculo se denomina **Pipeline de Suma**. Estas operaciones, necesitan el desempaquetado previo de sus dos operandos y, posteriormente a la suma, la post-normalización, empaquetado y redondeo del resultado.

Se analizan ambas situaciones por separado, para luego proponer una lógica de integración circuital.

4.1. Trivialidades y Excepciones

Se trata aquí la lógica de aquellas sumas consideradas triviales o que conducen a indeterminaciones matemáticas y excepciones. Llamando N a cualquier número regular, las operaciones incluidas en este grupo, son:

- $N + (\pm 0)$
- $N + (\pm \infty)$
- $(\pm)0 + (\pm \infty)$
- $NaN + (\pm 0)$
- $NaN + N$
- $NaN + (\pm \infty)$
- $(+\infty) + (-\infty)$

Entonces, es necesario clasificar a los operandos entrantes, para así poder separar las operaciones triviales y las excepciones. Un análisis de los operandos singulares muestra que con una lógica simple, se detecta si un operando es regular o si es alguno de los tres singulares (cero, infinito, NaN), ver fig.1.

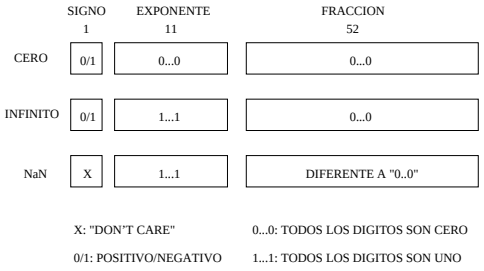

Figura 1: Números Singulares en IEEE 754

Como se expuso anteriormente, las operaciones triviales dan siempre por resultado, o uno de los números singulares, o uno de los operandos ingresados.
Por lo tanto, se debe plantear una segunda etapa de lógica que, recibiendo la identificación de los operandos entrantes, pueda:

- Generar una señal identificatoria del resultado correspondiente, si éste es un número singular.
- Pasar el operando que será resultado cuando así corresponda.

El último paso es generar las constantes correspondientes a resultados singulares en base a las señales identificatorias correspondientes. Elegido un **Pipeline** como solución al sumador siempre será preferible pasar señales de un bit hasta la última etapa (generando en ella las constantes), que pasar constantes de 64 bits a lo largo del mismo.

4.2. Tratamiento de Números Regulares

Se analiza ahora el caso más común en el tramiento de la suma: Cuando los dos operandos son regulares. Para este fin, y siguiendo con el estándar IEEE 754, se debe desempaquetar los operandos. Esto implica agregar el bit implícito al significando, para así convertirlo en la auténtica mantisa del número, (ahora de 53 bits). Ya desempaquetados ambos operandos, se inicia la operación propiamente dicha.
La operación de suma algebraica sigue la siguiente lógica de análisis:

- Se suma (o se resta) al operando más chico respecto del más grande.
- Se deben igualar los exponentes, aumentando el valor del exponente del operando menor hasta el del operando mayor. Para mantener al número en su correcta magnitud, al mismo tiempo, se desplaza a razón de un bit hacia la derecha su mantisa por cada bit de incremento del exponente. Si a consecuencia de esto, la mantisa completa del operando desplazado quedara fuera del rango de suma de las mantisas, el resultado requiere de una precisión no otorgada por esta arquitectura, por lo que el resultado es igual al operando de mayor magnitud con su signo.
- Se realiza la suma de los operandos si son del mismo signo, siendo el signo del resultado igual al signo de los operandos. Si son de signos opuestos, se realiza la resta de los operandos, siendo el signo del resultado igual al signo del operando de mayor valor absoluto.
- De ser posible, se normaliza el resultado, dando a la mantisa la estructura que permite su posterior empaquetado y ajustando al exponente en consecuencia.
- Se redondea según el criterio elegido el resultado y de ser posible se lo empaqueta.

5. Enrutamiento

Resta a continuación enrutar la operación; ya sea eligiendo el tratamiento de casos singulares (**Pipeline de Excepciones**) o de casos regulares (**Pipeline de Suma**). En principio se requiere de una lógica que detecte si ambos números son regulares o si al menos uno es singular para definir cuál de los Pipelines se encargará de ejecutar la operación. Es sencillo verificar analizando el esquema de la figura 1, que en el estándar IEEE 754 todo número no singular es regular. Por lo tanto, basta con verificar la negación de las señales que orientan Tivialidades y Excepciones para poder, en base a esto, enrutar los operandos regulares hacia el **Pipeline de Suma**, mientras que la lógica directa enruta el operando o las señales correspondientes hacia el **Pipeline de Excepciones**.

6. Esquema Propuesto para el Sumador - Orientación a Implementación FPGA

La tarea que sigue es armar un emblocamiento de módulos funcionales que permitan implementar el sumador. En particular se orienta la solución a Tecnologías FPGA. Partiendo de la elección de arquitectura de Pipeline para los bloques constitutivos, se trata de balancear lo más posible los retardos por etapa. Además se tratará de obtener el mejor balance posible entre la longitud y el procesamiento paralelo del Pipeline para evitar demoras ante dependencias de datos del tipo “Read after Write” al tiempo de ajustar la demora por etapa al mínimo estricto. Si bien no ocurre en todos los casos, se puede plantear la situación de que una implementación en particular admita un mayor grado de paralelismo el cual se vea severamente limitado por restricciones topológicas de la implmentación en FPGA. Concretamente, si se requieren excesivas rutas internas del dispositivo FPGA en una sola etapa, se compromete la disponibilidad del recurso para el resto de la implementación derivando esto en “retardos por enrutamiento”. Estos retardos comienzan a pesar en el balance de retardos de la lógica haciendo contraproducente el excesivo paralelismo.

Todas estas consideraciones previas conducen a las siguientes decisiones de implementación:

- La lógica de enrutamiento de la operación no excede los 6 niveles de compuerta para identificación de operandos y no excede los 6 niveles de compuerta para decidir la ruta.
- la lógica del **Pipeline de Excepciones** o pasa operandos o genera constantes, por lo que no excede los 8 niveles de compuertas.
- La lógica del **Pipeline de Suma** requerirá de múltiples operaciones encadenadas en el tiempo, entre las que se cuentan:
    - Comparación de exponentes, (por resta).
    - Comparación de Mantisas, (uso de comparadores).
    - Comparación de signos, (una compuerta X-OR).
    - Desplazamiento de la mantisa menor, (uso de barrel-shifters).
    - Suma de mantisas con 64 bits de precisión interna, (uso de sumador/restador de 64 bits o múltiples unidades en cascada).
    - Ajustes mantisa/exponente del post-sumador, (uso de barrel-shifters y de sumador).
    - Ajustes de redondeo.

El simple análisis de esta enumeración de etapas permite comprobar que etapas como barrel shifters y grandes sumadores exceden con creces los niveles de compuerta necesarios para decidir sobre el enrutamiento de operandos, así como para computar las excepciones. Por lo tanto, se escoge el desarrollo del enrutador en una sola etapa, distribuyendo los operandos a ambos pipelines mediante un Bus. Por su parte el **Pipeline de Excepciones** posee también una lógica posible de implementar en una sola etapa. El pipeline de la suma requerirá desde ya múltiples etapas. Además, por su complejidad, requerirá de un tratamiento más preciso, razón por la cual será analizado aparte.

7. Etapa de Enrutamiento

Se compone de las etapas lógicas que cumplen con la operatoria antes descripta. Primero se analizan los dos operandos mediante funciones booleanas, generando las señales que identifican a los operandos singulares y a los regulares, (cero, infinito, NaN, Normal). Luego en una segunda etapa de lógica booleana se procede a identificar los casos de excepciones o de sumas regulares, de donde se obtienen las señales que pasan a las siguientes etapas:

- En caso de suma regular se pasan los dos operandos al pipeline de suma, incorporando el bit implícito (desempaquetado).
- En caso de excepciones, si el resultado es un operando se pasa al pipeline de excepciones, y si el resultado es un número singular se pasa la señal correspondiente con valor lógico “1”.
- Como los Pipeline se encuentran moviendo datos entre etapas permanentmente, para validar los datos útiles, se genera y distribuye a sendas rutas un bit de validación que se establece en valor lógico “1” cuando la etapa del pipeline conduce datos válidos.

8. Lógica del Pipeline de Excepciones

Esta es la etapa más sencilla. Siempre se reciben un operando y las señales indicadoras de resultado singular. Una lógica combinacional simple establece si se pasa el operando entrante o si se presenta a la salida un número singular, mientras que con las señales correspondientes a las singularidades, elige la constante correcta de entre las posibles. Se presentan además a la salida todos los indicadores (flags) necesarias a la salida del Sumador completo.

9. Lógica del Pipeline de Suma

Este es el punto de desarrollo más importante del sumador, razón por la cual se adjuntan los cauces de ejecución en dos versiones: Cauce de cinco etapas para altas prestaciones (Ver Figura 2) y cauce de ocho etapas para lógicas de bajo costo (Ver Figura 3). En principio se tratan por separado los bloques Sumador, Post-Sumador y Redondeo para mantener coherencia con el desarrollo funcional del Módulo General. Para lograr un balance adecuado entre cantidad de etapas y la cantidad de lógica por etapa, se procede a desarrollar según la siguiente secuencia.

1. Se desarrolla el módulo funcional en modo combinacional puro.
2. Se comienza por simular individualmente los bloques constitutivos que lo componen, registrando los retardos combinacionales de cada bloque.
3. A continuación se implementa en modo combinacional puro el módulo funcional. Se compila y se simula su operación obteniendo tiempos que servirán sólo como referencia.
4. Con los tiempos por bloque y la disposición en el diagrama del módulo se procede a establecer los límites entre etapas del pipeline, de modo de balancear la carga de tiempos y estableciendo el compromiso de cantidad de etapas versus retardo por etapa más conveniente. Puede ser necesario separar un bloque constitutivo en partes para descargar etapas y/o reacomodar el(los) sub-bloque(s), agregar una(varias) etapa(s) al pipeline.

5. Se procede a compilar el conjunto y simular operaciones. Se comparan los resultados contra los del módulo combinacional puro.

9.1. Desarrollo Final del Sumador

9.1.1. Sumador

El sumador internamente desarrolla tres tareas principales:

- Análisis de los operandos entrantes y decisión de rutas internas
- Ajuste de mantisas y exponentes
- Suma propiamente dicha.

Por requerimiento del estándar IEEE 754 se extienden las mantisas de 53 bits a 64 bits utilizando "zero padding" a derecha. Para analizar los operandos se comparan los exponentes mediante restadores, ($exp(A) - exp(B)$ y $exp(B) - exp(A)$), las mantisas con comparadores de 53 bits, y los signos (con compuerta X-OR). Con el resultado de la comparación de exponentes se genera el exponente del resultado, y se determina el enrutamiento correcto de las mantisas a operar. El sumador debe respetar el signo de los operandos entrantes, razon por la cual la comparación de los signos define si la operación suma o resta las mantisas. Para ajustar la mantisa del operando de menor exponente se utilizan barrel shifters en cascada, ubicándolos en etapas diferentes del pipeline. Esto se debe a que el barrel shifter es una de las etapas sensibles en el estudio de retardos por etapa del pipeline. El sumador propiamente dicho se implementa con un sumador/restador de 64 bits. Según sea la familia lógica sobre la que se implementa el Sumador, la incidencia relativa de los bloques constitutivos en los tiempos por etapa se modifica fuertemente. Esto hace variar la disposición en etapas del bloque sumador permitiendo mejorar el compromiso entre velocidad y longitud del pipeline.
La figura 3 nos muestra el sumador con cuatro etapas de suma, mientras que la figura 2 presenta la versión con una sola etapa de suma.

9.1.2. Post-Sumador

La idea conceptual de este módulo es, de ser posible, normalizar la mantisa del número entrante y realizar las correcciones correspondientes a los indicadores (flags) y al exponente. Para poder realizar estas tareas se debe desplazar la mantisa hacia la izquierda hasta lograr que la cifra más significativa de la misma sea "1". A fin de mantener el valor correcto del número representado, se corrige el exponente, decrementándolo la cantidad de cifras desplazadas anteriormente. Los problemas que se pueden presentar son los siguientes:

1. Que todas las cifras de la mantisa sean cero.
2. La mantisa debe ser desplazada hacia la izquierda una cantidad de posiciones mayor al valor numérico del exponente.

En el primer planteo, el resultado de la operación es cero. Se indica esta situación encendiendo el indicador de ZERO. En el segundo caso, no es posible normalizar el número. Aquí nuevamente se nos presentan dos situaciones, caracterizadas por la cantidad de cifras a desplazar a la mantisa y al valor del exponente.
Llamemos EXP al valor del exponente, y DESPL a la cantidad de cifras a desplazar. Surge de comparar su diferencia contra valores particulares el siguiente análisis:

1. $EXP - DESPL \leq -53$: Resultado no representable. Se enciende el indicador (flag) de UNDERFLOW.
2. $-53 < EXP - DESPL \leq 0$: El resultado es un número del tipo DENORMAL. Se ajusta el exponente a cero desplazando EXP cifras a la mantisa. Se enciende el indicador (flag) de DENORMAL.

Si se observa con atención el diagrama en bloques del Post-Sumador, se encuentra una señal llamada man_ov, salida del bloque Sumador.
Esta señal representa un desbordamiento (OVERFLOW) en la cadena de sumadores del bloque anterior. Es el Post-Sumador el bloque responsable de atender esta situación, realizando el apropiado ajuste de exponente y de ser necesario, encendiendo el indicador de OVERFLOW.

9.1.3. Redondeo

Este módulo aplica el modo de redondeo que se selecciona externamente utilizando dos bits según el cuadro 1. Básicamente, se pude esquematizar su operatoria en tres sub-operaciones

1. Ajuste de mantisa y exponente por redondeo.
2. Generación de constantes.
3. Selección del Resultado.

Los ajustes de mantisa y esponente dependen del modo de redondeo seleccionado y van desde el simple truncamiento de la cifra hasta el caso (único) de recuperación del mayor denormal para su conversión en el mínimo número representable. La lógica comprende el uso de sumadores y multiplexores de un bit. Se contemplan los casos especiales de redondeo para los que se producen transiciones entre el campo numérico representable, las bandas de Overflow y Underflow y los Números Denormalizados. La generación de constantes no reviste mayor complejidad, y mediante el uso de Multiplexores y lógica combinacional sencilla se procede a seleccionar el resultado de la operación. Esta etapa, cumple con el empaquetado del resultado siempre y cuando el mismo no sea un resultado denormalizado. Además y como última función presenta todas las señales de indicadores (flags) necesarias a la salida del sumador completo.

Código	Descripción
00	Hacia $+\infty$
01	Hacia 0
10	Hacia el número más cercano
11	Hacia $-\infty$

Cuadro 1: Modos de Redondeo

9.2. Notas sobre el Criterio de Balanceo de Cauces

El balance entre longitud de cauce y retardo por etapa es la característica principal a tener en cuenta para un correcto de las arquitecturas basadas en segmentación de cauce. Merece la pena detallar en qué puntos del trabajo fué necesario apoyarse en la optimización de este compromiso de parámetros. Una de las implementaciones que en particular requiere de este tratamiento es la etapa de "barrel-shifter" utilizado para desplazar las mantisas, tanto en el sumador de cinco como de ocho etapas. En ambos casos su retardo excede en tiempo al de los demás bloques constitutivos, justificando su segmentación en dos etapas logrando el deseado balance general del "pipeline". Si bien esto agrega una etapa más a la longitud del cauce, la ventaja en frecuencia operativa resulta notable en ambos casos. En forma similar, la implementación final de la suma de mantisas requiere de una segmentación en cuatro etapas al utilizar dispositivos de bajo costo, tales como APEX 20K de Altera. Debido a que estos dispositivos no poseen bloques internos de arquitectura optimizados para fines particulares (Multiplicadores, FFT, etc.), segmentar el cauce de suma en cuatro etapas eleva notablemente la frecuencia operativa del sumador pagando como precio ciclos de reloj en la carga y drenaje del cauce.

10. Lógica de extracción

Los alcances de este trabajo no contemplan el desarrollo de una lógica de extracción de los datos. Tal tarea, se considera fundamental dado el desbalance de longitud entre los pipelines desarrollados. Sólo a los fines de prueba del presente trabajo, se implementa un multiplexor a la salida como última etapa, de forma tal de permitir la extracción de resultados e indicadores de cualquiera de las dos vías de procesamiento. Por otra parte, tampoco se desarrolla la lógica de control que gestiona los ciclos de detención de las ramas de pipeline, (Stall Cycles), pues dicho desarrollo se encuentra muy relacionado al de la lógica de extracción.

11. Notas sobre la Compilación y Simulación en FPGA

El código VHDL desarrollado toma ventaja de los componentes LPM (Library of Parametrized Modules) desarrollados por Altera Corporation. Gracias a tales bibliotecas se optimiza el rendimiento individual de cada bloque constitutivo, redundando esto en una mejora del rendimiento general. Los bloques que fueron optimizados mediante el mencionado criterio son: Sumadores, comparadores y multiplexores. Bloques como el barrel-shifter se desarrollan utilizando multiplexores LPM. Aquellos bloques que implementan funciones lógicas sencillas han sido confeccionados mediante funciones lógicas de nivel uno (basadas en compuertas e inversores). Los registros que delimitan las etapas del cauce son implementados mediante arreglos de flip-flop D convencionales. A continuación se presentan los resultados obtenidos por los autores del presente trabajo compilando las dos versiones del Sumador cuya diferencia reside sólo en la arquitectura del bloque sumador. Uno de ellos implementa un sumador de una sola etapa de 64 bits de longitud de palabra, mientras que el otro implementa el sumador en 4 etapas de 16 bits cada una. El primero da por resultado un pipeline para el Sumador de cinco etapas para un total de diez etapas contabilizando todo el camino de datos. El segundo posee un pipeline de ocho etapas, y el total del camino de datos alcanza a trece. Luego se compiló cada uno de los desarrollos para las familias lógicas ACEX1K, APEX II, FLEX20KE, CYCLONE, y STRATIX de la firma ALTERA. En todos los casos se solicitó al compilador, (QUARTUS II WEB Edition Ver 2.1) optimizar el resultado para máxima velocidad. El cuadro 2 sintetiza los resultados obtenidos, permitiendo tener una idea de las frecuencias de reloj a obtener con cada familia lógica y esquema de implementación utlilizados en este trabajo. Como habitualmente ocurre, tener idea de la cantidad real de operaciones por segundo que se podría alcanzar en estos casos está íntimamente relacionado con la naturaleza de las aplicaciones en las cuales el sumador puede ser utilizado. La secuencia de las mismas y la dependencia de datos influirán en la cantidad de operaciones por segundo que se podrán resolver. No habiéndose tratado la lógica de extracción en el presente trabajo, sólo se puede hablar de resultados parciales del desempeño de cada rama del sumador. A raíz de esto, sólo se puede garantizar que libre de dependencia de datos es posible extraer por cada rama del sumador una instrucción por ciclo, resultando

Familia	Sumador 5 etapas		Sumador 8 etapas	
	Cantidad LEs	Velocidad (MHz)	Cantidad LEs	Velocidad (MHz)
Acex1k	2480	56.04	2814	51.22
Apex II	2535	76.35	2828	90.19
Flex20KE	2537	57.57	2824	55.24
Cyclone	2060	93.05	2134	94.32
Stratix	2171	114.46	2258	109.39

Cuadro 2: Resultados de Compilación

la cantidad de operaciones por segundo igual a la frecuencia máxima alcanzada por cada implementación.

12. Conclusiones

Surge del análisis de los valores obtenidos que factores ajenos a los retardos de cada etapa (retardos combinacionales) inciden en el resultado de la compilación. Las rutas críticas en recorridos internos de las soluciones generadas por el compilador distorsionan los parámetros temporales esperados. En gran medida se debe a la inmadurez del compilador disponible para la ejecución del presente trabajo. Como consecuencia de esto, hay situaciones en las que agregar lógica a un desarrollo modifica el ruteo interno de la FPGA al compilar y en contra de lo a priori esperado se obtiene una mejora en la velocidad, y viceversa. Por otra parte, las familias más modernas de Chips FPGA, (CYCLONE y STRATIX) permiten obtener soluciones de mayor velocidad más allá del costo inherente a la familia de dispositivos. Esto se debe fundamentalmente al reciente progreso de las tecnologías FPGA. Cabe aclarar que el presente desarrollo está integramente escrito en lenguaje VHDL, y que se utilizaron componentes LPM, lo que no hace al código portable para su uso en otras tecnologías. Los resultados obtenidos serán en todos los casos ligeramente diferentes de los esperados debido a que el sumador aquí presentado en sus dos variantes es parte constitutiva de un desarrollo mayor, (FPU de 64 bits según IEEE 754). A causa de ello, los módulos compilados poseen un muy bajo porcentaje de lógica ajena al objetivo del presente documento. La incidencia de estas divergencias se considera mínima y no modifica en lo absoluto la esencia y contenido del presente trabajo.

Referencias

[1] IEEE Standards Asociation, *IEEE 754-1985: Standard for Binary Floating Point Arithmetic.*

[2] John L. Hennesy David A. Patterson, *Computer Architecture: A Cuantitative Approach*, Second Edition 1996, Morgan Kauffman.

[3] Sun Microsystems, *Numerical Computation Guide*, 1996, publicado por Sun Press.

[4] Zainalabedin Navabi, *VHDL: Analysis and Modeling of Digital Systems*, International Edition 1993, McGraw Hill.

[5] Peter J. Ashenden, *The VHDL Cookbook*, First Edition 1990, University of Adelaide, South Australia.

[6] Rudolf Usselmann, *Open Floating Point Unit*, The Free IP Cores Projects.

[7] Martin Kasprzyk, *Floating Point Unit - Digital IC Project 2001*, Enero 2002, European Space Agency.

[8] AMD, *Am29C327 User's Manual*, 1988, publicado por Advanced Micro Devices.

[9] ALTERA, *ACEX 1k Programmable logic Device Family Data Sheet Ver 3.3*, Septiembre 2001, publicado por Altera Corporation.

[10] ALTERA, *APEX II Programmable logic Device Family Data Sheet Ver 3.0*, Agosto 2002, publicado por Altera Corporation.

[11] ALTERA, *FLEX 10k Embedded Programmable logic Family Data Sheet Ver 4.1*, Marzo 2001, publicado por Altera Corporation.

[12] ALTERA, *CYCLONE FPGA Family Data Sheet Ver 1.0*, Septiembre 2002, publicado por Altera Corporation.

[13] ALTERA, *STRATIX Programmable logic Device Family Data Sheet Ver 2.1*, Agosto 2002, publicado por Altera Corporation.

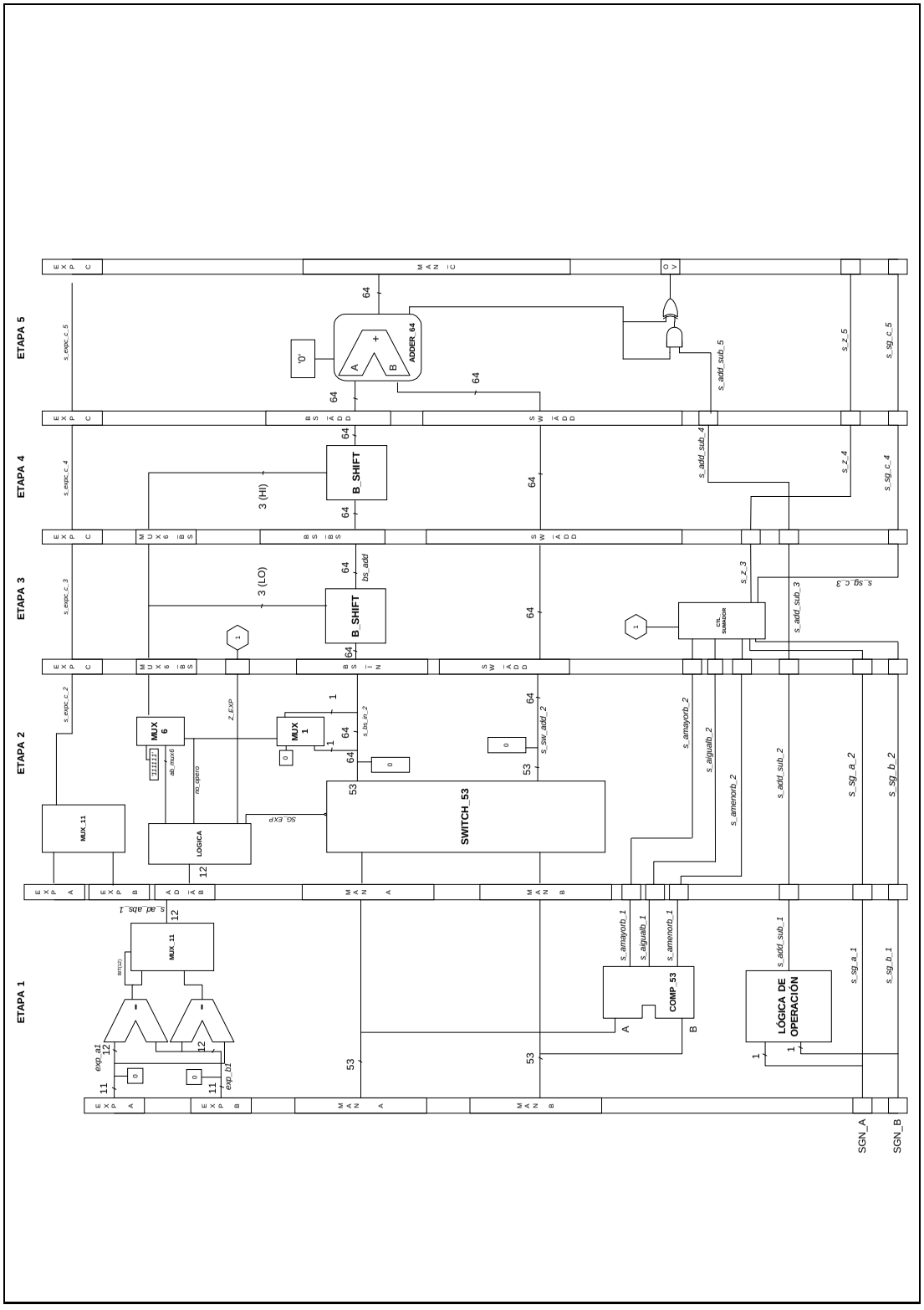

Figura 2: Sumador interno de 1 etapa

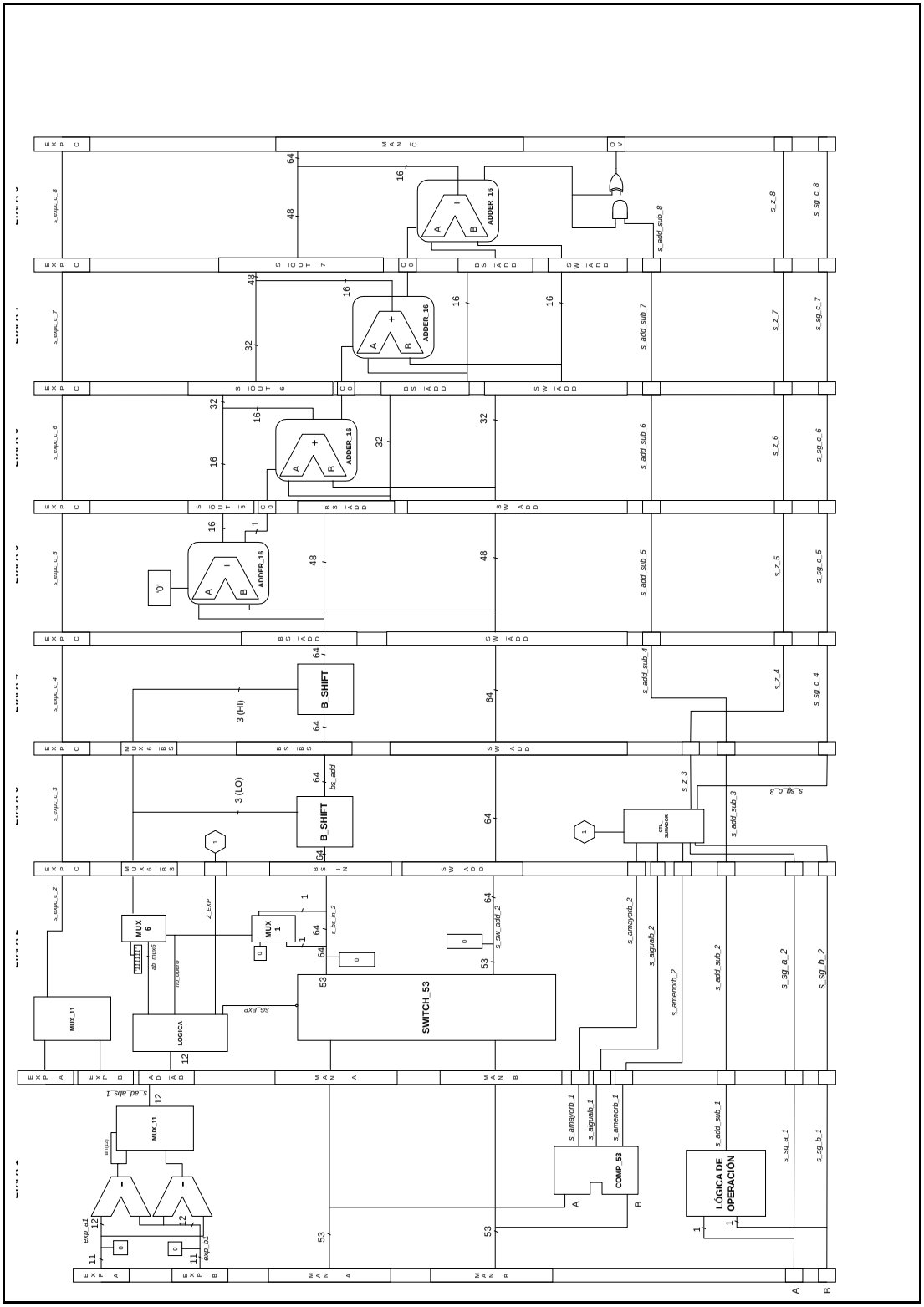

Figura 3: Sumador interno de 4 etapas