DISEÑO DE UN COPROCESADOR ASÍNCRONO QUE IMPLEMENTA EL ESTANDAR IEEE 754 EN SIMPLE PRECISIÓN

HENRY DELGADO ANGULO.

UNIVERSIDAD DEL VALLE - FACULTAD DE INGENIERIAS

ESCUELA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

PROGRAMA ACADEMICO DE INGENIERÍA ELECTRÓNICA

SANTIAGO DE CALI

2012

DISEÑO E IMPLEMENTACIÓN DE UN COPROCESADOR ASÍNCRONO QUE IMPLEMENTA EL ESTÁNDAR IEEE 754 EN SIMPLE PRECISIÓN

HENRY DELGADO ANGULO

Trabajo de grado presentado como requisito parcial

para optar al título de Ingeniero Electrónico

Director

RUBÉN DARÍO NIETO LONDOÑO, *PhD.*

UNIVERSIDAD DEL VALLE - FACULTAD DE INGENIERIAS

ESCUELA DE INGENIERÍA ELÉCTRICA Y ELECTRÓNICA

PROGRAMA ACADÉMICO DE INGENIERÍA ELECTRÓNICA

SANTIAGO DE CALI

2012

Nota de aceptación:

_____

_____

_____

Maribell Sacanamboy, *M.Sc*

_____

Ing. Nathaly Nieto Ramírez.

Santiago de Cali, 16 de enero de 2013.

CONTENIDO

LISTA DE TABLAS.

LISTA DE FIGURAS.

INTRODUCCIÓN

El presente trabajo de grado consiste en la concepción, diseño e implementación en hardware reprogramable, de un sistema que implementa el estándar para aritmética binaria en punto flotante IEEE 754 en simple precisión (32 bits) en sus operaciones básicas de suma/resta, multiplicación y división. Lo novedoso de esta propuesta es que la implementación en hardware se realiza utilizando especificación asíncrona de los módulos que conforman el sistema completo.

La descripción asíncrona del sistema se realizó utilizando la herramienta de especificación, síntesis, simulación e implementación comportamental conocida como Balsa System [1]. Esta herramienta se seleccionó teniendo en cuenta las siguientes consideraciones:

- Actualmente es la herramienta de libre distribución más avanzada para la descripción de sistemas asíncronos.
- Utiliza un lenguaje de alto nivel (también llamado Balsa).
- Integra ambientes para especificación, síntesis, y simulación de circuitos asíncronos, además de herramientas que permiten generar archivos de implementación sobre hardware real.

El sistema Balsa fué desarrollado en el *APT Group* (*The Advanced Processor Technologies Group*) de la *School of Computer Science, University of Manchester* (Inglaterra). *Balsa* es el nombre de un conjunto de herramientas de síntesis de sistemas de hardware asíncrono y también el nombre del lenguaje para describir esos sistemas. Las herramientas pueden correr sobre cualquier ambiente POSIX con X11 que manejen enteros al menos de 32 bits (*Linux*, *FreeBSD*, *MacOS X*, *Solaris*). Sin embargo, para producir una implementación en hardware real, ya sea en Silicio o en *FPGA*, se requiere el complemento de herramientas comerciales específicas: por ejemplo el software de diseño de *Xilinx*, o el ambiente de diseño de *Cadence* con una apropiada tecnología de librería de celdas [11].

El desarrollo del sistema se hizo utilizando los dos estilos de diseño digital: *síncrono y asíncrono*. Los módulos que implementan las funciones aritméticas básicas fueron desarrollados de manera individual y luego se integraron en cada sistema como un circuito completo. La idea fundamental de realizar la implementación en ambos estilos de diseño es la de comparar las figuras de mérito de desempeño, área utilizada y consumo de potencia, cuyos resultados se consignan en el capítulo final del documento.

Se decidió no incluir los conceptos sobre sistemas asíncronos debido a la restricción sobre el número de páginas del informe final y por considerar de mayor prioridad la inclusión de los conceptos básicos sobre el estándar IEEE 754 y los detalles de implementación de los distintos módulos en ambos estilos de diseño. Sin embargo, el lector que desee estudiar los conceptos más relevantes sombre diseño asíncrono puede consultar en [4], [12], [13].

1 ESTÁNDAR IEEE 754 PARA ARITMÉTICA BINARIA EN PUNTO FLOTANTE.

1.1 FORMATOS.

El Instituto de Ingenieros Eléctricos y Electrónicos (*Institute of Electrical and Electronics Engineers:* IEEE) definió el estándar que describe las operaciones aritméticas y su representación en punto flotante mediante el documento IEEE-754 de 2008 [3]. Aunque existen otras representaciones, esta es la representación más usada para números en punto flotante.

El estándar especifica:

- Tres formatos binarios, con codificación de 32, 64 y 128 bits.
- Dos formatos decimales, con codificación de 64 y 128 bits.
- Suma, Resta, Multiplicación, División, Raíz Cuadrada, operaciones de comparación.
- Conversión entre el formato entero y punto flotante.
- Conversión entre diferentes formatos de punto flotante.
- Conversión entre el formato en punto flotante básico y el carácter decimal.
- Excepciones en punto flotante y su manejo, incluyendo los que no son un número (*Not a Number: NaN*).

Signo: para el formato IEEE 754 en precisión simple, este bit es el más significativo (posición 31 del formato) como se indica en la Figura 1.1. Si el bit es 0 el dato es positivo y si es 1 el dato es negativo.

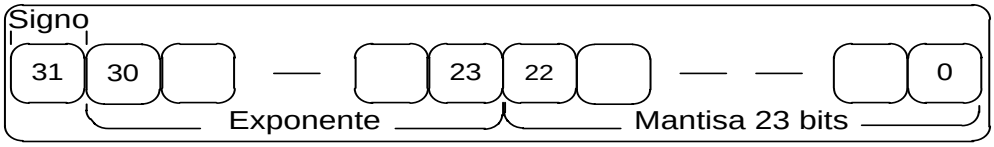

Figura 1.1. Formato IEEE 754.

Exponente: es de 8 bits, cubre los bits 23 al 30. El rango de valores va de 0 hasta 255, además utiliza un sesgo de 127 para representar números negativos sin usar bit de signo.

Mantisa: tiene 23 bits, va desde el bit 0 hasta el bit 22. La representación binaria permite un bit implícito que no se codifica ya que este siempre es 1 para números normalizados y se aprovecha para tener un bit más de resolución.

Los números se representan de la siguiente manera:

$$\boldsymbol{n = (-1)^{s} 2^{e} * \ 1, f \ (mantisa\ normalizada)} \qquad \textbf{(1)}$$

$$\boldsymbol{n = (-1)^{s} 2^{-126} * \ 0. \ f \ (mantisa\ sin\ normalizar)} \qquad \textbf{(2)}$$

Donde

$$f = \left(b_{23}{}^{-1} + b_{22}{}^{-2} + b_i{}^{n} + \cdots + b_0{}^{-23}\right) cuando\ b_i{}^{n} = 1\ o\ 0 \qquad (3)$$

$$s = signo\ (0\ es\ positivo, 1\ es\ negativo\ )$$

La Tabla 1.1 resume de forma detallada los valores especiales del estándar y algunos detalles mencionados anteriormente.

Tabla 1.1. Valores especiales de los números en punto flotante para el formato IEEE 754 en precisión simple.

S	*Exponente*	*Mantisa*	*Valor*
1	**00000000**	**00000000000000000000000**	***-cero***
0	**00000000**	**00000000000000000000000**	***+ cero***
1	**00000000**	**10000000000000000000000**	$-2^{0-127} * 0.2^{-1} = -2^{0-127} * 0.5$
0	**00000000**	**00000000000000000000001**	$+2^{0-127} x 0.2^{-23}$ $= +2^{0-127}$ $* 0.5\ el\ valor\ mas\ pequeño$
0	**00000001**	**01000000000000000000000**	$+2^{1-127} * 0.2^{-2} = +2^{1-127} * 1.25$
0	**10000001**	**00000000000000000000000**	$+2^{129-127} * 1.0 = 4$
0	**11111111**	**00000000000000000000000**	***+ infinito***
1	**11111111**	**00000000000000000000000**	***- infinito***
0	**11111111**	**10000000000000000000000**	***No es número (NaN)***
1	**11111111**	**10000100010000001100000**	***No es número (NaN)***

1.2 Excepciones.

El estándar IEEE-754 define cinco tipos de excepciones que pueden ser indicadas mediante un bit de un registro o una bandera de estado. En la Tabla 1.1 están consignadas varias de las excepciones (las utilizadas en las operaciones básicas definidas en este proyecto) como *operación inválida*, *cero e infinito*.

1.2.1 Operación inválida.

Algunas operaciones aritméticas son inválidas, tales como la división por cero y la raíz cuadrada de un número negativo. Existen dos tipos de NaN: QNaN y SNaN. El resultado de una operación inválida puede ser:

QNaN: cuando el resultado de una operación no es un número, como es el caso de la división por cero o la raíz cuadrada de un número negativo.

SNaN: cuando el resultado de una operación es un carácter o si alguno de los operando de entrada es un carácter.

1.2.2 Cero.

Esta excepción se señala siempre que el resultado sea cero sin tener en cuenta la forma en que se produjo. El algoritmo utilizado permite calcular rápidamente el resultado en el evento que este ocurra.

1.2.3 Infinito.

Esta excepción se activa siempre que el resultado de una operación algebraica genere un valor infinito, este se puede producir cuando alguno de los dos operando de entrada (en el caso de la operación de suma) sea infinito.

2 DISEÑO E IMPLEMENTACIÓN DEL COPROCESADOR ASÍNCRONO PARA PUNTO FLOTANTE DE PRECISIÓN SIMPLE (32 BITS).

El coprocesador asíncrono es un circuito digital que realiza operaciones aritméticas de suma, resta, multiplicación y división en punto flotante de dos operando de 32 bits sin utilizar una señal de reloj. Para lograr la sincronización, la ejecución y el procesamiento de datos, el circuito utiliza un protocolo de *handshake*, que puede ser definido desde la especificación de la implementación asíncrona con Balsa System.

El diseño de cada módulo se realizó utilizando la herramienta de descripción de circuitos asíncronos *Balsa System* [1] y se organizó de forma modular con la ayuda de la plataforma de síntesis e implementación de circuitos digitales *Xilinx Ise versión 13.1 [10]* utilizando el lenguaje de descripción de *hardware Verilog [7]*. En la Figura 2.1 se muestra la organización del coprocesador asíncrono con cada uno de sus módulos.

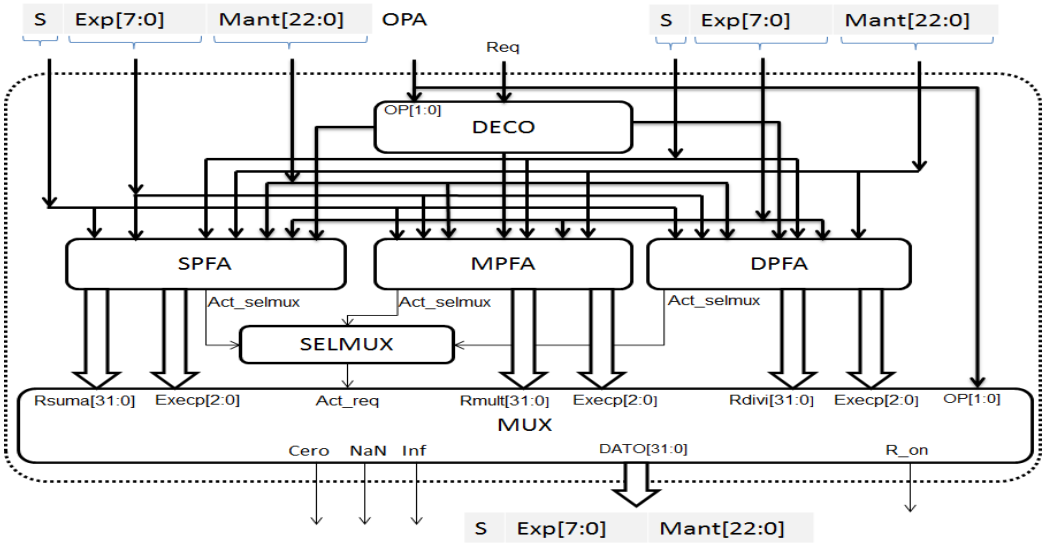

Figura 2.1. Coprocesador asíncrono para punto flotante de 32 bits (Asínchronous Floating Point Unit: AFPU).

El coprocesador asíncrono realiza las operaciones de suma/resta, multiplicación y división debido a que está conformado por cuatro unidades que son:

- **SFPA**: es la unidad de ejecución asíncrona de suma/resta para punto flotante de precisión simple.
- **MPFA**: es la unidad de ejecución asíncrona de la multiplicación para punto flotante de precisión simple.
- **DPFA**: es la unidad de ejecución asíncrona de división para punto flotante de precisión simple.
- **DECO**: es la unidad encargada de seleccionar las unidades de ejecución de las operaciones matemáticas.
- **SELMUX**: es la unidad encargada de activar la unidad MUX cuando una unidad de ejecución de operaciones haya terminado la operación.
- **MUX**: esta unidad se encarga de entregar los resultados de la operación realizada.

A continuación se describe en detalle estas unidades.

2.1 UNIDAD DE SUMA EN PUNTO FLOTANTE ASÍNCRONA (SUMA-PFA: SPFA).

Esta unidad se diseñó siguiendo las especificaciones del estándar para suma/resta en simple precisión y está conformada por cuatro módulos: t*dsuma_ctr1, shr_ctr2, suma_ctr3, nor_ctr4.* En la Figura 2.2 se detalla la organización de la unidad de suma en punto flotante asíncrona con todos sus módulos.

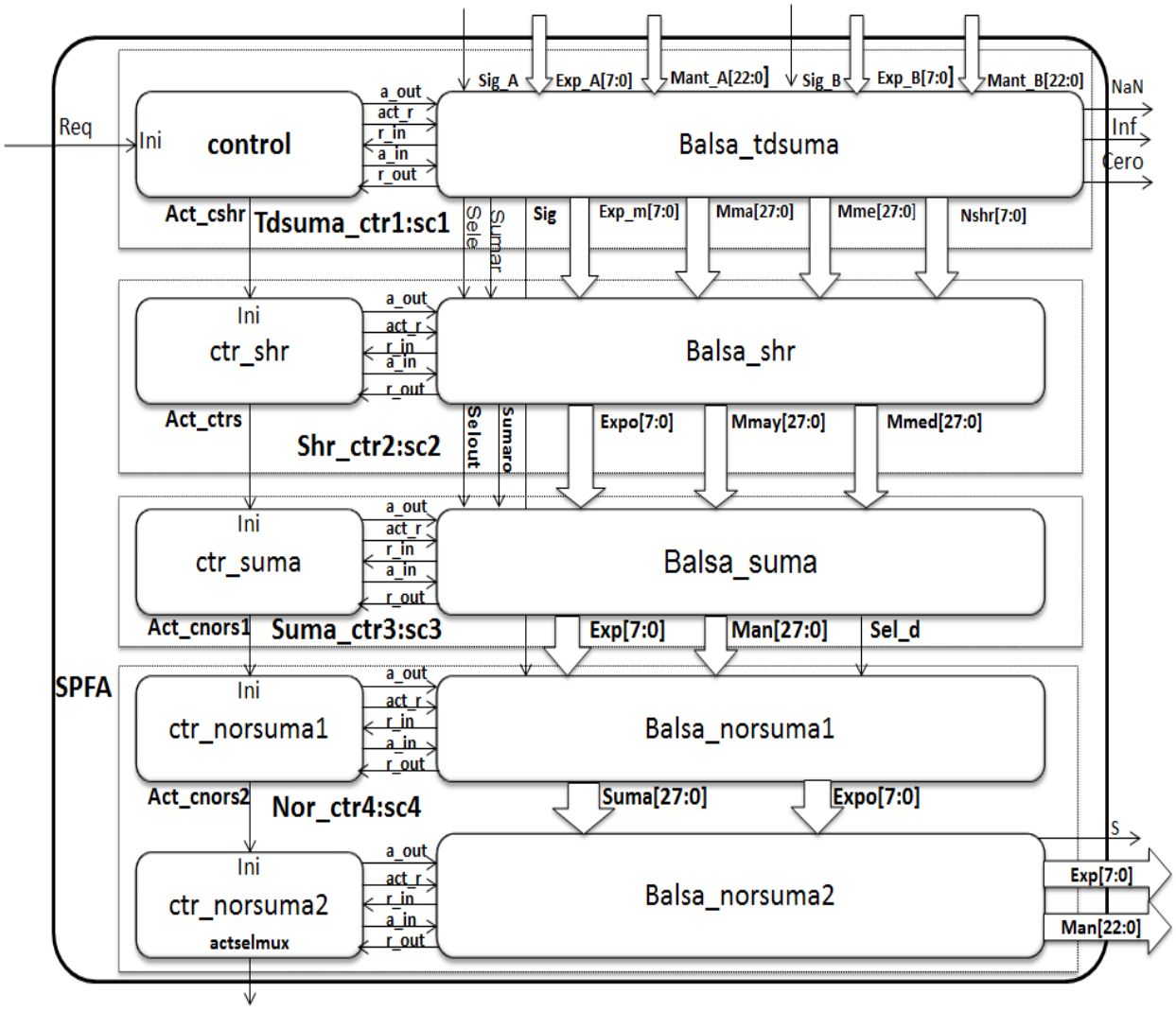

Figura 2.2. Unidad asíncrona de suma/resta para punto flotante (SPFA).

2.1.1 Módulo Tdsuma_ctr1.

Las funciónes de este módulo es la de clasificar los datos de las entradas, calcular el operando mayor y el número de desplazamiento para la mantisa menor. También define si se realiza la suma o la resta y verificar si hay alguna excepción para activar la bandera respectiva (NaN, Inf (infinito), Cero). Las entradas y salidas que conforman este módulo se muestra en la siguienteTabla 2.1 describe cada una de las entradas y salidas del módulo.

Tabla 2.1. Entradas y salidas del módulo tdsuma_ctr1.

ENTRADAS	***TIPO***	***DESCRIPCIÓN***
Exp_A	Bus de 8 bits	Exponente del operando A
Exp_B	Bus de 8 bits	Exponente del operando B
Mant_A	Bus de 23 bits	Mantisa del operando A
Mant_B	Bus de 23 bits	Mantisa del operando B
Req	bit	Señal de inicio
Sig_A	bit	Signo del operando A
Sig_B	bit	Signo del operando B
SALIDAS	***TIPO***	***DESCRIPCIÓN***
Exp_M	Bus de 8 bits	Exponente mayor
Mma	Bus de 28 bits	Mantisa mayor
Mme	Bus de 28 bits	Mantisa menor
CERO	bit	1 = el resultado es cero 0 = resultado diferente de cero
INF	bit	Número infinito
NaN	Bit	Operación invalida
sele	Bit	Signo del operando mayor
Sig	bit	Signo del resultado
Sumar	bit	1= restar operando 0= sumar operando
Nshr	Bus de 8 bits	Número de veces quece debe desplazar la mantisa menor.
Act_cshr	Bit	Activar el controlador del módulo de desplazamiento.

El módulo tdsuma_ctr1 está conformado por dos módulos ***Balsa_tdsuma*** y ***control***

La descripción comportamental de este módulo está resumida en el diagrama de flujo de la Figura 2.3 y el código *Balsa* se incluye en el **anexo 1**.

El comportamiento es el siguiente: cuando se activa la entrada *INI* el módulo verifica si el operando de entrada es un *NaN (no numero),* Inf (infinito*)* o si algún operando es *cero*. Si no existen excepciones se procede a definir el operando mayor de acuerdo con los exponentes, si los exponentes son iguales se verifican las mantisas y si la mantisas son iguales concluye que los operando son iguales, luego se examinan los signos para indicar si el resultado de la operación será cero (signos distintos) o diferente de cero (signos iguales), por último se define el signo del resultado y la operación a realizar (suma/resta).

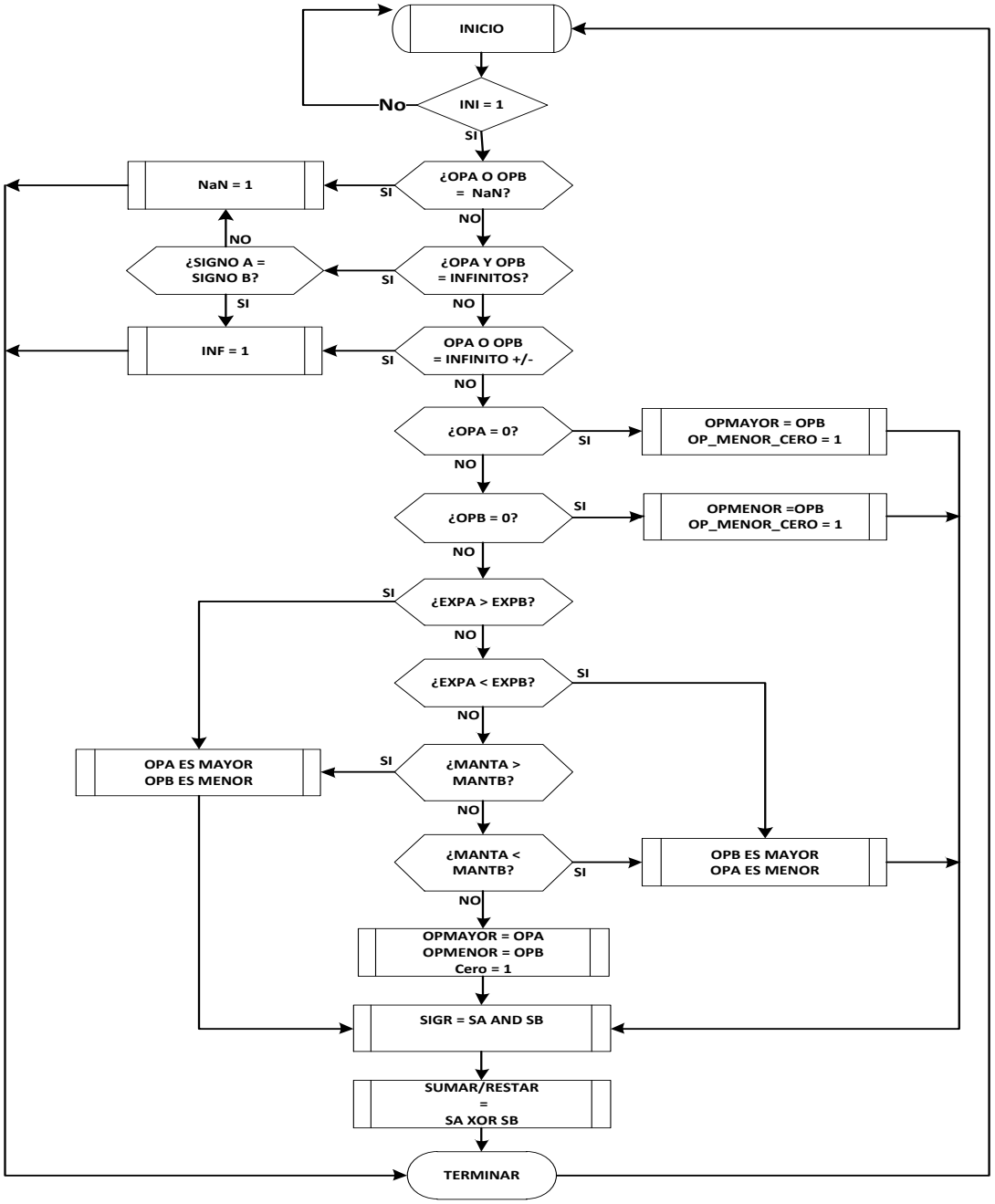

Figura 2.3. Diagrama de flujo del funcionamiento del módulo tdsuma_ctr1.

El diagrama de handshake del módulo *Balsa_tdsuma* que se obtuvo de realizar la descripción del algoritmo con el código Balsa se agrega en el anexo 2 debido a que presenta muchos componentes de *handshake* intrconectados de forma compleja y difícil de interpretar y no aporta claridad.

Módulo *control:* todos los módulos de control se describierón con el lenguaje *Verilog* en *Xilinx Ise 13.1* su función principal es la de establecer el *handshake* con los módulos elaborados en balsa. La Figura 2.4 contiene el diagrama de flujo de este módulo.

Cuando el módulo control recibe un pulso en la entrada de requerimiento envía un pulso de activación al modulo balsa_tdsuma. Luego con la entrada R_ints verifica si balsa_tdsuma realizó las solicitudes para el envio de datos a la entrada. Cuando esto ocurre el modulo control envía pulso de reconocimiento (ack) a Balsa_tdsuma y queda en la espera de que la entrada R_outs tenga todos sus bits activados en uno los cuales indican que balsa_tdsuma lla termino

de realizar su tarea y hay un dato valido en la salida. El modulo control activa la salida A_ctrshr para que el controlador del módulo de desplazamiento inicie su operación. El código del controlador combinacional se incluye en el **anexo 3**.

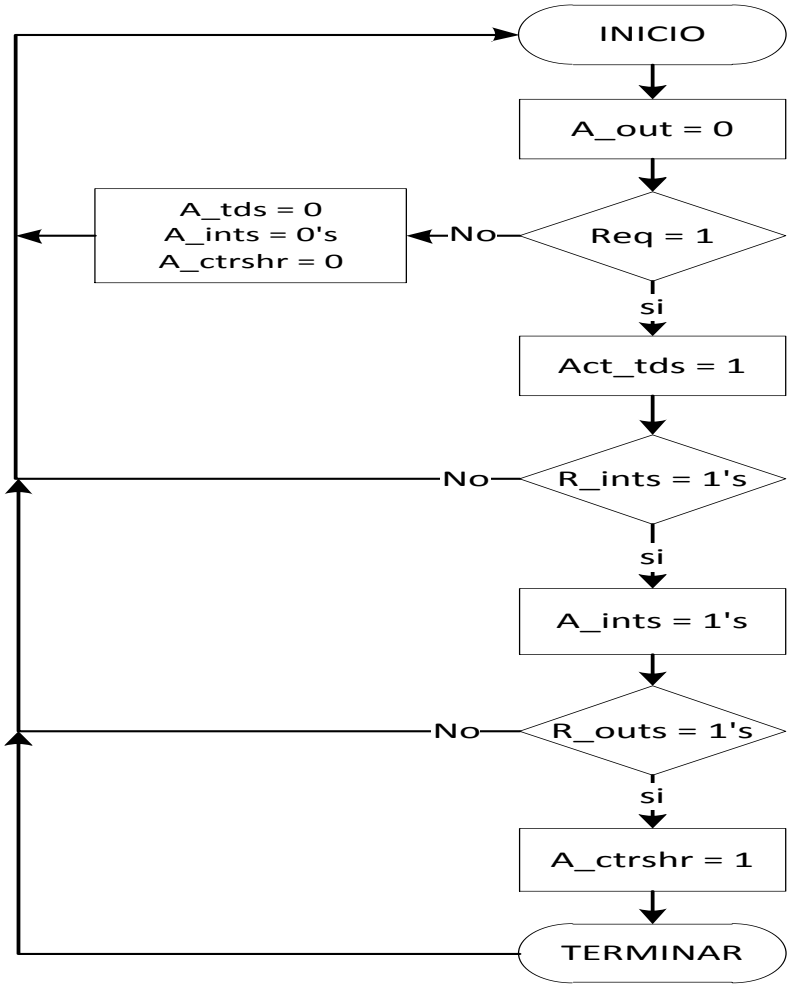

Figura 2.4. Diagrama de flujo del algoritmo para el módulo control.

Síntesis del diseño asíncrono del módulo tdsuma_ctr1: con la herramienta *Balsa* se generó un archivo de conexiones en lenguaje *Verilog* (*Verilog-Netlist*) del módulo Balsa_tdsuma el cual es sintetizado por la herramienta *Xilinx Ise 13.1*.

La Tabla 2.5 contiene los resumenes de los reporte de compilación de todos los módulos para la suma de punto flotante asíncrono sintetizado con la herramienta de Xilinx Ise.

2.1.2 Módulo shr_ctr2.

La función de este módulo es la de desplazar la mantisa menor de acuerdo a la diferencia algebraica de los exponentes de los operando A y B. Las entradas y salidas se describen en la Tabla 2.2.

Tabla 2.2. Entradas y salidas del módulo shr_ctr2.

ENTRADAS	**TIPO**	**DESCRIPCIÓN**
Exp_m	Bus de 8 bits	Exponente mayor
Mma	Bus de 28 bits	Mantisa mayor
Mme	Bus de 28 bits	Mantisa menor
Nshr	Bus de 8 bits	Cantidad para desplazar la mantisa menor
Sele	bit	Indicador de excepciones
Sumar	bit	Suma/Resta
Ini	bit	Indica el comienzo
SALIDAS	***TIPO***	***DESCRIPCIÓN***
Expo	Bus de 8 bits	Exponente mayor
Mmed	Bus de 28 bits	Mantisa menor desplazada.
Mmay	Bus de 28 bits	Mantisa mayor
selout	bit	Indica bandera de excepcion al módulo de suma/resta
act_ctrs	Bit	Activa el controlador de suma
Sumaro	bit	Suma/Resta

Módulo shr_ctr2: está compuesto por dos módulos llamados ***Balsa_shr*** y el ***ctr_shr.***

La descripción comportamental del módulo *Balsa_shr* esta resumida en el diagrama de flujo de la Figura 2.5 y el código realizado se encuentra en el ***anexo 4.***

Si ***sele = 1*** indica que hubo excepciones por lo tanto no se hace desplazamiento de la mantisa menor y todos los datos de la entrada pasan a la salida sin ninguna modificación, si ***sele = 0*** se desplaza la mantisa menor la cantidad de veces que indica la variable ***nshr.*** Cuando esto ocurre se muestran todos los datos en la salida. En este módulo el único dato que se modifica es la mantisa menor si no hay excepciones y si ***nshr*** es mayor que cero.

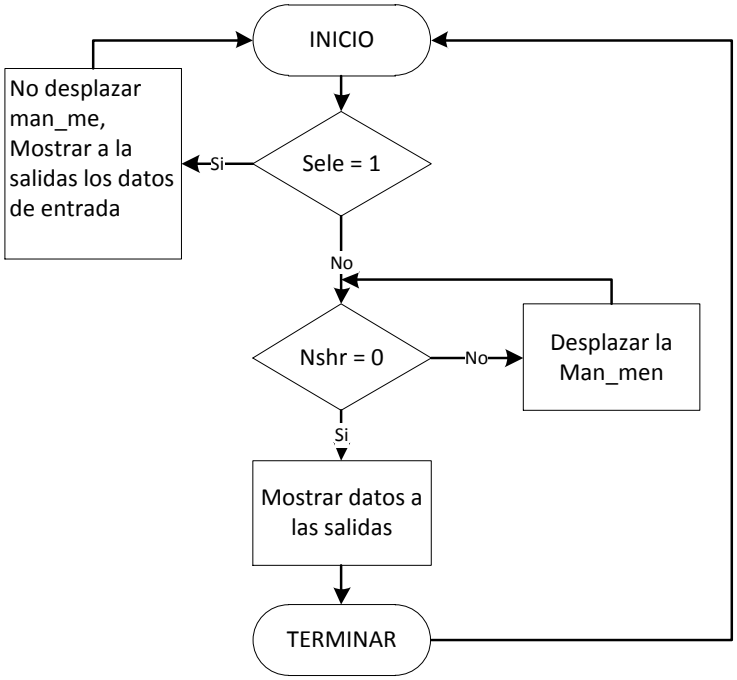

Figura 2.5. Diagrama de flujo del funcionamiento del módulo shr_ctr2.

En el ***anexo 5*** se encuentra el diagrama de handshake del módulo de desplazamiento de mantisa *Balsa_shr.*

Módulo *ctr_shr*: este módulo tiene el mismo algoritmo del módulo *control*, su función principal es la de enviar pulsos para establecer el protocolo de *handshake* con el módulo *Balsa_shr* y lograr que la mantisa menor sea desplazada si es el caso. La Figura 2.4 contiene el diagrama de flujo del algoritmo de control.

Síntesis del diseño asíncrono de shr_ctr2: el archivo verilog generado con *balsa* llamado ***Balsa_shr*** es sintetizado en *Xilinx Ise 13.1*. La entidad obtenida se muestra en la Figura 2.6. Como es característico de los circuitos asíncronos en esta figura se observa las entradas y salidas de datos con sus entradas de reconocimiento *acknowledge (ack)* y salidas de requerimiento *request* (req).

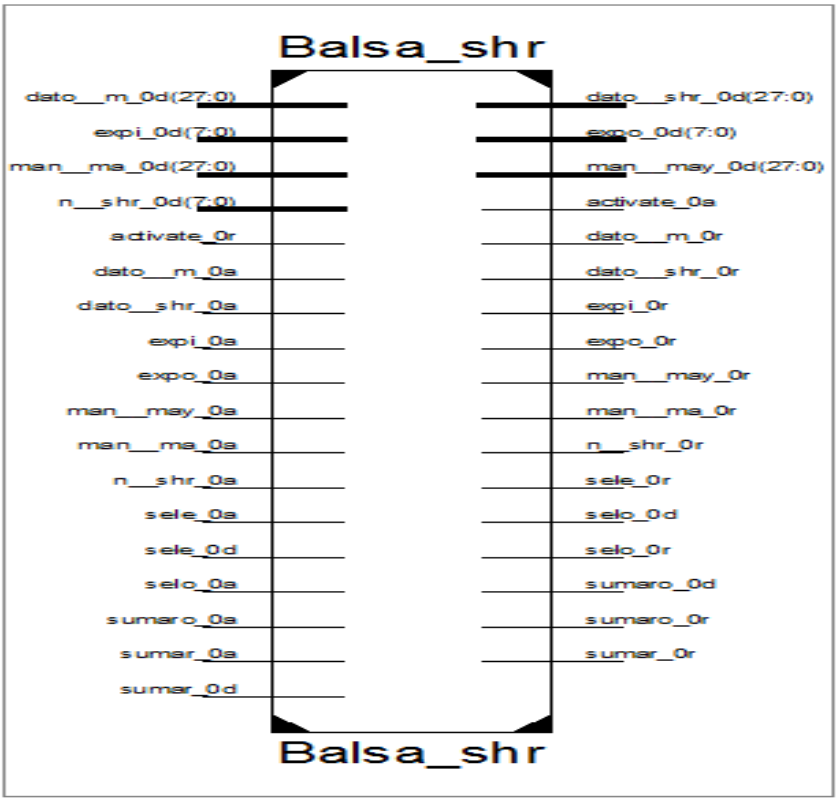

Figura 2.6. Módulo Balsa_shr sintetizado en Xilinx Ise 13.1.

Para que este módulo pueda operar se diseñó un circuito de control llamado *ctr_shr* el cual se encarga de manejar las señales de *req* y *ack* del módulo *Balsa_shr*.

La cantidad de elementos lógicos, registros y *latches* utilizados en la síntesis del módulo *shr_ctr* se encuentra en la tabla 2.5.

2.1.3 Módulo suma_ctr3.

La función de este módulo es la de sumar/restar la mantisa menor con la mayor. En la Tabla 2.3 están consignadas cada una de las entradas y salidas.

Tabla 2.3. Entradas y salidas del módulo suma_ctr3.

ENTRADAS	***TIPO***	***DESCRIPCIÓN***
Expo	Bus de 8 bits	Exponente mayor
Mmay	Bus de 28 bits	Mantisa mayor
Mme	Bus de 28 bits	Mantisa menor
Selout	Bit	Indica bandera de excepciones
Sumaro	Bit	Suma/Resta
Ini	Bit	Indica el comienzo
SALIDAS	***TIPO***	***DESCRIPCIÓN***
Exp	Bus de 8 bits	Exponente mayor
Man	Bus de 28 bits	Mantisa resultado
sel_d	Bit	Indica bandera de excepciones
act_ctrnor1	Bit	Activa el controlador del normalizador

Este módulo esta compuesto por dos módulos llamados *Balsa_suma* y el *ctr_suma.*

El módulo *Balsa_suma* se encarga de realizar la suma/resta de las mantisas. En el diagrama de flujo de la Figura 2.7 se muestra el algoritmo de funcionamiento. En el ***anexo 6*** está el código *balsa* realizado en *Balsa System.*

Si ***sele = 1*** indica que hubo excepciones por lo tanto no se hace la suma de las mantisas y todos los datos de la entrada pasan a la salida sin ninguna modificación para mostrar a la salida la excepción presentada.

Si ***sele = 0*** se verifica si la operación a realizar es suma o resta de mantisa de acuerdo al valor de la entrada sumar.

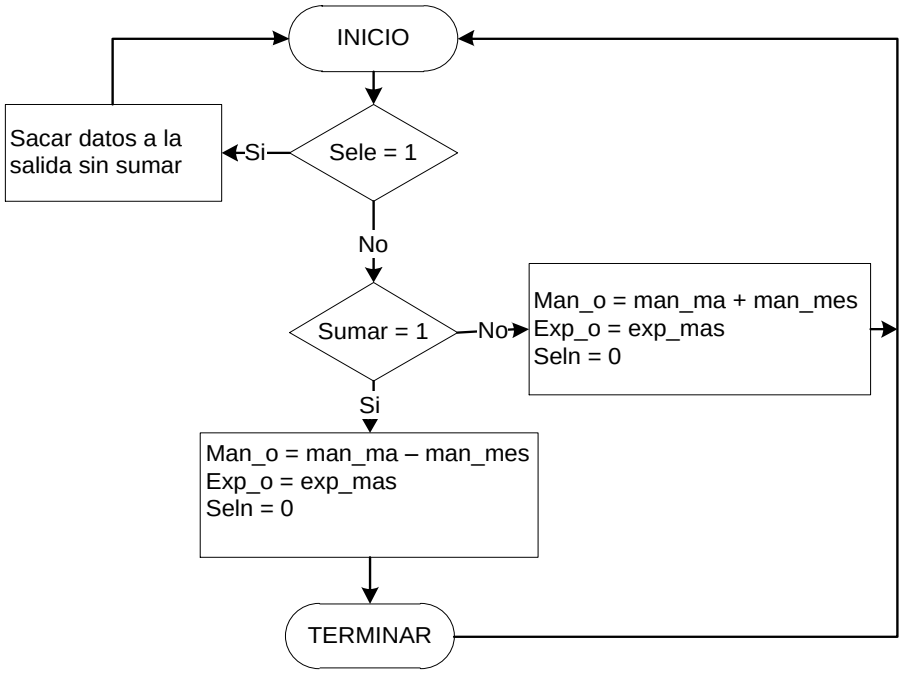

Figura 2.7. Diagrama de flujo del funcionamiento del módulo suma_ctr3.

Módulo *ctr_suma***.** La función de este módulo como la de los otros módulos de control es la de controlar el funcionamiento del módulo *Balsa_suma p*ara que este pueda operar y entregar la suma/resta de las dos mantisas. El algoritmo de funcionamiento esta detallado en la Figura 2.4.

En la Figura 2.8 se muestra el circuito de handshake equivalente al código *Balsa* realizado para el módulo ***suma_ctr3*** que realiza la operación de suma de la mantisa mayor y menor. En la parte derecha están las entradas y salida del circuito, en la parte superior está la entrada de activación del módulo.

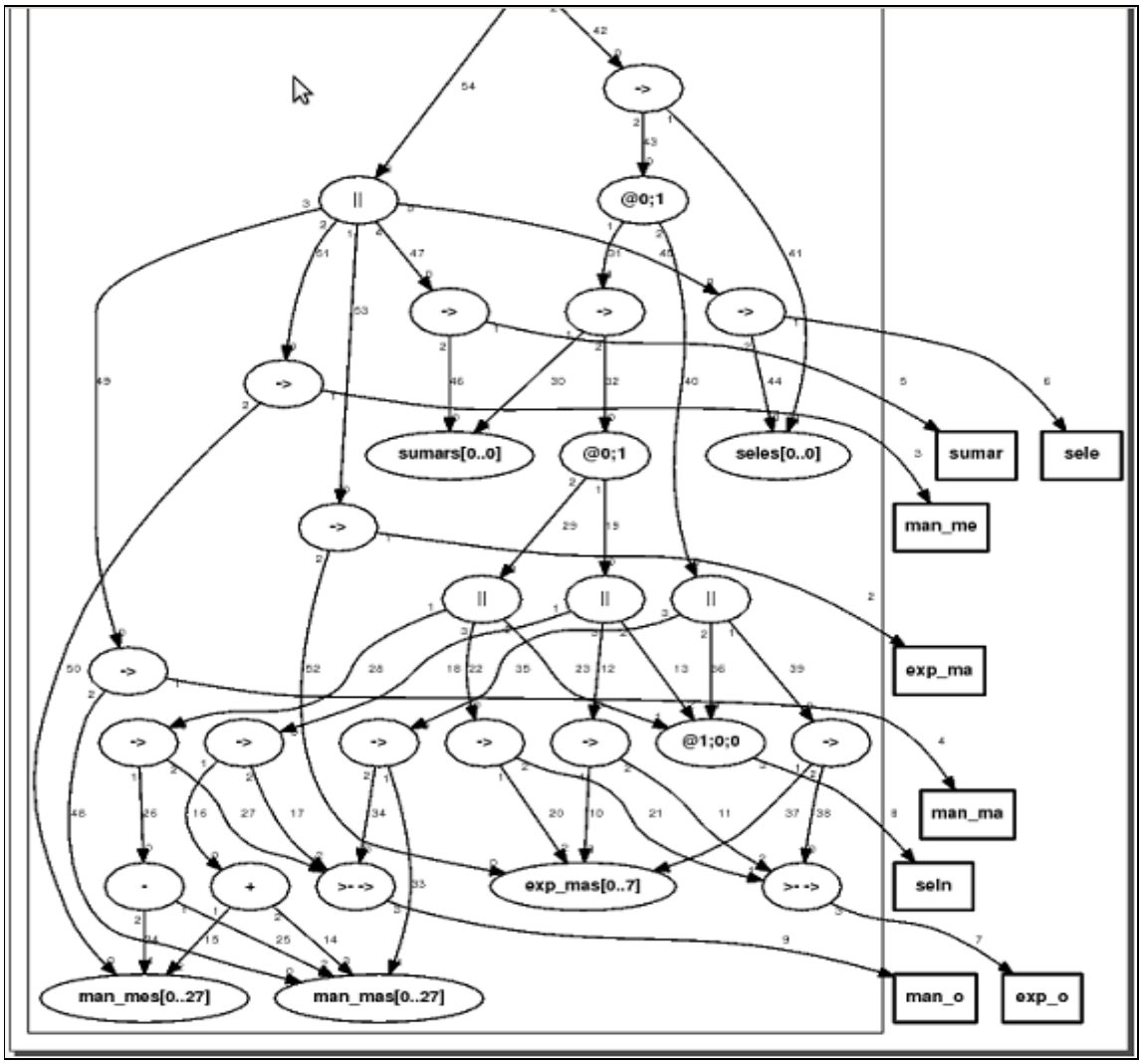

Figura 2.8. Circuito de handshake equivalente del módulo suma_ctr3.

Internamente están los diferentes componentes de *handshake* como el secuenciador de operaciones, los canales que transportan los datos y mensajes entre cada uno de los componentes, las variables, los componentes de paralelismo y los comparadores.

Prueba de la simulación temporal de *Balsa_suma:* los datos que se utilizó para realizar la prueba del circuito son: la mantisa menor es ***man_me_data(27:0) = 7214203***, la mantisa mayor es ***man_ma_data[27:0] = 4766826*****,** el número de exponente ***exp_ma_data[7:0]= 130*****,** la entrada que indica si hubo excepciones es ***sel_data = 0*****,** y la entrada de suma o resta es ***sumar_data = 1*** que indica que las mantisas se van a restar como se puede ver en la Figura 2.9.

La salida que tiene la respuesta más rápida es ***seln_data = 0*** la cual indica que no hubo excepciones y la resta se realizó en **t = 5.5us**, luego le sigue ***expo_data[7:0] = 130*****,** la mantisa resultante ***man_o_data[27:0] = 265961079*****,** con ***t = 6.9us.***

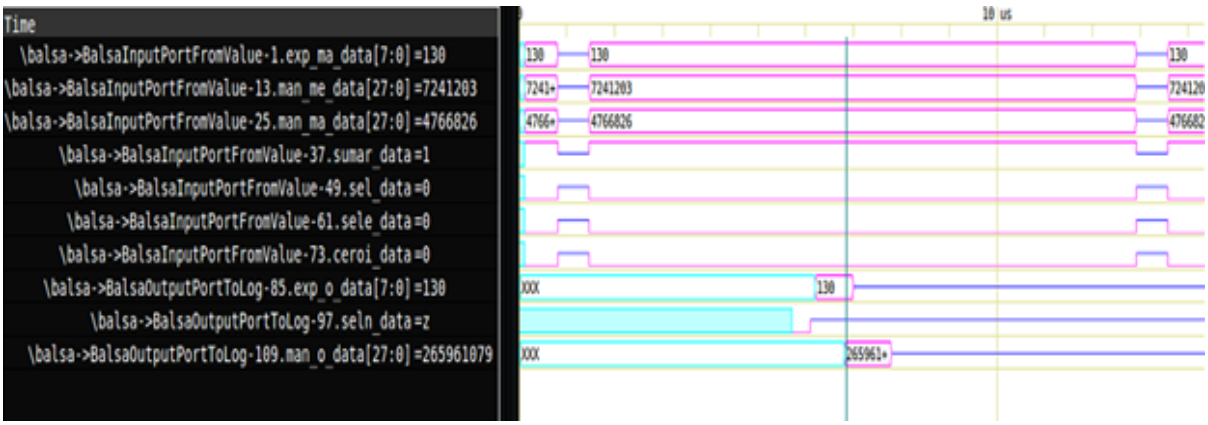

Figura 2.9. Simulación temporal del circuito asíncrono Balsa _suma.

Síntesis del diseño asíncrono de suma_ctr3. El archivo *Balsa* del módulo *Balsa_suma* se sintetizado en *Xilinx Ise 13.1*.

En la Figura 2.10 se visualiza el circuito sintetizado en X*ilinx Ise* con las entradas y salidas de datos con sus correspondientes señales de requerimiento y *acknowledge.*

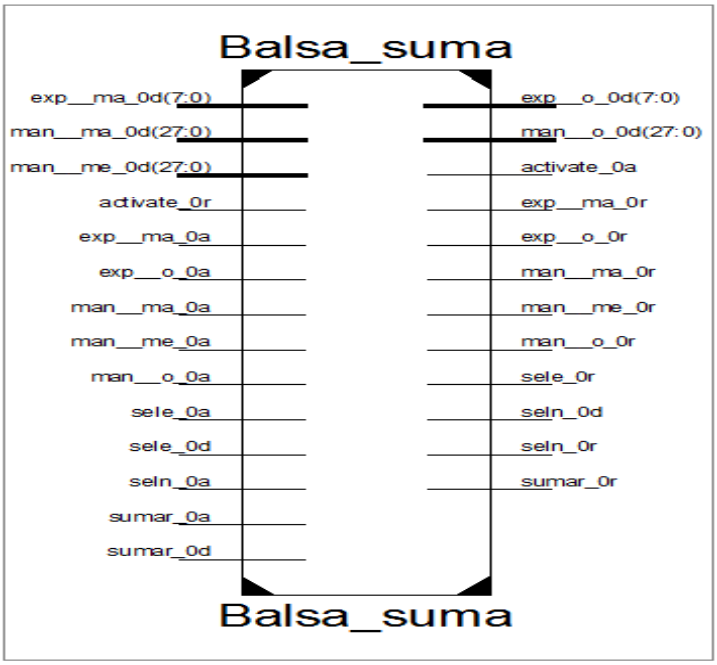

Figura 2.10. Módulo Balsa_suma sintetizado en Xilinx Ise 13.1.

El módulo *ctr_suma* es el encargado de manejar las señales de *req* y *ack* del módulo *Balsa_suma*, como se puede ver en la organización de las conexiones del módulo *suma_ctr3* de la Figura 2.11.

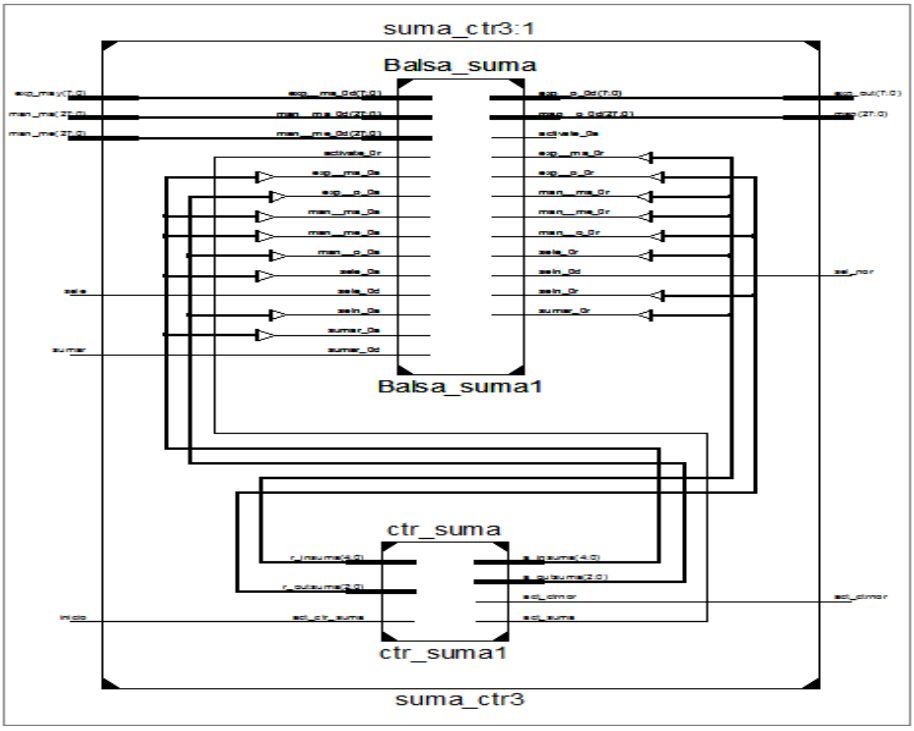

Figura 2.11. Conexión del módulo Balsa_suma con el módulo ctr_suma.

En la tabla 2.5 está consignada la cantidad de elementos lógicos, registros y *latches* utilizados en la síntesis***.***

Verificación del circuito asíncrono *suma_ctr3* en Xilinx Ise: en la Figura 2.12. Está la simulación realizada con los datos que se utilizó para probar el comportamiento del módulo suma_ctr3, los cuales son los mismos que se utilizó para probar el circuito en *System Balsa.*

Mantisa menor ***man_me[27:0] = 00000000011100011111110001010.***

Mantisa mayor ***man_ma[27:0] = 00000110111000010100011111011.***

El exponente ***exp_may[7:0] = 10000010, Inicio = 1, Sele = 0, sumar = 1.***

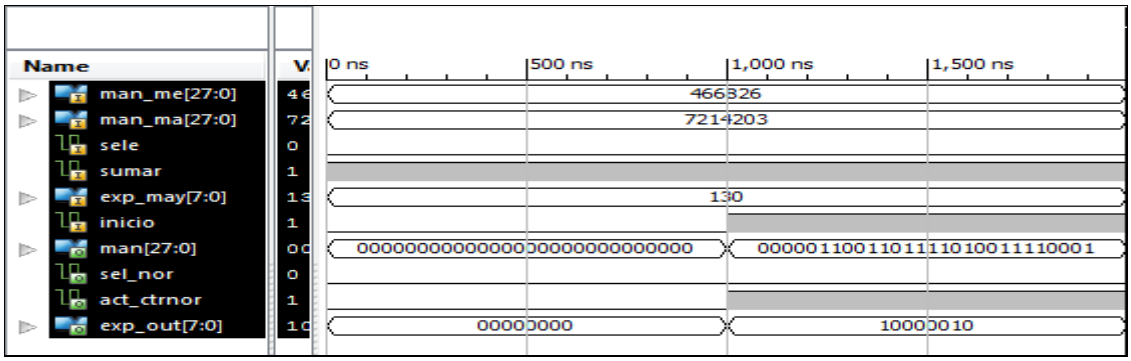

Figura 2.12. Prueba con el simulador isim de xise para el Circuito Asíncrono sc3.

Los resultado obtenidos son los siguientes: la mantisa resultado es ***Man[27:0]= 00000110011011110100111110001*****,** el indicador de excepciones ***Sel_nor = 0*****,** el cual indica que no se ha presentado excepciones. La salida de activación del controlador de normalización es ***act_ctrnor = 1,*** el exponente no se modifica hasta llegar a la etapa de normalización ***expo[7:0] = 10000010*****.**

2.1.4 Módulo nor_ctr4.

La función de este módulo es de normalizar la mantisa que entrega el bloque sumador para cumplir con el estándar IEEE 754. En la Tabla 2.4 se describe cada una de las entradas y salidas.

Tabla 2.4. Entradas y salidas del módulo nor_ctr4.

ENTRADAS	*TIPO*	*DESCRIPCIÓN*
Exp	Bus de 8 bits	Exponente
Man	Bus de 28 bits	Mantisa sin normalizar
Sel_d	bit	Indica bandera de excepciones
sig	bit	Suma/Resta
Ini	bit	Indica el comienzo
SALIDAS	*TIPO*	*DESCRIPCIÓN*
Suma	Bus de 28 bits	Mantisa resultado
Act_cnors2	Bit	Activa el controlador del *ctr_norsuma2*
Expo		Resultado del exponente

El módulo nor_ctr4 está compuesto por los módulos llamados ***Balsa_norsuma1, Balsa_norsuma2, ctr_norsuma1 y crt_norsuma2***.

Módulos *Balsa_norsuma1 y Balsa_norsuma2:* realiza la función de normalización y la descripción comportamental esta resumida en el diagrama de flujo de la Figura 2.13. En los ***anexos 4 y 5*** se entrega el código en el lenguaje *Balsa* tal como está en la Figura 2.13.

Si ***sele = 1*** indica que hubo excepciones por lo tanto no se hace normalización de la mantisa resultante y se prepara los datos para mostrarlos en formato IEEE 754.

Si ***sele = 0*** se verifica si el bit de desbordamiento de la mantisa resultante es 1. Si es 1 se coloca el bit de signo en el formato IEEE 754. Al exponente se le suma 1 y se toma desde bit 25 hasta el bit 3 de la mantisa resultante para colocar la mantisa en el formato IEEE 754.

Si el bit 27 de la mantisa resultante es cero se procede a verificar el bit 26. Si es 1 el exponente no cambia y se colocan los 23 bits de la mantisa resultante desde el bit 3 hasta el bit 25. Si el bit 26 es cero se calcula la cantidad de ceros que hay desde este bit hacia la derecha y se desplaza la mantisa esa misma cantidad y se resta la misma cantidad al exponente, luego se colocan los datos en el formato IEEE 754.

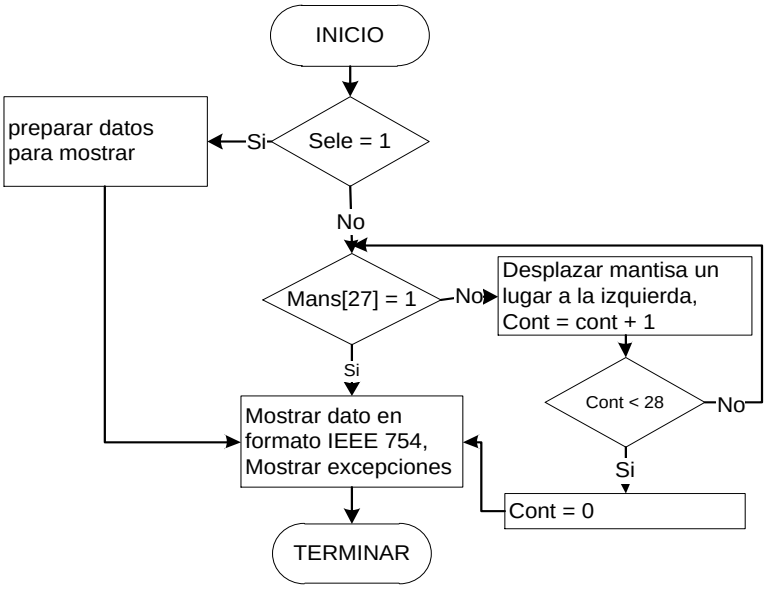

Figura 2.13. Diagrama de flujo del funcionamiento del módulo nor_ctr4.

Los módulos *ctr_norsuma1 y ctr_norsuma2* tienen la función de controlar los Módulos *Balsa_norsuma1* y *Balsa_norsuma2* a través de sus entradas y salidas asociadas para establecer el *handshake* (*ack, req*).

Este módulo de normalización se dividió en dos para poder calcular el exponente resultante antes de desplazar la mantisa a normalizar. En el ***anexo 6*** se muestra el diagrama de circuito de *handshake* del módulo *Balsa_norsuma1* el cual es extenso y no es muy legible. El diagrama de handshake del módulo *Balsa_norsuma2* se visualiza a continuación en la Figura 2.14 donde la función básica de este módulo es la de concatenar los datos de las entradas para mostrarlos a la salida en el formato IEEE 754.

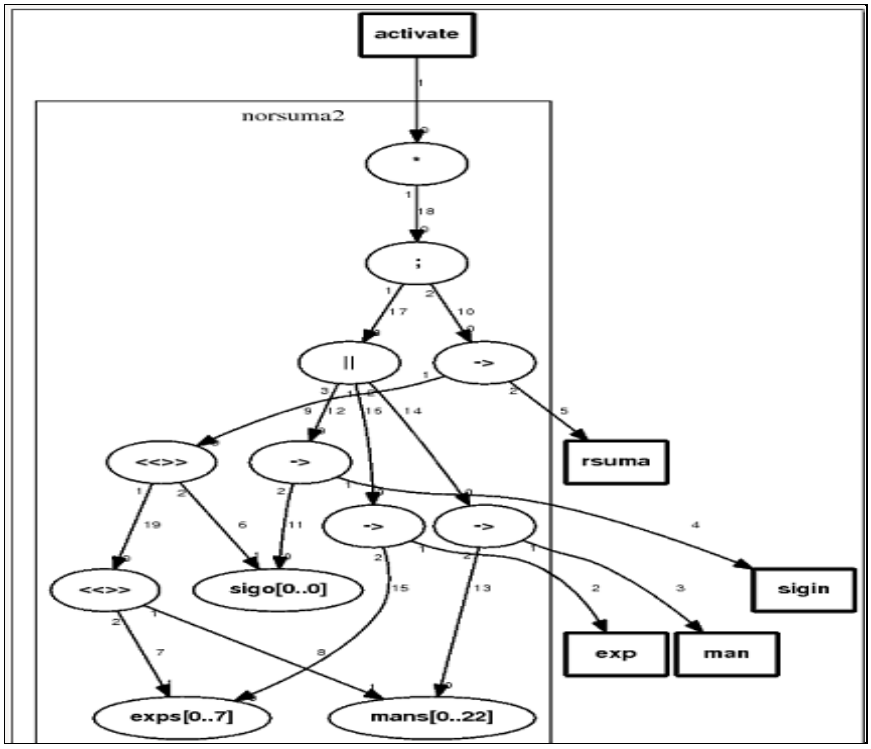

Figura 2.14. Diagrama de handshake del módulo Balsa_norsuma2.

El símbolo (*) se encarga de repetir las operaciones del circuito al realizar el handshake con el operador de secuencia (;). el operador de sucuencia establece el handshake con el componente de paralelismo (||) atraves del canal 1; este componente establece el hanshake con los canales para transmitir los datos de la entrada a las variables mans[0..22], exps[0..7] y sigo[0..0]. En la siguiente secuencia se concatenan los datos y se envían a la salida rsuma.

Síntesis del diseño asíncrono de nor_ctr4: los archivos *Balsa_norsuma1* y *Balsa_norsuma2* realizado en *Balsa* se sintetizaron en la herramienta *Xilinx Ise 13.1*. Los módulos de control *ctr_nor1* y ctr_nor2 se realizaron en *Xilinx Ise 13.1* como los otros controladores descritos anteriormente.

En la Figura 2.15. Se muestra la organización interna del módulo *nor_ctr4* que contiene todos los módulos diseñados para normalizar la mantisa y ajustar el exponente.

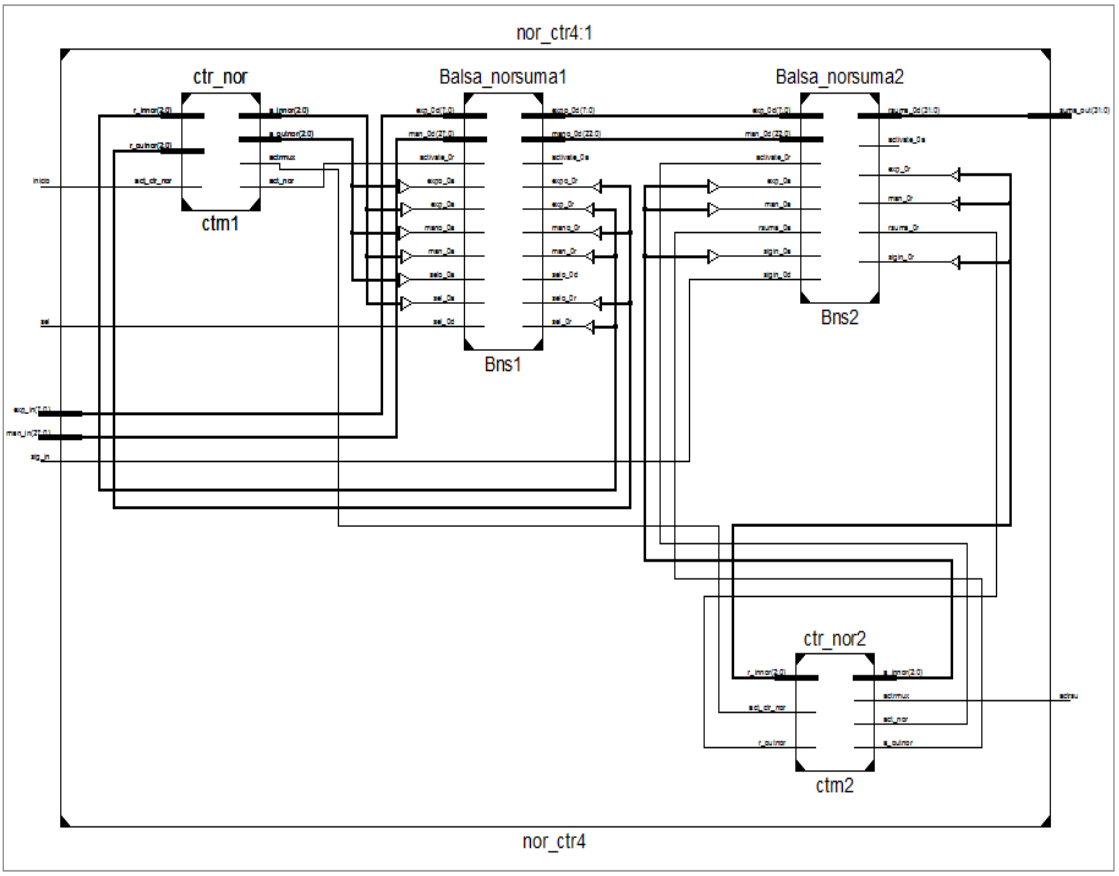

Figura 2.15. Módulo nor_ctr4 sintetizado en Xilinx Ise 13.1

En la Tabla 2.5 está consignada la cantidad de elementos lógicos, registros y *latches* utilizados en la síntesis del módulo *Balsa_shr* y el *ctr_shr.*

Verificación del circuito asíncrono nor_ctr4 en Xilinx Ise 13.1: para la simulación del circuito asíncrono nor_ctr4 se empleó los mismos datos que se utilizaron para probar el módulo en balsa. En la Figura 2.16 se visualizan los datos de la entrada y los resultados.

Mantisa sin normalizar ***man_in[27:0] = 0000010010000101111000110101O.***

El exponente ***exp_in[7:0] = 10000010, Inicio = 1, Sel = 0, sig_in = 1.***

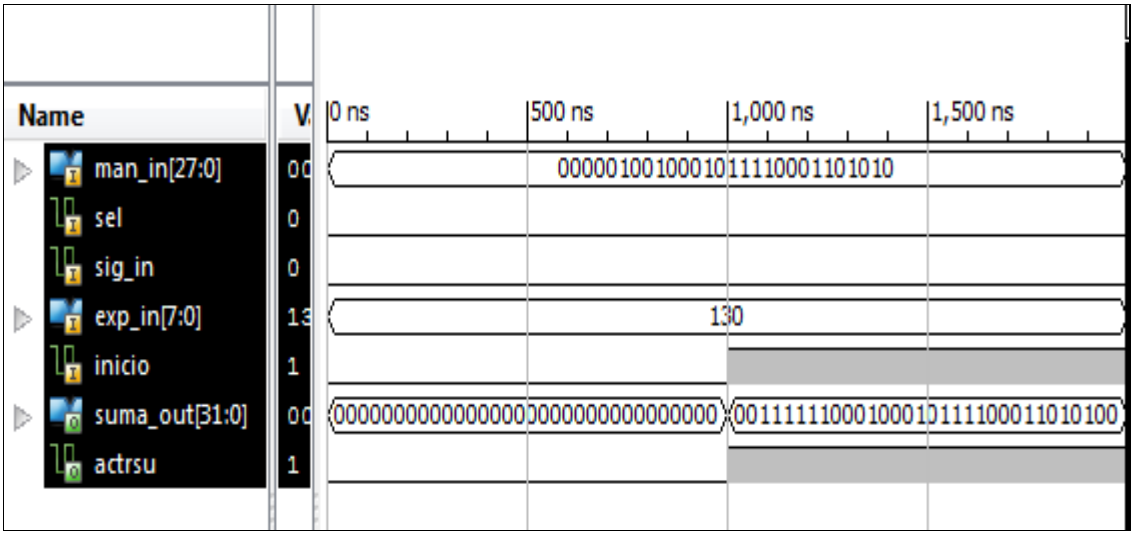

Figura 2.16. Prueba con el simulador isim de xise para el Circuito Asíncrono sc4.

El resultado obtenido es un dato en formato IEEE 754 de 32 bits donde el bit 31 es el bit de ***signo = 1.*** *Este bit* indica que el dato es negativo. El exponente *es:* ***01111110*** el cual se decremento 4 veces para normalizar la mantisa, la mantisa normalizada es ***00100010111100011010100*****,** el bit implícito no se coloca porque para las mantisas normalizadas siempre es uno.

2.1.5 Módulo SPFA (sumapf_asyn).

Este módulo asíncrono contiene a todos los módulos para realizar la suma/resta como se mira en el diagrama de bloques de la Figura 2.2. La organización interna de todos sus componentes sintetizados en *xilinx ise* se visualiza en la Figura 2.17.

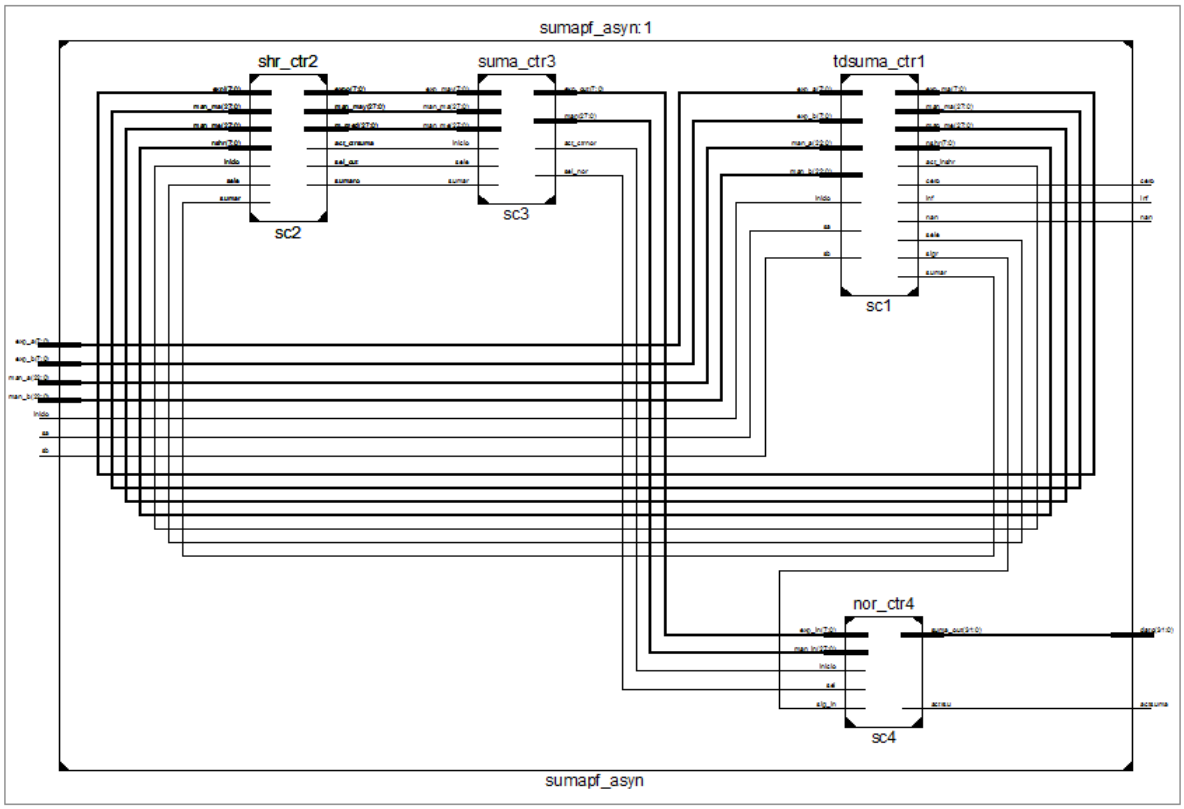

Figura 2.17. Organización interna del módulo SPFA.

El resumen de la síntesis del módulo SPFA de punto flotante asíncrono se muestra en la tabla 2.5.

Verificación del circuito asíncrono *SPFA:* *p*ara probar el sumador asíncrono se utilizó los siguientes datos: el operando A= 12,546 y el operando B = -0,465. Estos datos son pasados a binario y luego al formato IEEE 754 para cada uno de los operando de entrada.

Los exponentes A y B son*:* ***Exp_a[7:0] =10000010, exp_b[7:0] = 01111101.***

Las mantisas A y B son: ***man_a[22:0] = 10010001011110001101010.***

Man_b[22:0] = 11011100001010001111011.

Los signo A y B son**: *sa = 0, sb = 1*,** como se muestra en la Figura 2.18.

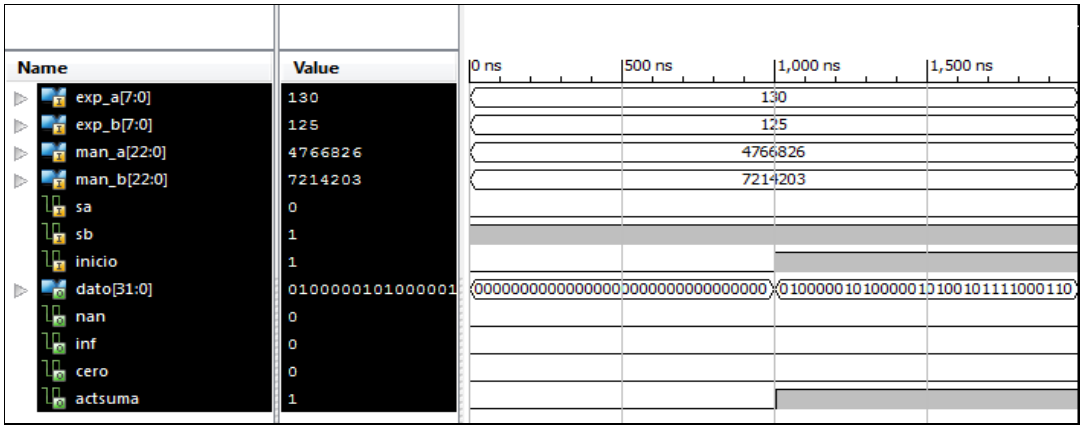

Figura 2.18. Resultados de la simulación del módulo asíncrono sumapf_asyn.

El resultado obtenido es ***dato[31:0] = 01000001010000010100101111000110*** el cual está en formato IEEE 754 de 32 bits como se muestra en la Figura 2.18. El signo del resultado es cero (positivo). El exponente resultado es el exponente mayor de los exponentes de las entradas ***10000010***. La mantisa ***10000010100101111000110*** es el resultado de la suma de las dos mantisas normalizadas.

La tabla 2.5 contiene los resumenes de los reporte de compilación de todos los módulos para la suma de punto flotante asíncrono sintetizado con la herramienta de Xilinx Ise.

Tabla 2.5. Resumen de la síntesis del los módulos de suma.

Unidad	*No. De elementos lógicos*	*No. De registros*	*No. De latches (IOB)*	*No. De flip flop (IOB)*
Tdsuma_ctr1	1347	1	0	64
Shr_ctr2	257	0	0	66
Suma_ctr3	1119	0	0	69
Nor_ctr4	1347	1	0	64
sumapf_asyn	1736	0	1	74

2.2 UNIDAD DE MULTIPLICACIÓN EN PUNTO FLOTANTE ASÍNCRONA (MPFA).

Esta unidad fue diseñada utilizando *[1]* para la descripción asíncrona como son las unidades ***Balsa_tdproducto, Balsa_producto*** **y** ***Balsa_norprod***. El lenguaje *Verilog* se utilizó para instanciar los módulos ***tdprod_ctd***, ***producto_ctp***, ***npctr***, en el entorno de desarrollo *Xilinx Ise 13.1*. Esta unidad se diseñó siguiendo las especificaciones del estándar IEEE 754 [3] para multiplicación en precisión simple. En la Figura 2.19 se detalla la organización con todos sus módulos.

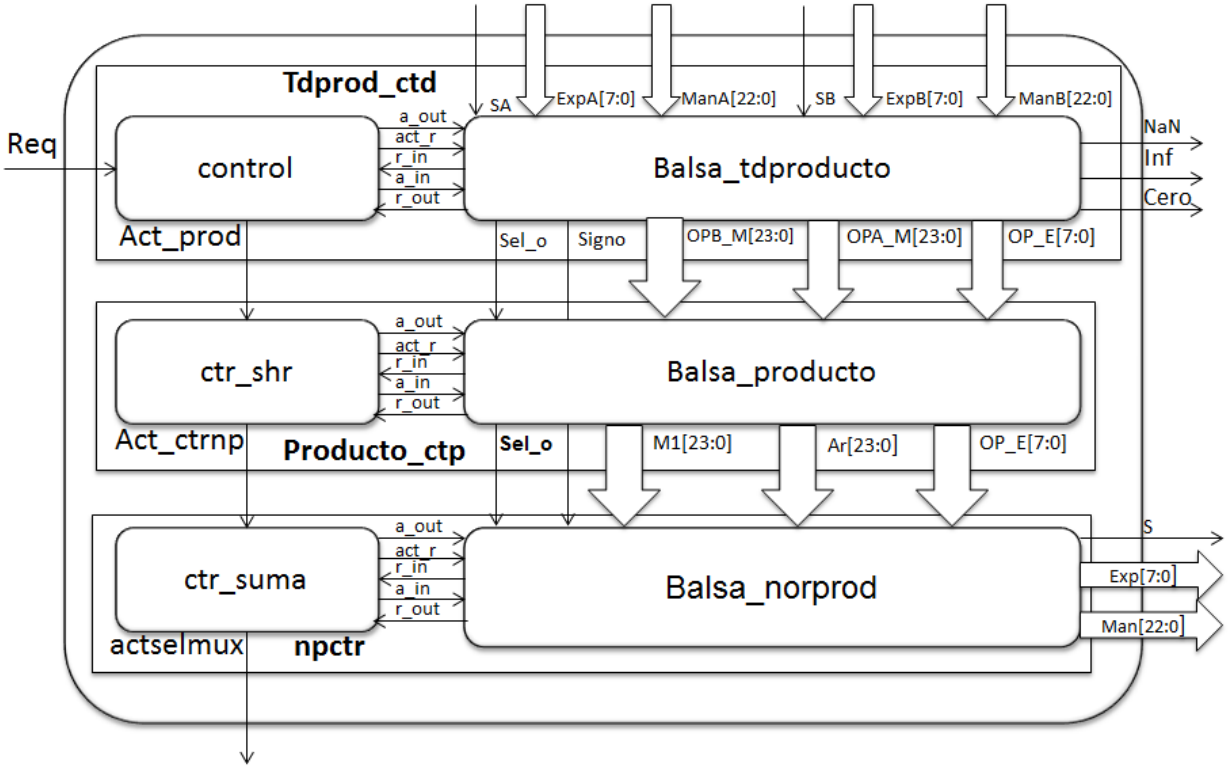

Figura 2.19. Unidad asíncrona de multiplicación para punto flotante (MPFA).

Esta unidad realiza la multiplicación de dos números en punto flotante de simple precisión y está conformada por tres módulos principales

- Módulo *tdprod_ctd.*
- Módulos *producto_ctp.*
- Módulo *ncp1.*

2.2.1 Módulo tdprod_ctd.

La función de este módulo es de seleccionar el tipo de datos a la entrada y preguntar si hay alguna excepción para activar la bandera respectiva (NaN, Inf, Cero) y determinar el resultado parcial del exponente. También se encarga de agregarle el bit implícito a las mantisas para prepararlas para la multiplicación. En la Tabla 2.6 se describe cada una de las entradas y salidas del módulo.

Tabla 2.6. Entradas y salidas del módulo tdprod_ctd.

ENTRADAS	***TIPO***	***DESCRIPCIÓN***
ExpA	Bus de 8 bits	Exponente del operando A
ExpB	Bus de 8 bits	Exponente del operando B
ManA	Bus de 23 bits	Mantisa del operando A
ManB	Bus de 23 bits	Mantisa del operando B
INICIO	bit	Señal de inicio (Req)
SA	bit	Signo del operando A
SB	bit	Signo del operando B
SALIDAS	***TIPO***	***DESCRIPCIÓN***
OP_E	Bus de 8 bits	Resultado del Exponente
OPA_M	Bus de 23 bits	Mantisa A
OPB_M	Bus de 23 bits	Mantisa B
CERO	bit	1 = el resultado es cero 0 = resultado diferente de cero
INF	bit	Número infinito
NaN	Bit	Operación invalida
SEL_O	Bit	Indica si hubo excepciones
,ACT_PROD	Bit	Activa el controlador del multiplicador
SIGNO	bit	Signo del resultado

El módulo *tdprod_ctd* esta conformado por los módulos *Balsa_tdproducto* y el controlador *ctdprod*.

La descripción comportamental del módulo *Balsa_tdproducto* se encuentra en el diagrama de flujo de la Figura 2.20. Debido a que el código y el diagrama de circuito son extensos se agregó el código *Balsa* en el ***anexo 7*** y el diagrama de circuito en el ***anexo 8.***

El algoritmo se comporta de la siguiente manera: cuando se activa la entrada ***INICIO*** el módulo verifica las excepciones a la entrada como son la de operación invalida (*NaN*), Infinito **(***Inf*) y operando cero (*cero*), si no hay excepciones calcula el exponente resultante sumando los exponentes de entrada y le resta el exceso 127. El exponente de todas forma queda formateado con el exceso debido a que los dos exponentes tienen el exceso 127, Luego se normaliza las mantisas y se entregan todos los resultados incluyendo las excepciones con valor cero. El signo se determina usando la operación lógica ***AND***. Luego se termina la operación.

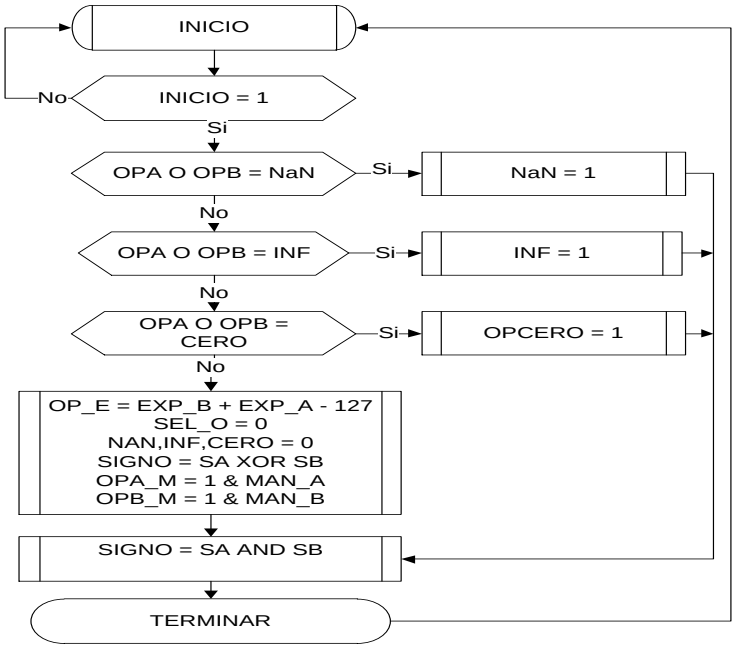

Figura 2.20. Algoritmo de funcionamiento del módulo Balsa_tdproducto.

Módulo de *control ctdprod:* este módulo se diseñado con el lenguaje *Verilog* en *Xilinx Ise 13.1*. La función principal es la de establecer el *handshake* con el módulo *Balsa_tdproducto* para que este pueda tomar los datos de la entrada y entregarlo al módulo de multiplicación. La Figura 2.4 contiene el diagrama de flujo del módulo controlador **ctdprod**.

Síntesis del diseño asíncrono del módulo *tdprod_ctd:* el módulo *Balsa_tdproducto* está conectado al controlador *ctdprod* que tiene la misma función del controlador de la Figura 2.4. El cual a través de las salidas de requerimiento y las entradas de reconocimiento interactúa con *Balsa_tdproducto* para que pueda tomar los datos de la entrada, procesarlos y luego mostrarlos en la salida.

Esta conexión también se realizó para conformar un solo módulo que permite ver las entradas y salidas como los circuitos convencionales, este módulo se observa en la Figura 2.21

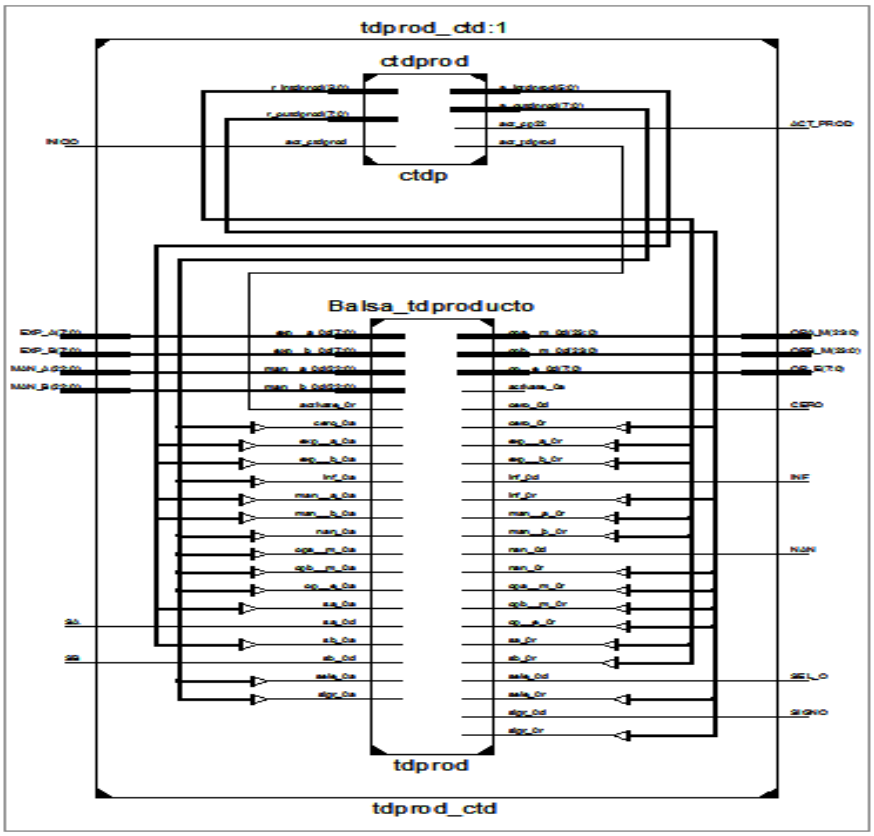

Figura 2.21. Organización interna del controlador del módulo tdprod_ctd.

En la **Tabla 2.13** está consignada la cantidad de elementos lógicos y de registros utilizados en la síntesis del módulo *tdprod_ctd.*

Verificación del circuito asíncrono *tdprod_ctd* en Xilinx Ise: la gráfica de la Figura 2.22 está los datos de entrada para verificar el circuito asíncrono *tdprod_ctd.* Se utilizó el operando A. con A= 12,546 y el operando B. con B = -0,465.

El dato equivalente en el formato IEEE 754 queda de la siguiente manera: ***Sb = 1, exp_b = 01111101, man_b = 11011100001010001111011, exp_a = 10000010, man_a = 10010001011110001101010, sa = 0.*** Como se muestra en la parte baja de la Figura 2.22.

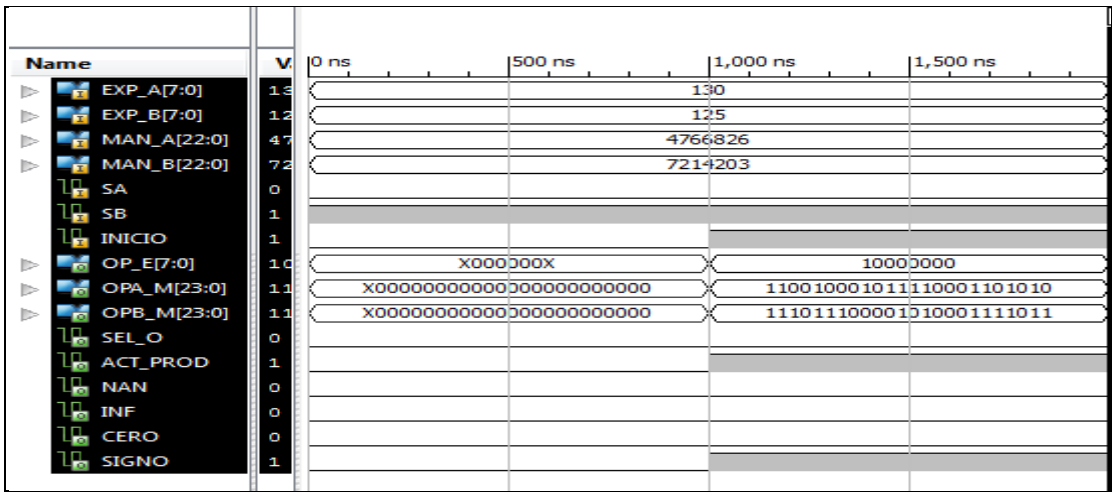

Figura 2.22. Prueba con el simulador isim para el Circuito Asíncrono tdprod_ctd.

Los resultados obtenidos son: ***OPA_M[23:0] = 11011100001010001111011, OPB_M[23:0] = 11001000101111000110101O, OP_E[7:0] = 10000000, SEL_O = 0, CERO = 0, SIGNO = 1, NaN = 0, INF = 0, ACT_PROD = 1.***

A las salidas ***OPA_M[23:0]*** y ***OPB_M[23:0]*** se les agregó el bit implícito a la izquierda para normalizar las mantisas. Esto ocasiona que el dato de las mantisas tenga un valor mayor. El dato del exponente ***OP_E[7:0]*** es el resultado de la suma de los exponentes de entada. Las excepciones ***NaN = 0, INF = 0, CERO = 0*** no se activaron. El signo como era de esperarse resulto negativo.

2.2.2 Módulo producto_ctp.

La función de este módulo es la de multiplicar las mantisas normalizadas entregadas por el módulo *tdprod_ctd*. En la Tabla 2.7 se describe cada una de las entradas y salidas.

Tabla 2.7. Entradas y salidas del módulo producto_ctp.

ENTRADAS	*TIPO*	*DESCRIPCIÓN*
MAI	Bus de 24 bits	Mantisa A normalizada
MBI	Bus de 24 bits	Mantisa B normalizada
OP_E	Bus de 8 bits	Exponente
Sele	Bit	Indica bandera de excepciones
Act_prod	Bit	Indica el inicio
SALIDAS	*TIPO*	*DESCRIPCIÓN*
Arn	Bus de 24 bits	Acumulado de la multiplicación
M1n	Bus de 24 bits	multiplicador
OP_E	Bus de 8 bits	
Selon	Bit	Indica bandera de excepciones
Signo	Bit	Bit de signo
Actcnor	Bit	Activa el controlador de normalización

El módulo *producto_ctp* está compuesto internamente por dos módulos llamados *Balsa_producto* y el *ctr_prod.*

El módulo *Balsa_producto* es un macro módulo que representa 24 módulos realizados con la herramienta B*alsa System* para realizar la multiplicación. La descripción comportamental se realizó utilizando el algoritmo de sumas y desplazamiento para multiplicación binaria ya que el *System Balsa* no sintetiza el multiplicador de forma directa.

Los módulos de la multiplicación en *Balsa* se les dieron los nombres de ***Balsa_p1***, ***Balsa_p2****. Con Balsa_p1* se realiza la primera secuencia de la multiplicación y con *Balsa_p2* se realiza las 23 siguientes secuencias*.* El código Balsa del módulo ***Balsa_p1*** se muestra en el **anexo 9** y el del módulo ***Balsa_p2*** en el ***anexo 10.***

El módulo *ctr_prod* **e**s un macro módulo de control que contiene 24 controles. Cada uno para un módulo *balsa_pn* diferente. Este controlador cumple la misma función de todos los controladores antes descritos y se muestra en la Figura 2.4.

En la Tabla 2.9 está consignada la cantidad de elementos lógicos, registros y latches utilizados en la síntesis del módulo *producto_ctp.*

Verificación del circuito asíncrono *producto_ctp* en Xilinx Ise 13.1: para la simulación del circuito asíncrono *producto_ctp* se utilizó los siguientes datos los cuales se pueden ver en la Figura 2.23.

Mantisa A ***MAI[23:0] = 11001000101111000110101O.***

Mantisa B ***MBI[23:0] = 11101110000101000111101l.***

La bandera de excepciones ***sele = 0.***

La entrada de inicialización ***Inicio = 1.***

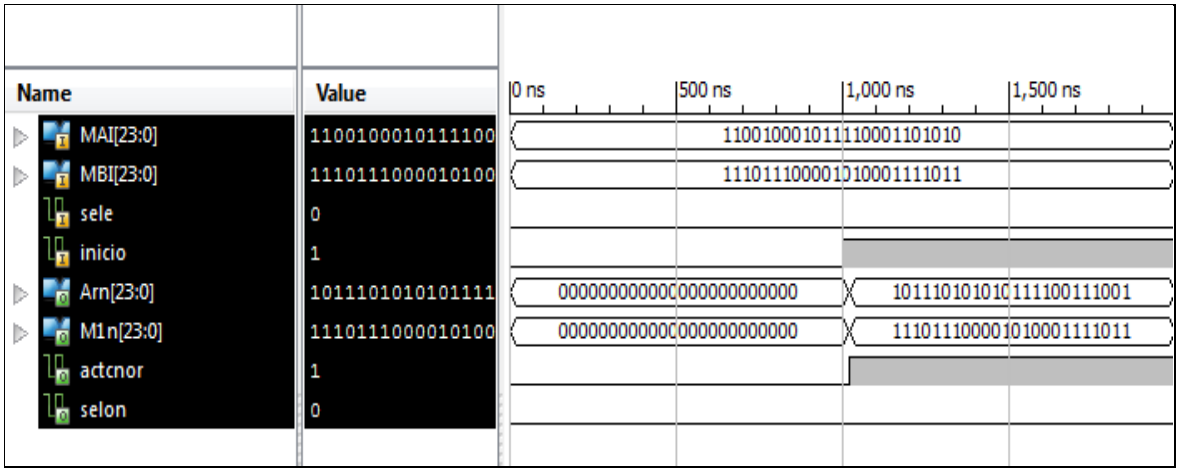

Figura 2.23. Prueba con el simulador isim para el Circuito Asíncrono producto_ctp.

El resultado obtenido de la multiplicación de las mantisas normalizadas es el siguiente:

La parte alta de la multiplicación ***Arn[23:0] = 101110101010111100111001.***

La parte baja de la multiplicación ***M1n[23:0] = 011011100001010001111011.***

La salida de activación del controlador del normalizador ***actcnor = 0.***

La salida de excepciones ***selon = 0.***

Estos datos corresponde a la multiplicación de los operando de entrada, lo que sigue es normalizar los resultados para culminar el multiplicador de punto flotante de 32 bits.

2.2.3 Módulo *npctr.*

La función de este módulo es la de normalizar el producto de la multiplicación entregado por el módulo asíncrono ***producto_ctp***. Las entradas y salidas se describen en la siguiente
Tabla 2.8

Tabla 2.8. Entradas y salidas del módulo npctr.

ENTRADAS	*TIPO*	*DESCRIPCIÓN*
EXP	Bus de 8 bits	Exponente
Mi	Bus de 24 bits	Mantisa multiplicando
Ai	Bus de 24 bits	Mantisa menor
Sele	Bit	Indica bandera de excepciones
Signo	Bit	Suma/Resta
Inicio	Bit	Indica el comienzo de la operación
SALIDAS	***TIPO***	***DESCRIPCIÓN***
DATO	Bus de 32 bits	Resultado de la multiplicación
Actproxic	Bit	Activa a *selmux*

El módulo *npctr* está compuesto internamente por dos módulos llamados ***Balsa_norprod*** y el ***ctrnor*.**

El módulo *Balsa_norprod* tiene la función de realizar la normalización de la multiplicación por medio de sus componentes de *handshake* generado por la descripción realizada con el software *System Balsa*, la descripción comportamental esta resumida en el diagrama de flujo de la Figura 2.24 y en el ***anexo 11*** se muestra el código realizado con lenguaje *Balsa*.

El algoritmo se comporta de la siguiente manera: si ***sele = 1*** indica que hubo excepciones por lo tanto no se hace normalización de la multiplicación y se muestra el dato de la excepción en formato IEEE 754.

Si ***sele = 0*** se comprueba si el bit más significativo del producto A[47] es 1. Si es 1 se desplaza el producto una vez hacia la izquierda y se suma 1 al exponente. Si es cero se desplaza dos veces el producto y el exponente no se modifica. Por último se entrega los resultados en el formato IEEE 754.

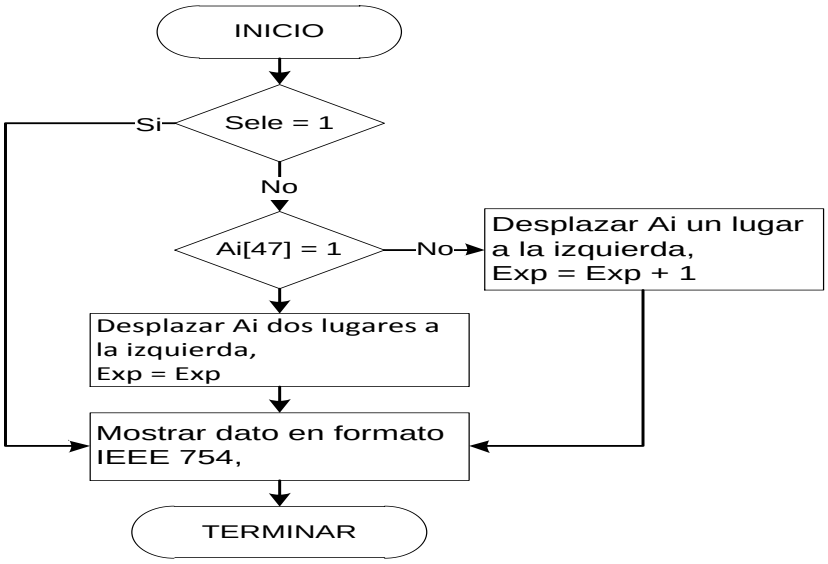

Figura 2.24. Algoritmo de funcionamiento del módulo Balsa_norprod.

El módulo ctrnor se encarga de controlar al módulo *Balsa_norprod* que tiene la tarea de realizar la normalización del producto de forma asíncrona utilizando los componentes de *Handshake* para remplazar los ciclos de reloj.

Simulación temporal de Balsa_norprod: Los datos que se utilizó para realizar la prueba del circuito son los datos que entrego el módulo multiplicador de mantisas normalizadas *producto_ctp*.

La parte alta del producto es ***Ai[23:0] = 101110101010111100111001.***

La parte baja del producto es ***Mi[23:0] = 011011100001010001111011.***

La bandera de excepciones ***sele = 0.***

El resultado del signo ***sigin_data = 1.***

El exponente es ***expin_data[7:0] = 10000000.***

La salida ***dato_data[31:0] = 11000000011101010101111001111001*** contiene la respuesta del multiplicador con la mantisa normalizada y en el formato IEEE 754 la respuesta fue entregada en el tiempo t = 6.7 us como se puede ver en la Figura 2.25.

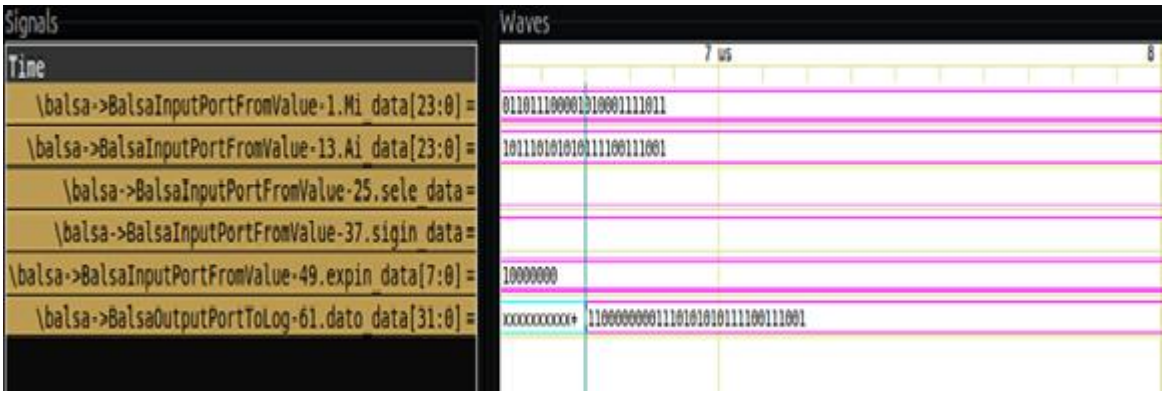

Figura 2.25. Simulación temporal del circuito asíncrono Balsa _suma.

Síntesis del diseño asíncrono de Balsa_norprod: después de haber sintetizado el código balsa en *Xilinx Ise 13.1* se integro los módulos ***ctrnor*** y ***Balsa_norprod*** para conformar el módulo ***npctr*** el cual se muestra en la **Figura 2.26.**

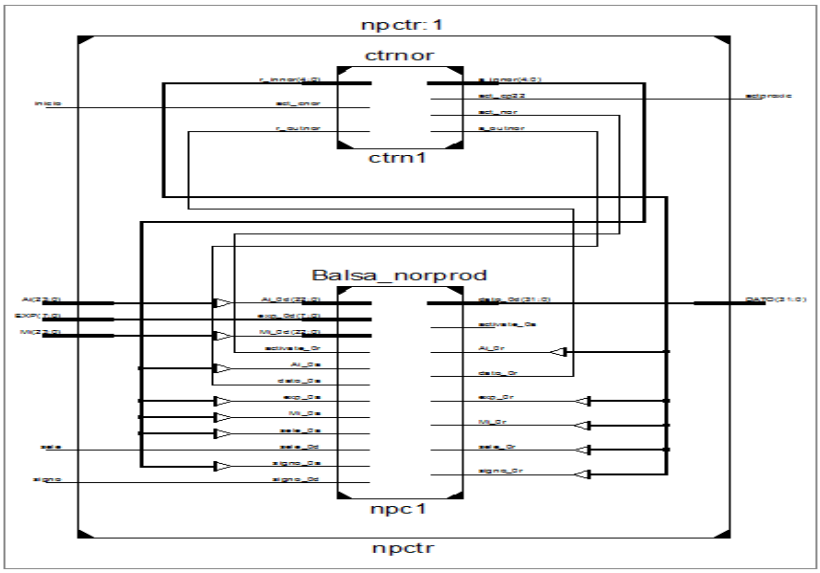

Figura 2.26. Arquitectura del módulo npctr.

En la Tabla 2.9 está consignada la cantidad de elementos lógicos, registros y *latches* utilizados en la síntesis del módulo *npctr.*

Verificación del circuito asíncrono *npctr* en Xilinx Ise: para la simulación del circuito asíncrono *npctr* se utilizó los siguientes datos los cuales se muestra en la Figura 2.27.

La parte alta de la multiplicación ***Ai[23:0] = 10111010101011110011100 1.***

La parte baja de la multiplicación ***Mi[23:0] = 01101110000101000111101 1.***

El exponente es ***EXP[7:0] = 01111100.***

La salida de excepciones ***sele = 0.***

El signo resultado ***signo = 1.***

El inicio de operación es ***inicio = 1.***

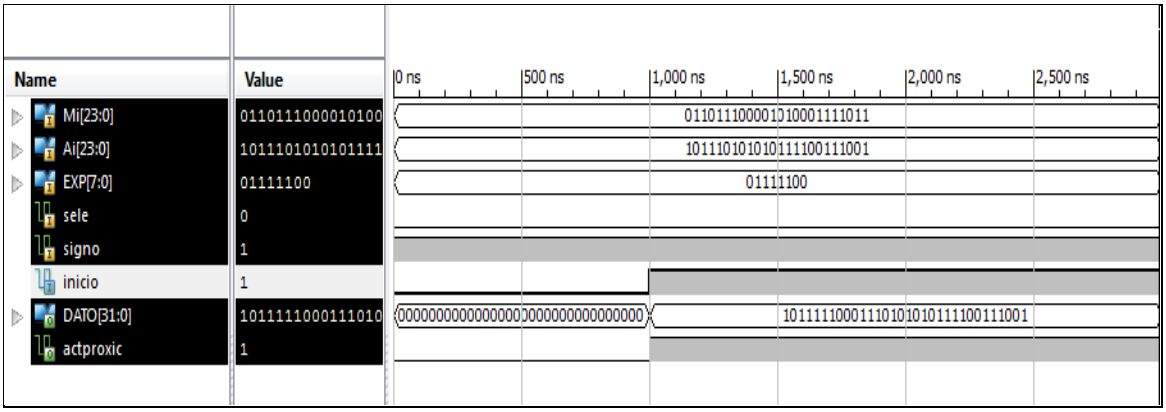

Figura 2.27. Prueba con el simulador isim de xise para el Circuito Asíncrono npctr.

El producto normalizado en formato IEEE 754 es el siguiente:

Dato[31:0]= 10111110001110101010111100111001.

El activador de ***selmux*** es ***actproxic = 0.***

El signo del resultado es ***1*** negativo. El exponente es ***01111100***. El dato del exponente que se utilizó para la prueba en *Balsa* es diferente al dato utilizado en *Xilinx Ise 13.1*. La mantisa sin el bit implícito es ***00111010101011110011100 1***. Estos datos fueron verificados y corresponden al dato esperado para esta multiplicación.

La salida para activa a ***selmux*** es ***1*** después de ***1ns*** de activarse el inicio del circuito.

2.2.4 Módulo MPFA (multiplicador_asin).

Este módulo asíncrono contiene a todos los módulos para realizar la Multiplicación de los operando ***A*** y ***B*** de 32 bits como se mira en el diagrama de bloques de la Figura 2.19. La arquitectura interna de todos sus componentes conectados se visualiza en la Figura 2.28.

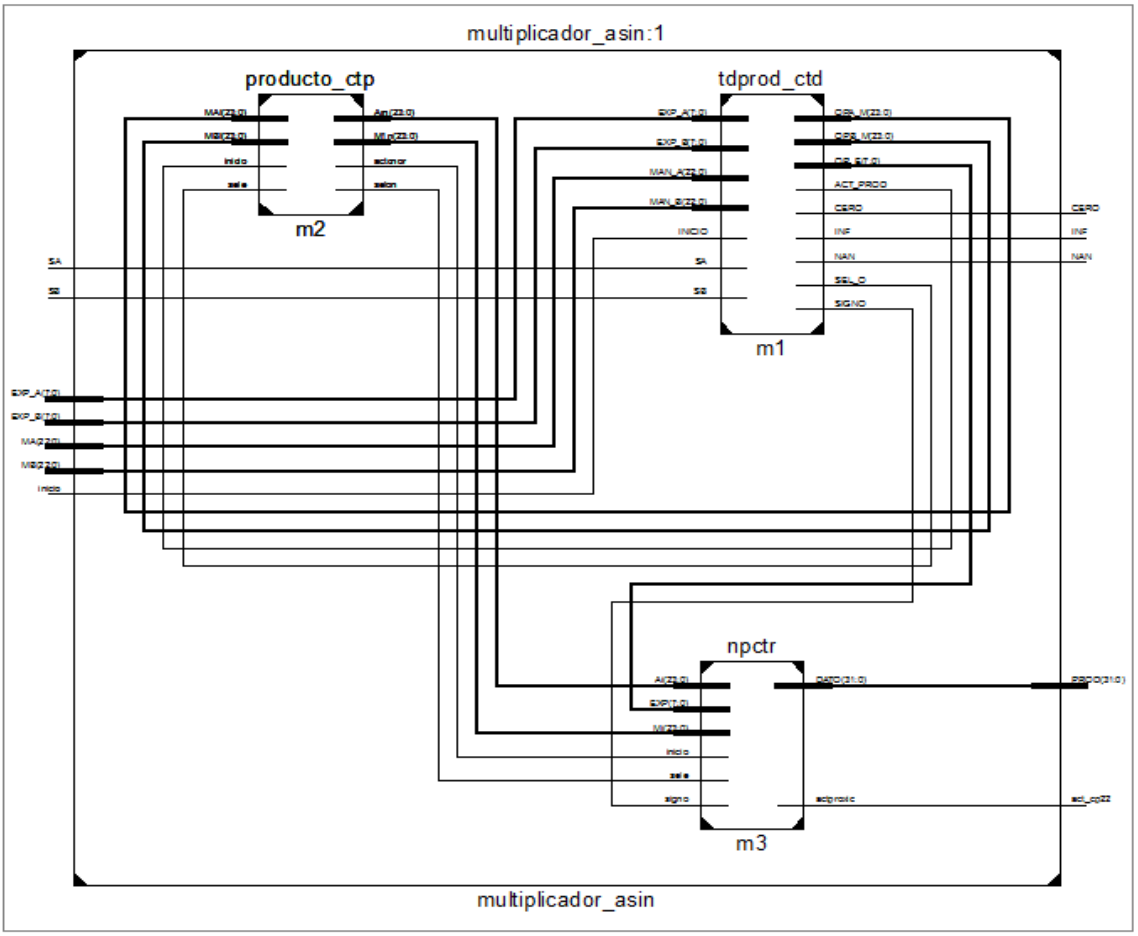

Figura 2.28. Arquitectura interna del módulo multiplicador_asin.

En la Tabla 2.9. Está consignada la cantidad de elementos lógicos, registros y latches utilizados en la síntesis del módulo *multpilicador_asin.*

Verificación del circuito asíncrono *multiplicador_asin:* para probar el multiplicador de punto flotante asíncrono se utilizó los siguientes datos:

El operando A= 12,546 y el operando B = -0,465, estos datos son pasados a binario y luego al formato IEEE 754 para cada uno de los operando de entrada.

Los exponentes A y B son: ***Exp_a[7:0] =10000010, exp_b[7:0] = 01111101.***

Las mantisas A y B son: ***man_a[22:0] = 10010001011110001101010.***

man_b[22:0] = 11011100001010001111011.

Los signo A y B son**: *sa = 0, sb = 1*,** como se muestra en la Figura 2.29.

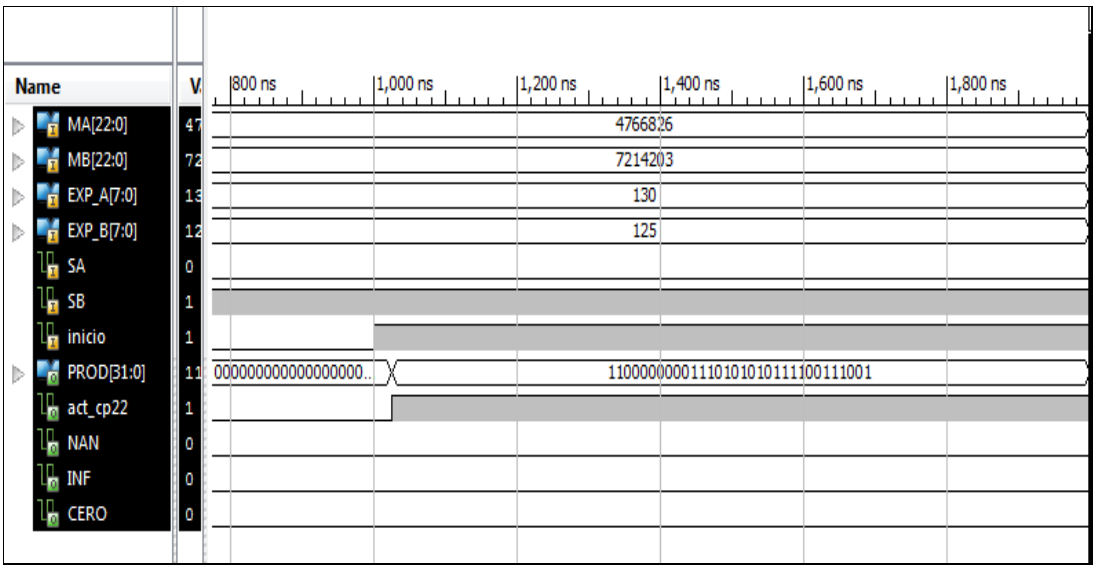

Figura 2.29. Resultados de la simulación del módulo asíncrono multiplicador_asin.

El resultado obtenido es un dato en formato IEEE 754 de 32 bits como se muestra en la Figura 2.29. ***PROD[31:0] = 11000000011101010101111001111001.*** El signo del resultado es uno (negativo). El exponente resultado es la suma algebraica de los exponentes de las entradas ***10000000***. La mantisa es el resultado de la multiplicación de las dos mantisas normalizadas ***01110101010111100111001.***

Tabla 2.9. Resumen de la síntesis del módulo tdprod_ctd.

Unidad	No. De elementos lógicos	No. De registros	No. De latches (IOB)	No. De flip flop (IOB)
tdprod_ctd	519	0	0	64
producto_ctp	5667	1977	0	49
Npctr	59	0	0	56
MPFA (multpilicador_asin)	8215	1853	0	64

2.3 UNIDAD DE DIVISIÓN PARA NÚMEROS EN PUNTO FLOTANTE ASÍNCRONA DE 32 BITS (DPFA).

Esta unidad tiene la función de realizar la división de dos operandos en punto flotante de 32 bits. Esta conformado por los módulos funcionales realizados en código Balsa ctrltddivision, Diman, Normaldiv y los modulos combinatorios para control ctrltddivision, ctrdivman, ctrlenor y divctrlnor. En la Figura 2.30. Se detalla la organización con todos sus módulos.

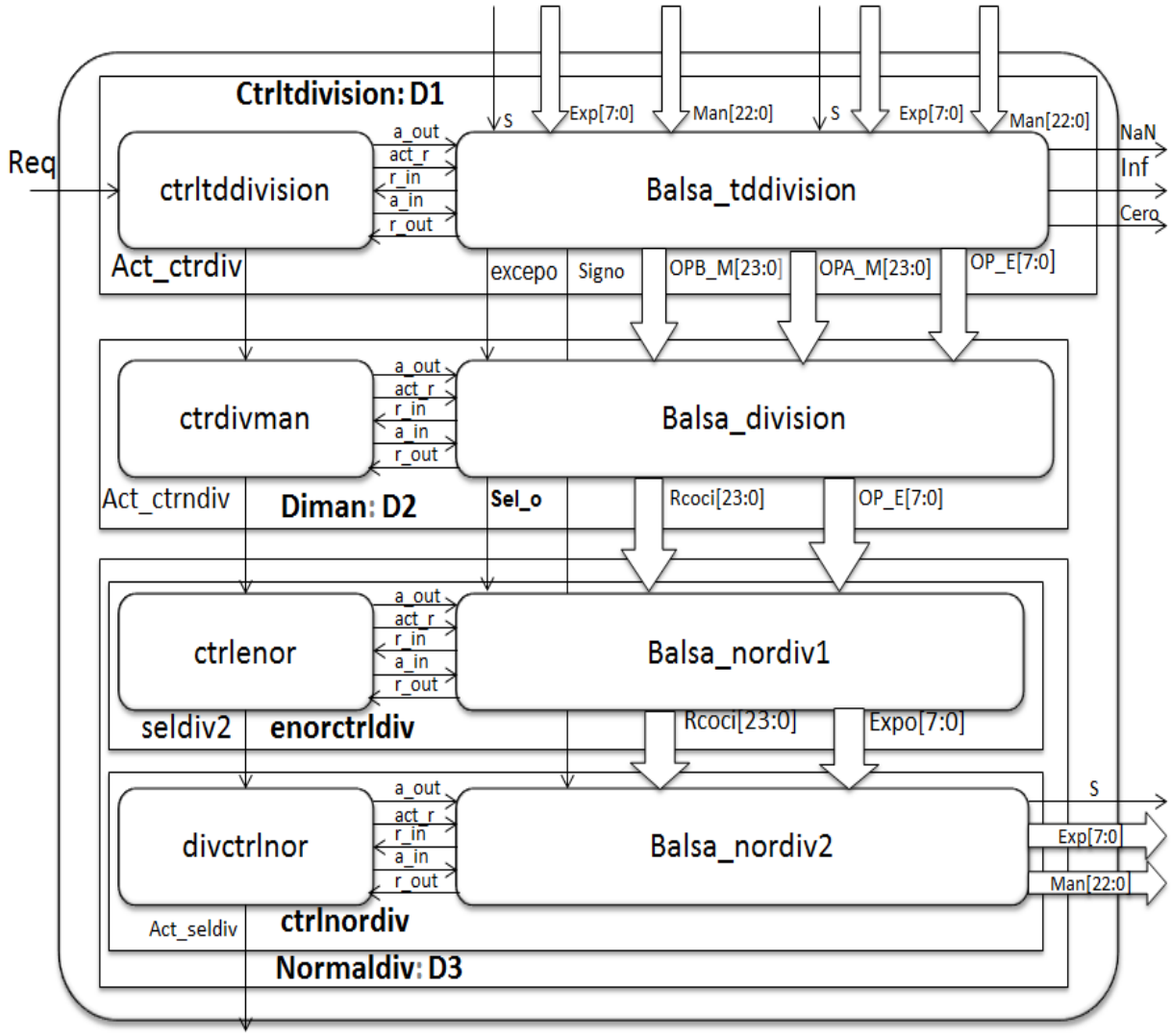

Figura 2.30. Unidad de división asíncrona para punto flotante de 32 bits (DPFA).

2.3.1 Módulo *ctrltdivision*

La función de este módulo es de seleccionar el tipo de datos a la entrada y preguntar si hay alguna excepción para activar la bandera respectiva (*NaN, Inf, Cero*), también calcula el exponente realizando la resta algebraica entre los exponentes de entrada. En la Tabla 2.10 se describe cada una de las entradas y salidas del módulo.

Tabla 2.10. Entradas y salidas del módulo ctrltdivision.

ENTRADAS	***TIPO***	***DESCRIPCIÓN***
expA	Bus de 8 bits	Exponente del operando A
expB	Bus de 8 bits	Exponente del operando B
Ma	Bus de 23 bits	Mantisa del operando A
Mb	Bus de 23 bits	Mantisa del operando B
Inicio	Bit	Señal de inicio
Sa	Bit	Signo del operando A
Sb	Bit	Signo del operando B
SALIDAS	**TIPO**	**DESCRIPCIÓN**
Expout	Bus de 8 bits	Resultado del Exponente
Mao	Bus de 24 bits	Mantisa A normalizada
Mbo	Bus de 24 bits	Mantisa B normalizada
Cero	Bit	1 = el resultado es cero, 0 = resultado diferente de cero
Inf	Bit	Número infinito
Nan	Bit	Operación invalida
Excep	Bit	Indica si hubo excepciones
Actnex	Bit	Activador del ctrl del multiplicador
Sig	Bit	Signo del resultado

El módulo ctrltdivision internamente tiene el módulo *Balsa_tddivision* y el controlador ***ctrltddivision*** los cuales se describirán a continuación.

Módulo *Balsa_tddivision:* la función de este módulo es de detectar si los operando de entrada son normales o si presentan excepciones. La descripción comportamental esta resumida en el diagrama de bloques de la Figura 2.31. La descripción realizada en código *Balsa* se encuentra en el ***anexo 12.*** A continuación se explica el funcionamiento del algoritmo.

Cuando se activa la entrada *INICIO* el módulo verifica las excepciones a la entrada como son la de operación invalida (*NaN*), Infinito **(***Inf*) y operando cero (*cero*). Si no hay excepciones calcula el exponente resultante realizando la resta de los exponentes de entrada y se suma el exceso 127 para que el exponente quede formateado con el exceso 127. Luego se normaliza las mantisas y se entregan todos los resultados incluyendo las excepciones con valor cero. El signo se determina usando la operación lógica ***AND.***

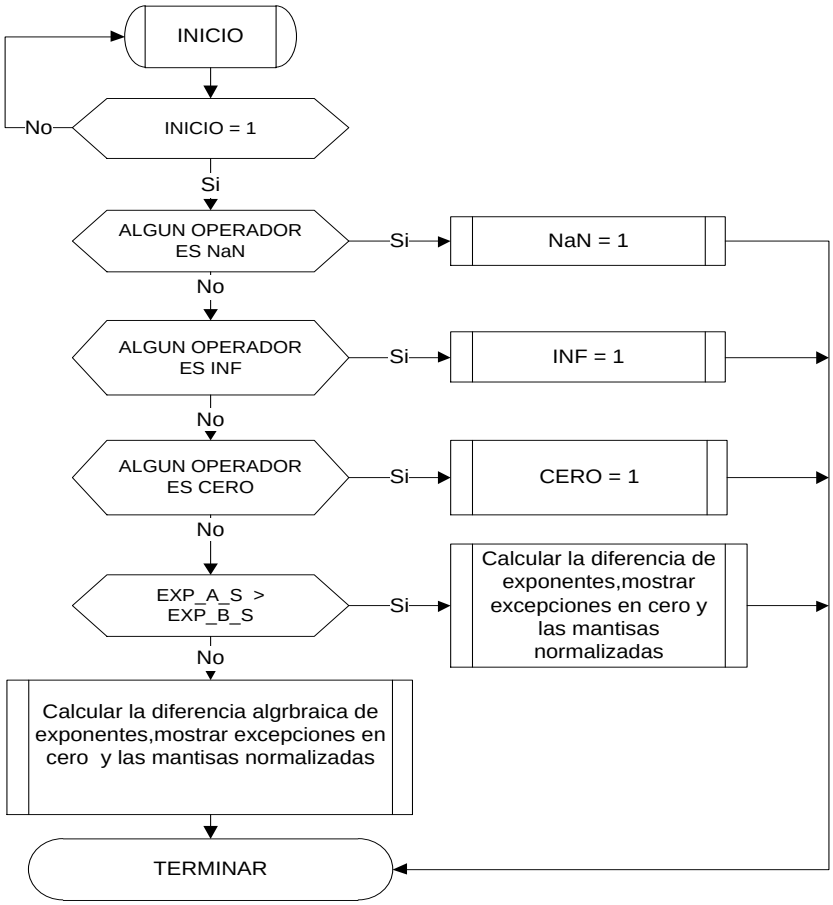

Figura 2.31. Diagrama de flujo del funcionamiento del módulo Balsa_tddivision.

Módulo *ctrlddivision:* la función principal es la de establecer el *handshake* con el módulo *Balsa_tddivision* para que pueda recibir los datos de la entrada y entregarlo al módulo de división de mantisa. La Figura 2.4 contiene el diagrama de flujo del algoritmo de control.

Síntesis del diseño asíncrono del módulo *ctrltdivision:* el módulo *Balsa_tddivision* está conectado al controlador *ctrltddivision* de la Figura 2.32 a través de las salidas de requerimiento y a las entradas de reconocimiento para que pueda tomar los datos de la entrada, procesarlos y luego mostrarlos en la salida. Esta conexión también se realizó para conformar un solo módulo que permite ver las entradas y salidas como los circuitos convencionales.

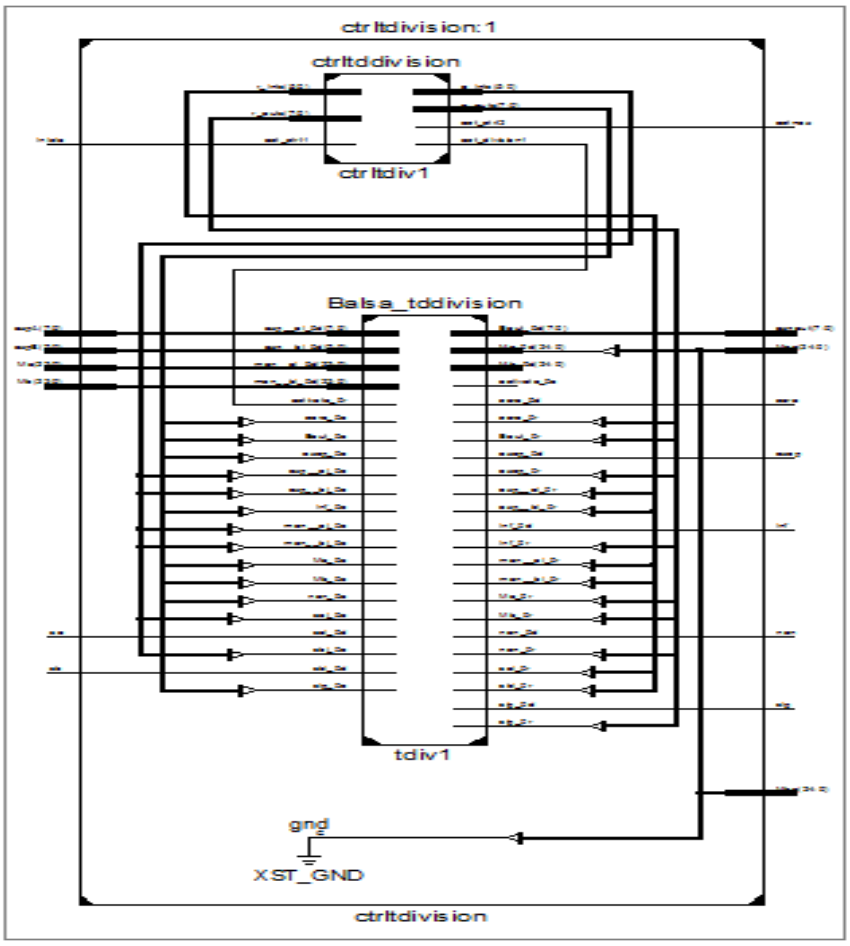

Figura 2.32. Arquitectura interna del módulo ctrltdivision con el controlador ctrltddivision y el módulo Balsa_tddivision.

El principio de funcionamiento de este circuito es el mismo que se explicó para el módulo de selección de tipo de dato de la suma y la multiplicación *tdprod_ctd.* Hay que tener en cuenta que las entradas y salidas son diferentes, pero su finalidad es la misma.

En la Tabla 2.13 está consignada la cantidad de elementos lógicos y de registros utilizados en la síntesis del módulo *Balsa_tddivision* y el de control.

Verificación del circuito asíncrono ctrltdivision en Xilinx Ise 13.1: para verificar el circuito asíncrono ctrltdivision se utilizó el operando A= 12,546 y el operando B = -0,465.

El dato equivalente en el formato IEEE 754 queda de la siguiente manera:

sb = 1. expB = 01111101. Ma = 11011100001010001111011. expA = 10000010. Mb = 10010001011110001101010. sa = 0. Como se muestra en la parte baja de la Figura 2.33.

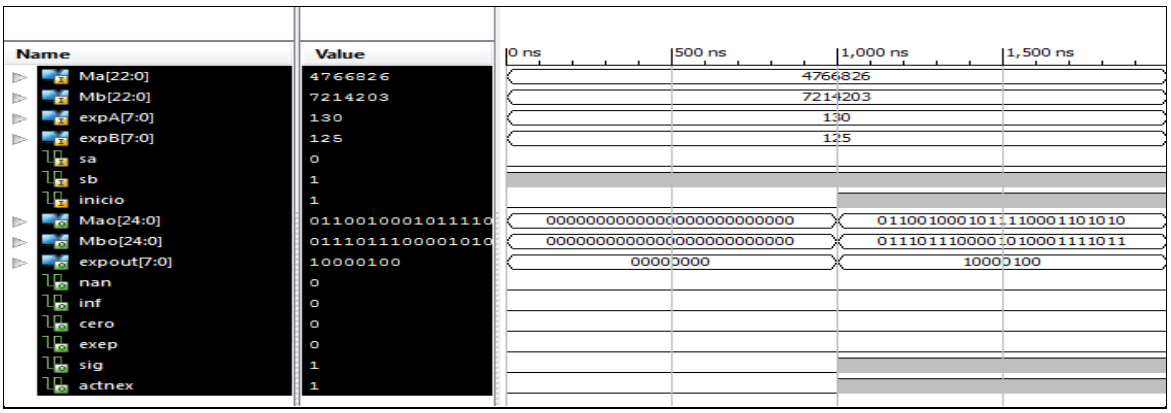

Figura 2.33. Prueba con el simulador isim para el Circuito Asíncrono ctrltdivision.

Los resultados obtenidos son:

Mao[24:0] = 011001000101111000110101O.

Mbo[24:0] = 011101110000101000111101l.

expout[7:0] = 10000100. excep = 0. cero = 0. sig = 1. nan = 0. inf = 0. actnex = 1.

A las salidas *Mao[24:0]* y *Mbo[24:0]* se les agregó el bit implícito y el bit de desbordamiento a la izquierda para para garantizar un buen resultado. Esto ocasiona que las mantisas aumenten su tamaño. El dato del exponente ***expout[7:0]*** es el resultado de la resta de los exponentes de entada. Las excepciones ***NaN = 0, INF = 0, CERO = 0*** no se activaron. El signo como era de esperarse resulto negativo.

2.3.2 Módulo *divman.*

La función de este módulo es la de dividir las mantisas normalizadas entregadas por el módulo ***ctrltdivision***. En la Tabla 2.11 se describe cada una de las entradas y salidas.

Tabla 2.11. Entradas y salidas del módulo divman.

ENTRADAS	*TIPO*	*DESCRIPCIÓN*
Mai	Bus de 24 bits	Mantisa A normalizada
Mbi	Bus de 24 bits	Mantisa B normalizada
Excep	Bit	Indica bandera de excepciones
Inicio	Bit	Indica el comienzo
SALIDAS	*TIPO*	*DESCRIPCIÓN*
Rcoci	Bus de 24 bits	Cociente de la división de mantisas
Actctrlnor	Bit	Activa el controlador de normalización

El módulo divman está compuesto internamente por dos módulos llamados *Balsa_division* y el *ctrdivman.* La Figura 2.34 contiene el diagrama de flujo del algoritmo general del funcionamiento de este módulo.

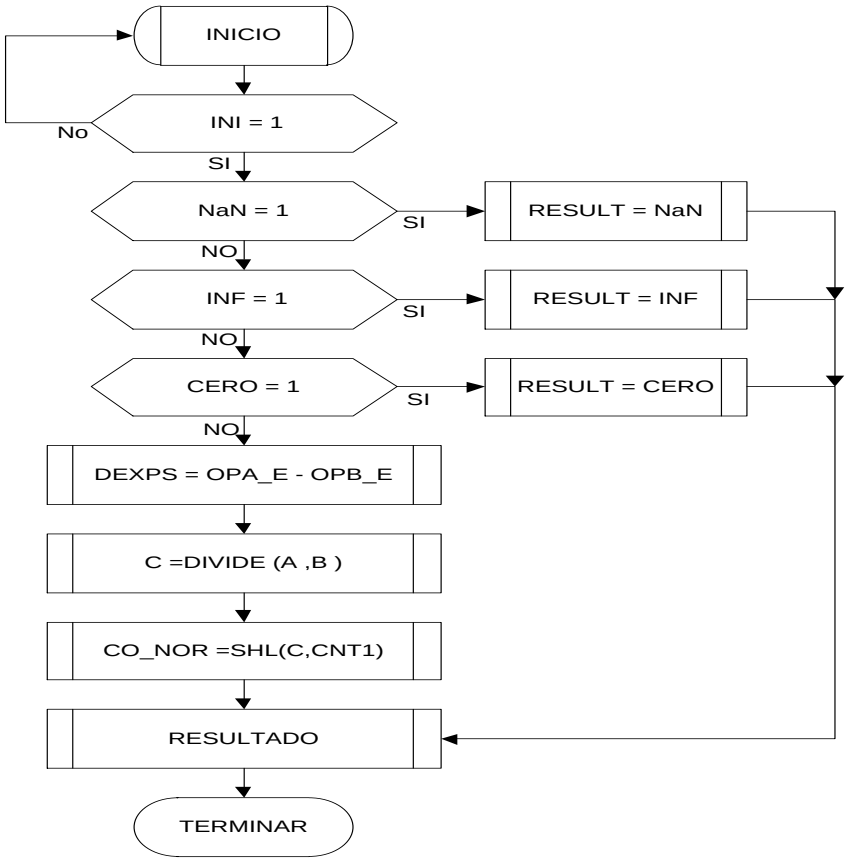

Figura 2.34. Algoritmo general del módulo divman.

Como se puede mirar en la figura anterior la división solo se realiza si no se presentan excepciones, en caso de que se presenten el resultado dependerá de la excepción que ocurra.

El módulo *Balsa_division e*s un macro módulo que representa 24 módulos provenientes de la descripción en *Balsa* llamados ***Balsa_d1*** hasta ***Balsa_d24.*** La idea es de recrear las 24 iteraciones que realiza la división debido a que los operando son de 24 bits. Para la descripción en *Balsa* se utilizó el algoritmo de restas y desplazamiento para la división binaria. Por razones de tamaño solo se agrega el código y el diagrama de handshake de los módulos ***Balsa_d1*** se muestra en el ***anexo 13*** y el del módulo ***Balsa_d24*** en el **anexo *14***.

Módulo *ctrdivman:* *es* un macro módulo de control que contiene 24 controles. Cada uno para un módulo *Balsa_dn* diferente, todos cumplen con la misma función de los controladores antes descritos. En la Figura 2.4 se muestra la gráfica del diagrama de flujo del módulo controlador.

En la **¡Error! No se encuentra el origen de la referencia.** está consignada la cantidad de elementos lógicos, registros y *latches* utilizados en la síntesis del módulo *divman.*

Verificación del circuito asíncrono *divman* en el nivel de compuertas: para la simulación del circuito asíncrono *divman* se utilizó los siguientes datos:

Mantisa A ***Mai[24:0] = 01100100010111100011010 10.***

Mantisa B ***Mbi[24:0] = 01110111000010100011110 11.***

La bandera de excepciones ***excep = 0.***

La entrada de inicialización ***Inicio = 1.*** Los cuales se muestran en la Figura 2.35.

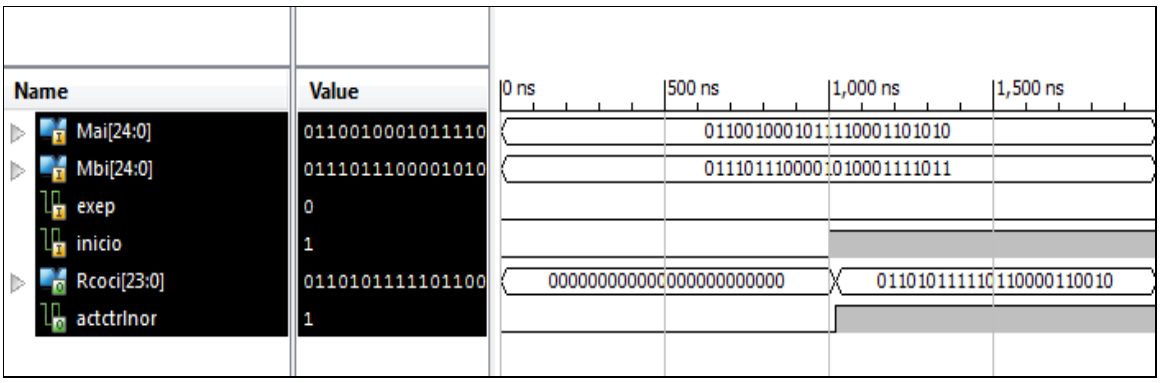

Figura 2.35. Prueba con el simulador isim para el Circuito Asíncrono divman.

El resultado obtenido de la división de las mantisas normalizadas es el siguiente:

El resultado de la división es ***Rcoci[23:0] = 011010111110110000110010.***

La salida de activación del controlador del normalizadores ***actctrlnor = 0.***

Estos datos corresponde a la división de los operando de entrada, lo que sigue es normalizar los resultados para culminar la división de punto flotante de 32 bits de acuerdo al estándar IEEE 754.

2.3.3 Módulo *normaldiv.*

La función de este módulo es de normalizar las mantisas de la división entregada por el módulo asíncrono divman, en la Tabla 2.12 se describe las entradas y salidas.

Tabla 2.12. Entradas y salidas del módulo normaldiv.

ENTRADAS	***TIPO***	***DESCRIPCIÓN***
Exp	Bus de 8 bits	Exponente
Data	Bus de 24 bits	Mantisa del cociente sin normalizar
Excep	Bit	Indica bandera de excepciones
Sig	Bit	Signo resultado
Ini	Bit	Indica el comienzo de la operación
SALIDAS	***TIPO***	***DESCRIPCIÓN***
Datout	Bus de 32 bits	Resultado de la división en punto flotante de 32 bits
Actcdi	Bit	Activa a selmux

El módulo *normaldiv* está compuesto internamente por dos módulos llamados ***enorctrldiv*** y el ***ctrlnordiv,*** el diagrama de flujo del algoritmo de funcionamiento se muestra a continuación en la Figura 2.36.

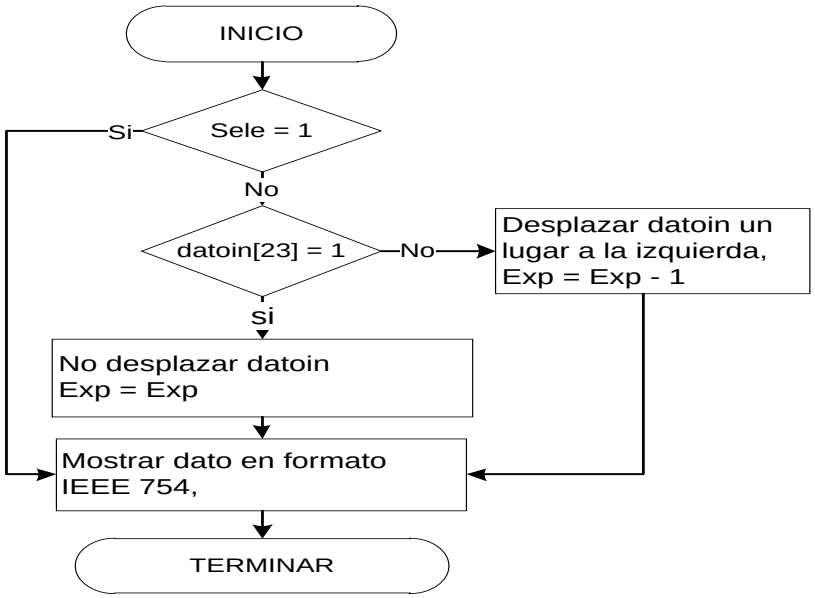

Figura 2.36. Diagrama de flujo del Algoritmo de funcionamiento del módulo normaldiv.

Si ***sele = 1*** indica que hubo excepciones por lo tanto no se hace normalización de la multiplicación y se muestra el dato de la excepción en formato IEEE 754.

Si ***sele = 0*** se comprueba si el bit más significativo de la división es 1. Si es 0 se desplaza el producto una vez hacia la izquierda y se resta 1 al exponente. Si es cero no se desplaza el cociente de la división y el exponente no se modifica. Por último se entrega los resultados en el formato IEEE 754.

El módulo *enorctrldiv* tiene la función de ajustar el exponente en caso de realizar el desplazamiento del cociente de la división para realizar la normalización, esto se logra gracias a la arquitectura interna que está conformada por los módulos ***balsa_enordiv*** y ***ctrlenor*.** Mirar la Figura 2.37. La cual muestra las conexiones realizadas entre los dos módulos.

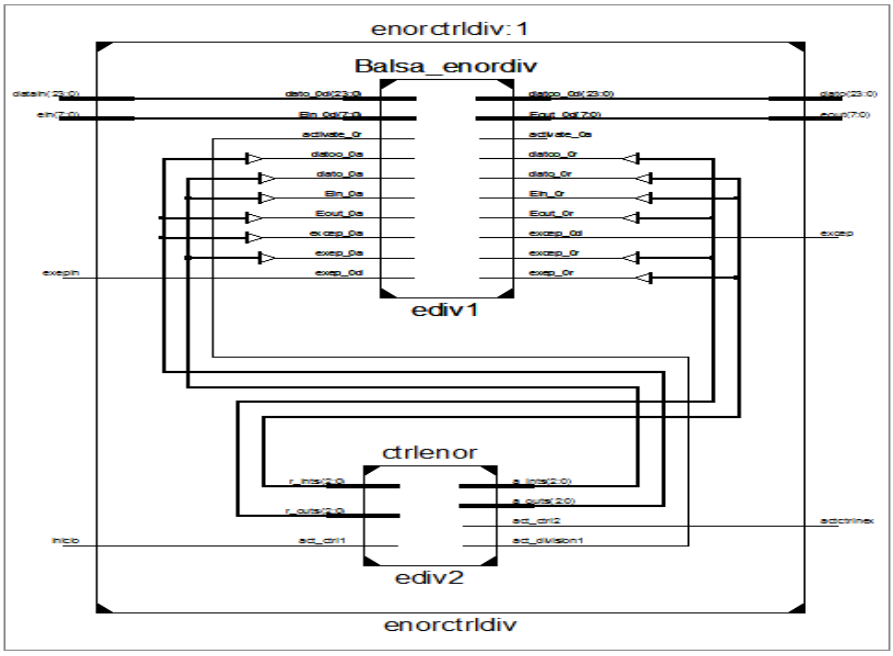

Figura 2.37. Arquitectura del módulo enorctrldiv.

El módulo *ctrlnordiv* tiene la función de realizar el desplazamiento del resultado de la división de las dos mantisas realizadas por el módulo divman, esto se consigue por la conexión interna de su arquitectura de los módulos ***Balsa_nordiv*** y ***divctrlnor*****,** esta conexión se muestra en la siguiente Figura 2.38.

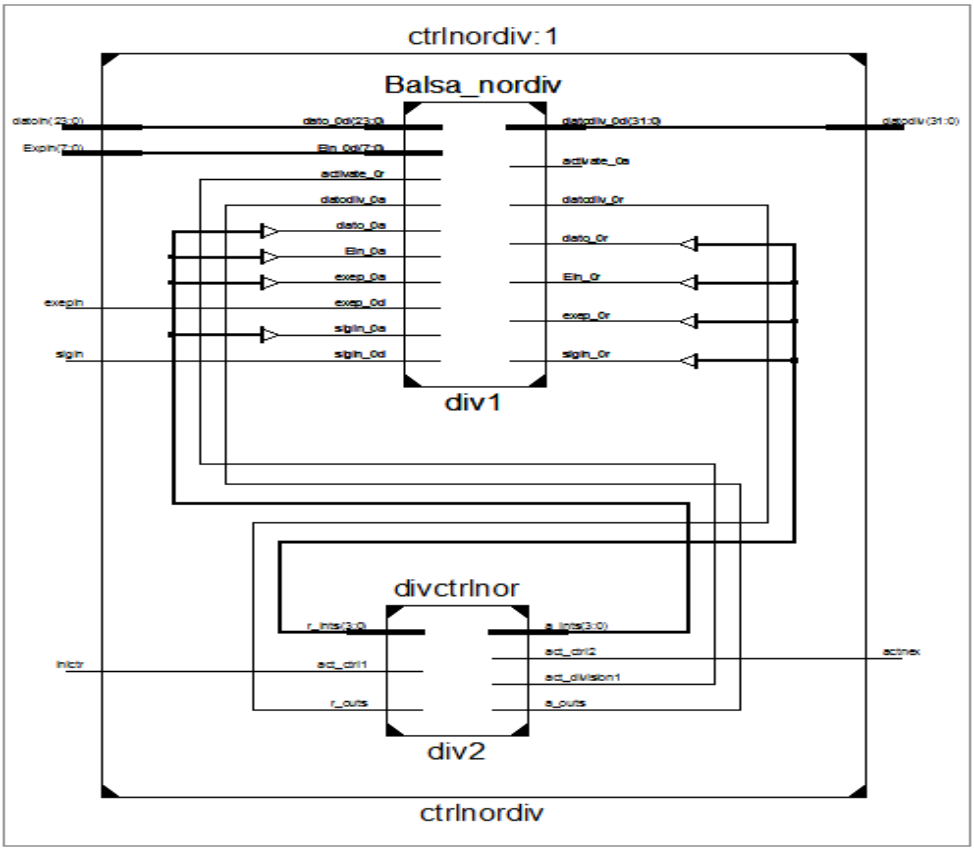

Figura 2.38. Arquitectura del módulo ctrlnordiv.

Síntesis del diseño asíncrono de *normaldiv:* después de haber realizado las conexiones de los módulos *enorctrldiv* y el *ctrlnordiv* se creó la entidad ***normaldiv*** que engloba la idea general de normalizar el resultado de la división. La arquitectura y la entidad se visualizan en la Figura 2.39.

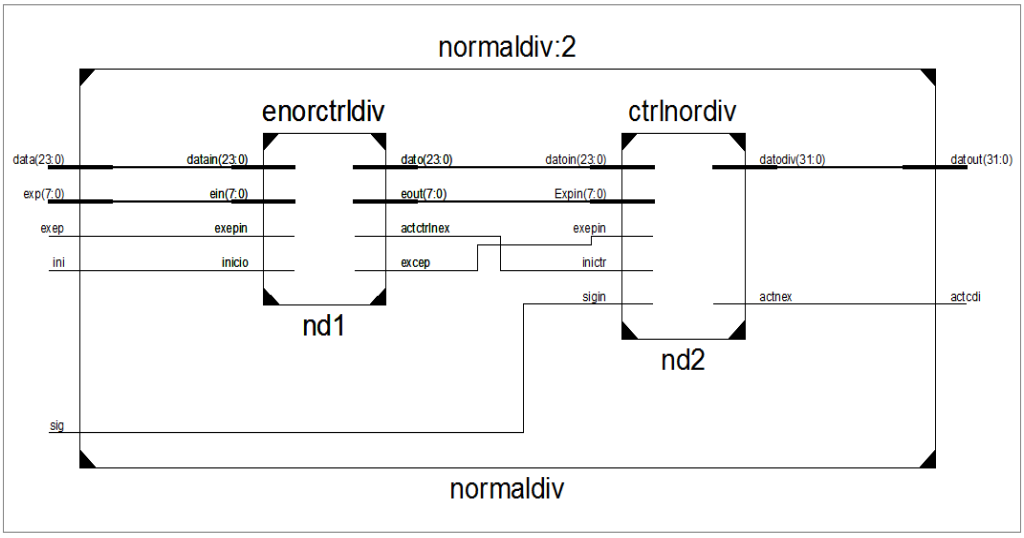

Figura 2.39. Arquitectura del módulo normaldiv.

En la Tabla 2.13 está consignada la cantidad de elementos lógicos, registros y *latches* utilizados en la síntesis del módulo *normaldiv.*

Verificación del circuito asíncrono normaldiv en el nivel de compuertas: para la simulación del circuito asíncrono *normaldiv* se utilizaron los siguientes datos:

El resultado de la división de las mantisas es: ***Rcoci[23:0] = 011010111110110000110010 = 7072818.***

El exponente *es* ***EXP[7:0] = 10000100.***

La salida de excepciones ***sele = 0.***

El signo resultado ***signo = 1.***

El inicio de operación es ***inicio = 1.*** *Estos datos* se visualizan en la Figura 2.40.

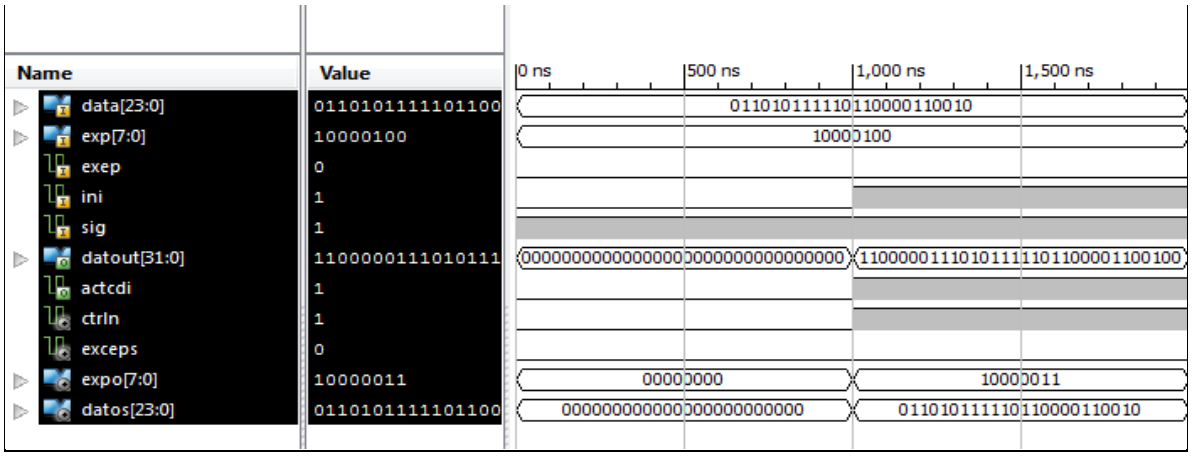

Figura 2.40. Prueba con el simulador isim de xise para el Circuito Asíncrono normaldiv.

El producto normalizado en formato IEEE 754 es el siguiente:

Dato[31:0]= 11000001110101111101100001100100.

El activador de selmux es ***actcdi = 1.***

El signo del resultado es ***1*** negativo. El exponente es ***10000011.*** Fue modificado debido a la normalización de la mantisa resultante. La mantisa sin el bit implícito es ***10101111101100001100100***. Estos datos fueron verificados y corresponden al dato esperado para esta división. La salida para activar a ***selmux*** es ***1*** después de ***1ns*** de activarse el inicio del circuito.

2.3.4 Módulo *divisiónpf.*

Este módulo asíncrono contiene a todos los módulos para realizar la división de los operando ***A*** y ***B*** de 32 bits como se mira en el diagrama de bloques de la Figura 2.30. La arquitectura interna de todos sus componentes conectados se visualiza en la Figura 2.41.

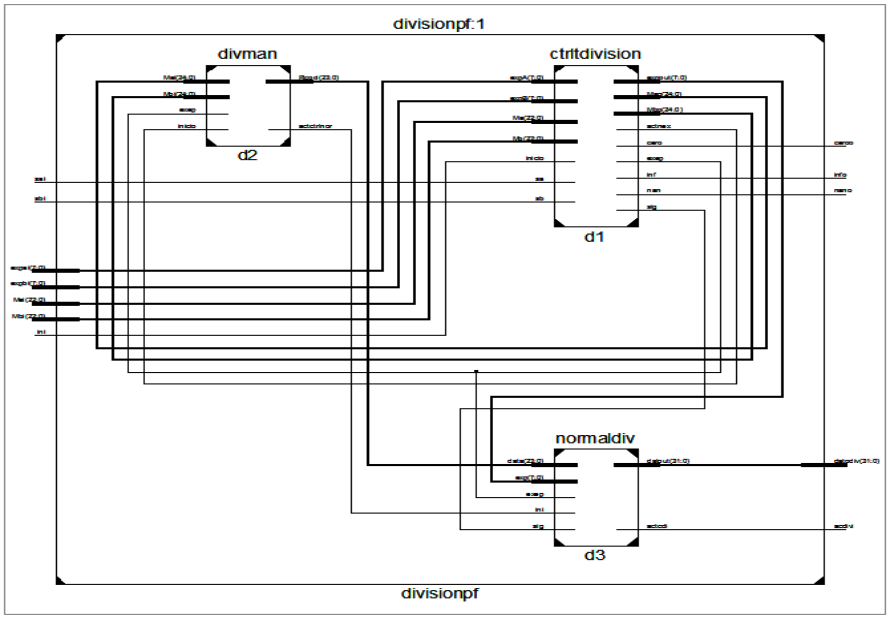

Figura 2.41. Arquitectura interna del módulo divisionpf.

El resumen de la síntesis del módulo divisor de punto flotante asíncrono se muestra en la Tabla 2.13.

Verificación del circuito asíncrono *divisionpf:* *p*ara probar el divisor de punto flotante asíncrono se utilizó los siguientes datos: el operando A= 12,546 y el operando B = -0,465, estos datos son pasados a binario y luego al formato IEEE 754 para cada uno de los operando de entrada.

Los exponentes A y B son: ***Expai[7:0] =10000010, expbi[7:0] = 01111101.***

Las mantisas A y B son: ***Mai[22:0] = 10010001011110001101010.***

Mbi[22:0] = 11011100001010001111011.

Los signo A y B son**: *sai = 0, sbi = 1*,** como se muestra en la Figura 2.42.

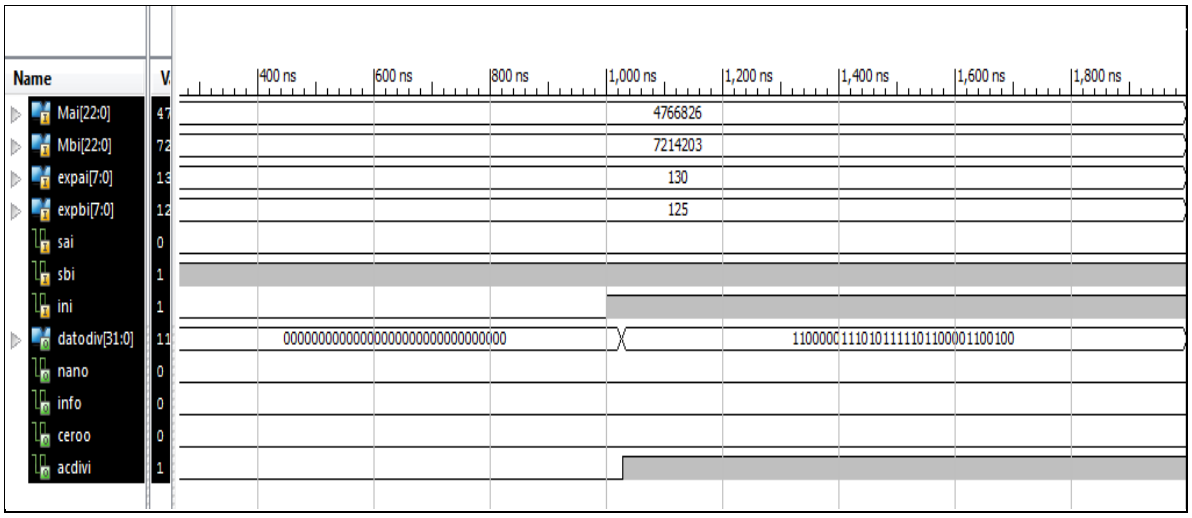

Figura 2.42. Resultados de la simulación del módulo asíncrono divisionpf.

El resultado obtenido es un dato en formato IEEE 754 de 32 bits como se muestra en la Figura 2.42. ***datodiv[31:0] = 11000001110101111101100001100100.*** El signo del resultado es uno (negativo). El exponente resultado es ***10000011*** equivale a la resta algebraica de los exponentes de las entradas. La mantisa es ***10101111101100001100100*** determinado por la división de las dos mantisas normalizadas***.***

Tabla 2.13. Resumen de la síntesis del los módulos DPFA.

Unidad	*No. De elementos lógicos*	*No. De registros*	*No. De latches(IOB)*	*No. De flip flop (IOB)*
ctrltdivision	5667	1977	0	49
Divman	5667	1977	0	49
normaldiv	183	33	0	34
divisionpf	17373	4155	0	4

2.4 MODULO *DECO.*

Es básicamente un decodificador que funciona con el protocolo de handshake. La función de este módulo es de tomar el código de operación colocado por el usuario y activar el módulo que tiene la función de realizar dicha operación como se muestra en el diagrama de bloques de la Figura 2.1. La Tabla 2.14 contiene las entradas y salidas. La Tabla 2.15 muestra el funcionamiento del módulo ***deco*** de acuerdo al código de operación.

La siguiente Figura 2.43 muestra el diagrama de bloques del módulo ***deco*** que se detalla en profundidad.

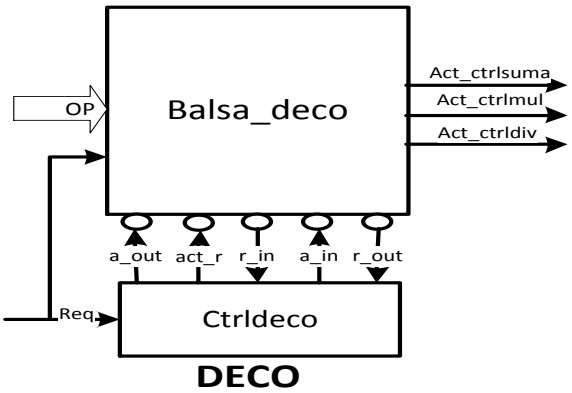

Figura 2.43. Diagrama de bloques del módulo deco.

Tabla 2.14. Entradas y salidas del módulo deco 1 de 3.

ENTRADAS	*TIPO*	*DESCRIPCIÓN*
OP	Bus de 2 bits	Código de operación
Ini	Bit	Indica el comienzo de la operación
SALIDAS	*TIPO*	*DESCRIPCIÓN*
Sumar	Bit	Activa el controlador del módulo suma
Divir	Bit	Activa el controlador del módulo multiplicador
Multr	Bit	Activa el controlador del módulo divisor

Tabla 2.15. Tabla de operación del módulo deco 1 de 3.

Ini	*OP[1:0]*	*sumar*	*multr*	*divir*
0	00	0	0	0
0	01	0	0	0
0	10	0	0	0
0	11	0	0	0
1	00	1	0	0
1	01	0	1	0
1	10	0	0	1
1	11	0	0	0

El módulo *deco* está compuesto internamente por dos módulos llamados ***Balsa_deco*** y el ***ctrldeco*.**

La Tabla 2.15 muestra el comportamiento de este módulo. Cuando la entrada ***ini = 0*** no se selecciona ningún bloque de ejecución. Cuando ***ini*** es ***1*** se puede elegir con el código ***00*** la operación ***suma.*** Con el código ***01*** la operación de ***multiplicación.*** Con el código ***10*** se elige la operación de ***división*** y con el código ***11*** no selecciona ninguna operación.

Módulo *Balsa_deco:* *e*ste módulo fue diseñado en *Balsa* y sintetizado en *Xilinx Ise 13.1*. Este módulo es el encargado de realizar la función de decodificación y solo depende del controlador para realizar su tarea. En la Tabla 2.15 y en la Figura 2.44 se muestra el comportamiento.

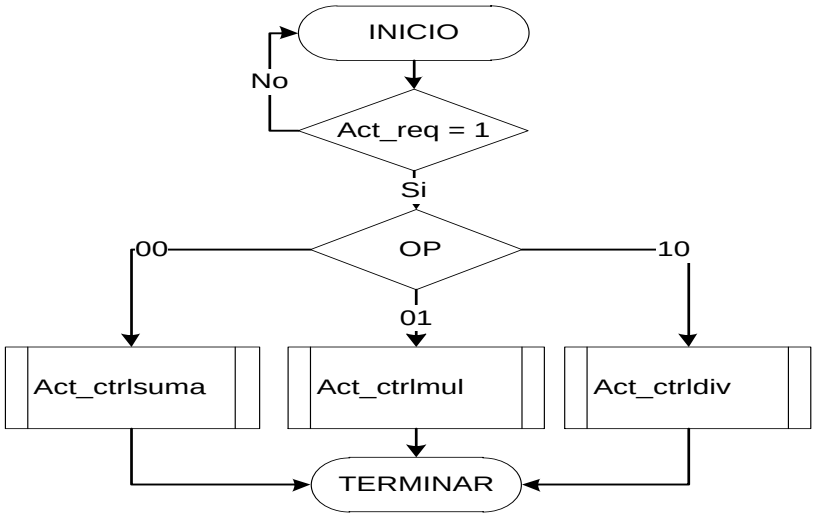

Figura 2.44. Algoritmo del módulo deco.

Módulo *ctrldeco:* la función de este módulo es la de controlar el módulo *deco* para que pueda realizar las tares especificada en la Tabla 2.15.

En la Figura 2.45 se muestra el código del módulo *deco* realizado en *System Balsa 4.0*, el cual está hecho de acuerdo a las funciones especificadas en la Tabla 2.15.

```
Files
deco.balsa*
 1
 2  import [balsa.types.basic]
 3  procedure deco ( input op : 2 bits;
 4                   input ini : 1 bits;
 5                   output sumar,divir,multr : 1 bits      )is
 6
 7  variable inis : 1 bits
 8  variable ops : 2 bits
 9
10  begin
11
12  loop
13  ini->inis||
14  op -> ops;
15
16  if inis = 1 then
17          case ops of 0 then
18                  sumar <- 1 ||
19                  divir <- 0 ||
20                  multr <- 0
21
22          |1 then
23
24                  sumar <- 0 ||
25                  divir <- 0 ||
26                  multr <- 1
27
28          |2 then
29
30                  sumar <- 0 ||
31                  divir <- 1 ||
32                  multr <- 0
33
34          |3 then
35
36                  sumar <- 0 ||
37                  divir <- 0 ||
38                  multr <- 0
39          end
40
41  else
42                  sumar <- 0 ||
43                  divir <- 0 ||
44                  multr <- 0
45  end
46  end
47  end--begin
```

Figura 2.45. Código en balsa para el módulo deco.

Síntesis del diseño asíncrono de *deco:* con el archivo *Verilog* generado por *Balsa* se realizo la síntesis RTL del módulo *Balsa_deco* en la herramienta síncrona *Xilinx Ise 13.1*. Luego se creó el módulo *deco* el cual tiene en su organización el controlador *ctrldeco* y el módulo *Balsa_deco* como se puede mirar en la Figura 2.46.

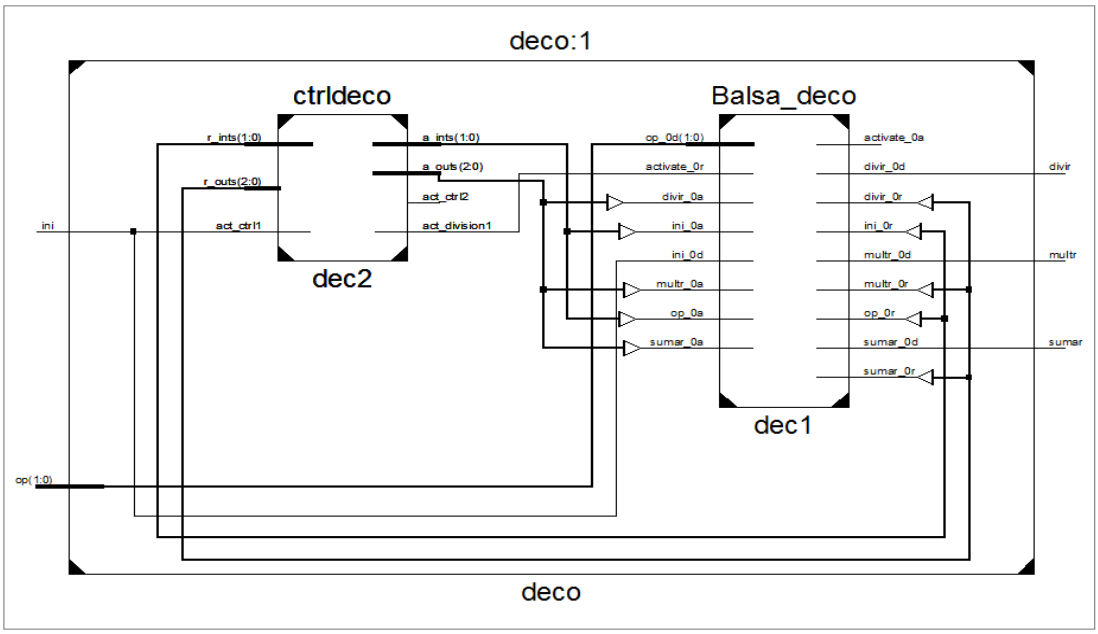

Figura 2.46. Arquitectura del módulo deco con el módulo Balsa_deco y ctrldeco.

En la Tabla 2.16 están consignados la cantidad de elementos lógicos, registros y *latches* utilizados en la síntesis del módulo *normaldiv.*

Tabla 2.16. Resumen de la síntesis del módulo deco.

Unidad	*No. De elementos lógicos*	*No. De registros*	*No. De latches(IOB)*	*No. De flip flop (IOB)*
Deco	81	1	0	2

2.5 Módulo *SELEOR.*

Es básicamente una compuerta ***or*** que se utiliza para activar al controlador del módulo ***MUX.*** La salida de este módulo se activa cada vez que una de las unidades de operación termina de realizar su tarea para habilitar el módulo de selección de resultado. En la Figura 2.47 se muestra la entidad que contiene la compuerta ***or*** realizada con ***AND*** y ***NOT***.

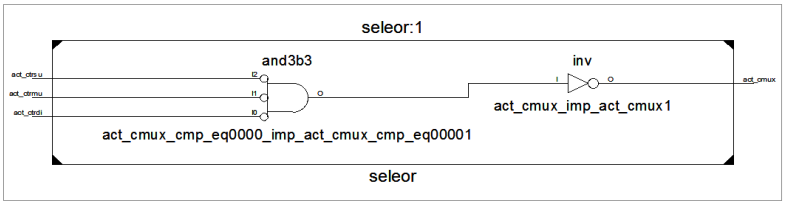

Figura 2.47. Módulo seleor.

En la Tabla 2.17 está consignada la cantidad de elementos lógicos, registros y *latches* utilizados en la síntesis del módulo *seleor*.

Tabla 2.17. Resumen de la síntesis del módulo seleor.

Unidad	*No. De elementos lógicos*	*No. De registros*	*No. De latches(IOB)*	*No. De flip flop (IOB)*
Seleor	1	0	0	0

2.6 MÓDULO *MUX*.

Es básicamente un multiplexor. La función de este módulo es la de entregar a la salida el resultado del número en punto flotante de la operación seleccionada en la entrada ***OP*** y las banderas de excepciones. Las entradas y salidas se muestran en la Tabla 2.18. El diagrama de bloque se muestra en la Figura 2.48.

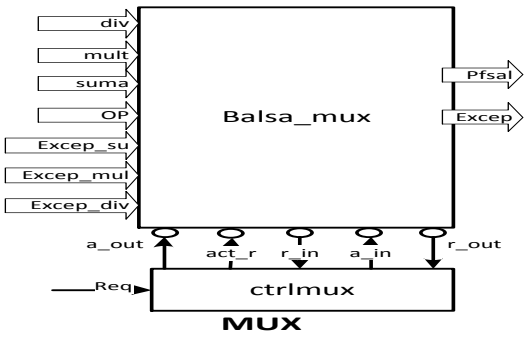

Figura 2.48. Diagrama de bloques del módulo mux.

Tabla 2.18. Entradas y salidas del módulo mux.

ENTRADAS	*TIPO*	*DESCRIPCIÓN*
Req	Bit	Entrada de requerimiento
Op	Bus 2 bits	Código de operación
Suma	Bus 32 bits	Resultado de la suma en formato IEEE 754
Mult	Bus 32 bits	Resultado de la multiplicación en formato IEEE 754
Divi	Bus 32 bits	Resultado de la división en formato IEEE 754
Nans	Bit	Operación invalida de la suma
Nanm	Bit	Operación invalida de la multiplicación
Nandi	Bit	Operación invalida de la división
Infs	Bit	Número infinito de la suma
Infm	Bit	Número infinito de la multiplicación
Infd	Bit	Número infinito de la división
Ceros	Bit	El resultado de la operación de la suma es cero
Cerom	Bit	El resultado de la operación de la multiplicación es cero
Cerod	Bit	El resultado de la operación de la división es cero
SALIDAS	*TIPO*	*DESCRIPCIÓN*
Salida	Bus 32 bits	Resultado
Nano	Bit	Operación invalida
Info	Bit	Número infinito
Ceout	Bit	cero
Act_ctrl2	Bit	Indica que el dato está listo

Tabla 2.19. Tabla de operación del módulo mux.

REQ	*OP[1:0]*	*RESULTADO EN IEEE 754 32 BITS*
0	00	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0	01	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0	10	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
0	11	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
1	00	Suma[31:0]
1	01	Multiplicacion[31:0]
1	10	División[31:0]
1	11	xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

El módulo *mux* está compuesto internamente por dos módulos llamados *Balsa_mux* y el *ctrlmux.* La Tabla 2.19 muestra el comportamiento de este módulo. Cuando la entrada ***ini = 0*** no entrega ningún dato a la salida, cuando ***ini = 1*** la salida entrega el dato correspondiente al código de operación ***OP[1:0]*** seleccionado por el usuario.

Módulo *Balsa_mux:* *e*ste módulo fue diseñado en *Balsa* y sintetizado en *Xilinx Ise 13.1*. La función principal es la de realizar la selección de los datos de una de las cuatro operaciones aritméticas de acuerdo al código de operación indicado por el usuario.

En la Figura 2.49 muestra el diagrama de flujo del funcionamiento del módulo *mux* que explica de una manera gráfica la información de la Tabla 2.19.

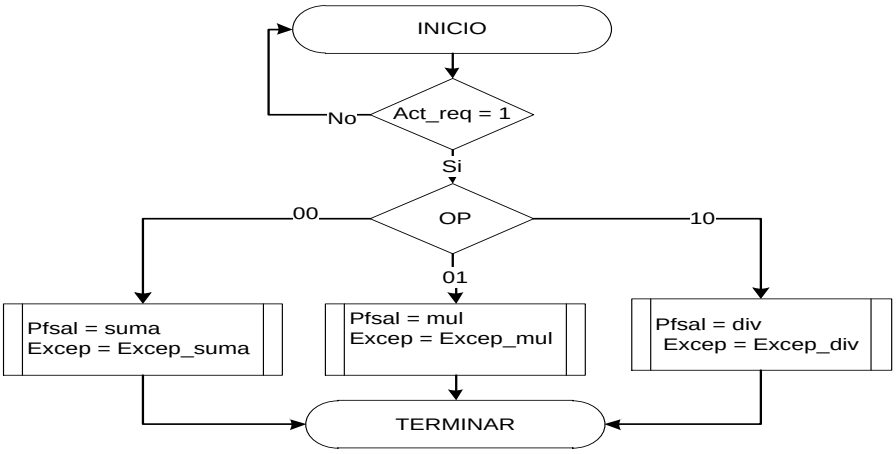

Figura 2.49. Algoritmo del módulo mux.

El módulo *ctrlmux* tiene la función de controlar el módulo *mux* para que pueda realizar las tareas especificadas en la Tabla 2.19.

En la Figura 2.50 se muestra el código del módulo *mux* realizado en *System Balsa 4.0* en el cual se especifica el funcionamiento de la información de la Tabla 2.19.

Files

deco.balsa | mux.balsa*

```
4  import [balsa.types.basic]
5  procedure mux ( input op:2 bits;
6                  input suma,mult,divi: 32 bits;
7                  input nans,nanm,nand : 1bits;
8                  input infs,infm,infd:1 bits;
9                  input ceros,cerom,cerod : 1 bits;
10                 output salida : 32 bits;
11                 output nano,info,ceout:1 bits   )is
12 variable inis,cerosu,ceromu,cerodi,nansu,nanmu,nandi,infsu,infmu,infdi : 1 bits
13 variable ops : 2 bits
14 variable sumas,multi,divis : 32 bits
15 begin
16 loop
17 ceros->cerosu||
18 cerom->ceromu||
19 cerod->cerodi||
20 nans ->nansu||
21 nanm ->nanmu||
22 nand ->nandi||
23 infs ->infsu||
24 infm ->infmu||
25 infd ->infdi||
26 suma ->sumas||
27 mult ->multi||
28 divi -> divis||
29 op -> ops;
30             case ops of 0 then
31                  salida <- sumas ||
32                  nano <- nansu  ||
33                  info <- infsu  ||
34                  ceout <- cerosu
35             |1 then
36
37                  salida <- multi ||
38                  nano <- nanmu  ||
39                  info <- infmu  ||
40                  ceout <- ceromu
41             |2 then
42                  salida <- divis ||
43                  nano <- nandi  ||
44                  info <- infdi  ||
45                  ceout <- cerodi
46             |3 then
47                  salida <- 0 ||
48                  nano  <- 0 ||
49                  info  <- 0  ||
50                  ceout  <- 0
```

Figura 2.50. Código en balsa para el módulo mux.

Cuando el módulo *mux* está activado y la entrada ***op_data [1:0] = 00*** el módulo seleccionado es el de la suma como se muestra en la información de la Tabla 2.19.

Síntesis del diseño asíncrono del módulo *mux:* con el archivo *Verilog* generado por balsa se sintetizo el módulo *Balsa_mux* con la herramienta síncrona *Xilinx Ise 13.1.* Luego se creó el módulo *mux* el cual tiene en su estructura el controlador *ctrlmux* y el módulo *Balsa_mux* como se puede mirar en la Figura 2.51.

.

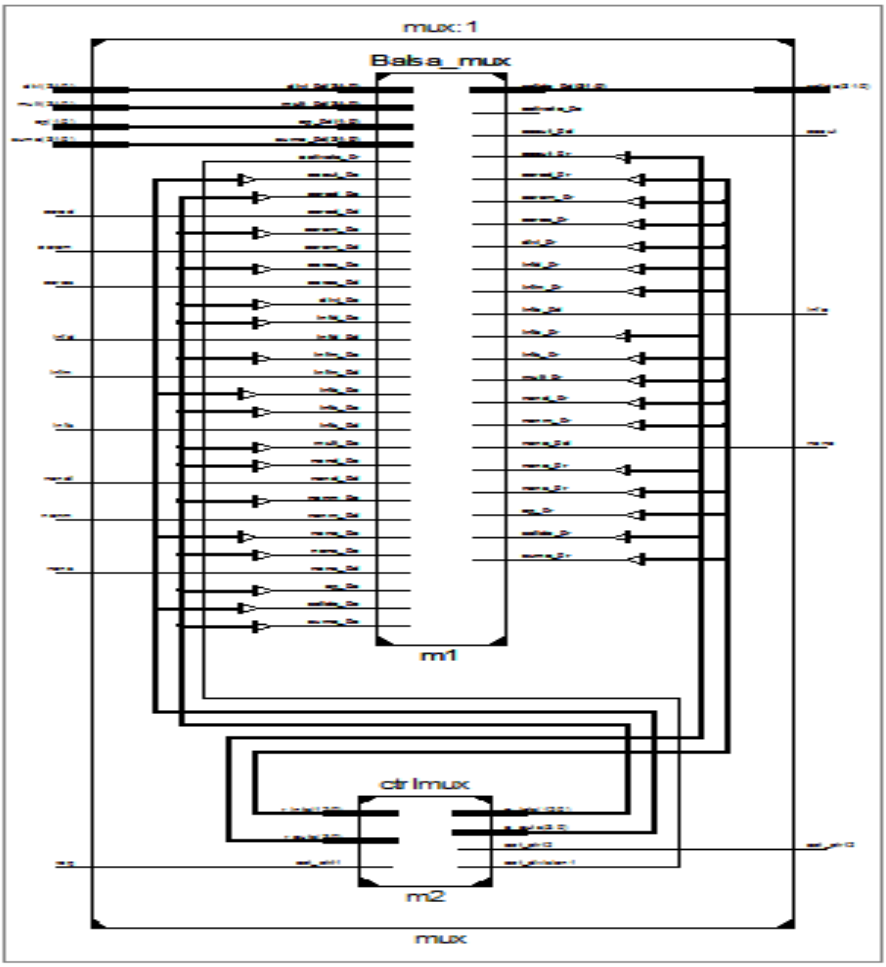

Figura 2.51. Arquitectura del módulo mux con el módulo Balsa_mux y ctrlmux.

2.7 MODULO AFPU.

Después de haber sintetizados los módulos asíncronos para las operaciones de suma (SPFA), multiplicación (MPFA), división (DPFA) para punto flotante de 32 bits y los módulos deco, *seleor* y *mux* se realizó la integración de estos módulos en una unidad llama AFPU como se muestra en el diagrama de bloque de la Figura 2.1. En la Tabla 2.20 se visualiza las entradas y salidas de este módulo.

Tabla 2.20. Descripción de las entradas y salidas del módulo AFPU.

ENTRADAS	***TIPO***	***DESCRIPCIÓN***
Req	Bit	Entrada de requerimiento
Op	Bus 2 bits	Código de operación
Mana	Bus 32 bits	Mantisa A
Manb	Bus 32 bits	Mantisa B
Expa	Bus 32 bits	Exponente A
Expb	Bit	Exponente B
Sai	Bit	Signo A
Sbi	Bit	Signo B
SALIDAS	***TIPO***	***DESCRIPCIÓN***
Pfsal	Bus 32 bits	Resultado en punto flotante
Nansal	Bit	Operación invalida
Infsal	Bit	Número infinito
Cerosal	Bit	cero
Ctrnext	Bit	Indica que el dato está listo

La estructura interna con todos sus componentes interconectados se visualiza en la Figura 2.52.

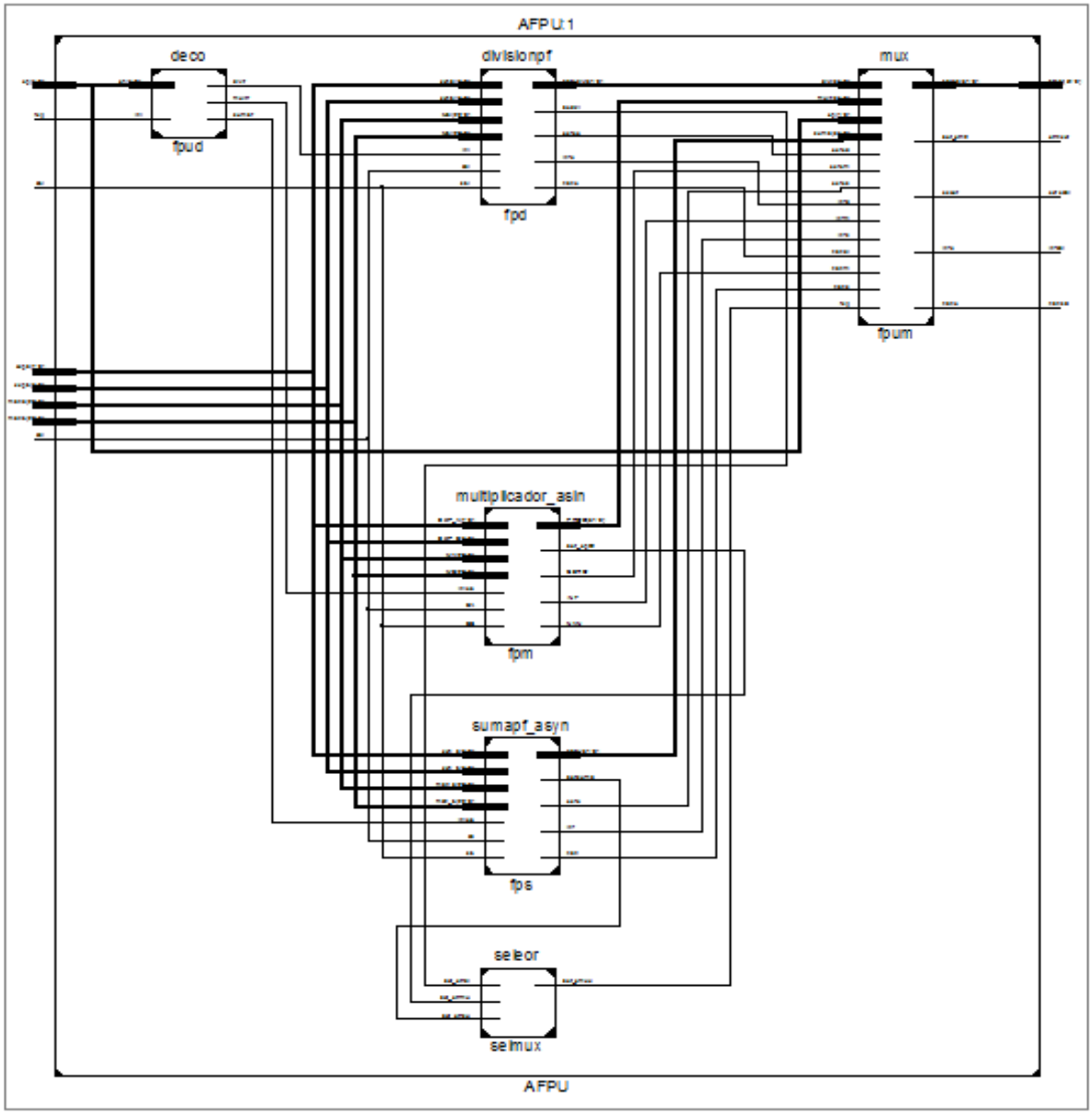

Figura 2.52. Arquitectura interna del módulo AFPU.

El resumen de la síntesis del módulo AFPU de punto flotante asíncrono se muestra en la Tabla 2.21.

Tabla 2.21. Resumen de la síntesis del módulo AFPU.

Unidad	*No. De elementos lógicos*	*No. De registros*	*No. De latches(IOB)*	*No. De flip flop (IOB)*
AFPU	17128	3839	0	4

Verificación del circuito asíncrono AFPU: para probar el módulo AFPU para punto flotante asíncrono se utilizó los siguientes datos: el operando A= 12,546 y el operando B = -0,465. Estos datos son pasados a binario y luego al formato IEEE 754 para cada uno de los operando de entrada.

Los exponentes ***A*** y ***B*** son: ***Expai[7:0] =10000010, expbi[7:0] = 01111101.***

Las mantisas ***A*** y ***B*** son: ***Mai[22:0] = 10010001011110001101010.***

Mbi[22:0] = 11011100001010001111011.

Los signo ***A*** y ***B*** son**: *sai = 0, sbi = 1,*** como se muestra en la Figura 2.53.

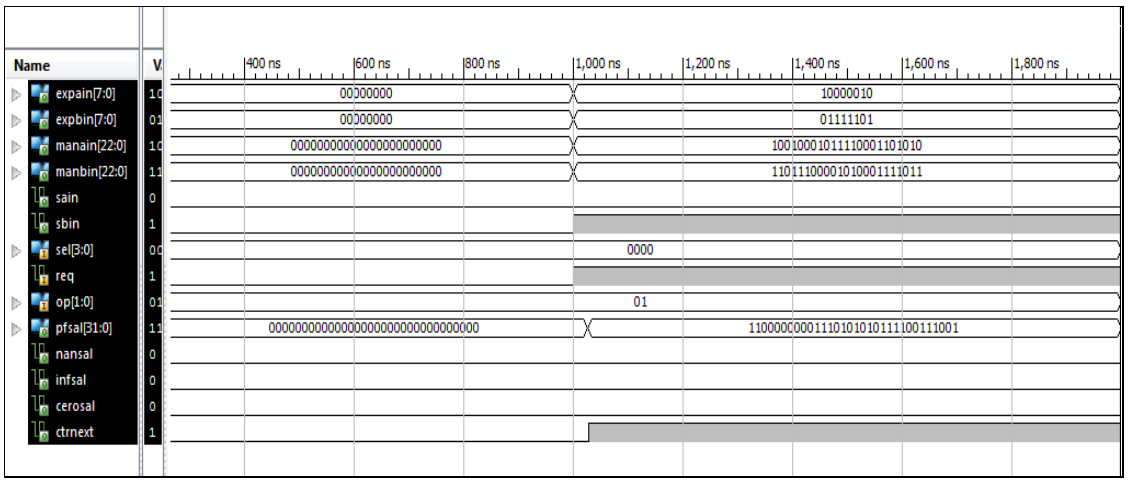

Figura 2.53. Simulación del coprocesador Asíncrono AFPU (multiplicación).

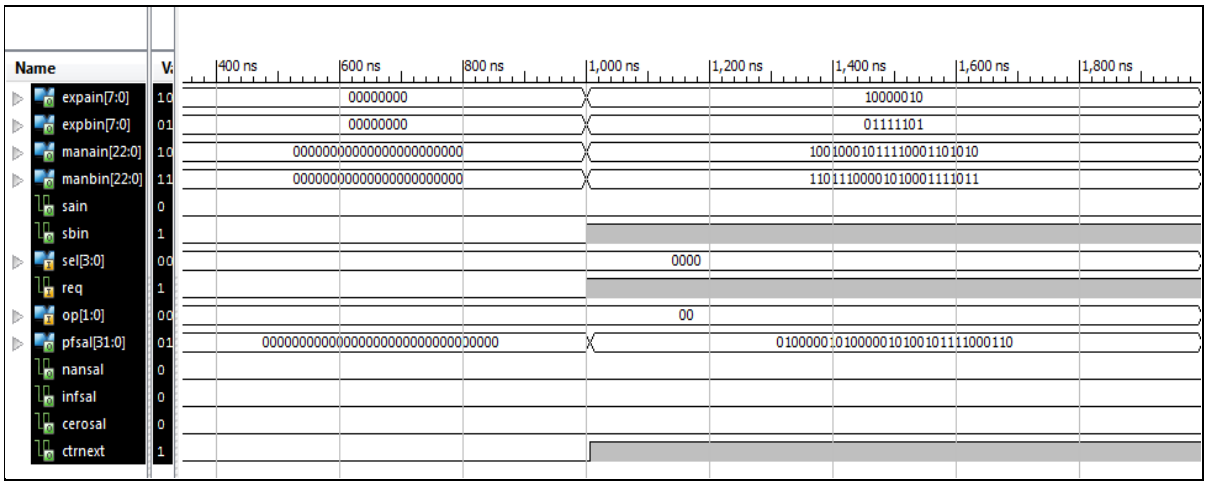

Figura 2.54. Simulación del coprocesador Asíncrono AFPU (suma).

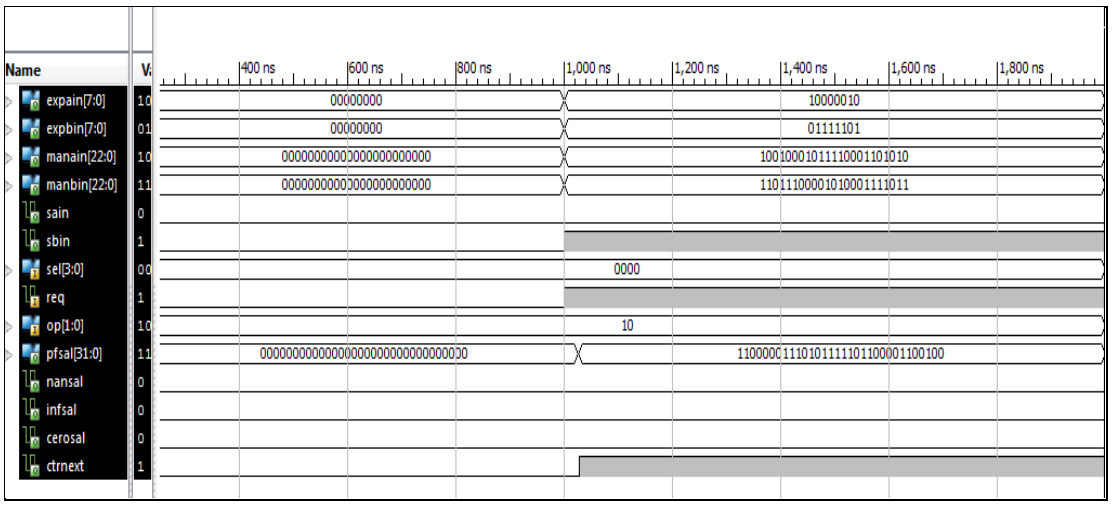

Figura 2.55. Simulación del coprocesador Asíncrono AFPU (división).

Los tiempos que aparecen como tiempo de respuesta de las simulaciones del módulo AFPU no corresponden al tiempo de respuesta debido a que a los módulos de control se les coloco un retardo de 1ns para poder visualizar los resultados de forma correcta en el simulador Isim

El resultado obtenido en la Figura 2.53 corresponde a la multiplicación de los operando ***A*** y ***B*** de 32 bits cuyo valor es:

PROD[31:0] = 11000000001110101010111100111001 el tiempo de respuesta es ***t = 26 ns***

El resultado de la suma se muestra en la Figura 2.54. El resultado obtenido es:

dato[31:0] = 01000001010000010100101111000110 el tiempo de respuesta es ***t = 5 ns.***

El resultado de la división de las dos mantisas en punto flotante de 32 bits está en la Figura 2.55.

datodiv[31:0] = 11000001110101111101100001100100 el tiempo de respuesta es ***t = 29 ns.***

La estimación de potencia del módulo AFPU se visualiza en la Figura 2.56. Esta estimación de potencia se realizó con la herramienta *PlanAhead* de *Xilinx Ise*.

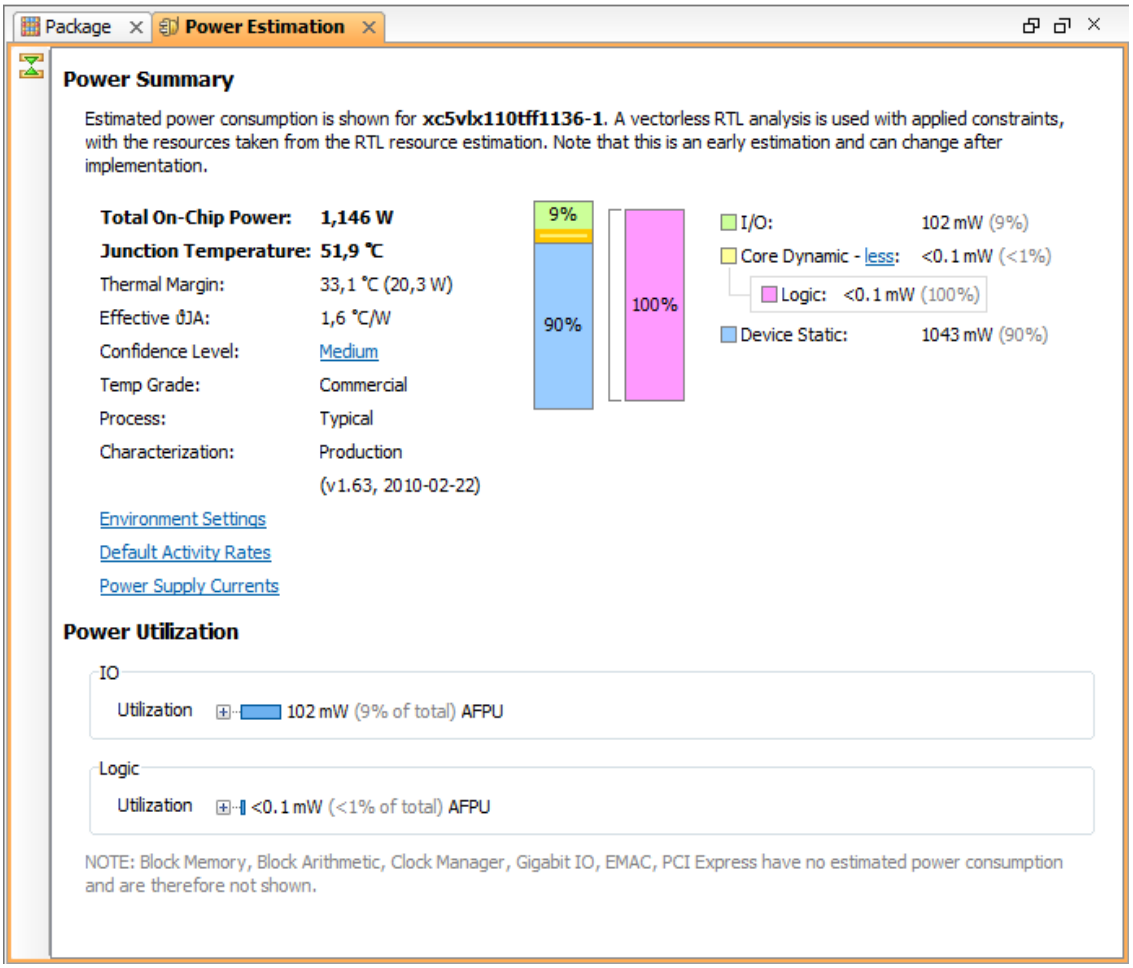

Figura 2.56. Estimación de potencia del módulo AFPU utilizando PlanAhead 13.1 de Xilinx Ise.

El consumo de potencia estimada total del módulo AFPU es 1146 mW. Distribuidos de la siguiente manera:

- Las entradas y salidas consumen 102 mW equivalente al 9% del porcentaje global
- La dinámica del núcleo (Lógica) consume menos de 0.1 mW equivalente al 100% de menos del 1%
- El dispositivo en estado estático consume 1043 mW equivalente al 90% del porcentaje global.

2.8 UNIDAD DE PUNTO FLOTANTE SÍNCRONA (SFPU).

Esta unidad se diseñó utilizando el mismo algoritmo del coprocesador asíncrono. Con el objetivo de poder comparar las características principales que tienen los dos tipos de circuitos digitales hasta a hora diseñados. En la Figura 2.57 se visualiza el diagrama de bloque del coprocesador síncrono con todos los componentes.

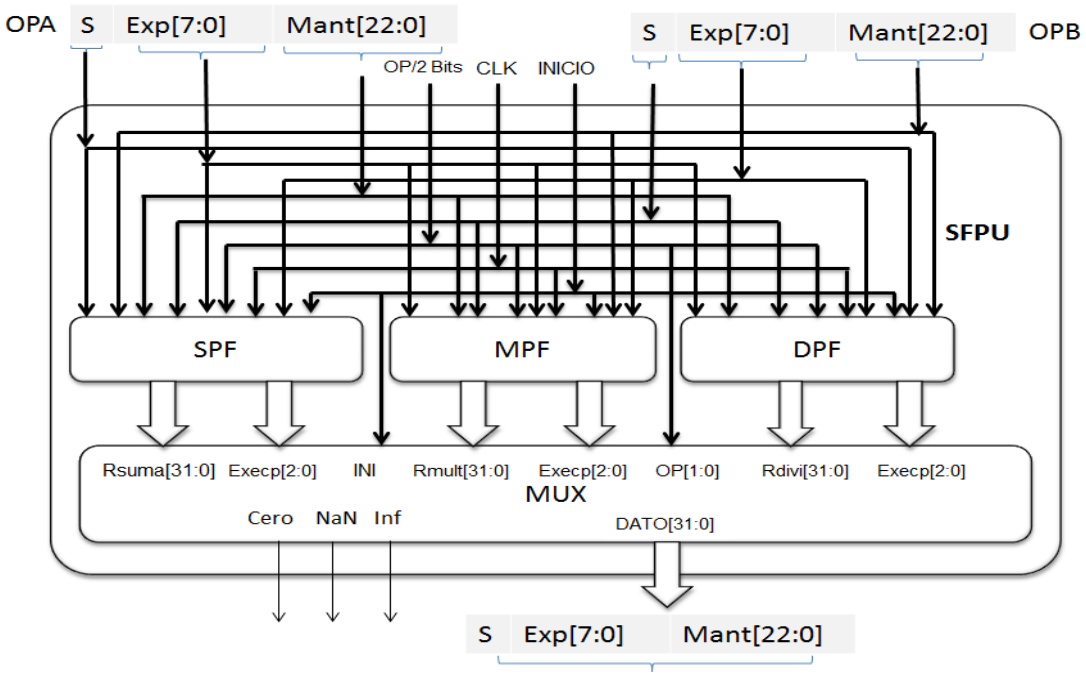

Figura 2.57. Diagrama de bloques de coprocesador síncrono para punto flotante de precisión simple.

El coprocesador síncrono es básicamente una SFPU que realiza operaciones aritméticas en punto flotante de dos operando en simple precisión, sincronizado con una señal de reloj.

Se diseñó utilizando el lenguaje de descripción de hardware VHDL en el entorno de *Xilinx Desing Ise 13.1.* Este modulo está conformado por cuatro unidades

- SPF.
- MPF.
- DPF.
- MUX.

En la siguiente Tabla 2.22 se especifican las entradas y salidas del módulo SFPU las cuales son las misma utilizadas para el módulo AFPU.

Tabla 2.22. Entradas y salidas del módulo SFPU.

ENTRADAS	*TIPO*	*DESCRIPCIÓN*
EXPA	Bus de 8 bits	Exponente del operando A
EXPB	Bus de 8 bits	Exponente del operando A
MANA	Bus de 23 bits	Mantisa del operando A
MANB	Bus de 23 bits	Mantisa del operando B
OPx	Bus de 2 bits	Código de operación
CLK	bit	Señal de reloj
INI	bit	Señal de inicio
SIGA	bit	Signo del operando A
SIGB	bit	Signo del operando B
SALIDAS	*TIPO*	*DESCRIPCIÓN*
Exep	Bus de 3 bits	Excepciones
Pfsin	Bus de 32 bits	resultado

La organización interna con todos sus componentes se visualizan en la Figura 2.58 donde se puede observar los módulos ***SUMADOR_PF***, ***PFPRODUCTO***, ***DIVISINPF*** y el ***MULTIPLEXOR*** de datos.

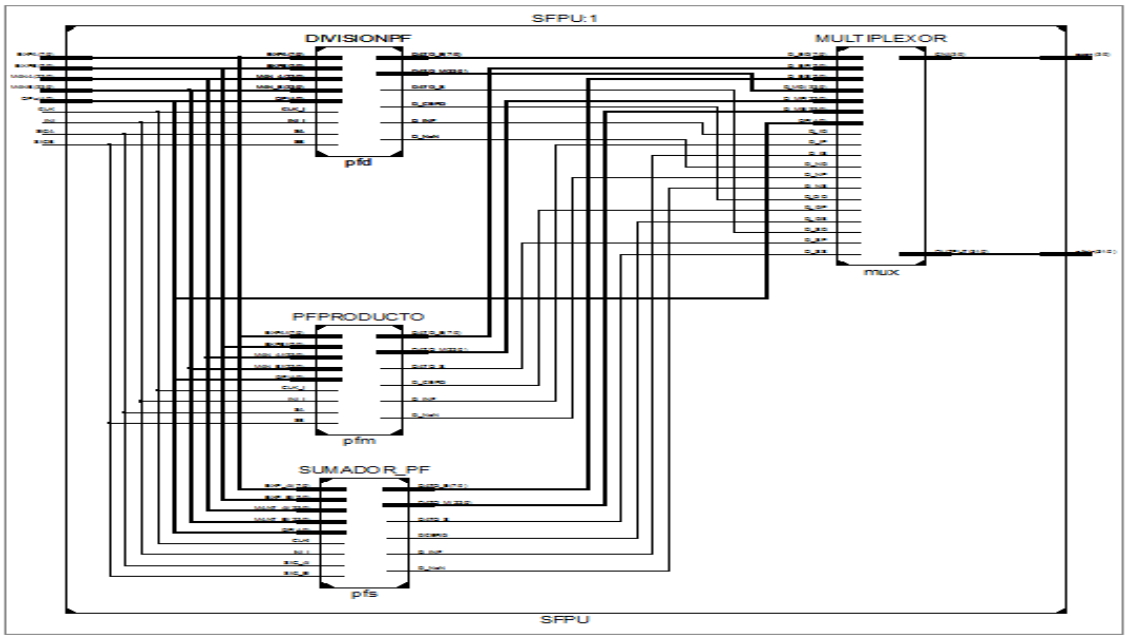

Figura 2.58. Arquitectura interna del coprocesador síncrono SFPU.

La señal ***clk*** se encarga de sincronizar todas las funciones del módulo SFPU cuando la señal de inicio está activada en ***1***.

El resumen de la síntesis del módulo *SFPU* para punto flotante asíncrono se muestra en la Tabla 2.23. Los datos más representativos, la cantidad de elementos lógicos y el número de registros utilizados.

Tabla 2.23. Resumen de la síntesis del módulo SFPU.

Unidad	*No. De elementos lógicos*	*No. De registros*	*No. De latches(IOB)*	*No. De flip flop (IOB)*
SFPU	3718	785	0	107

La Tabla 2.24 muestra el resumen de compilación y el número de ciclos de reloj de los módulos síncronos de la ***suma***, ***multiplicación*** y ***división*** que se obtuvieron al realizar la compilación y la simulación.

Tabla 2.24. Resumen de la compilación y simulación de cada uno de los módulos del coprocesador síncrono SFPU.

Unidad	*No. De ciclos de reloj*	*No. De elementos lógicos*	*No. De registros*
Suma/Resta	10.5	3718	785
Multiplicación	6.5	3023	294
División	6.5	3718	785

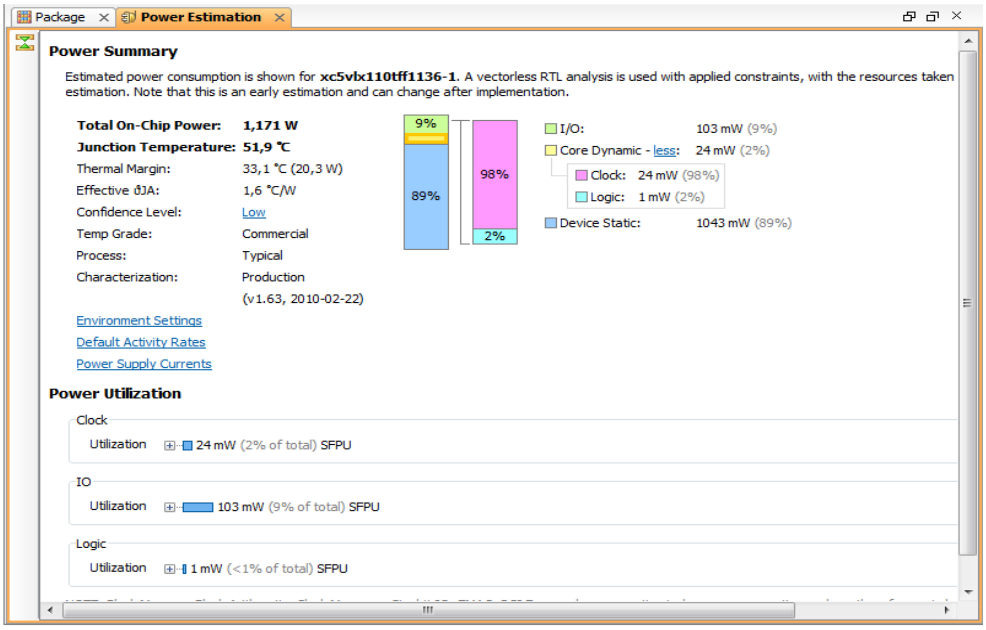

Figura 2.59. Estimación de potencia del módulo coprocesador síncrono SFPU realizado con PlanAhead 13.1 de Xilinx Ise.

En la Figura 2.59 está consignada la estimación del consumo de potencia del módulo *SFPU* de 1171 mW. Distribuidos de la siguiente manera.

- Las entradas y salidas consumen 103 mW equivalente al 9% del porcentaje global
- La dinámica del núcleo consume 24 mW distribuido de la siguiente manera:
  - La señal de reloj consume 24 mW equivalente al 98% del100% del porcentaje global del 2% para la dinámica del núcleo. La dinámica está regida por la señal de reloj.
  - Los elementos lógicos consumen 1 mW equivalente al 2% del 100% del porcentaje global del 2% para la dinámica del núcleo
- El dispositivo en estado estático consume 1043 mW equivalente al 90% del porcentaje global.

3 DATOS PRÁCTICOS Y ANÁLISIS DE RESULTADOS.

Los resultados que se muestran en las siguientes gráficas son tomadas del reporte de compilación que es generado por la herramienta *Xilinx Ise 13.1* al realizar la síntesis de cada módulo del coprocesador síncrono.

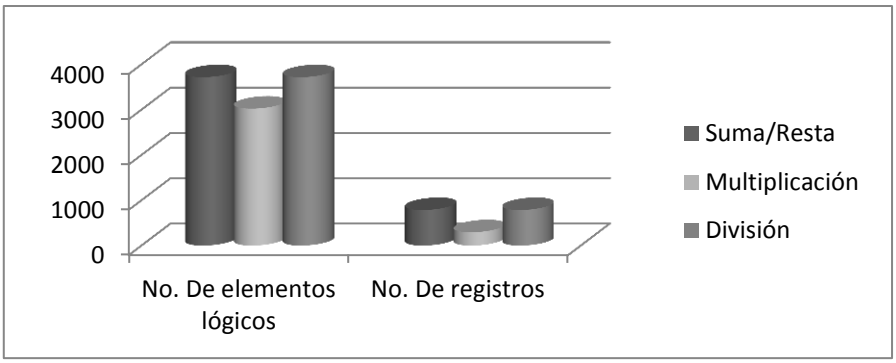

Figura 3.1. Reporte de compilación del coprocesador síncrono SFPU.

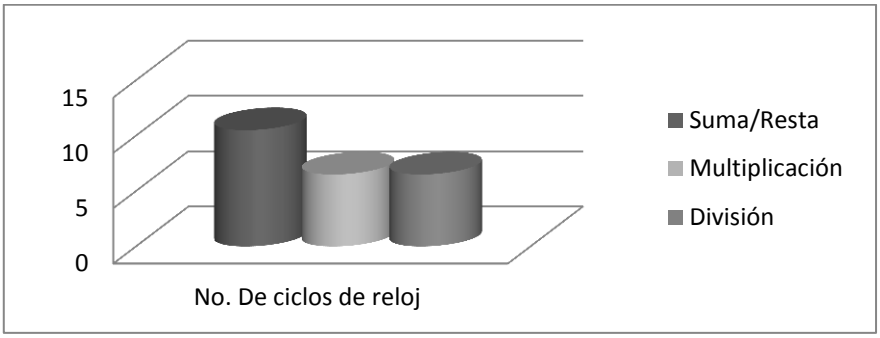

Figura 3.2. Ciclo de reloj de las unidades del coprocesador síncrono

En la gráfica de la Figura 3.1 y Figura 3.2 se observa que la unidad de suma síncrona se demora más tiempo en entregar la respuesta a la salida con 10,5 ciclos de reloj. También tiene la mayor cantidad de elementos lógicos y de registro junto con la unidad de división con 3718 y 785 respectivamente. La unidad de multiplicación con la unidad de división utilizan 6.5 ciclos de la señal de reloj para entregar los datos validos a la salida. En la cantidad de elementos lógicos y número de registros la multiplicación utiliza menos con solo 3023 y 294 respectivamente.

En la gráfica de la Figura 3.3 se observa que la división asíncrona utiliza el mayor número de elementos lógicos con 17373 mientras que la multiplicación le sigue con 8215, la unidad Asíncrona que menos elementos lógicos utiliza es la suma con 1736, esto es debido a que los algoritmos de multiplicación y división para punto flotante son más extensos porque hay que realizar algoritmos de la multiplicación y división de mantisas por aparte.

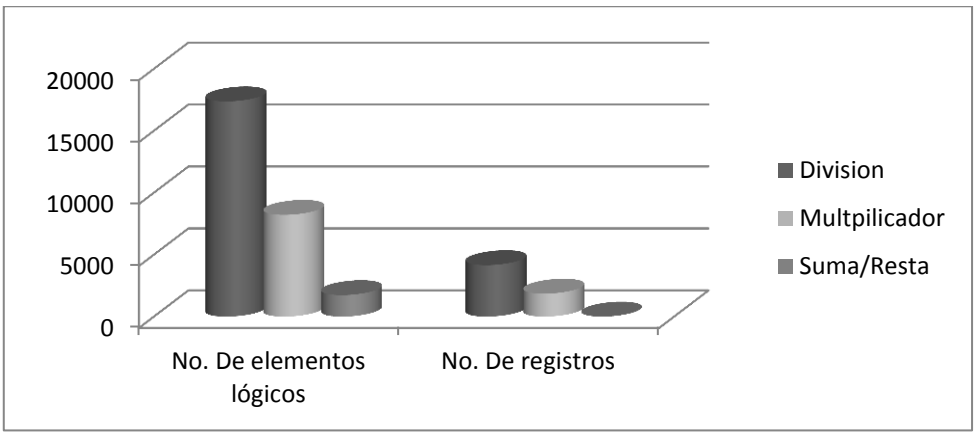

Figura 3.3. Reporte de compilación del coprocesador Asíncrono AFPU.

La unidad que más número de registro utiliza es la división con 4155, le sigue la multiplicación con 1853 y la suma no utiliza registros.

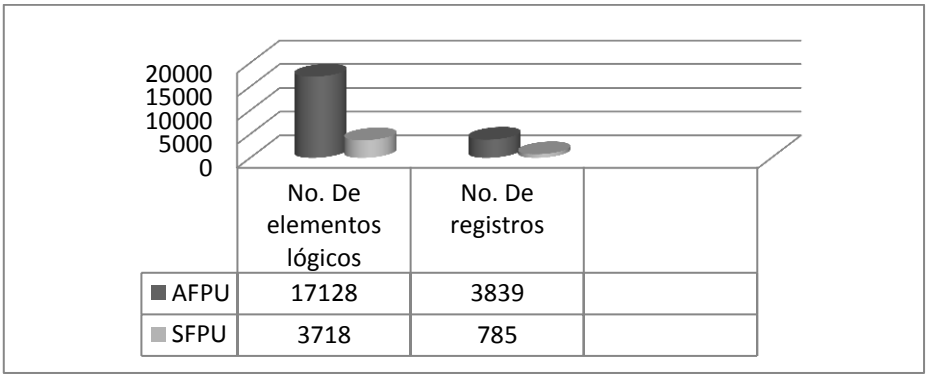

Figura 3.4. Reporte de compilación delos coprocesadores Asíncrono y Síncrono (AFPU, SFPU).

En la gráfica de la Figura 3.4 se observa que el coprocesador Asíncrono ***AFPU*** utiliza mayor cantidad de elementos lógicos con 17128 mientras que el coprocesador síncrono (***SFPU***) solo utiliza 3718.

En la cantidad de registros el coprocesador asíncrono también utiliza la mayor cantidad con 3839 mientras que el coprocesador síncrono utiliza 785.

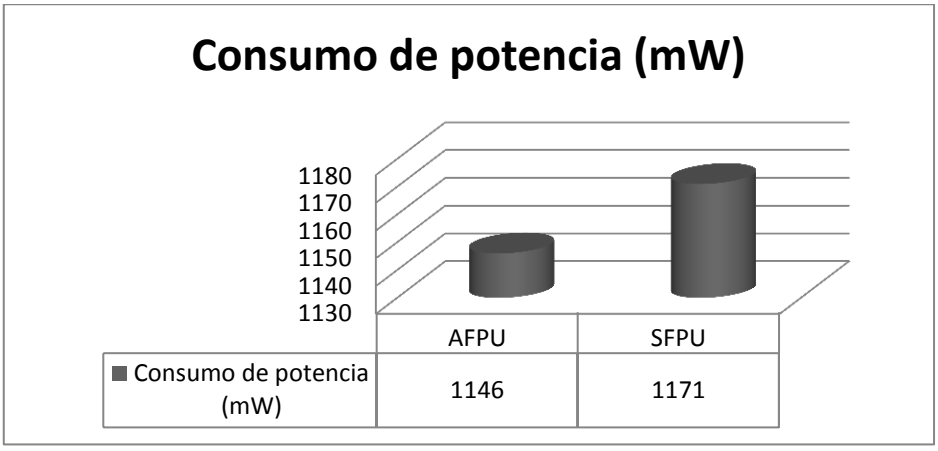

Figura 3.5. Reporte de la estimación de potencia de los coprocesadores Asíncrono y Síncrono (AFPU, SFPU).

[2] la validación de potencia para los circuitos VLSI se basa en la rapidez de consumo de potencia dinámica para un circuito CMOS está gobernada por la siguiente ecuación

$p = C.V^2.E.F$ Donde

P es la potencia medida en mW, C es la capacitancia en faradios, V es el voltaje medido en vol, E es la variación de la conmutación (promedio no de ciclo de transición) y F es la frecuencia en Hz.

La Figura 3.5 presenta una estimación de potencia de los coprocesadores síncrono y asíncrono se realizó con la herramienta PlanAhead de Xilinx Ise. La cual entrega un consumo de potencia de 1171 mW para el coprocesador síncrono mientras que para el coprocesador Asíncrono utiliza 1146 mW. La diferencia de consumo de potencia entre los dos coprocesadores es de 25 mW.

4 RESULTADOS PRÁCTICOS.

En la Figura 4.1 se observa los resultados obtenidos de la división, suma y multiplicación del coprocesador asíncrono funcionando en la *FPGA Xilinx Virtex IV XC5VLX110T-1FF1136 [9]*, utilizando la herramienta de verificación de *hardware [8],* estos resultados son los mismos que se obtuvieron al realizar la simulación *con Isim de Xilinx Ise 13.1.*

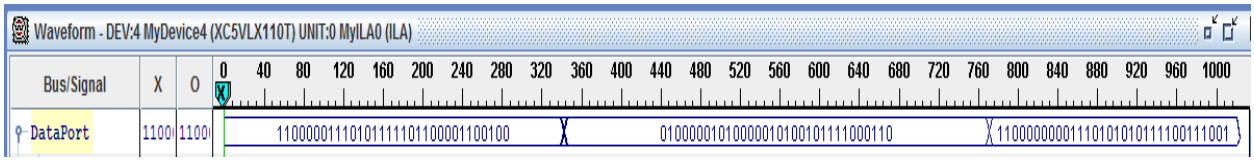

Figura 4.1. Resultado obtenido con Chipscope Pro.

Las Figura 4.2, Figura 4.3 y Figura 4.4 muestran los resultados obtenidos con el coprocesador síncrono (*SFPU*). Los cuales corroboran el buen funcionamiento de este circuito en la *FPGA Xilinx Virtex IV XC5VLX110T-1FF1136.*

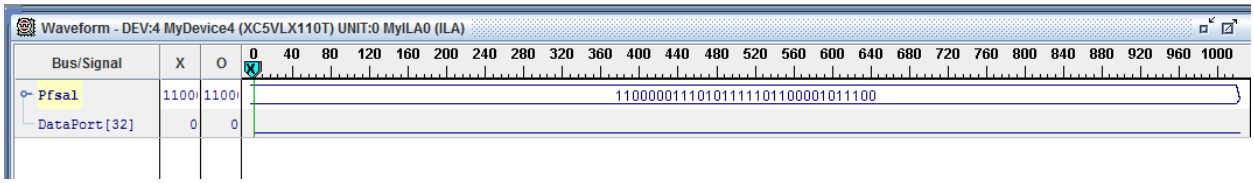

Figura 4.2. Resultado de la división del coprocesador síncrono obtenido con Chipscope Pro.

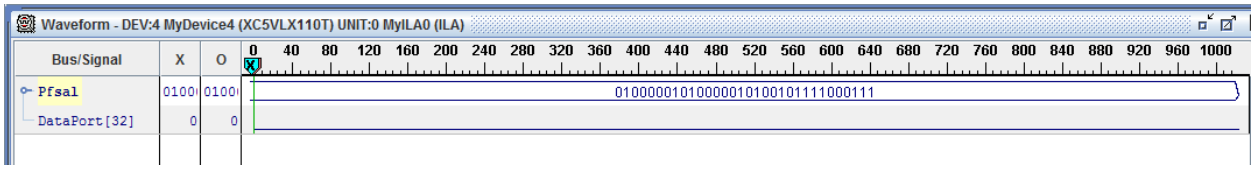

Figura 4.3. Resultado de la suma con el coprocesador síncrono obtenido con Chipscope Pro.

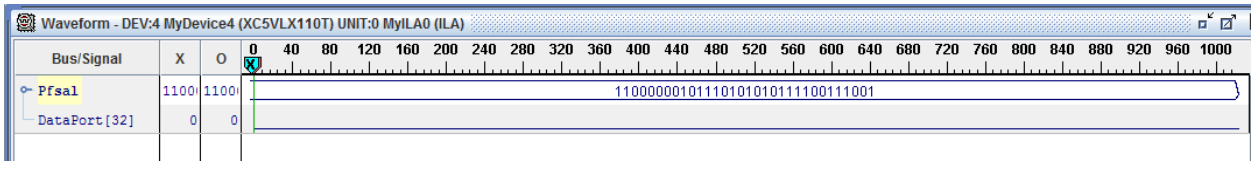

Figura 4.4. Resultado de la multiplicación con el coprocesador síncrono obtenido con Chipscope Pro.

En la Tabla 4.1 se muestra los datos de corriente y voltaje para determinar la potencia de los dos coprocesadores síncrono y asíncrono tomados de la *FPGA Xilinx Virtex IV XC5VLX110T-1FF1136* en funcionamiento.

Tabla 4.1. Consumo de potencia con requerimiento (req = 0 y req = 1) de los coprocesadores síncrono (SFPU) y asíncrono (AFPU).

			Consumo con req=0		*Consumo con req=1*	
Unidad	***Voltaje (V)***	***Corriente S.P. (mA)***	***Corriente (mA)***	***Consumo de Potencia (mW)***	***Corriente (mA)***	***Consumo de Potencia (mW)***
AFPU	4.97	0.120	0.124	0.616	0.143	0.7107
SFPU	4.97	0.120	0.140	0.6958	0.140	0.6958

En la Tabla 4.1 se observa que el coprocesador síncrono tiene el mismo consumo de potencia si el requerimiento de dato este en 1 ó 0 debido a que el reloj sigue operando.

Si el requerimiento del coprocesador asíncrono está en cero la potencia consumida es 0.616 mW. Cuando se activa el requerimiento del coprocesador asíncrono experimenta un consumo pico de potencia de 0.7107 mW. Cuando el requerimiento es desactivado debido a que ya termino la operación requerida el consumo retorna a 0.616mW. Esto ocasiona que haya un ahorro considerable en el consumo de potencia con relación al coprocesador síncrono.

5 CONCLUSIONES

5.1. Se ha diseñado e implementado en una FPGA un sistema digital asincrono que responde a las características mas importante del estándar IEEE 754 en simple precisión utilizando tanto herramientas de diseño síncrono como asíncrono.

5.2. Se ha descripto, mediante una herramienta síncrona, el estándar IEEE 754 de simple precisión utilizando la herramienta de sintesis Xilinx Ise 13.1. Esta herramienta permitió simular el comportamiento del coprocesador, conocer los ciclos de operación, el área de los dispositivos utilizados y el consumo de potencia, tanto para la implementacion sincrona como asincrona del sistema que implementa el estandar IEEE 754.

5.3. Se logró la integración de las herramientas para el diseño asíncrono y síncrono para la implementación del estándar en hardware real (FPGA) las descripciones en Balsa se deben probar en Xilinx Ise, ya que hay instrucciones que no son compactibles con Xilinx Ise 13.1.

5.4. De acuerdo con los resultados dados se puede determinar que el coprocesador Asíncrono utiliza mayor área de dispositivos para realizar su función, la diferencia en número de elementos lógicos y de registro es bastante grande como se observó en la Figura 3.2.

5.5. El coprocesador asíncrono consume menos potencia que el coprocesador síncrono como se observó en la gráfica de potencia y la tabla 4.1 debido a que evita el consumo de potencia dinámica ocasionado por la señal de reloj.

5.6. La operación de suma es la que más retardo de reloj utiliza debido a que el algoritmo para suma en [3] requiere mayor cantidad de módulos con respecto a la operación de multiplicación y división.

5.7. Para la implementación de los circuitos asíncronos realizados con *Balsa System* en la herramienta de *xilinx Ise* se debe tener especial cuidado con las secuencias entre operaciones ya que *Xilinx Ise* utiliza concurrencia y solo se puede hacer secuencia dentro de los procesos.

5.8. El software *Isim* presenta muchos inconvenientes para simular el comportamiento de estos circuitos es recomendable verificar el comportamiento en un dispositivo reprogramable como las *FPGAs* para determinar los comportamientos de los diseños.

5.9. Debido a que no hay reglas establecidas para el diseño de circuitos asíncronos, se debe tener cuidado con los retardos de propagación cuando se trabaja con muchos módulos ya que este puede ocasionar que los datos en la salidas sean incorrectos.

5.10. No todos los circuitos que funcionan en Balsa System son sintetizables en Xilinx Ise, por tal motivo, el diseñador debe replantear los diseños de tal forma que se puedan implementar con esta herramienta. Esto es un gran desafío donde hay que tener mucha paciencia e ingenio.

5.11. Esta primera experiencia con el diseño del coprocesador asíncrono ha sido muy enriquecedora y es una puerta que se abre para la elaboración de circuitos más eficientes en cuanto a consumo de energía y rendimiento.

6. TRABAJO FUTURO

6.1. Implementación en hardware asíncrono el estándar IEEE 754 completo (que incluya operaciones como logaritmos, circuitos de exponenciación, operaciones con números complejos) utlizando algoritmos aritméticos rápidos y optimizados para los diferentes formatos descritos en el estándar.

6.2. Implementar el coprocesador asíncrono de forma genérica, de tal manera que se pueda seleccionar cualquiera de los formatos de 32, 64 o 128 bits.

REFERENCIAS.

[1]. *Doug Edwards, Bardsley Andrew, Janin Lilian, Plana Luis & Toms Will. Balsa:A Tutorial Guide, 2006, University of Manchester*

[2].Hardeep Singh. Power Aware Design and Implementation of 8-bit Asynchronous Arithmetic and Logic Unit, 2009, Thapar University, Patiala Punjab, India.

[3]. IEEE Computer Society. IEEE Standard for Floating-Point Arithmetic IEEE 754 -2008. Disponible en:http://ieeexplore.ieee.org.bd.univalle.edu.co/stamp/stamp.jsp?tp=&arnumber=4610935 (consulta: martes 15/02/2011).

[4]. Jens Sparso. Asynchronous Circuit Design A Tutorial, 2006. Technical University of Denmark.

[5]. Jidan Al-Eryani, unidad aritmética de punto flotante, 2006. Disponible en:http://www.opencores.org (consulta: miércoles 9/03/2011). jidan@gmx.net

[6]. Lumbiarres López Rubén. Co-diseño hardware-software de una unidad en coma flotante para microprocesador de 32 bits, 2003, universitat politécnica de Catalunya.

[7]. *Verilog-1995 Quick Reference Guide by Stuart Sutherland of Sutherland HDL, Inc., Portland, Oregon, USA at.* Disponible en internet: *http://www.sutherland-hdl.com/online_verilog_ref_guide/vlog_ref_top.html.*

[8]. Xilinx® hardware devices. ChipScope Pro 13.1 Software and Cores,UG029 (v13.1) March 1, 2011 disponible en internet:*www.xilinx.com*.

[9]. Xilinx® hardware devices. ML505/ML506/ML507Getting Started Tutorial, UG348 (v3.0.3) June 18, 2009 disponible en internet: *www.xilinx.com.*

[10]. Xilinx® ISE® Design Suite. ISE Design Suite Software Manuals -PDF Collection disponible en internet: http://www.xilinx.com/support/download/index.htm.

[11]. Nieto Londoño Rubén Darío. Diseño e Implementación De Un Cripto-procesador Asíncrono De Bajo Consumo Basado En El Algoritmo De Rijndael, 2009. Universidad Del Valle.

[12]. Davis, A. Nowick S. M. "An Introduction to Asynchronous Circuit Design". Computer Science University of Utah, Salt Lake City, UT 84112; Computer Science, Columbia University, New York, NY 10027. September 19, 1997.

[13].Liu, J. Ph.D. Thesis: "Arithmetic and Control Components for an Asynchronous System". University of Manchester, Faculty of Science and Engineering, Department of Computer Science, 1997.