1. Resumen

En el presente se crea una ALU con instrucciones específicas para el micro-procesador que se encuentra en desarrollo en la escuela de ingeniería en electrónica. Dicho proceso cuenta con la implementación de la misma desde la descripción en Verilog, pasando por la síntesis de la misma, simulación post-síntesis, implementación física y simulación post-implementación, lo anterior por medio del uso de las herramientas de Synopsys.

2. Introducción

La ALU es una de las partes más importante de los microprocesadores, dado que es la encargada de ejecutar las operaciones aritméticas y lógicas que, en términos básicos, es la función principal de éstos dispositivos; de ahí la importancia de desarrollar una unidad Aritmética-Lógica con variedad de funciones y banderas que permitan interpretar de mejor manera el resultado de dichas operaciones.

Para la implementación de la ALU se va a tomar en cuenta las especificaciones de [1] las cuales detallan las funciones necesarias que se desean implementar en una primera iteración de esta unidad; además se van a tomar en cuenta las banderas implementadas en la versión de ALU de [2] y otros aspectos relevantes de esta fuente.

3. Objetivos

- Generar una ALU en HDL que realice las funciones especificadas.
- Ejecutar el proceso de síntesis de la ALU en la tecnología de IMB 8RF de 130 nm.
- Realizar las simulaciones post-síntesis y post-layout que permitan la verificación y caracterización de la ALU.

4. Creación de la ALU en HDL

La etapa inicial del proyecto consiste en realizar la descripción en alto nivel, es decir, por medio del uso de lenguaje Verilog en nuestro caso. Para realizar dicho diseño en alto nivel se deben tener las consideraciones de la micro-arquitectura para la cual se está realizando esta unidad, las cuales se encuentran mostradas en la tabla 1; además de las mencionadas en esta tabla se generan otras banderas llamadas overflow, carry y negative, cuya lógica de funcionamiento se toma de [2] específicamente del capítulo 5 página 250.

Cabe destacar que en la tabla 1 cuando las entradas se encuentran en negrita significa que representan números sin signo. Con todas las especificaciones mencionadas se comienza con la descripción de la unidad, que va a ser meramente un circuito lógico combinacional; para lo anterior se hace uso de Vivado como IDE, debido a que nos permite detectar errores de sintaxis y realizar una síntesis inicial para que antes de realizar la síntesis en la tecnología deseada se hayan eliminado la mayor cantidad de errores posible.

Tabla 1: Especificaciones de la ALU

Alu_Cntrl	Función ID	OUT	Zero
0000	EQU	0	A == B ? 1 : 0
0001	LESS_THAN	0	A ¡B ? 1 : 0
0010	LESS_THAN UNSIGNED	0	**A ¡B** ? 1 : 0
0011	GREATER_THAN	0	A ¿B ? 1 : 0
0100	GREATER_THAN UNSIGNED	0	**A ¿B** ? 1 : 0
0101	ADD	A + B	0
0110	ADD UNSIGNED	**A + B**	0
0111	SUB UNSIGNED	**A - B**	0
1000	SHIFT LEFT LOGICAL	A ≪ B extendido con 0's)	0
1001	SHIFT RIGHT LOGICAL	A ≫ B extendido con 0's)	0
1010	SHIFT RIGHT ARITMETIC	A ≫ B extendido con signo)	0
1011	BITWISE OR	A or B	0
1100	BITWISE XOR	A xor B	0
1101	BITWISE AND	A and B	0

En la figura 1 se puede apreciar el diagrama de bloques de la ALU, en la cual se muestra la forma en que se implementan las banderas y las demás funciones que se mencionan tanto en [2] como en la tabla 1.

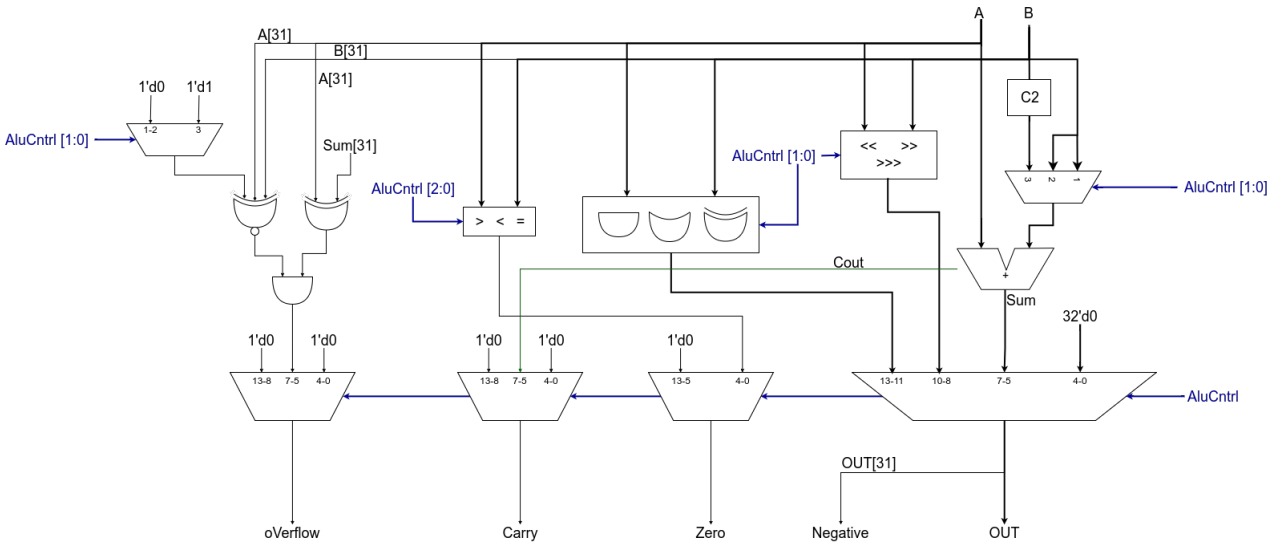
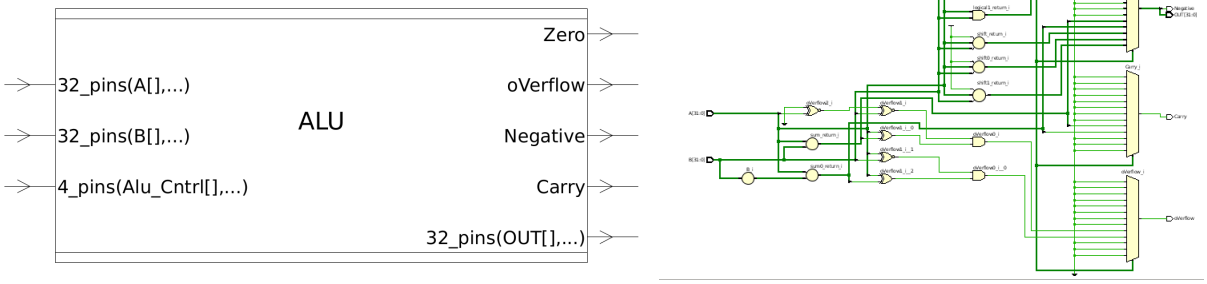


Figura 1: Diagrama de bloques de la ALU

En la figura 2 se pueden apreciar el primer y segundo nivel de la ALU, en donde el segundo nivel es obtenido de Vivado por lo cual hace uso de algunas celdas que no van a estar disponibles al momento de realizar la síntesis para la tecnología requerida.



(a) Primer Nivel (b) Segundo Nivel

Figura 2: ALU en niveles de bloques iniciales.

5. Síntetis RTL

A la hora de realizar la síntesis de la ALU como primer paso se realizan las respectivas configuraciones donde se especifican las bibliotecas y los directorios que requiere el Design Compiler. Luego se desarrollan los scripts correspondientes a la síntesis y restricciones de la misma, para proceder a la ejecución de dichos scripts en la herramienta de síntesis.

Como resultado de la síntesis se generan los archivos correspondientes a la descripción del módulo a nivel de celdas (GLN), el archivo de base de datos sintetizado y los reportes de área, potencia, celdas y tiempo. Los archivos GLN serán utilizados posteriormente en la simulación lógica post-síntesis y en la implementación física, por otro lado los reportes permiten tener una primera aproximación del comportamiento del módulo además del área que va a consumir basada en las celdas estándar que lo conforman.

En la figura 3 se observa como la ALU adquiere una estructuración más compleja a nivel de bloques en comparación de la figura 2 además se observa la gran cantidad de cableado necesario para la interconexión de las celdas. Luego de los reportes se obtiene la tabla 2 donde se detalla el área que van a abarcar las 1370 celdas que van a conformar la ALU, además se obtiene que va a consumir 1,27 mW y que va a tener un retardo aproximado de 18.49 ns.

Tabla 2: Reportes post-síntesis de la ALU

Parámetro	Valor
Área utilizada	10627.2 μm^2
Potencia	1.2745 mW
Retardo	18.49 ns
Celdas	1370

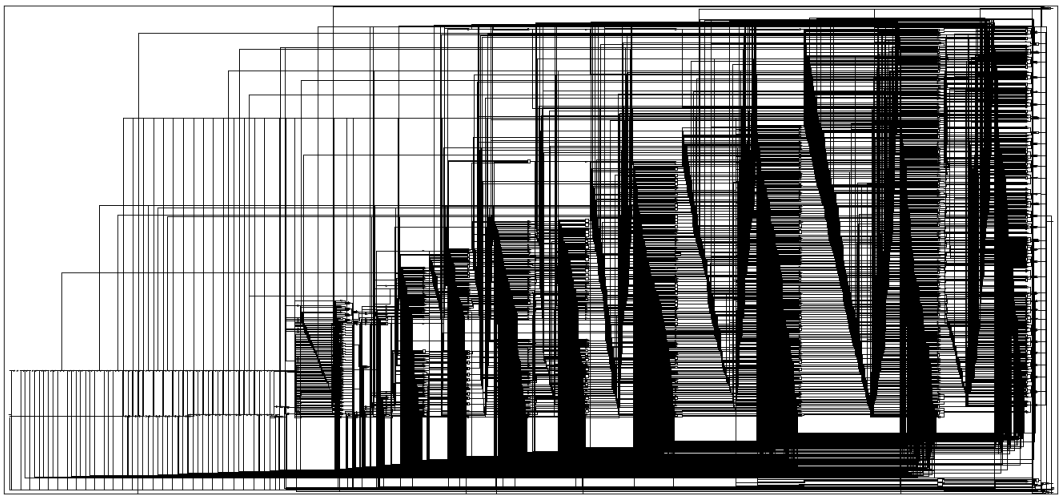


Figura 3: Conformación de la ALU a nivel de celdas estándar.

6. Simulación Lógica Post-Síntesis

Para la simulación post-síntesis se toman los archivos GLN resultado de la síntesis, el testbench correspondiente además de un archivo donde se especifican las rutas de los archivos mencionados y la biblioteca de celdas estándar, con esto se ejecuta la herramienta VCS que carga todos estos archivos para generar el comportamiento del módulo sintetizado, este ya considera ciertos aspectos de las celdas que aportan a los retardos en la respuesta de la ALU.

Como resultado de la simulación se obtuvo las figuras 4 y 5 donde se evidencia el funcionamiento de la ALU con los cambios de las señales de control y los cambios en las entradas además del funcionamiento de las banderas implementadas.

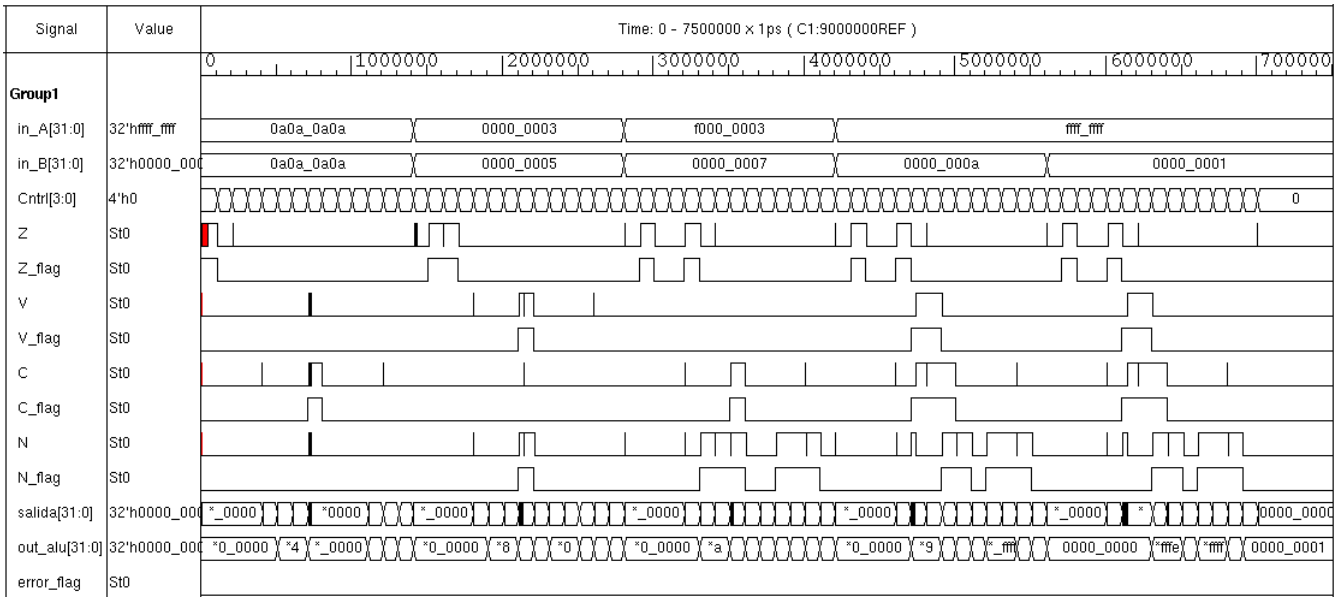


Figura 4: Simulación del testbench completo post-síntesis.

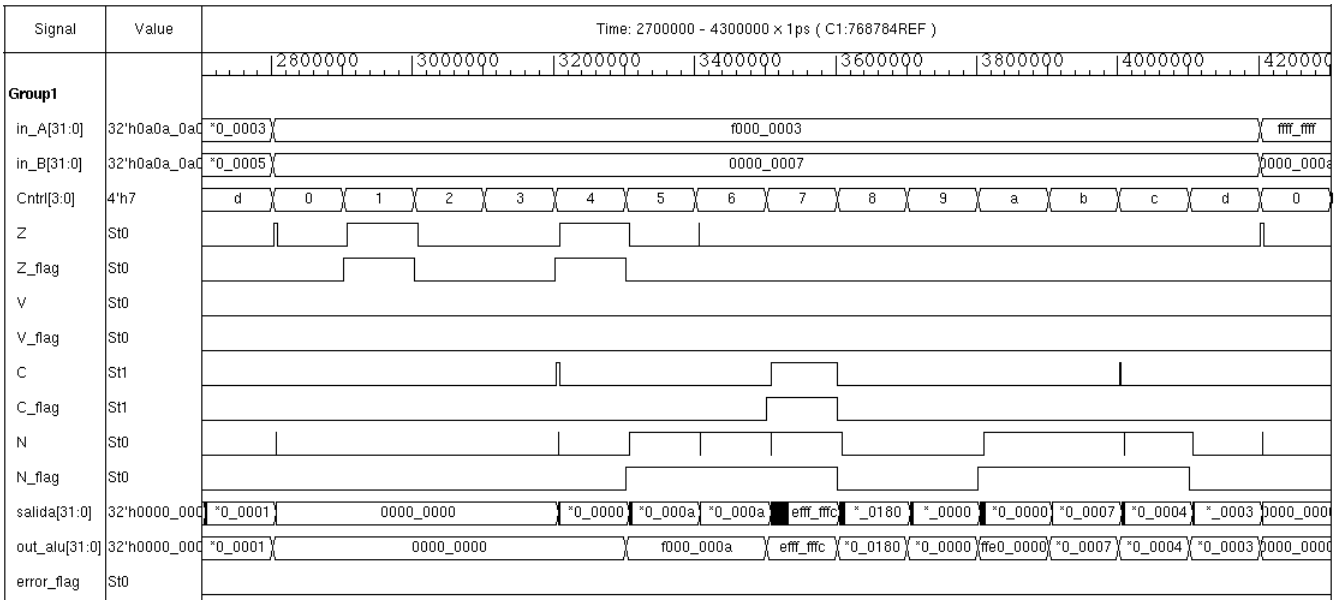


Figura 5: Simulación de una parte del testbench post-síntesis.

7. Implementación Física

Al realizar la implementación física por medio de comandos se genera el acomodo de las celdas estándar, se crean los anillos y líneas de conexión a la alimentación, se conectan las mismas con las celdas y por ende se genera un esquemático más ordenado (ver figura 6) respecto al que se observa en la figura 3.

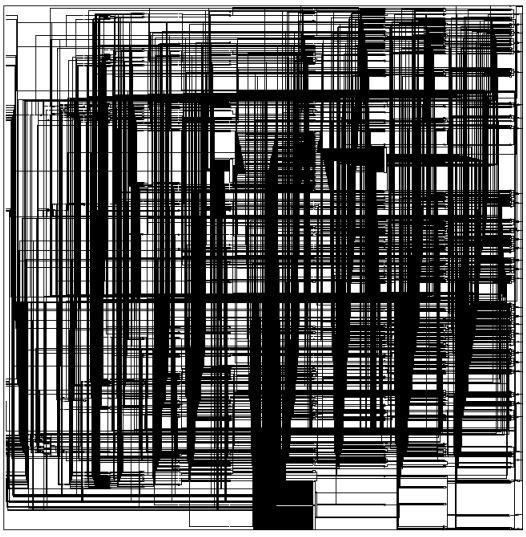
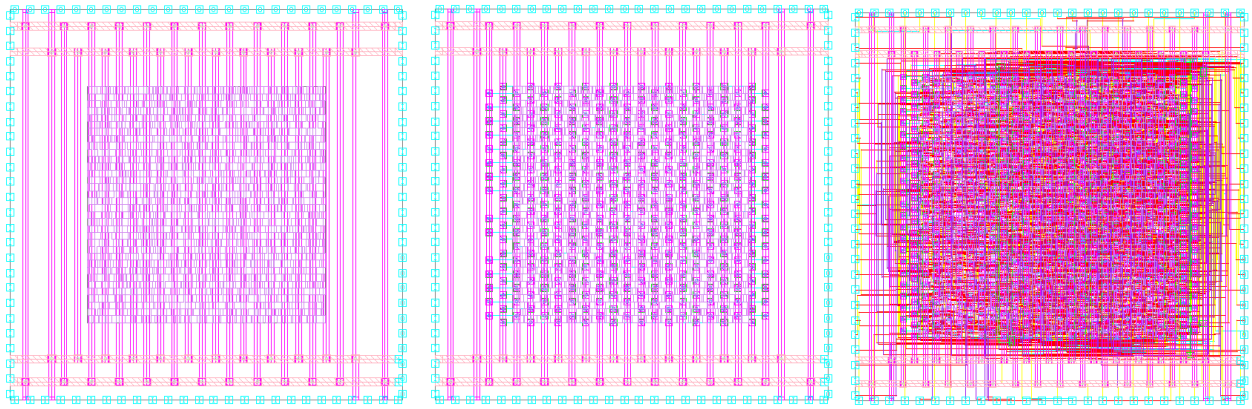


Figura 6: Esquemático generado después de realizar la implementación física.

En la figura 7 se pueden observar algunos de los pasos que se realizan en esta etapa del proyecto, en la subfigura (a) se tiene el diagrama del dispositivo cuando solo se tienen las celdas posicionadas, los anillos de alimentación y las líneas para realizar las uniones; en la (b) se observan ya las celdas conectadas a la alimentación; finalmente en la (c) se tienen todas las celdas conectadas a sus respectivas entradas y salidas.

Al generar la implementación física se presentan algunos errores que tienen como causa la no utilización de todas las terminales de las celdas estándar usadas, lo cual genera un desperdicio en términos de área e incluso de consumo de energía. Dichos errores son persistentes pese a que se tiene un DRC con cero violaciones, y se deben ignorar debido a que no se pueden modificar las celdas estándar.

De igual forma después de crear el layout se generan los reportes de potencia, área, retardo y celdas empleadas, estos se ubican en la tabla 3 donde se observa un aumento en todas las variables exceptuando el numero de celdas dado que no se están considerando las celdas que se utilizan para rellenar el espacio entre las celdas que utiliza la ALU.



(a) (b) (c)

Figura 7: Algunos pasos en la implementación física.

Tabla 3: Reportes post-layout de la ALU

Parámetro	Valor
Área utilizada	58564 μm^2
Potencia	3,3206 mW
Retardo	29.22 ns
Celdas	1370

8. Simulación Lógica Post-Implementación Física

Para esta simulación, de igual forma que en la simulación post-síntesis, se toman los archivos generados por el proceso de post-layout donde de toma en cuenta el cableado y otros efectos que influyen en el comportamiento del sistema, como resultado se obtienen las figuras 8 y 9 donde se aprecia que se mantiene el comportamiento de la ALU.

9. Análisis de resultados

Respecto a los esquemáticos generados en la etapa de síntesis y en la implementación física se puede ver una clara diferencia entre ellas; en la primera mencionada, que se puede ver en la figura 3, hay un

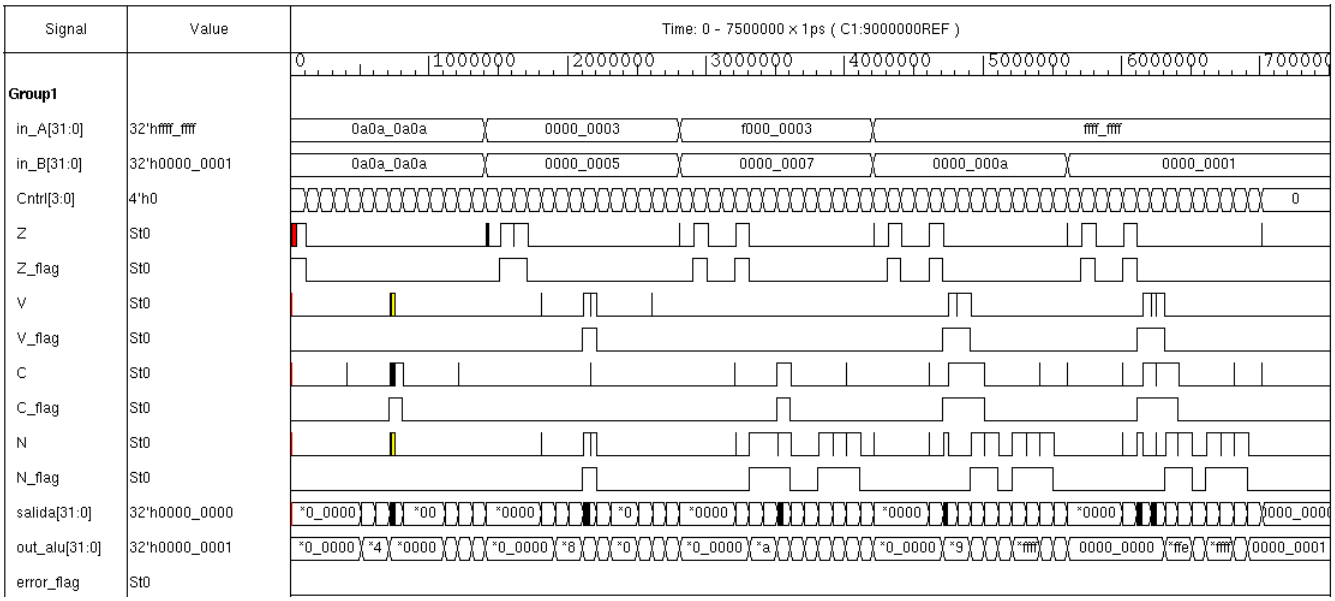


Figura 8: Simulación del testbench completo post-layout.

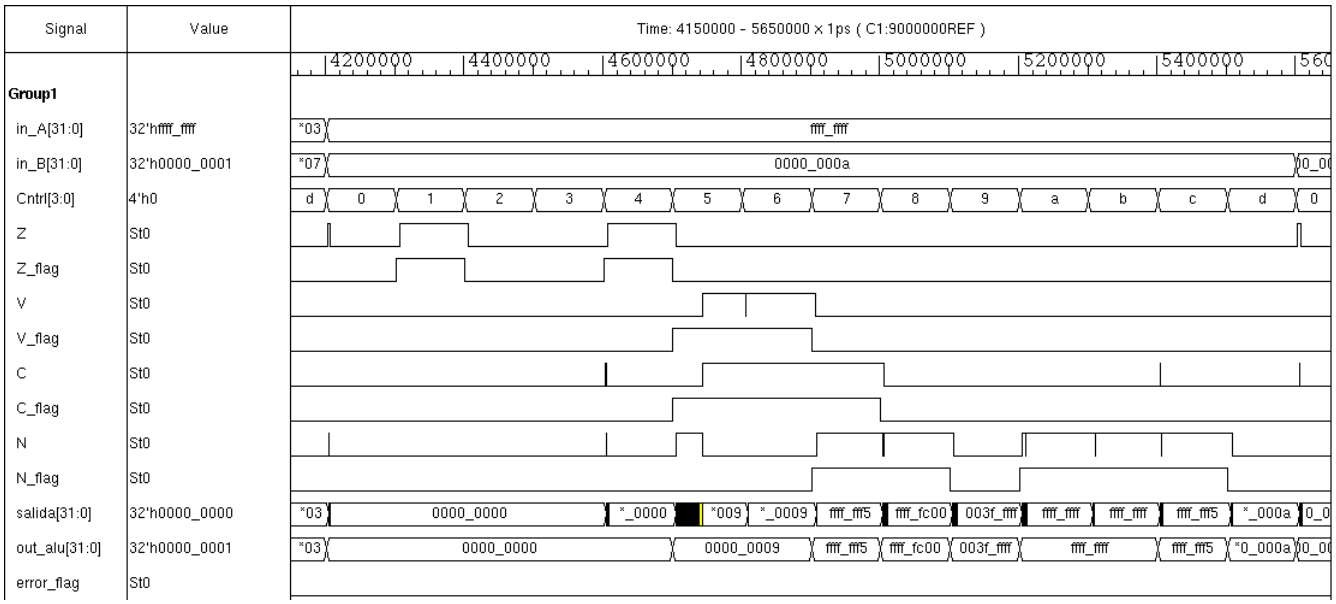


Figura 9: Simulación de una parte del testbench post-layout.

menor orden en las celdas por lo cual se puede, de primera mano, decir que hay una menor simetría respecto a la figura 6, lo cual puede generar errores conocidos, como lo son glitches.

Luego analizando los retardos obtenidos en ambas simulaciones se puede ver que esta ALU no puede operar en un sistema de alta frecuencia, esto se debe a que se está realizando toda la operación en una sola etapa, para optimizar el diseño de este módulo es necesario agregar una estructuración de pipeline de modo que exista un mayor control de los retardos internos y de este forma optimizar la ruta crítica y a su vez eliminar los glitches que se observan en las simulaciones.

Luego observando los reportes de las simulación post-síntesis y post-layout se observa como se da un aumento en el retardo, la potencia y el área, esto se debe a que se añade el efecto de las conexiones entre celdas y las conexiones de alimentación, dado que estas añaden capacitancias, resistencias y además consumen área del módulo, es por esto que es de gran importancia llevar el diseño a nivel de layout para considerar todas las variables involucradas principalmente en el retardo y consumo de potencia. Lo mencionado anteriormente es observable en la figura 7, con la cual se puede ir sospechando de este aumento en retardo y consumo de energía y área.

9.1. Complicaciones de la tarea

Al realizar la síntesis del diseño RTL de la primera ALU realizada se tienen errores que fueron solucionados fácilmente, pues todos eran causados por errores en la escritura del código que no ocasionaban ningún problema en el funcionamiento del sistema. Al realizar la segunda versión de la ALU se tiene más cuidado en la descripción por lo cual no se generan errores.

Al realizar la simulación lógica post-síntesis se generan errores debido a que no se le brinda el tiempo suficiente a las señales para que pasen a través de toda la lógica combinacional y la comparación que realiza el testbench se da con valores erróneos. Dicho fallo se elimina creando un delay mayor en el banco de pruebas, de forma que se le de más tiempo a la señal para estar lista antes de ser comparada.

En la implementación física se llega a complicar el uso del área, pues el área que por estándar coloca la herramienta no es suficiente espacio para que se de correctamente la colocación de las conexiones entre celdas, lo cual se nota hasta que se realiza el chequeo, después de haber realizado los anillos y líneas de conexión a la alimentación. Para corregir este problema se varió el parámetro que define el porcentaje de uso de espacio de las celdas respecto al área asignada al módulo, para obtener un ajuste libre de errores se ha debido realizar este proceso varias veces hasta lograr un DRC de cero, lo cual indica que no se tienen violaciones en las reglas de diseño.

10. Conclusiones y recomendaciones

10.1. Conclusiones

- El cableado juega un papel importante en el consumo de potencia, área y retardos dado que estos parámetros aumentan de la etapa de síntesis a la de layout.
- El área total del circuito no solo debe considerar el lugar en donde se colocarán las celdas, sino que se debe considerar el espaciado requerido del cableado así como las líneas de alimentación

necesarias para conectar todas las celdas estándar.

- Realizar la implementación física por medio de comandos no implica consecuentemente que la herramienta coloque las celdas de forma que el circuito creado sea simétrico, por lo cual se pueden tener tiempos de propagación distintos dependiendo de la ruta de los datos. De aquí la importancia de generar lógicas con pipeline.
- El uso de celdas personalizadas que se adecuen a las necesidades de ésta unidad en vez de usar celdas estándar eliminaría los errores en la etapa de implementación ocasionando a su vez una disminución en el área y el consumo de energía de la unidad.

10.2. Recomendaciones

- Tener en cuenta el retardo de los bloques combinacionales en el diseño del testbench para no tener errores de temporizado.
- Definir un área considerable a la hora de implementar el layout para que la herramienta pueda realizar las conexiones entre celdas en pocas iteraciones.
- Separar etapas del módulo diseñado con registros para sincronizar señales internas y de esta forma eliminar glitches.
- Investigar a profundidad los comandos a utilizar en los scripts usados para la síntesis y la implementación, de forma que se pueda ahorrar tiempo cuando se está realizando este proceso.
- Poder realizar tanto desde la interfaz, como por medio de los comandos, los pasos para la implementación física, pues se puede ahorrar una cantidad considerable de tiempo.

Referencias

[1] García, R. (2017). *CR TEC RISCV, Microarchitectural Specification.* Instituto Tecnológico de Costa Rica.

[2] Harris, S., & Harris, D. (2015). *Digital Design and Computer Architecture: ARM Edition.* Morgan Kaufmann.