

Arithmetic Logic Unit (ALU) de 4 bits

Trabajo Final de Circuitos Logicos Programables

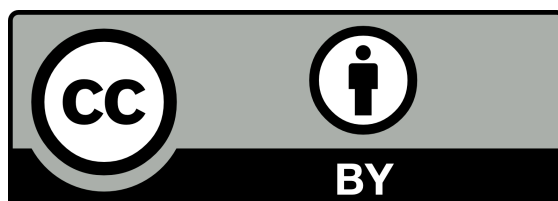
Agustín Jesús Vazquez

(vazqueza193@gmail.com)

22/04/2025

versión A

Esta obra está bajo una
Licencia Creative Commons Atribución
4.0 Internacional.



Historial de cambios

Versión	Fecha	Descripción	Autor	Revisores
A	16/04/2025	Versión Original	Agustin Vazquez	

Índice de contenido

1. Unidad aritmético lógica (ALU)	4
1.1. Selector de operaciones	4
2. Componentes principales	5
2.1. Suma de 4 bits	5
2.2. Resta de 4 bits	5
2.3. Multiplicación de 4 bits	6
2.4. División de 4 bits	7
2.5. Desplazamiento de 4 bits	7
3. Bloque principal de la ALU	8
4. Resultados de simulación	10

1. Unidad aritmético lógica (ALU)

Una Unidad aritmética lógica (ALU) es un circuito digital que se utiliza para realizar operaciones aritméticas y lógicas. Es uno de los componentes esenciales en los microcontroladores. Es capaz de realizar las siguientes operaciones:

1. Suma
2. Resta
3. Multiplicación
4. División
5. Desplazamiento
6. Operaciones lógicas (AND, OR, NOT y XOR)

El uso de FPGAs permite diseñar ALUs personalizadas que se ajustan a las necesidades específicas de una aplicación. Esto se hace utilizando lenguajes de descripción de hardware como VHDL. En la *figura 1* se muestra la representación de la ALU de 4 bits que tiene como entrada dos parámetros (A y B) y como salida se obtiene el resultado (Result) de la operación que se le indicó realizar.

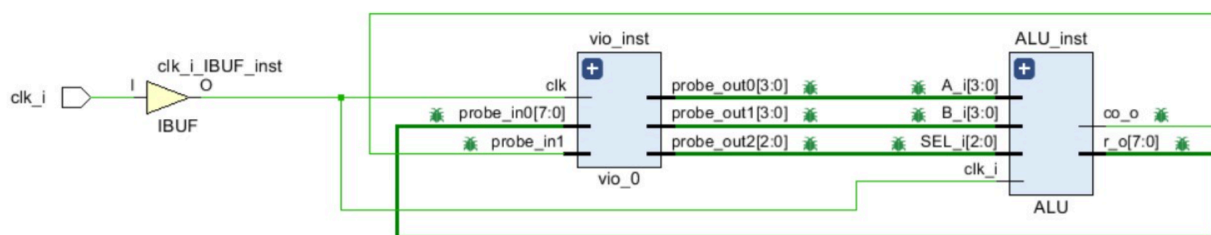


Figura 1. Diagrama de bloques de la ALU de 4 bits.

1.1. Selector de operaciones

Para seleccionar qué operación aritmética se quiere realizar se utiliza un número de operación. Las operaciones que podemos realizar según el valor indicado se indican en la *Tabla 1*.

Tabla 1. Operaciones de la ALU.

ALU_Sel	Operación	Resultado
000	Suma	$A + B$
001	Resta	$A - B$

010	Multiplicación	$A * B$
011	División	A / B
100	Módulo	$A \% B$
101	Desplazamiento	$A \gg B$

2. Componentes principales

2.1. Suma de 4 bits

Realiza la suma de dos números de 4 bits con acarreo. La *figura 2* detalla los archivos utilizados para la simulación en el software *Modelsim*.

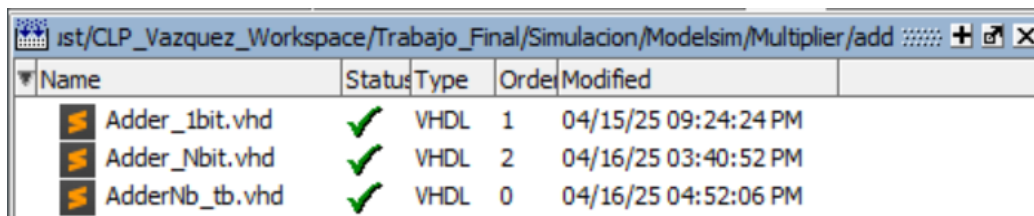


Figura 2. Banco de pruebas de la operación de suma realizada en Modelsim.

addNb es el sumador de N bits; para $N = 4$ instancia cuatro sumadores de 1 bit (**add1b**) en cascada, propagando el acarreo de un bit al siguiente y generando un acarreo final. En la *figura 3* puede observarse el resultado de la simulación, cuyos casos de prueba fueron redactados en el testbench.

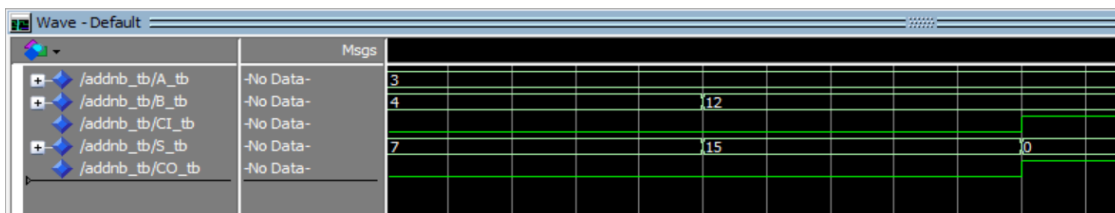


Figura 3. Simulación del componente Adder_Nb.

2.2. Resta de 4 bits

Calcula la diferencia de dos números de 4 bits y detecta el préstamo. La *figura 4* detalla los archivos utilizados para la simulación en el software *Modelsim*.

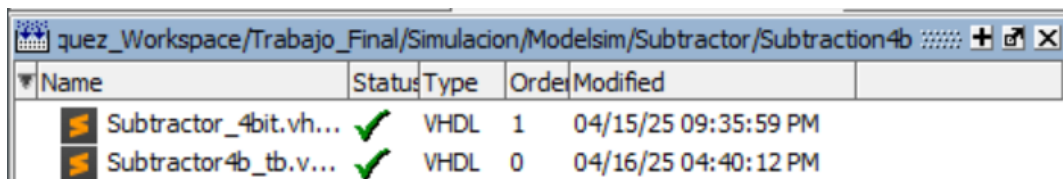


Figura 4. Banco de pruebas de la operación de resta realizada en Modelsim.

sub4b convierte ambos operandos a unsigned de 5 bits (añadiendo un 0 MSB), efectúa la resta y usa el quinto bit resultante para indicar si hubo préstamo (underflow). En la *figura 5* puede observarse el resultado de la simulación, cuyos casos de prueba fueron redactados en el testbench.

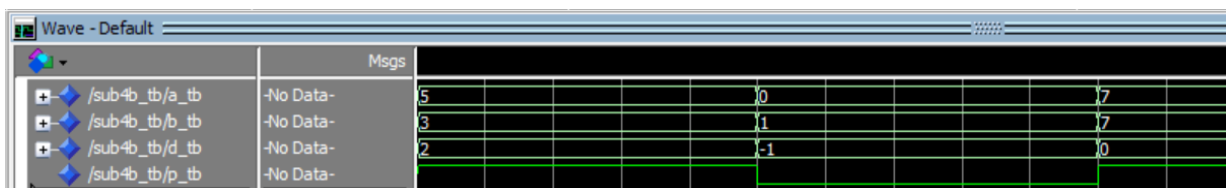


Figura 5. Simulación del componente Subtractor_4bit.

2.3. Multiplicación de 4 bits

Obtiene un producto de hasta 8 bits mediante sumas parciales. La *figura 6* detalla los archivos utilizados para la simulación en el software *Modelsim*.

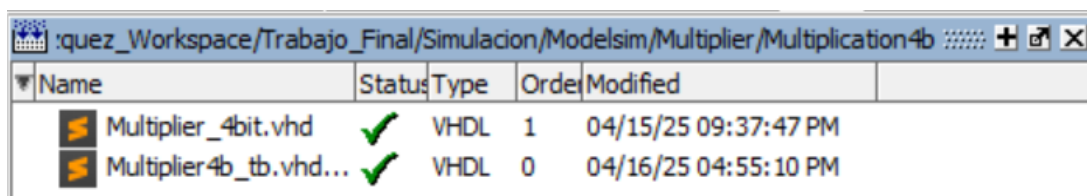


Figura 6. Banco de pruebas de la operación de multiplicación realizada en Modelsim.

mult4b aplica el algoritmo “shift-and-add”: por cada bit ‘1’ del multiplicador desplaza el multiplicando la posición correspondiente y suma el resultado al acumulador de 8 bits. En la *figura 7* puede observarse el resultado de la simulación, cuyos casos de prueba fueron redactados en el testbench.

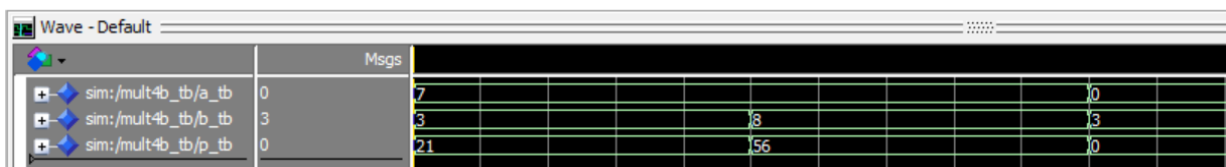
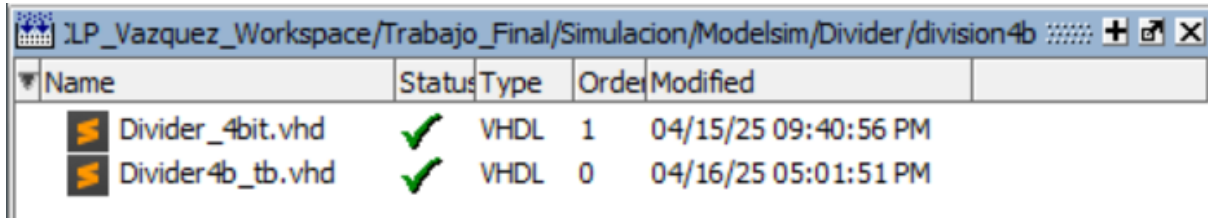


Figura 7. Simulación del componente Multiplier_4bit.

2.4. División de 4 bits

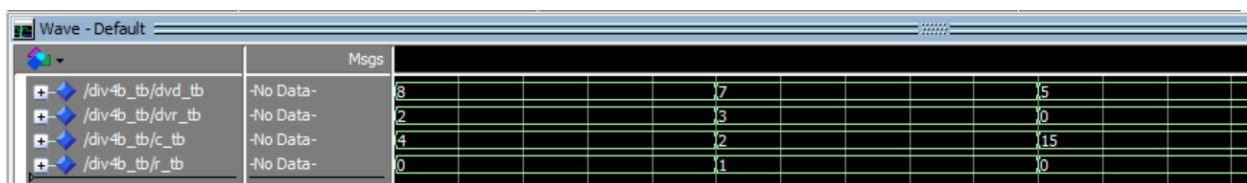
Calcula el cociente y el resto de la división entera de dos números de 4 bits. La *figura 8* detalla los archivos utilizados para la simulación en el software *Modelsim*.



Name	Status	Type	Order	Modified
Divider_4bit.vhd	✓	VHDL	1	04/15/25 09:40:56 PM
Divider4b_tb.vhd	✓	VHDL	0	04/16/25 05:01:51 PM

Figura 8. Banco de pruebas de la operación de división realizada en Modelsim.

div4b implementa un divisor secuencial sincronizado: en cada ciclo de reloj resta el divisor al resto parcial y, según el signo, ajusta el bit de cociente correspondiente; incluye detección de división por cero. En la *figura 9* puede observarse el resultado de la simulación, cuyos casos de prueba fueron redactados en el testbench.

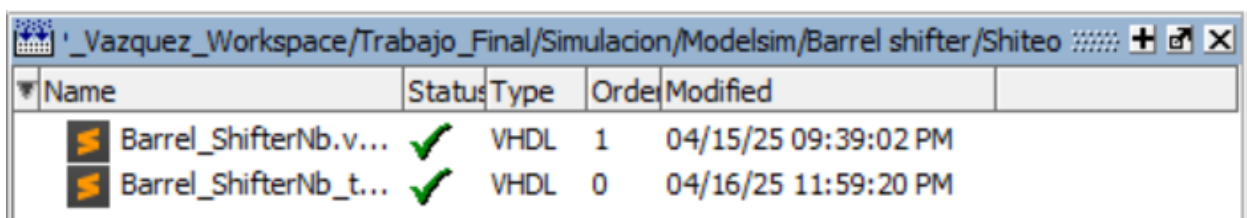


Wave - Default	Msgs
/div4b_tb/dvd_tb	-No Data-
/div4b_tb/dvr_tb	-No Data-
/div4b_tb/c_tb	-No Data-
/div4b_tb/r_tb	-No Data-

Figura 9. Simulación del componente Divider_4bit.

2.5. Desplazamiento de 4 bits

Desplaza lógicamente un vector de 4 bits a la derecha o a la izquierda. La *figura 10* detalla los archivos utilizados para la simulación en el software *Modelsim*.



Name	Status	Type	Order	Modified
Barrel_ShifterNb.v...	✓	VHDL	1	04/15/25 09:39:02 PM
Barrel_ShifterNb_t...	✓	VHDL	0	04/16/25 11:59:20 PM

Figura 10. Banco de pruebas de la operación de desplazamiento realizada en Modelsim.

bShifterNb convierte la entrada a unsigned y usa la función *shift_right* con el valor de desplazamiento para mover los bits, rellenando con ceros los espacios vacíos.

En la *figura 11* puede observarse el resultado de la simulación, cuyos casos de prueba fueron redactados en el testbench.

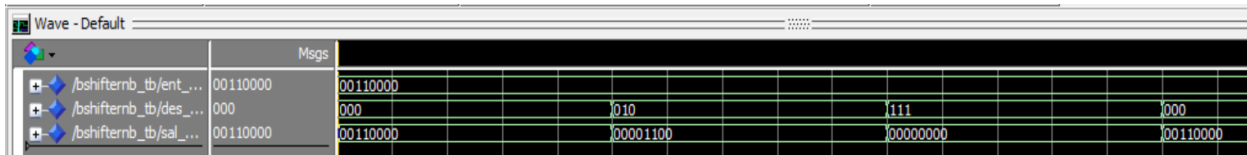


Figura 11. Simulación del componente Barrel_ShifterNb.

3. Bloque principal de la ALU

La ALU utiliza los archivos detallados en la *figura 12*, como se muestra a continuación:

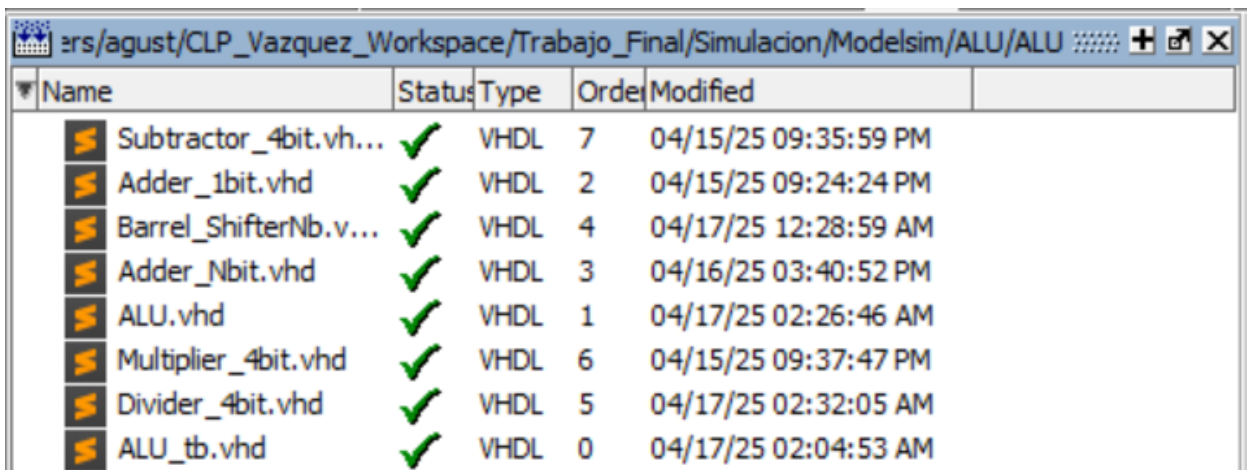
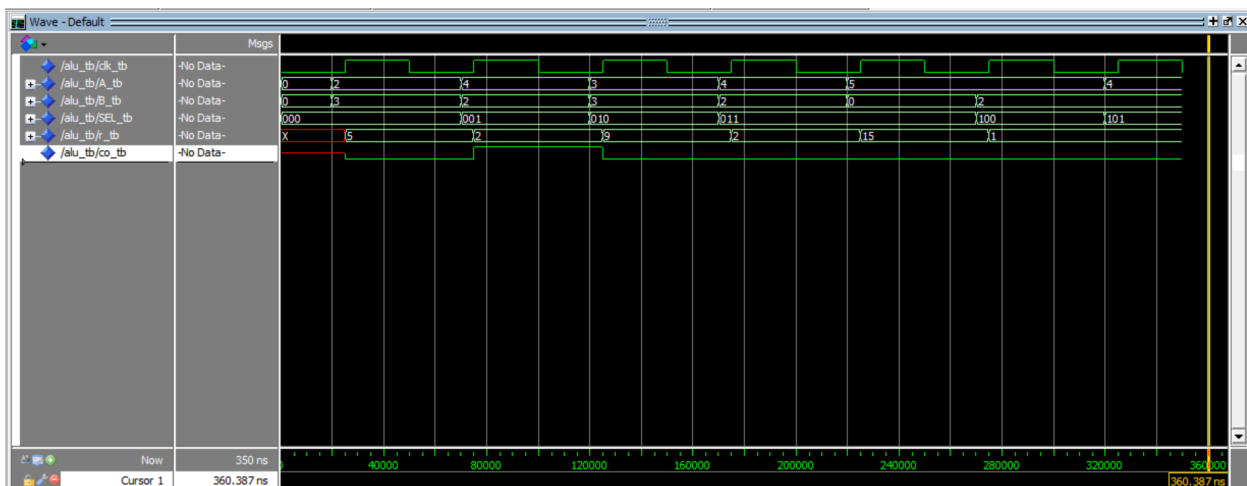


Figura 12. Banco de pruebas de la ALU realizada en Modelsim.



En las siguientes figuras se detallan esquemático y ruteo de las etapas de síntesis e implementación del proyecto en el software Vivado.

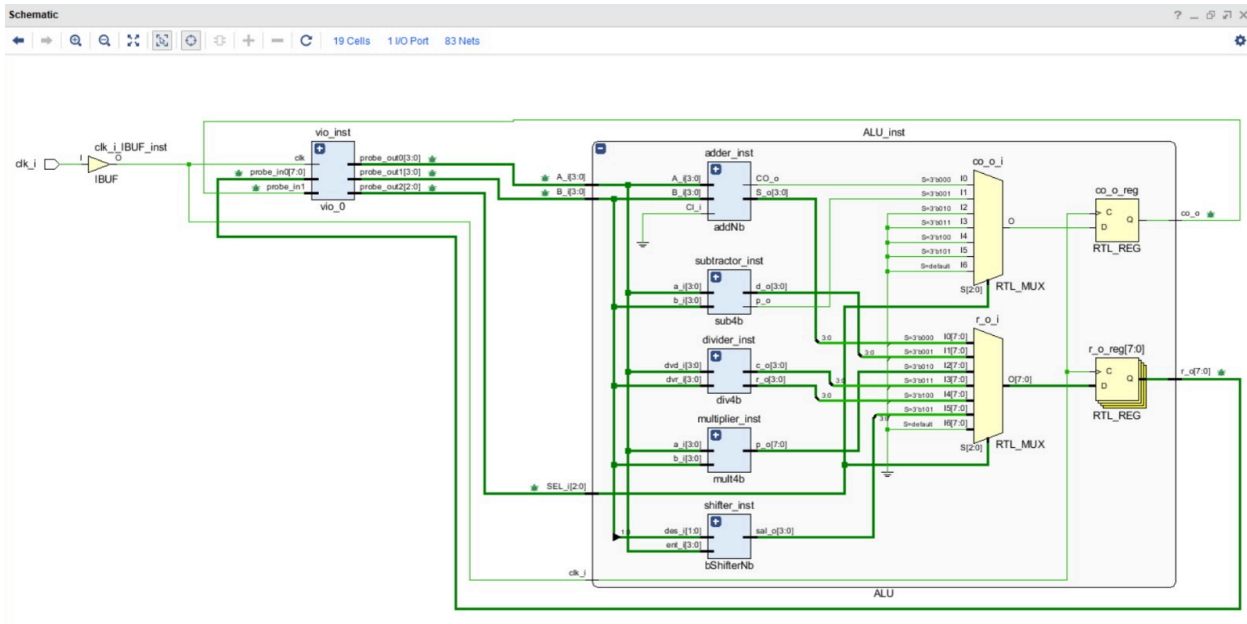


Figura 13. Esquemático de la ALU.

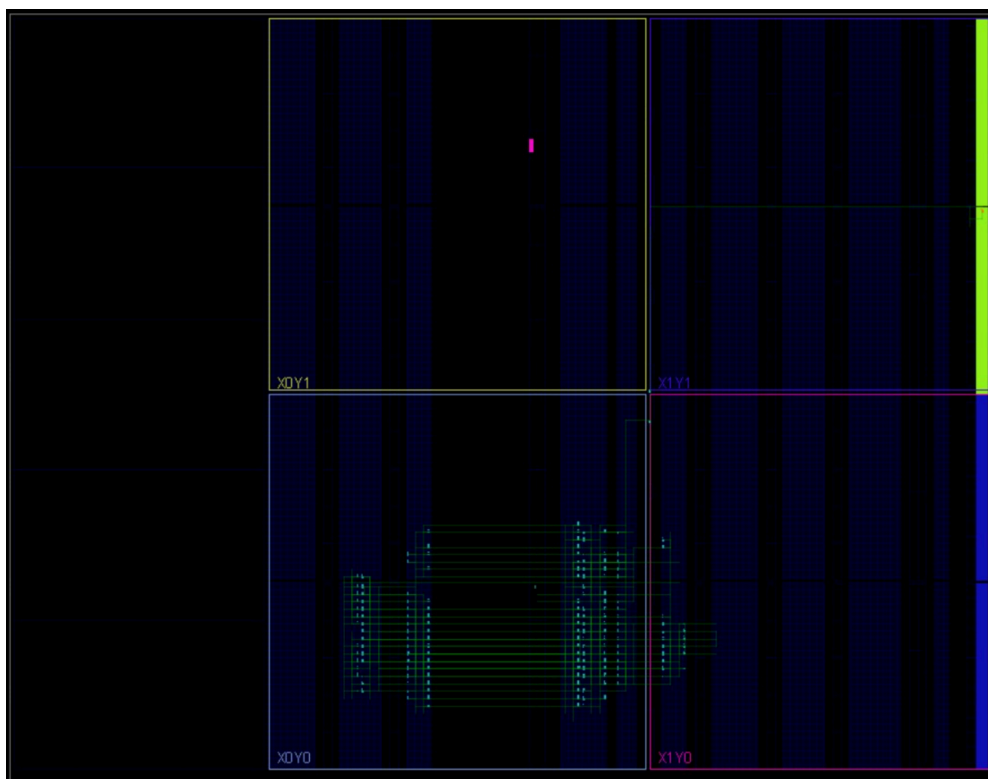
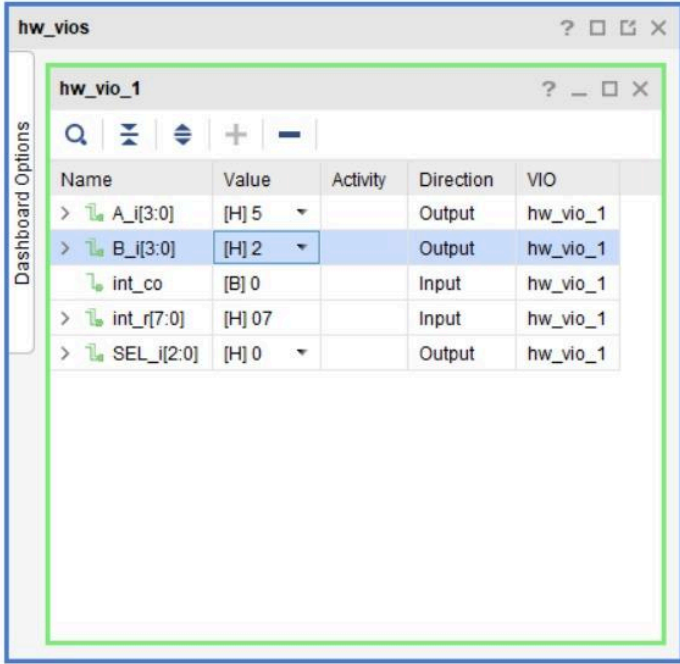


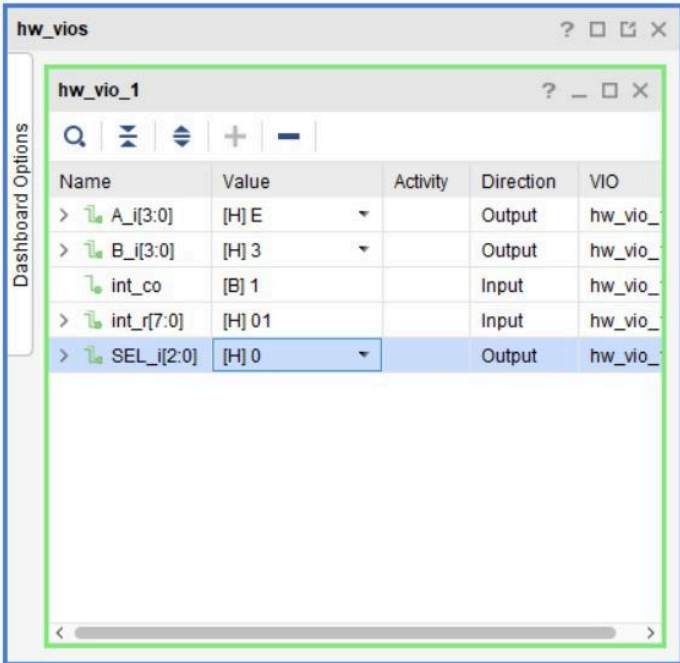
Figura 14. Vista del ruteo realizado.

4. Resultados de simulación



Name	Value	Activity	Direction	VIO
> A_i[3:0]	[H] 5		Output	hw_vio_1
> B_i[3:0]	[H] 2		Output	hw_vio_1
int_co	[B] 0		Input	hw_vio_1
> int_r[7:0]	[H] 07		Input	hw_vio_1
> SEL_i[2:0]	[H] 0		Output	hw_vio_1

Figura 15. Se realiza la suma ($SEL_i = '000'$) de dos números sin carry.



Name	Value	Activity	Direction	VIO
> A_i[3:0]	[H] E		Output	hw_vio_1
> B_i[3:0]	[H] 3		Output	hw_vio_1
int_co	[B] 1		Input	hw_vio_1
> int_r[7:0]	[H] 01		Input	hw_vio_1
> SEL_i[2:0]	[H] 0		Output	hw_vio_1

Figura 16. Se realiza la suma ($SEL_i = '000'$) de dos números con carry.

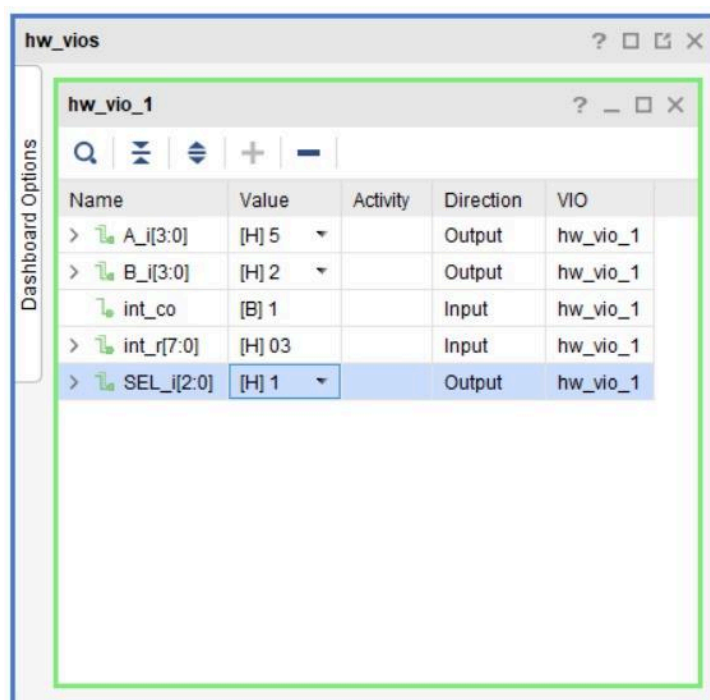


Figura 17. Se realiza la resta (SEL_i = '001') de dos números.

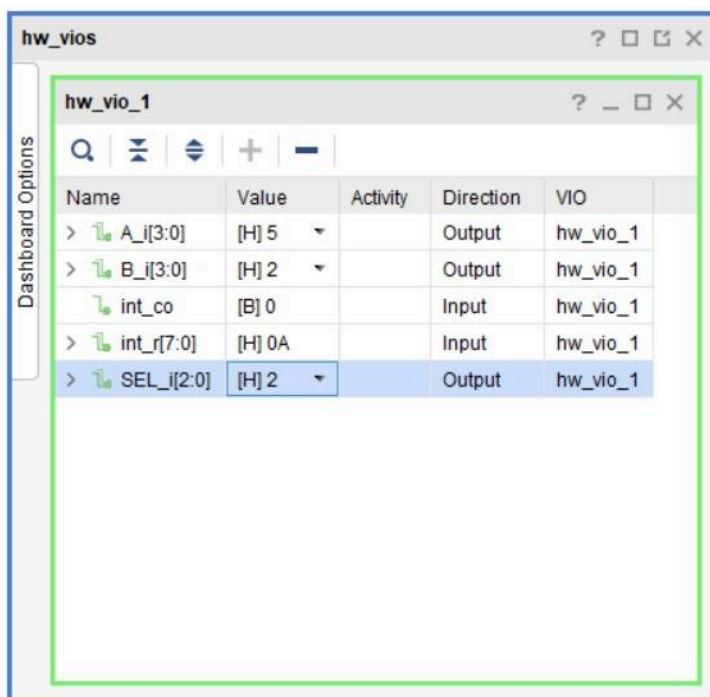
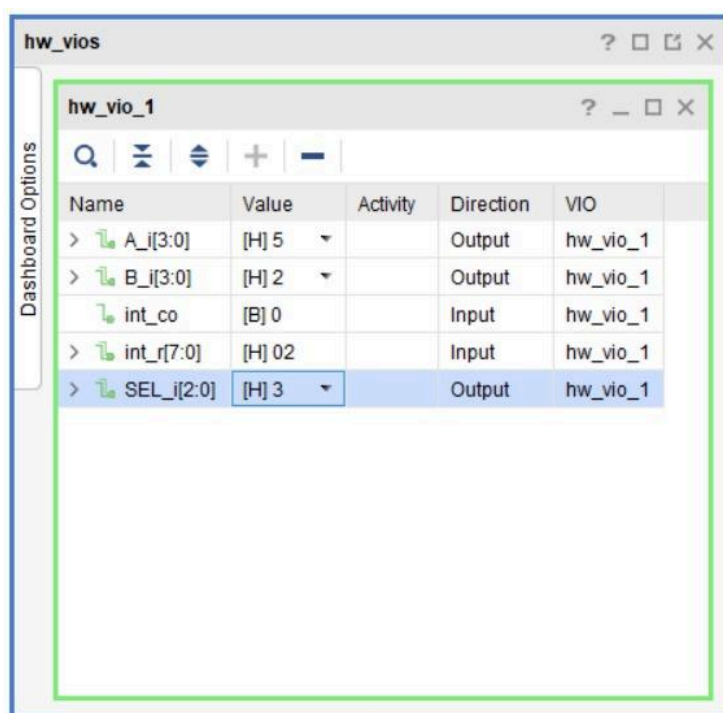
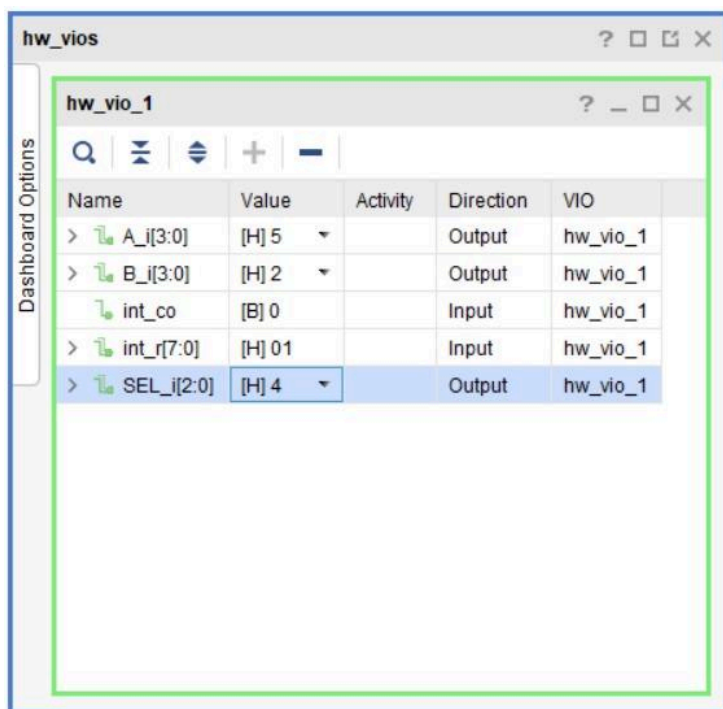


Figura 18. Se realiza la multiplicación (SEL_i = '010') de dos números de 4 bits con salida de 8 bits.



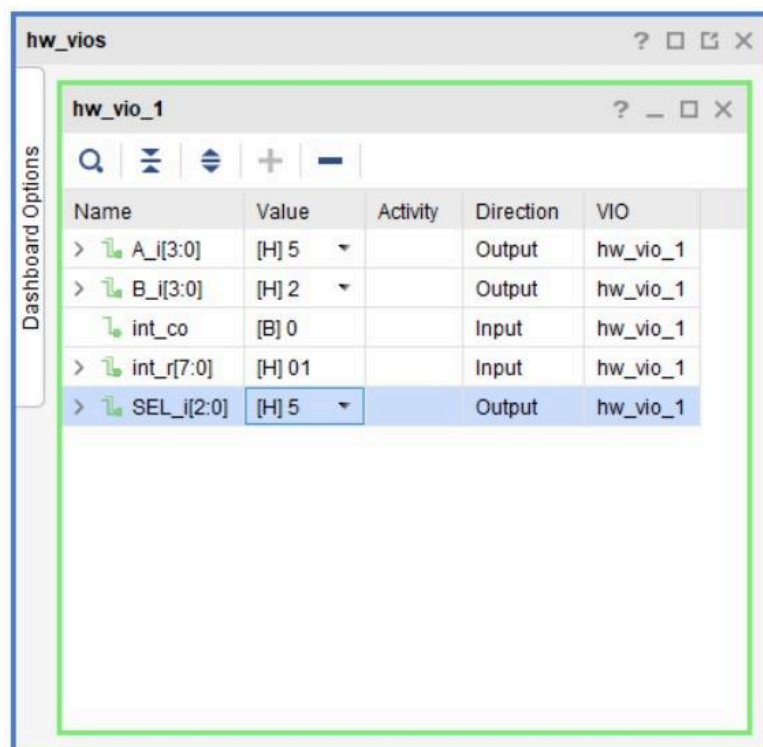
Name	Value	Activity	Direction	VIO
> A_i[3:0]	[H] 5		Output	hw_vio_1
> B_i[3:0]	[H] 2		Output	hw_vio_1
int_co	[B] 0		Input	hw_vio_1
> int_r[7:0]	[H] 02		Input	hw_vio_1
> SEL_i[2:0]	[H] 3		Output	hw_vio_1

Figura 19. Se realiza la división (SEL_i = '011') de dos números. La salida es el cociente de la división.



Name	Value	Activity	Direction	VIO
> A_i[3:0]	[H] 5		Output	hw_vio_1
> B_i[3:0]	[H] 2		Output	hw_vio_1
int_co	[B] 0		Input	hw_vio_1
> int_r[7:0]	[H] 01		Input	hw_vio_1
> SEL_i[2:0]	[H] 4		Output	hw_vio_1

Figura 20. Se realiza el módulo (SEL_i = '100') de dos números. La salida es el resto de la división.



Name	Value	Activity	Direction	VIO
> A_i[3:0]	[H] 5		Output	hw_vio_1
> B_i[3:0]	[H] 2		Output	hw_vio_1
int_co	[B] 0		Input	hw_vio_1
> int_r[7:0]	[H] 01		Input	hw_vio_1
> SEL_i[2:0]	[H] 5		Output	hw_vio_1

Figura 21. Se realiza el desplazamiento de un número A (SEL_i = '101') B veces a la derecha.