Universidad de Costa Rica
Escuela de Ingeniería Eléctrica
Estructuras de computadores digitales II

Tarea #3

Implementación de multiplicador en Verilog

Arturo Apú Chinchilla, B20386
Fabián Meléndez Bolaños, B24056
Luis Felipe Rincón Riveros, B25530

Profesor: Erick Carvajal

15 de mayo de 2015

1. Introducción

El presente laboratorio consistió en implementar un multiplicador iterativo de números enteros en Verilog. Esto se logró principalmente mediante el control de una máquina de estados y de un módulo con bloques aritméticos, multiplexores y registros llamado datapath.

El laboratorio se terminó exitosamente. El tamaño del multiplicador es variable, por lo que se puede hacer para cualquier cantidad de bits en los operandos.

La descripción utilizada tiene algunas ventajas. Con una buena sincronización de las banderas, se pueden usar varias multiplicadores en cascada y así multiplicar números muy grandes.

El módulo principal fue probado con entradas generadas iterativamente y aleatoriamente. La multiplicación fue efectiva en el 100 % de las pruebas.

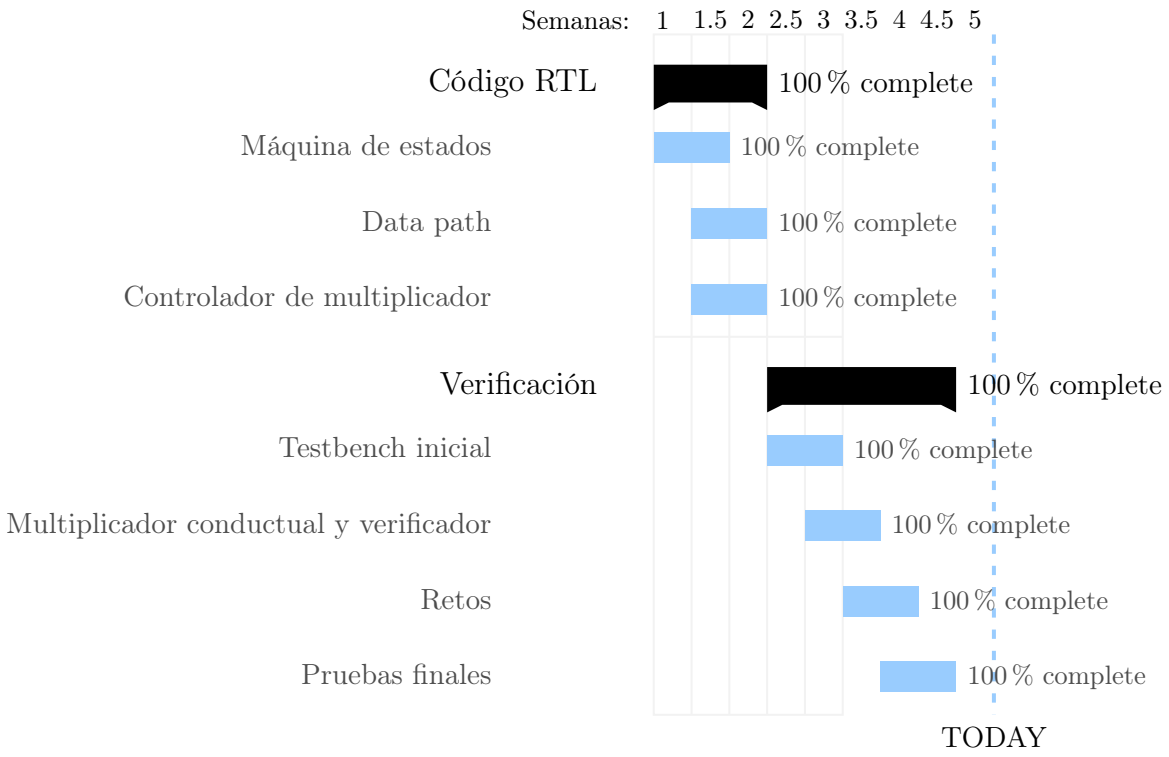
Además se crearon dos módulos multiplicadores extra, uno de tres entradas y otro de cuatro entradas. Ambos fueron exitosos, mutiplicaron de manera correcta en la totalidad de sus pruebas.

2. Manejo del proyecto

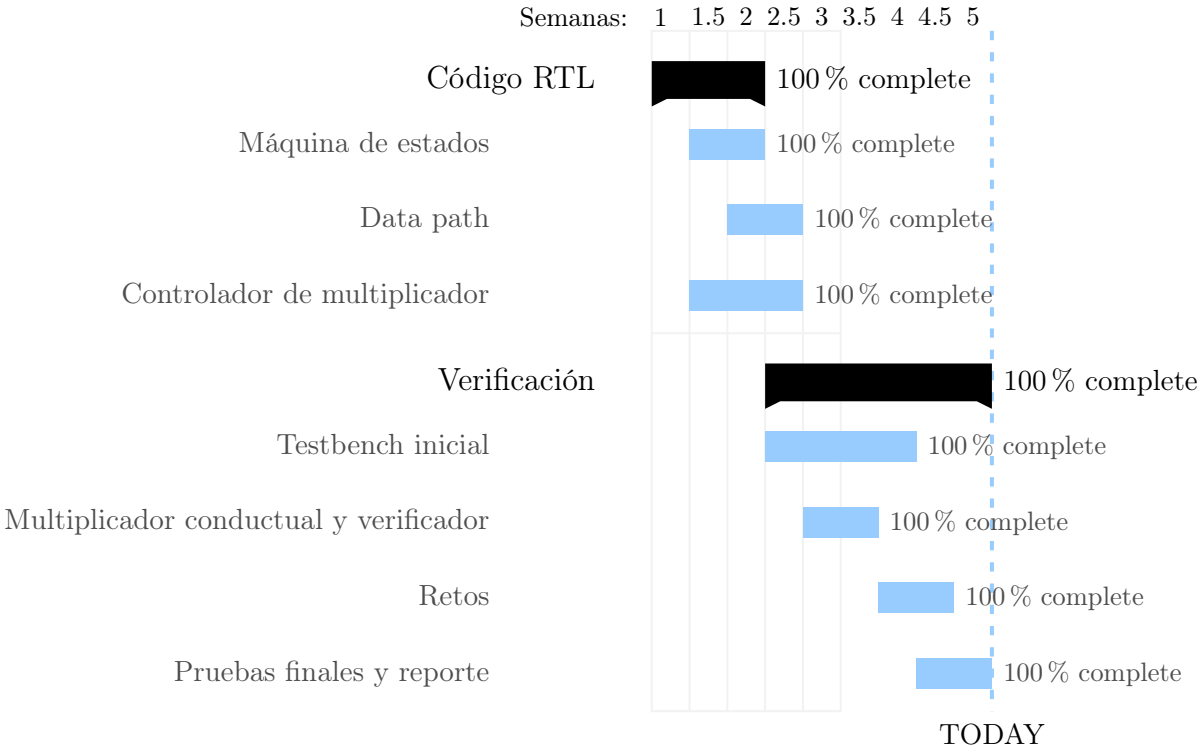
Los roles se dividieron de la siguente manera:

- Arquitecto: Luis Felipe Rincón R.
- Diseñador: Fabián Meléndez B.
- Verificador: Arturo Apú Ch.

2.1. Cronograma inicial



2.2. Cronograma real



Los contadores de tres y cuatro entradas fueron más difíciles de implementar por la sincronización de las banderas. Además, para esto se decidió hacer el tamaño del multiplicador un parámetro para así poder instanciar un multiplicador de cualquier tamaño.

3. Diseño

El mutiplicador diseñado recibe 4 señales de entrada y tiene 3 señales de salida, tal y como se muestra en la Figura 1. El módulo indica que está listo para recibir datos cuando la salida oIdle está en alto. Cuando esto sucede, se leen los dos números que se desean multiplicar, iA y iB. Luego, se le indica al multiplicador que puede leer estos datos poniendo en alto la señal iValid_Data.

Cuando el multiplicador termina de multiplicar, el producto se pone en oProduct y la señal oDone pasa a estar en alto. Finalmente, se le indica al multiplicador que ya se leyó el dato poniendo iAcknoledge en alto; con esto, el multiplicador pone la señal oIdle en alto y está listo para recibir nuevos multiplicandos.

Figura 1: Diagrama de bloque del multiplicador.

Para el diseño, se crearon tres módulos. Un contador de 5 bits con reset sincrónico para poder contar los ciclos de reloj y dos módulos que se encargan de resolver la multiplicación.

Uno de estos módulos es una máquina de estados con 4 estados. El primer estado un estado de Reset que definir las señales que se van a ultilizar; luego entra en un lazo de estados como el que se muestra en la Figura 2.

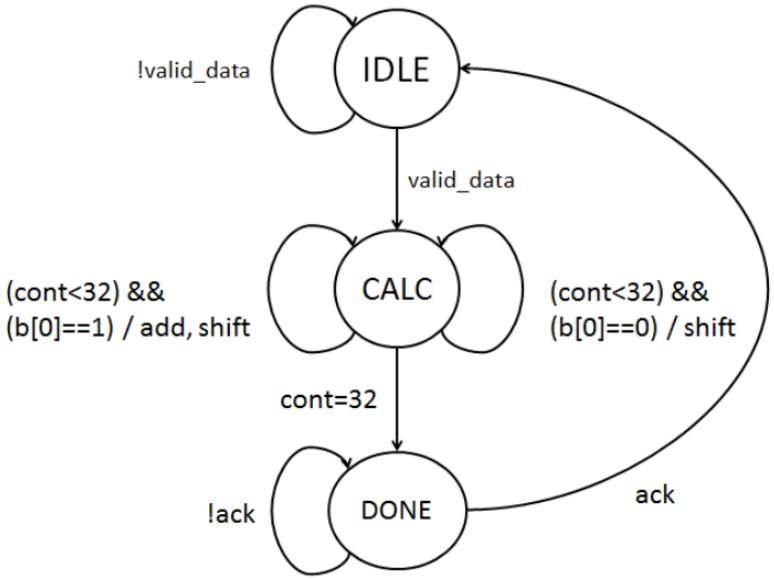


Figura 2: Máquina de estados implementada.

Como se puede observar, el controlador tiene un estado IDLE en el que el multiplicador está listo para recibir datos. Esto se informa al exterior con una bandera de oIDLE. Para salir de este estado, se le debe informar al multiplicador, por medio de una entrada iValid_Data, que se ya se le dieron datos válidos para multiplicar y que proceda a hacerlo.

Con esto se entra al estado CALC, aquí es donde se hace la multiplicación. Para esto se utiliza el módulo Data Path, y el controlador se limita a manejar la bandera de reset de los datos en el data path al inicio y asegurar de no hacerlo más. Se cuentan tantos ciclos de reloj como el tamaño de los operandos (que son los que se ocupan para multiplicar) y luego se sale de este estado.

Al finalizar se entra en el estado de DONE, el cual se encarga de informar que ya se tiene dato listo. Si se recibe la indicación de que ya se agarró el dato, se vuelve al estado IDLE.

El cálculo de la multiplicación sucede en el Data Path, el cual se muestra en la Figura 3. En este circuito, si se da un adecuado reinicio de los valores de reg_b, reg_a, y reg_prod y luego se dejan pasar 32 ciclos de reloj (o tantos como el tamaño de los operandos), entonces se obtendrá al final de los 32 ciclos el resultado de la multiplicación.

4. Estrategia de pruebas

Un multiplicador no necesita demasiadas pruebas. Un análisis completo consistiría en probar todas las combinaciones de entradas y revisar que cada salida sea correcta. Esto se consideró no necesario; sin embargo, el módulo se probó con una cantidad significativa de semillas.

La generación de semillas se decidió hacer de manera iterativa y de manera aleatoria; para lograr se diseñaron dos de pruebas: una que genera las semillas iterativamente y otra que las genera

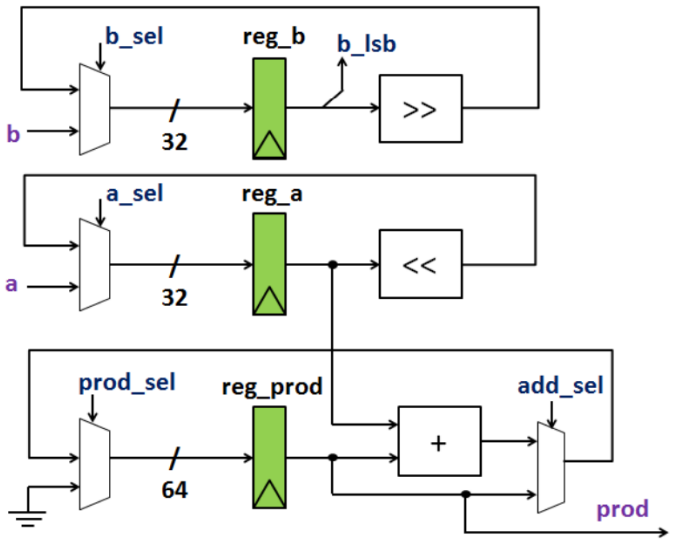


Figura 3: Circuito del Data Path.

aleatoreamente. Agregado a esto se implementaron dos estructuras de verificación. En la primera, un testbench genera tanto las señlas de control como los multiplicandos tal como se ilustra en la Figura 4.

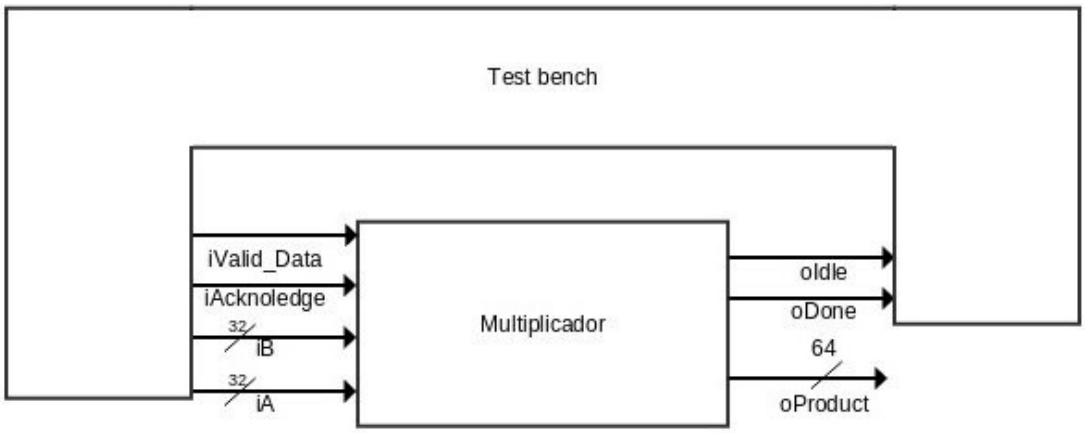


Figura 4: Diagrama de bloques de primera estructura de pruebas.

La otra estructura de prueba consiste en agregar un módulo multiplicador conductual y un módulo verificador. Un testbench igual al anterior genera las señales, estas se conectaron tanto a una instancia del multiplicador diseñado como a una del multiplicador conductual. Las salidas de ambos multiplicadores se conectan al verificador. Este último pone una señal de verificación si ambos productos son iguales, si no, pone la señal en cero. La estructura anterior se ilustra en la Figura 5.

Ambas pruebas se hicieron en ambas estructuras.

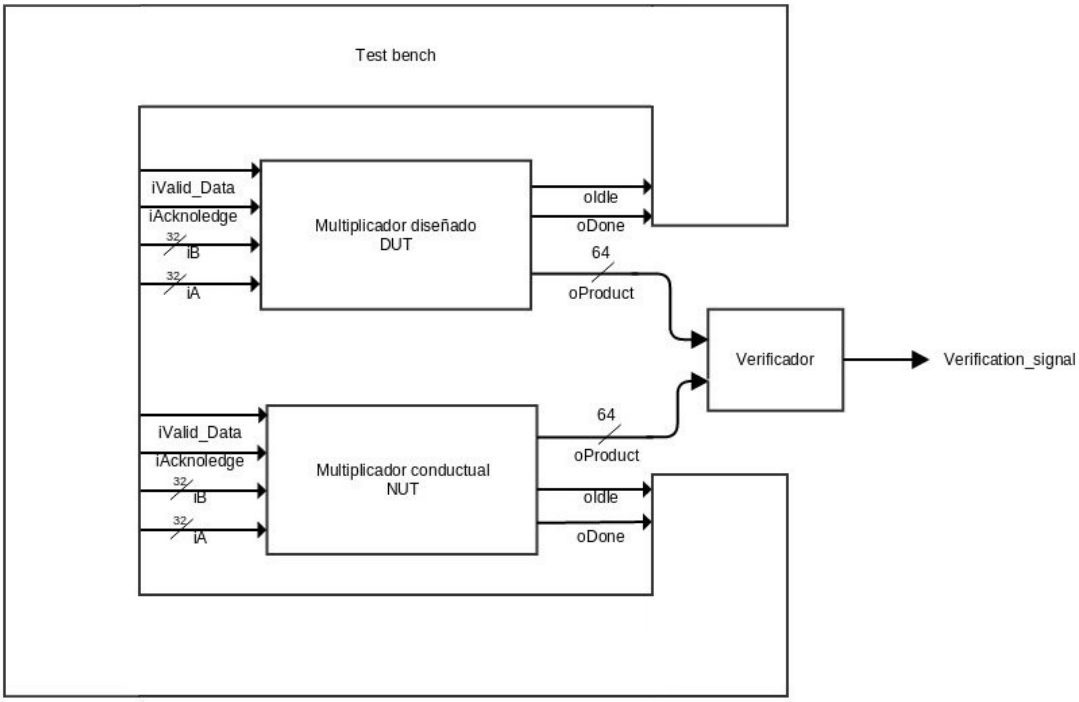


Figura 5: Diagrama de bloques segunda estructura de pruebas.

5. Evaluación

5.1. Entradas aleatorias

Se utilizó la función random de Verilog para generar números aleatorias en las entradas del multiplicador.

Primero se probó lo anterior con el módulo solo, la simulación se observa en la Figura 6. En dicha imagen se nota que $32 \cdot 83 = 5093$, $38 \cdot 82 = 3116$, $47 \cdot 90 = 3290$ entre otras multiplicaciones.

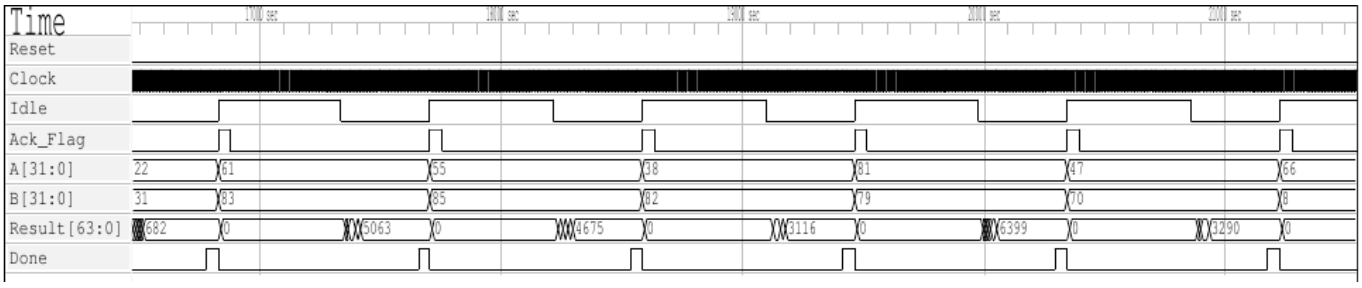


Figura 6: Prueba con entradas aletorias.

Se probó también con el módulo conductual y verificador. Esto se observa en la Figura 7. En esta simulación basta con revisar la señal verif para comprobar que el módulo funciona. Esta señal se mantiene en 1, excepto un pequeño intervalo cuando hay un cambio de resultado (esto porque el módulo conductual no tiene retardo comparado con el módulo estructural). Por lo tanto el módulo multiplicador cumple correctamente con las funciones para las que fue diseñado.

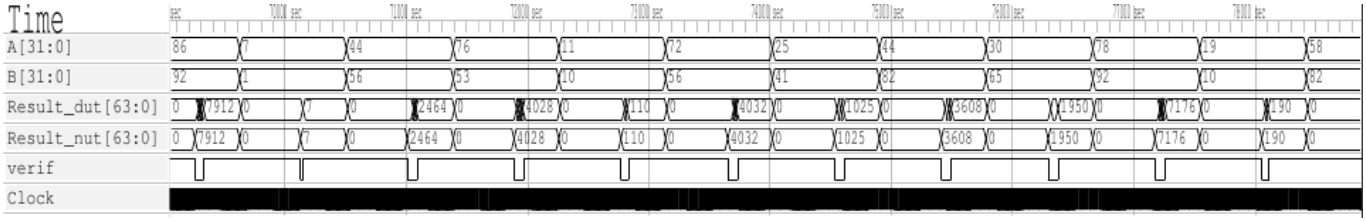


Figura 7: Prueba con entradas aletorias con verificador.

5.2. Entradas iterativas

Para la segunda prueba se incrementaron ambas entradas al mutiplicador en 1, iterativamente, de manera que todas las salidas son cuadrados perfectos.

Primero realizó la prueba con el módulo solo, la simulación se observa en la Figura 8. Aquí, se nota que multiplicaciones de enteros muy pequeños son realizadas sin problema.

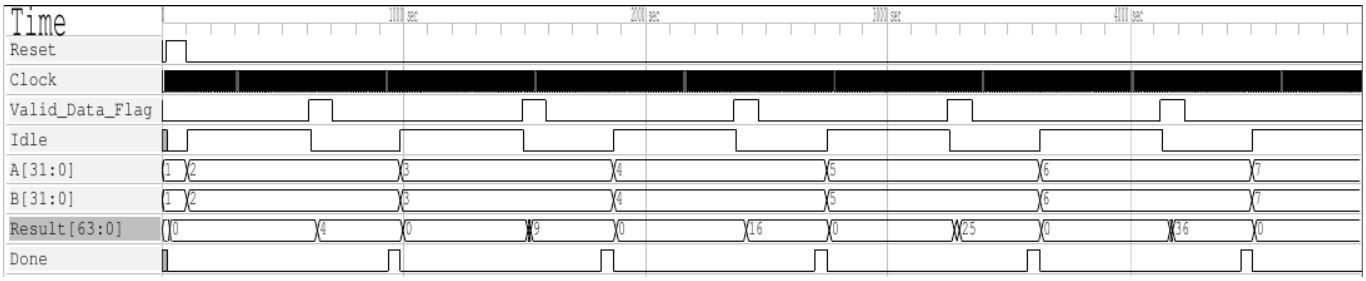


Figura 8: Prueba con entradas iterativas.

Con un tiempo de simulación algo elevado se probaron multiplicaciones de números más grandes, tal y como se aprecia en la Figura 9. Aquí se observan multiplicaiones como $535 \cdot 535 = 286225$ y semejantes.

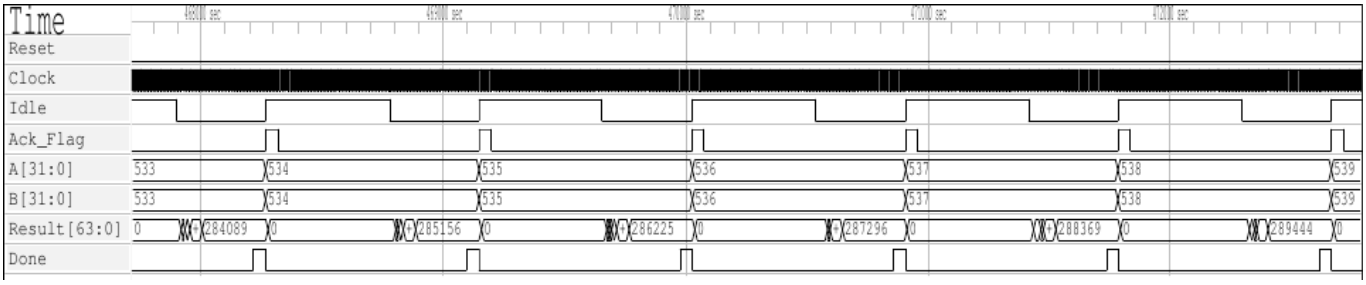


Figura 9: Prueba con entradas iterativas y mayor tiempo de simulación

Se probó también con el módulo conductual y verificador. Esto se observa en la Figura 10. Aquí, nuevamente, se observa que la señal verif se mantiene en alto, por lo que todas las multiplicaciones se realizan de manera correcta.

Análogamente, con un alto tiempo de simulación, se probaron multiplicaciones de número más grandes. Eso se observa en la Figura 11. Aquí, nuevamente, el verificador se mantiene en alto para cada iteración de las entradas.

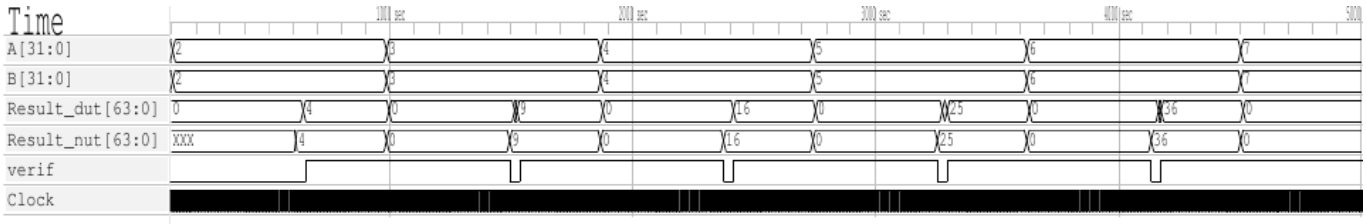


Figura 10: Prueba con entradas iterativas con verificador.

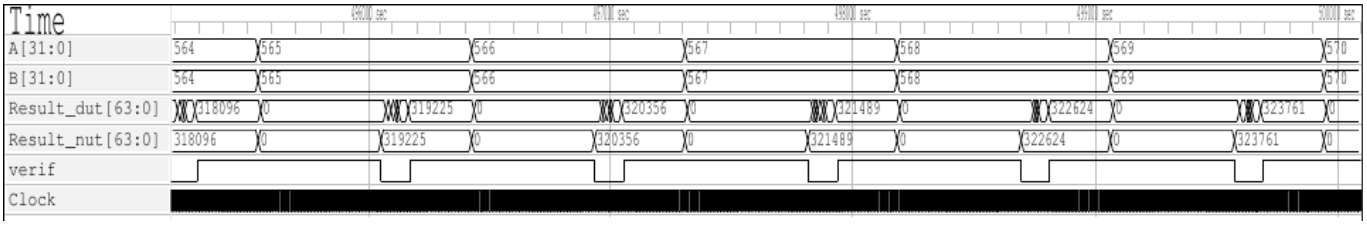


Figura 11: Prueba con entradas iterativas con verificador y mayor tiempo de simulación.

6. Retos

6.1. Multiplicador de 3 entradas

Se implementó un multiplicador de 3 entradas usando el bloque multiplicador diseñado. Básicamente dos entradas van a un multiplicador, y el resultado de este va junto a la tercera entradaa un segundo multiplicador.

Para lograr esto la tercera entrada se tuvo que sostener hasta que la primera multiplicaciópn se realizara. Además se le tuvo que concatenar 32 bits de ceros para que sea del mismo tamaño que el producto de las otras dos entradas.

Ahora, la señal iAcknoledge es la misma para ambos contadores. La entrada iValid le indica al primer contador que puede leer los primeros dos multiplicandos. La señal oDone de este primer multiplicador es la entrada iValid del segundo, así cuando termina de multiplicar le indica al segundo que empice.

La señal oIdle del multiplicador de tres entradas es una and lógica entre los oIdle de los multiplicadores instanciados. esto para que el dato se reporte como listo cuando los dos multiplicadores han terminado de trabajar.

En las Figuras 12 y 14 se muestran dos pruebas de este mutiplicador. Aquí se obtiene que $197 * 198 * 199 = 7762194$, y este número es 767112 en hexadecimal. Además se obtiene que $219 * 220 * 221 = 10647780$, y este número es $A278E4$ en hexadecimal.

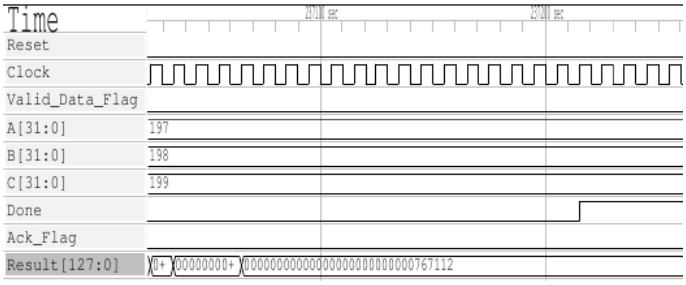


Figura 12: Prueba de mutiplicador de tres entradas.

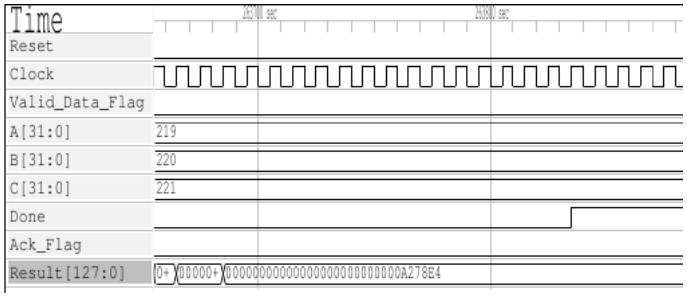


Figura 13: Prueba de mutiplicador de tres entradas.

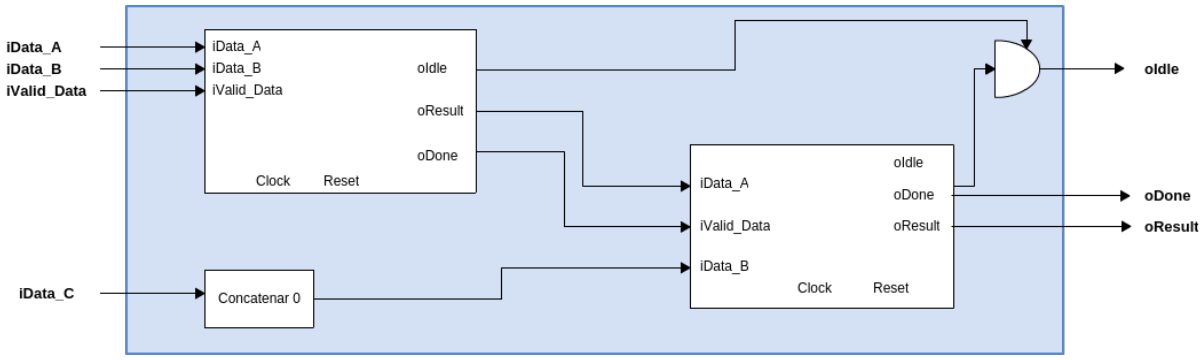


Figura 14: Diagrama de bloques del multiplicador de 4 entradas.

6.2. Multiplicador de 4 entradas

Para implementar el multiplicar de 4 entradas, se utilizaron 3 multiplicadores de 2 entradas, los cuales ya se habían verificado en pruebas anteriores. En la Figura 15 se muestra el circuito que se utilizó.

Cabe destacar que primero se pasan los 4 datos de dos en dos a ambos multiplicadores del primer nivel. Estos hacen la multiplicación y una vez que ambos esten listos, se envíe la señal de iValid_Data al tercer multiplicador. Este ya determina que todo está listo y da el resultado. Todos los multiplicadores utilizan las mismas señales de Reset y Clock. El oIdle se determina mediante una compuerta AND de tres entradas, las cuales son las banderas de estado Idle de los 3 multiplicadores.

En el circuito de la figura, no se muestran los tañamos de cada línea ya que son parametrizados. No obstante, la implementación si utiliza la cantidad de bits necesarios.

Para probar este circuito, se utilizó básicamente el mismo TestBench explicado anteriormente, excepto que se llama TestBench4x4.v. Se agregaron las entradas necesarias y la instancia del módulo Four_Multiplicador. Los resultados de esta simulación se muestan en la Figura 16.

En esta se puede verificar que la multiplicación se efectuá perfectamente. En este caso, en la prueba los datos son de 16bits para facilitar la presentación de los resultados. Sin embargo, el tamaño se puede cambiar fácilmente en el TestBench.

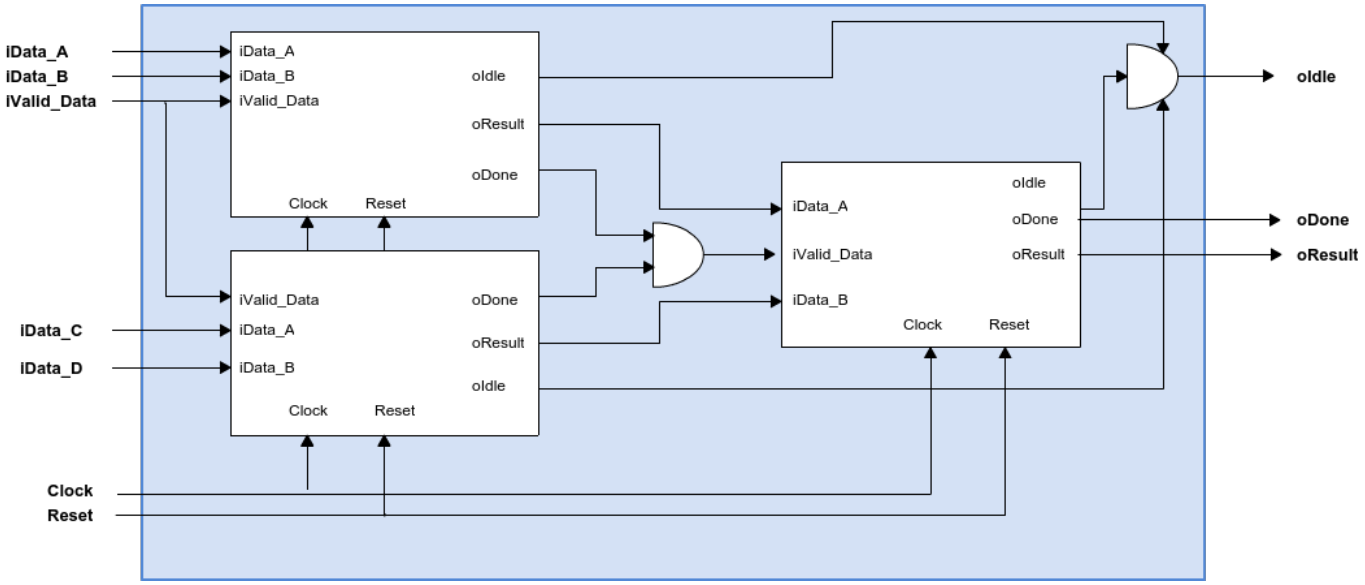


Figura 15: Estructura de diseño del multiplicador de 4 entradas

7. Instrucciones de utilización de la simulación

Para correr cualquier simulación, en caso de que se utilice un sistema operativo de LINUX/UNIX, se realizó un Makefile, el cual facilita la compilación y ejecución de comandos.

Se puede hacer:

```
>$ make
```

Este programa compilará el testbench que se tenga en esta carpeta. En caso de que no se tenga este programa (make), se deberán ejecutar los siguientes comandos:

```
>$ iverilog -c out.vvp <<testbench>>.v
>$ vvp out.vvp
>$ gtkwave test.vcd
```

Nota: el archivo .vcd puede cambiar de nombre.

Existen varios testbenchs. El más simple es Testbench.v, el cual realiza multiplicaciones con el multiplicador y luego está automatic_verificator.v el cual verifica el multiplicador con un verificador.

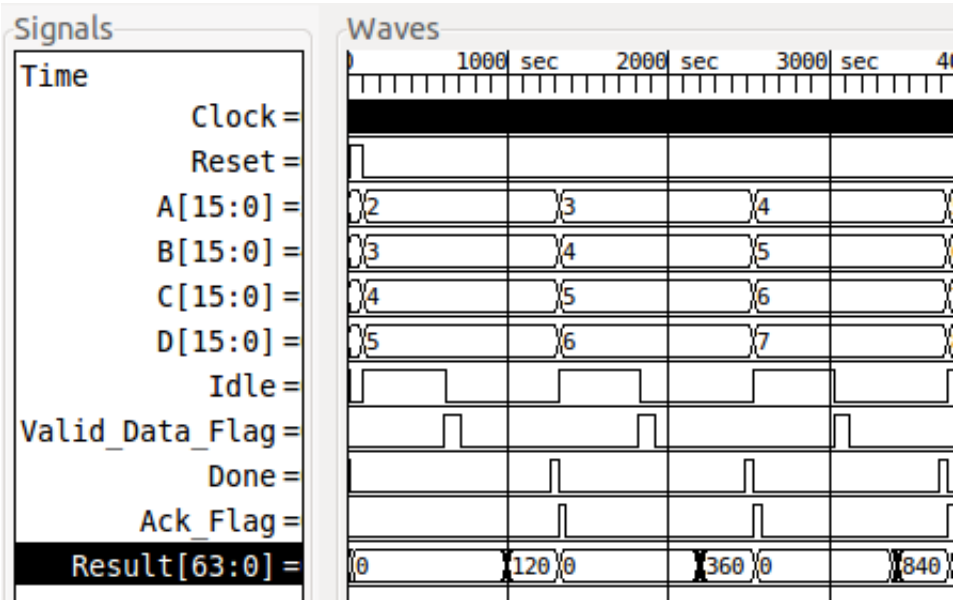


Figura 16: Resultados de simulación para multiplicador de 4 entradas