Informe de Resolución de la Práctica 9

- Alumno: CALDERÓN Sergio Leandro
- Legajo: 02285/4

1.1 – Entidad Multiplicador

La interfaz está compuesta por 4 puertos de entrada y 2 puertos de salida.

Tabla 1. Entradas del multiplicador.

Nombre	Tipo	Descripción
A	Bit_Vector(3 downto 0)	Primer operando de 4 bits
B	Bit_Vector(3 downto 0)	Segundo operando de 4 bits
STB	Bit	Strobe, mando de inicio de operación
CLK	Bit	Reloj utilizado para avanzar de estado

Tabla 2. Salidas del multiplicador.

Nombre	Tipo	Descripción
Result	Bit_Vector(7 downto 0)	Producto (resultado) de 8 bits
Done	Bit	Indicador de operación terminada

1.2 – Arquitectura de Sumas Sucesivas

Se utilizan los componentes diseñados de las prácticas 2 al 7.

- **Controller:** máquina de estado finito de Moore, con 5 estados definidos.
- **Adder8:** sumador de 8 bits asincrónico, con carry de entrada y de salida.
- **ShiftN:** registro de desplazamiento de N bits hacia la izquierda o derecha.
- **Latch8:** registro con memoria (Flip Flop tipo D) para números binarios de 8 bits.

Las instancias necesarias para la implementación del multiplicador se muestran en la Tabla 3 a continuación. En las próximas secciones se explicará en detalle cada una de ellas.

Tabla 3. Instancias de componentes utilizadas en la arquitectura de sumas sucesivas.

Nombre	Componente	Descripción
FSM_Controller	Controller (práct. 7)	Controla el funcionamiento del multiplicador
Reg_SRA	ShiftN (práctica 6)	Desplaza los bits de “A” hacia la derecha
Reg_SRB	ShiftN (práctica 6)	Desplaza los bits de “B” hacia la izquierda
Adder	Adder8 (práctica 5)	Suma la salida de Reg_SRB + resultado previo
Reg_ACC	Latch8 (práctica 3)	Almacena resultados intermedios del sumador
Reg_Res	Latch8 (práctica 3)	Almacena el resultado final de la operación

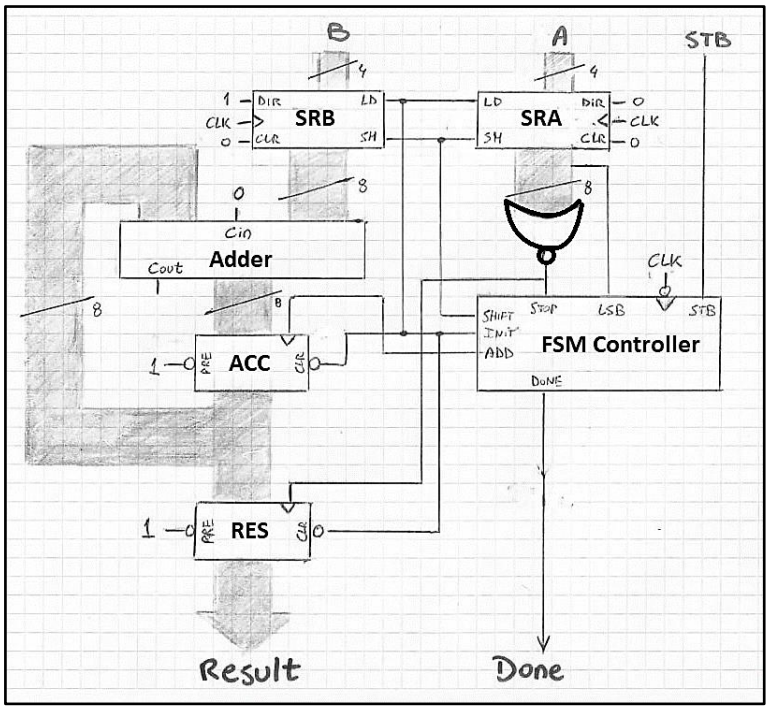

Figura 1. Modelo implementado para la arquitectura de sumas sucesivas.

A partir del modelo mostrado en la Figura 1, se declaran las siguientes señales internas a la arquitectura, mayormente nombradas a partir de la interfaz de FSM Controller.

```
-- Declaración de señales internas a la arquitectura
signal Stop: Bit;                           -- Entrada Stop del Controlador
signal Init : Bit;                          -- Salida Init del Controlador
signal Shift: Bit;                          -- Salida Shift del Controlador
signal Add: Bit;                            -- Salida Add del Controlador
signal Q_SRA: Bit_Vector(7 downto 0);       -- Salida del componente SRA
signal Q_SRB: Bit_Vector(7 downto 0);       -- Salida del componente SRB
signal Q_ACC: Bit_Vector(7 downto 0);       -- Salida del acumulador (ACC)
signal Q_Adder: Bit_Vector(7 downto 0);     -- Salida del sumador (Adder)
signal Co_Adder: Bit;                       -- Carry de salida del sumador
signal NotClock: Bit;                       -- Señal invertida de reloj
signal NotClearReg: Bit;                    -- Señal invertida de Clear para reg acumulador y resultado
signal Estable: Bit;                        -- Señal que indica si los FF están estabilizados
```

Figura 2. Porción del código VHDL donde se declaran las señales internas de la arquitectura

1.2.1 – FSM Controller

La máquina de estado finito define 5 estados: “End”, “Init”, “Check”, “Add” y “Shift”. Al ser sincrónica, requiere de una entrada de reloj, para poder realizar las transiciones entre estados en los flancos de bajada de dicha señal, según el diseño visto en la práctica 7. Por tal motivo, se utiliza una señal invertida del reloj en el puerto CLK de FSM Controller.

```
-- Instanciación de todos los componentes, mapeo de puertos
FSM_Controller: Controller port map (STB=>STB, CLK=>NotClock, LSB=>Q_SRA(0), Stop=>Stop,
                                     Init=>Init, Shift=>Shift, Add=>Add, Done=>Done);
```

Figura 3. Instanciación de FSM_Controller

El estado inicial por defecto es “End”, para el cual la señal de salida “Done” es 1, indicando que el multiplicador no se encuentra realizando ninguna operación. Este estado se mantiene hasta que la señal de entrada “STB” esté activa (sea 1), pasando al estado “Init”.

En el estado “Init”, se realiza la carga de los números A y B en los registros de desplazamientos, y se limpian los registros acumulador y resultado, que son de tipo Latch. En el próximo flanco de bajada de reloj, se pasa al estado “Check”, a partir del cual se vuelve al estado “End” si “Stop = 1” (esto es, cuando todos los bits de Q_SRA sean cero), sino se pasa al estado “Add” si “LSB = 1” o al estado “Shift” si “LSB = 0” (según el bit 0 de Q_SRA).

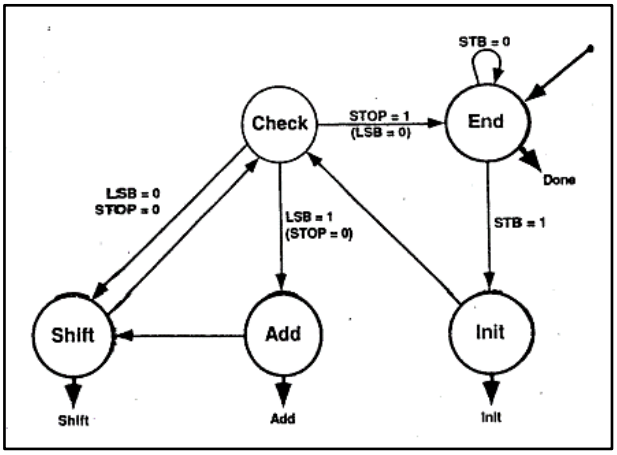

Figura 4. Diagrama de estados de FSM Controller

En el estado “Add”, se realiza la carga del acumulador, y en el próximo flanco de bajada de reloj se pasa al estado “Shift”. En dicho estado, se realiza un desplazamiento en ambos registros SRA y SRB, y en el próximo flanco de bajada de reloj se vuelve al estado “Check”.

1.2.2 – Registros de desplazamiento de bits

El componente ShiftN también es sincrónico, y permite realizar la carga y desplazamiento de bits en ambos sentidos de un número binario. En concordancia al diseño visto en la práctica 6, para habilitar el desplazamiento de bits, la entrada “SH” debe estar activa, por ende, la misma toma el valor de la salida “Shift” de FSM Controller. El sentido de desplazamiento lo define la entrada “DIR”, siendo 0 hacia la derecha, y 1 hacia la izquierda; por tal motivo, para Reg_SRA se indica “DIR => 0” y para Reg_SRB se indica “DIR => 1”.

```
Reg_SRA: ShiftN port map(CLK=>CLK, CLR=>'0', LD=>Init, SH=>Shift, DIR=>'0', D=>A, Q=>Q_SRA);
Reg_SRB: ShiftN port map(CLK=>CLK, CLR=>'0', LD=>Init, SH=>Shift, DIR=>'1', D=>B, Q=>Q_SRB);
```

Figura 5. Instanciación de Reg_SRA y Reg_SRB.

Como se mencionó anteriormente, la carga de los números A y B se realiza en el estado "Init" de FSM Controller; es por ello que la entrada "LD" de los registros es la señal "Init".

1.2.3 – Adder

El sumador de 8 bits realiza la suma de los operandos "Q_ACC" y "Q_SRB", que son la salida del registro acumulador y de desplazamiento de B, respectivamente. En este caso, no se requiere considerar un "carry" de entrada, por lo tanto, se establece la entrada "Cin" del sumador en bajo (0). Según el diseño visto en la práctica 5, este componente resuelve las operaciones de manera asincrónica, por lo que no necesita una entrada de reloj.

```
Adder: Adder8 port map(A=>Q_ACC, B=>Q_SRB, Cin=>'0', Cout=>Co_Adder, Sum=>Q_Adder);
```

Figura 6. Instanciación de Adder

1.2.4 – Registro acumulador

Se encuentra implementado mediante el componente Latch8, reutilizando el diseño de la práctica 3 sin modificación alguna. La entrada de datos "D" es la salida "Q_Adder" del sumador mencionado anteriormente, pero únicamente se copia dicha entrada a la salida "Q_ACC" del Latch cuando la máquina se encuentra en el estado "Add". Para conseguir esto, se establece la señal "Add" como entrada de reloj del Latch, ya que dicha señal únicamente se encuentra activa para dicho estado, y el Latch realiza la carga en los flancos de subida.

Por otra parte, en el estado "Init" debe realizarse la limpieza de este registro. Es por ello que se asigna una señal "NotClearReg" (dependiente a "Init") a la entrada "Clr" del registro acumulador, teniendo en cuenta que esta entrada es activa en bajo, al igual que "Pre" (Preset).

```
Estable <= '0', '1' after 2 ns;
NotClearReg <= Estable and not Init;

Reg_ACC: Latch8 port map(D=>Q_Adder, Clk=>Add, Pre=>'1', Clr=>NotClearReg, Q=>Q_ACC);
```

Figura 7. Instanciación de Reg_ACC

En un principio, el Latch se encuentra en un estado inestable, su salida toma valores alternantes entre el mínimo y el máximo, permaneciendo así hasta que se realiza un Clear o Preset de manera asíncrona. Como normalmente la puesta en 0 (Clear) recién se realizaría en el estado "Init", se utiliza una señal "Estable", con el objetivo de que la señal "NotClearReg" tome el valor 0 tanto al principio como también en el estado "Init" del controlador.

1.2.5 – Registro de resultado

Al modelo original planteado para la resolución de esta práctica se agregó un registro que permite mostrar el resultado final de la multiplicación una vez la operación haya terminado, mediante el uso de un Latch de 8 bits. La entrada de datos "D" es la salida "Q_ACC" del registro acumulador (resultados intermedios), pero dicha entrada se asigna a la salida "Result" únicamente en el estado "End". Por ende, se asigna la señal "Stop" a la entrada de reloj del Latch, ya que dicha señal se utiliza en valor alto para pasar desde el estado "Check" hacia "End", no pudiendo utilizar "Done" por tratarse de una salida de la entidad Multiplicador.

```
Reg_Res: Latch8 port map(D=>Q_ACC, Clk=>Stop, Pre=>'1', Clr=>NotClearReg, Q=>Result);
```

Figura 8. Instanciación de Reg_Res

De manera similar que en el registro acumulador, en este caso también se utiliza la señal "NotClearReg" para limpiar el resultado tanto al principio como al empezar una nueva multiplicación (estado "Init"), teniendo en cuenta que "Clr" y "Pre" son activas en bajo.

1.2.6 – Compuerta NOR

Como pudo observarse en la Figura 1, la señal Stop es la salida de una compuerta NOR, cuya entradas son los bits de Q_SRA (salida del registro de desplazamiento de A). La implementación de la misma se realizó mediante la operación OR a cada bit de la señal Q_SRA, y luego la operación NOT a dicho bit resultante, como se muestra a continuación.

```
-- Compuerta NOR entre SRA y la máquina de estados
Stop <= not(Q_SRA(7) or Q_SRA(6) or Q_SRA(5) or Q_SRA(4) or Q_SRA(3) or Q_SRA(2) or Q_SRA(1) or Q_SRA(0));
```

Figura 9. Asignación concurrente de la señal Stop

El código completo del multiplicador, que incluye las Figuras 2, 3, 5, 6, 7, 8 y 9, se encuentra en el archivo **Multiplicador.vhd**, anexado para su visualización y compilación. Las siguientes secciones hacen referencia al archivo **TestMultiplicador.vhd**, también anexado para poder ser simulado en el entorno Active-HDL.

2 – Testbench

Para verificar el funcionamiento del multiplicador se creó la entidad "TestMultiplicador", sin puertos. La arquitectura utilizada es "Driver", en la cual se realiza una prueba exhaustiva, pasando por todos los resultados posibles. Se declaran señales homónimas a cada puerto del componente "Multiplicador" y una señal auxiliar para verificar el resultado.

```
    -- Señales de entrada y salida del Multiplicador
    signal A, B: Bit_Vector(3 downto 0);
    signal Result: Bit_Vector(7 downto 0);
    signal STB, CLK, Done: Bit;

    -- Señal auxiliar para verificar cada operación
    signal resultado: NATURAL := 0;

begin
    -- Se instancia el multiplicador
    Multi: Multiplicador port map(STB, CLK, A, B, Result, Done);
```

Figura 10. Declaración de señales e instanciación del multiplicador.

En el caso del reloj, se le asigna un **período de 10 ns** (5 ns en bajo, 5 ns en alto) mediante el procedimiento "Clock" del paquete Utils (práctica 8), por lo que se declara el uso del mismo para la entidad de testbench como "use work.Utils.all".

Para testear todas combinaciones de los operandos A y B, se utilizan dos bucles: uno para incrementar los valores de A y otro para los valores de B, ambos desde 0 hasta 15. Las asignaciones de los operandos se consigue mediante la conversión del valor natural de la variable de iteración a número binario de 4 bits, también del paquete Utils.

```
-- Por cada valor de A
for i in 0 to 15 loop
    -- Se convierte el primer operando a binario
    A <= Convert(i, 4);

    -- Por cada valor de B
    for j in 0 to 15 loop
        -- Se convierte el segundo operando a binario
        B <= Convert(j, 4);
```

Figura 11. Conversiones a binario de las variables de iteración.

Una vez asignados los valores A y B, se inicia la multiplicación mediante la activación temporaria de la señal "STB", preferiblemente coincidiendo con el flanco de subida de reloj, y luego se espera al fin de la operación mediante: wait until Done = '1'. La verificación del resultado se realiza comparando la señal "Result" con la conversión a binario de "resultado".

```
-- Se verifica el resultado obtenido
assert Result = Convert(resultado, 8) report "Resultado incorrecto" severity FAILURE;
wait for 5 ns;
```

Figura 12. Verificación del resultado de la multiplicación.

El valor natural del resultado esperado se actualiza para cada combinación, sumando el valor del primer operando (i) al resultado anterior, o bien, reiniciando a cero para j = 0.

Aunque el testbench realiza las 256 multiplicaciones posibles, en un tiempo de aproximadamente 28500 ns para este caso particular, analizaremos únicamente los tres casos solicitados: cuando ambos operandos son iguales al valor mínimo, a ‘7’ y al máximo.

2.1 – Verificación para operandos iguales al valor mínimo

En el comienzo de la simulación, el primer caso evaluado es cuando A = B = “0000”, es decir, se realiza la multiplicación de 0 x 0. Como se puede observar en la Figura 13, el tiempo que demora esta operación es 20 ns, según la separación entre los flancos de la señal “Done”.

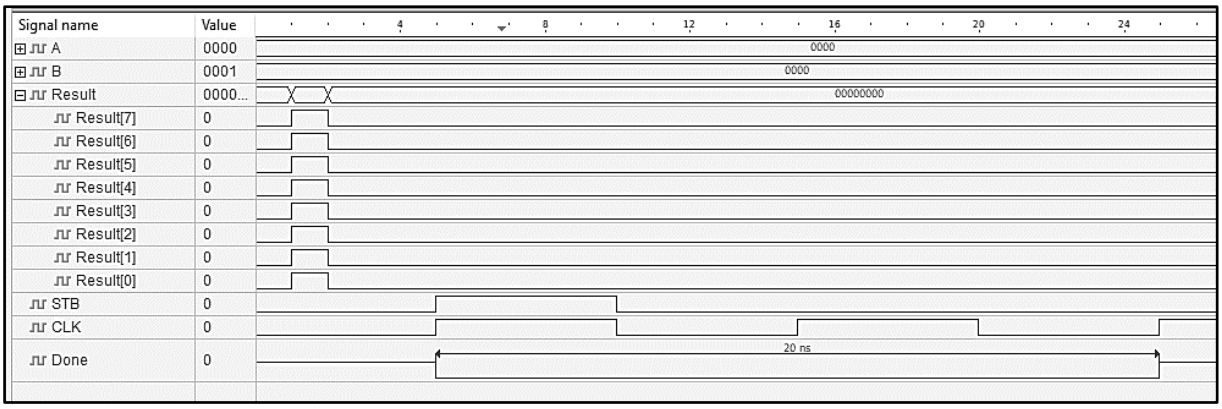

Figura 13. Intervalo de simulación para el caso mínimo de ambos operandos.

En el primer flanco de subida de reloj se inicia la multiplicación (transición al estado interno “Init” del multiplicador), en el segundo flanco se transiciona al estado “Check”, y como el valor del operando A es “0000”, en el tercer flanco se transiciona al estado “End”. Por tal motivo, como ocurren 2 períodos de reloj para la operación, en este caso demora 20 ns.

Ahora bien, la duración de la multiplicación no es proporcional al resultado, lo cual se puede demostrar analizando el caso A = “1111” y B = “0000”, con resultado cero. La duración de esta operación es 140 ns, que es 7 veces mayor que el caso mínimo. Esto se debe a que el multiplicador, en el estado interno “Check”, siempre transiciona al estado “Add” (y luego “Shift”) mientras la señal “LSB” esté activa (en este caso ocurre para los 4 desplazamientos de A hasta que Q_SRA toma el valor “00000000”). La duración total se explica sumando los 10 ns en estado “Init” + 5 x 10 ns en “Check” + 4 x 10 ns en “Add” + 4 x 10 ns en “Shift”.

La simulación del caso mencionado anteriormente puede observarse en la Figura 14.

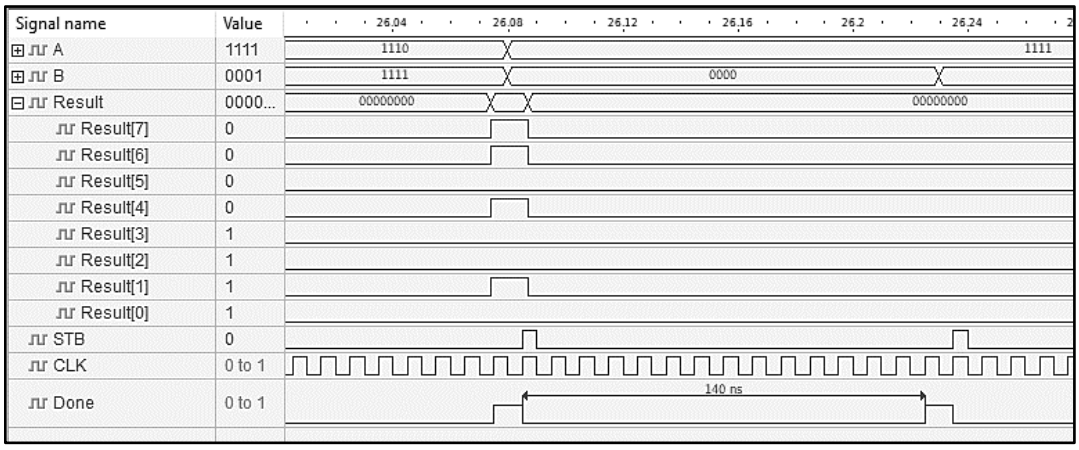

Figura 14. Intervalo de simulación para el caso A = “1111” y B = “0000”.

2.2 – Verificación para operandos iguales al valor 7

Se asigna el valor “0111”, que se trata del número 7 en binario, a los operandos A y B, y se inicia la multiplicación, esperando que el resultado sea “00110001”, el número 49 en representación binaria sin signo. Efectivamente, cuando la señal “Done” se activa nuevamente, se puede observar (Fig. 15) que se obtuvo dicho resultado, demorando 110 ns esta operación.

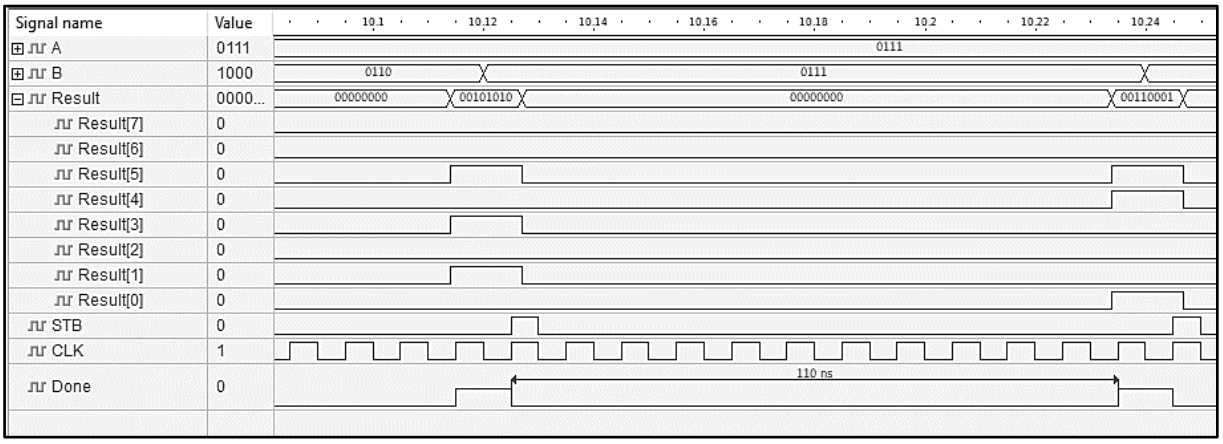

Figura 15. Intervalo de la simulación para el caso A = B = “0111”.

Para esta multiplicación en particular ocurre el siguiente proceso. Estableciendo instante cero cuando STB <= ‘1’, en los primeros 10 ns se cargan los números A y B en los registros de desplazamientos y se limpian los registros acumulador y de resultado (estado “Init”). En los siguientes 10 ns, se encuentra en estado “Check”. Como “LSB” está activo para los primeros 3 desplazamientos, se transiciona al estado “Add”, luego a “Shift” y se vuelve a “Check”, demorando 3 x 30 ns = 90 ns. Para este momento, ya no hay bits para desplazar en Reg_SRA, entonces la señal Stop está activa, y se transiciona a “End”. El tiempo total es: 10 ns de “Init” + 4 x 10 ns de “Check” + 3 x 10 ns de “Add” + 3 x 10 ns de “Shift” = 110 ns.

En la tabla a continuación se detallan los valores asignados a las salidas de los registros Reg_SRA, Reg_SRB y Reg_ACC, durante toda la multiplicación de este caso particular.

Tabla 4. Valores internos del multiplicador para el caso 7 x 7.

Instante [ns]	Est. actual	Q_SRA	Q_SRB	Q_ACC
0	Init	“0000 0111”	“0000 0111”	“0000 0000”
10	Check	“0000 0111”	“0000 0111”	“0000 0000”
20	Add	“0000 0111”	“0000 0111”	“0000 0111”
30	Shift	“0000 0011”	“0000 1110”	“0000 0111”
40	Check	“0000 0011”	“0000 1110”	“0000 0111”
50	Add	“0000 0011”	“0000 1110”	“0001 0101”
60	Shift	“0000 0001”	“0001 1100”	“0001 0101”
70	Check	“0000 0001”	“0001 1100”	“0001 0101”
80	Add	“0000 0001”	“0001 1100”	“0011 0001”
90	Shift	“0000 0000”	“0011 1000”	“0011 0001”
100	Check	“0000 0000”	“0011 1000”	“0011 0001”
110	End	“0000 0000”	“0011 1000”	“0011 0001”

Como se puede observar en la Tabla 4, las sumas sucesivas realizadas son, en primer lugar, 0 + 7 = 7, es decir, el valor del operando B, luego 7 + 14 = 21, que es el resultado anterior sumado el doble del operando B, y por último 21 + 28 = 49, donde 28 es el cuádruple de B.

2.3 – Verificación para operandos iguales al valor máximo

El último caso de la simulación consiste en la multiplicación de ambos operandos iguales al máximo valor representable de 4 bits en binario sin signo, es decir, A = B = “1111”, el valor 15 en decimal. Se activa la señal “STB” para iniciar esta operación y se espera a la activación de la señal “Done”, con una demora predecible de 140 ns para el reloj utilizado, según lo analizado en la sección 2.1 para el caso 15 x 0. La simulación se observa en la Fig. 16.

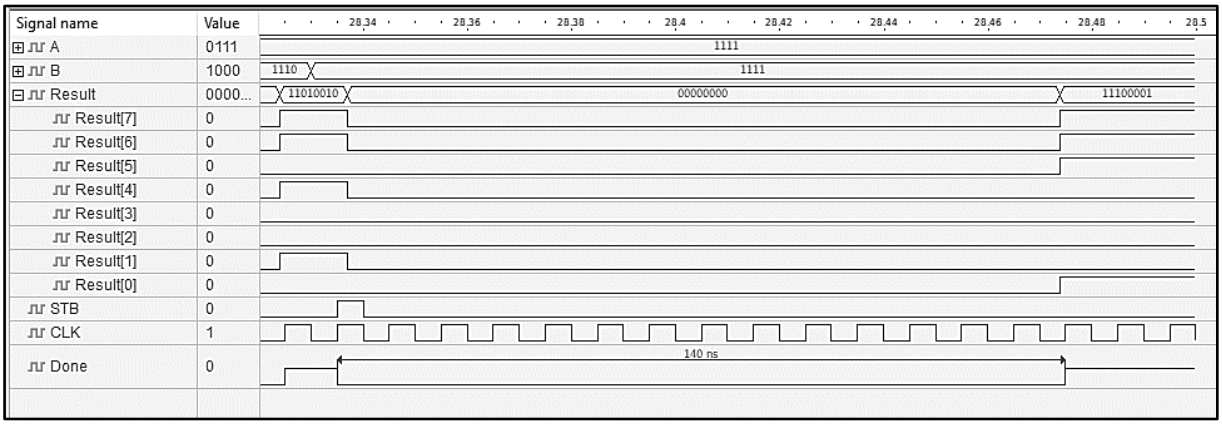

Figura 16. Intervalo de simulación para el caso A = B = “1111”.

El resultado de la operación, “11100001” (15 x 15 = 225 en decimal), es correcto.

En la tabla a continuación se detallan los valores asignados a las salidas de los registros Reg_SRA, Reg_SRB y Reg_ACC, durante toda la multiplicación de este caso particular.

Tabla 5. Valores internos del multiplicador para el caso 15 x 15.

Instante [ns]	Est. actual	Q_SRA	Q_SRB	Q_ACC
0	Init	“0000 1111”	“0000 1111”	“0000 0000”
10	Check	“0000 1111”	“0000 1111”	“0000 0000”
20	Add	“0000 1111”	“0000 1111”	“0000 1111”
30	Shift	“0000 0111”	“0001 1110”	“0000 1111”
40	Check	“0000 0111”	“0001 1110”	“0000 1111”
50	Add	“0000 0111”	“0001 1110”	“0010 1101”
60	Shift	“0000 0011”	“0011 1100”	“0010 1101”
70	Check	“0000 0011”	“0011 1100”	“0010 1101”
80	Add	“0000 0011”	“0011 1100”	“0110 1001”
90	Shift	“0000 0001”	“0111 1000”	“0110 1001”
100	Check	“0000 0001”	“0111 1000”	“0110 1001”
110	Add	“0000 0001”	“0111 1000”	“1110 0001”
120	Shift	“0000 0000”	“1111 0000”	“1110 0001”
130	Check	“0000 0000”	“1111 0000”	“1110 0001”
140	End	“0000 0000”	“1111 0000”	“1110 0001”

Como se puede observar en la Tabla 5, las sumas sucesivas realizadas son:

- 0 + 15 = 15, es decir, el valor del operando B.
- 15 + 30 = 45, es decir, el resultado anterior sumado el doble del operando B.
- 45 + 60 = 105, es decir, el resultado anterior sumado el cuádruple del operando B.
- 105 + 120 = 225, es decir, el resultado anterior sumado el óctuple del operando B.

2.4 – Conclusiones

El multiplicador diseñado a partir de los componentes vistos en las prácticas anteriores funciona correctamente para todas las combinaciones posibles de los operandos A y B. Sin embargo, en este diseño no aplica la propiedad conmutativa de los operandos si lo analizamos con respecto al tiempo que demora en realizarse la operación. Considerando la utilización de un reloj de período 10 ns y un mando de inicio coincidente con un flanco de subida de reloj, se puede observar que la multiplicación tarda un tiempo mínimo de 20 ns únicamente para los casos donde el operando A = “0000” (valor mínimo), y la espera va aumentando a medida que se incrementa A, hasta un máximo de 140 ns para A = “1111” (valor máximo de 4 bits).

A su vez, el **tiempo** de la multiplicación es independiente del operando B. En los análisis realizados y en la sección 1.2.1, se pudo observar que en este esquema de sumas sucesivas, la cantidad de sumas de múltiplos de B (transiciones) depende de los bits del operando A.