Informe de Resolución de la Práctica 10

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

Introducción

En el informe correspondiente a la práctica 9 se explicó el funcionamiento de un multiplicador utilizando una arquitectura de sumas sucesivas, compuesta por una máquina de estado finito, un sumador, registros de desplazamiento de bits y *latches*. Además, se analizaron 3 casos de prueba: la multiplicación para el menor número representable (0x0), el mayor número representable (de 4 bits, 15x15) y un caso intermedio particular (7x7).

Para la presente resolución, se efectuarán mediciones del tiempo que demora el multiplicador en realizar la multiplicación utilizando una frecuencia de reloj establecida, hallando y justificando los valores mínimo y máximo obtenidos. También se explicará brevemente la implementación de una interfaz de usuario amigable para este sistema.

1 – Tiempo de demora de multiplicación

Para su funcionamiento, el multiplicador requiere de una señal de entrada de reloj, utilizada para avanzar entre estados posibles en los flancos de bajada. Para esta práctica, se solicita la aplicación de un reloj de frecuencia **f = 50 MHz**, es decir, período T = 1/f = 20 ns.

La implementación de una señal binaria alternante de dicho período se puede conseguir mediante el uso del procedimiento **Clock** del paquete Utils, como se muestra a continuación:

```
-- Se asigna el período del reloj
Clock(CLK, 10 ns, 10 ns);
```

Figura 1. Invocación al procedimiento Clock para la señal CLK.

El tiempo de demora de la multiplicación no depende únicamente de la frecuencia del reloj empleado, sino también del valor del primer operando “A”, cuyos bits se desplazan hacia la derecha en el multiplicador, culminando la operación cuando todos sus bits sean ceros.

1.1 – Medición del tiempo

Para el cálculo de la demora asociada a cada valor del operando “A”, se realizó la simulación ingresando el valor correspondiente del primer operando y un valor cualquiera del segundo operando, ya que “B” únicamente influye en el valor del producto o resultado.

En los resultados mostrados a continuación, la medición se realiza desde el momento en que se activa la señal de Strobe (STB = ‘1’), en simultáneo a un flanco de subida de reloj, hasta la finalización de la multiplicación, indicada por la activación de la señal Done.

Tabla 1. Resultados de simulación de todos los valores posibles de “A”

Valor de “A” binario	Valor de “A” decimal	Demora del producto [ns]
0000	0	40
0001	1	100
0010	2	140
0011	3	160
0100	4	180
0101	5	200
0110	6	200
0111	7	220
1000	8	220
1001	9	240
1010	10	240
1011	11	260
1100	12	240
1101	13	260
1110	14	260
1111	15	280

Como se puede observar en la Tabla 1, la multiplicación “más rápida” se consigue cuando el primer operando “A” es el menor número representable (“0000”), con una demora de **40 ns**; mientras la multiplicación “más lenta” se consigue cuando “A” es el mayor número representable que admite el multiplicador (“1111”), con una demora de **280 ns**. Las simulaciones correspondientes a ambos casos descriptos se muestran a continuación:

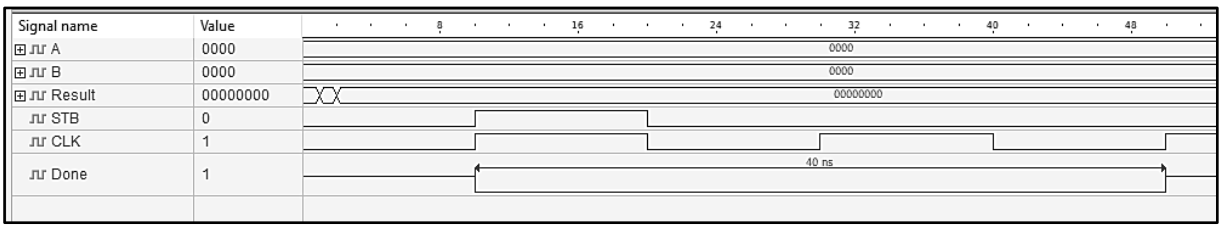


Figura 2. Simulación de un caso con demora mínima.

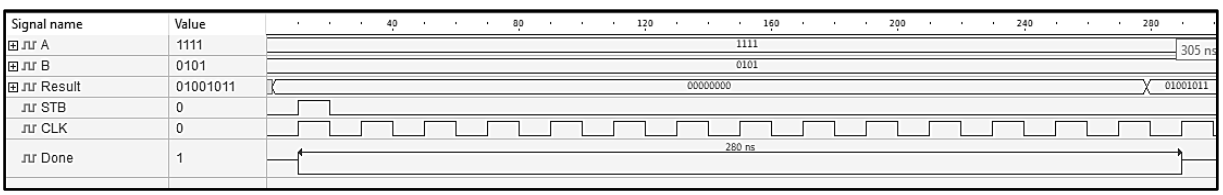


Figura 3. Simulación de un caso con demora máxima.

A partir de los resultados, se pueden deducir los siguientes aspectos de la demora:

- **NO** es directamente proporcional al valor de “A”, demostrado cuando “A” es 12, la operación se realiza en un tiempo menor que cuando “A” vale 11, en decimal.
- Es mayor cuanto más “N” bits se requieran para representar a “A” en binario, ya que se desplazará “N” veces los bits de “A” hacia la derecha (estado “Shift” del multiplicador).
- Es mayor cuantos más bits “1” haya en el operando “A” en binario, dicha cantidad coincide con las veces que el bit menos significativo es 1 (transición al estado “Add”).

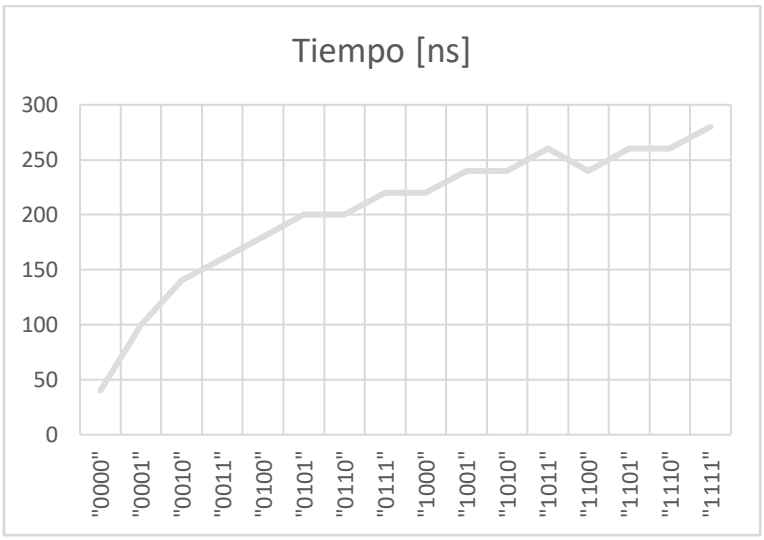
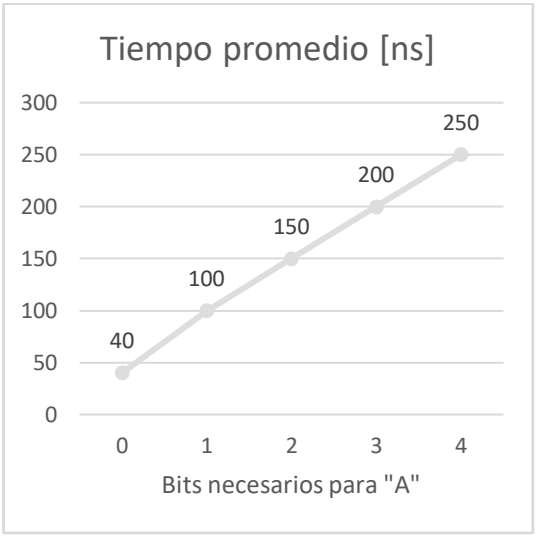
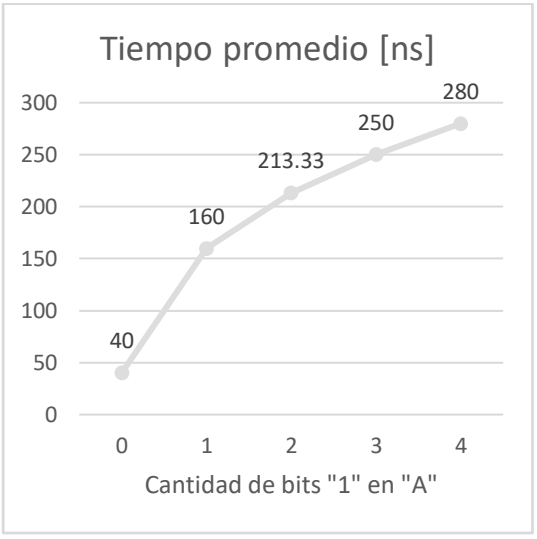


Figura 4. Comparación entre módulo de “A” y la demora de la operación.





Figuras 5 y 6. Comparación entre atributos de “A” en binario y la demora de la operación.

Las observaciones anteriores justifican que el tiempo de demora, por ejemplo, cuando el primer operando es “1001”, “1010” o “1100”, sea exactamente el mismo.

Recordando el funcionamiento de la máquina de estado finito del multiplicador, se puede llegar a la siguiente expresión para calcular el tiempo de cualquier multiplicación posible, donde **M** es la cantidad de bits "1" de "A", **N** la cantidad de bits de "A" y **T** el período de reloj.

$$t_{total} = t_{init} + t_{check} + M * \left(t_{shift} + t_{add} + t_{check}\right) + (N - M) * (t_{add} + t_{check})$$

$$t_{total} = t_{init} + t_{check} + M * t_{shift} + N * (t_{add} + t_{check})$$

$$t_{total} = 2\,T + M\,T + 2\,N\,T = T\,(2N + M + 2)$$

De esta manera, se comprueban los tiempos de demora mínimo y máximo:

- Cuando M = N = 0, es decir, "A" = "0000", se obtiene $t_{total} = 20\ ns * 2 = 40\ ns$
- Cuando M = N = 4, es decir, "A" = "1111", se obtiene $t_{total} = 20\ ns * 14 = 280\ ns$

1.2 – Tiempo promedio de demora

El multiplicador admite operandos de 4 bits, por lo que existen 256 combinaciones de "A" y "B" que pueden ingresarse al mismo. Sin embargo, se mencionó en la sección anterior que para un mismo valor de "A" se obtiene la misma demora independientemente de "B", por lo que el conjunto de combinaciones de interés se reduce a 16.

Utilizando los resultados de la Tabla 1, se obtiene una demora media de **202,5 ns**.

$$t_{prom} = \frac{1}{16} \sum_{i=0}^{15} t_{demora}(i) = \frac{3240\ ns}{16} = 202{,}5\ ns$$

2. Interfaz de usuario

Para la impresión de texto y lectura de números por consola se declara el uso de la librería "STD.textio.all" en el archivo de testbench. Los mensajes a mostrar son declarados como constantes en la arquitectura, como se muestra en la porción de código a continuación.

```
-- Mensajes personalizados para mostrar en consola
constant msj_bienvenida:string :=   "Bienvenido. Este sistema es un multiplicador de números enteros.";
constant msj_restriccion:string :=  "Unicamente se permite ingresar números entre 0 y 15";
constant msj_pide_num1:string :=    "Por favor, ingrese el primer operando";
constant msj_pide_num2:string :=    "Ahora, ingrese el segundo operando";
constant msj_error_input:string :=  "Entrada inválida. Ingrese un número entre 0 y 15";
constant msj_calculando:string :=   "Resolviendo la multiplicación ";
constant msj_resultado:string :=    "El resultado es ";
```

Figura 7. Declaración de constantes en la arquitectura de Interfase.

En el proceso, se declaran dos variables tipo "line", de modo que sirvan de búffer para entrada y salida respectivamente, una variable tipo "boolean" para identificar entradas inválidas, y variables tipo "integer" y "natural" para leer y mostrar números en decimal.

En la lectura, se intenta asignar la entrada a un “integer”, ya que, si se utilizara “natural”, el sistema reporta un error por asignación fuera de rango al ingresar números negativos.

```
Stimulus : process
    -- Declaración de variables
    variable lineaOut,lineaIn:line; -- Linea buffer para entrada y salida
    variable exito:boolean;         -- Indicador de asignación de lectura exitoso
    variable num1,num2:integer;     -- Número entero leido por teclado
    variable resultado:Natural;     -- Resultado (producto) en representación decimal
```

Figura 8. Declaración de variables en el proceso de multiplicación.

Sin embargo, en la lectura es necesario controlar que el número ingresado se encuentre en el rango de valores que admite el multiplicador, entre 0 y 15, de la siguiente forma:

```
-- Se solicita y lee el primer operando
write(lineaOut, msj_pide_num1);
writeline(output, lineaOut);

loop
    readline(input, lineaIn);
    read(lineaIn, num1, exito);
    -- Se verifica que sea un numero entre 0 y 15
    if (exito and num1 >= 0 and num1 < 16) then
        A <= Convert(num1,4);
        exit;
    else
        write(lineaOut, msj_error_input);
        writeline(output, lineaOut);
    end if;
end loop;
```

Figura 9. Bucle de solicitud de operando al usuario.

En caso de que el usuario ingrese letras o números fuera del rango permitido, se muestra un mensaje de aviso y se vuelve a solicitar el número hasta que la operación sea exitosa. Una vez ingresados ambos operandos, se muestra la operación que se está realizando y, luego, el resultado de dicha operación. Mediante **integer’image()** se convierte el número a texto.

```
-- Se muestra qué operación se está realizando y se espera al resultado
write(lineaOut, msj_calculando & integer'image(num1) & "x" & integer'image(num2));
writeline(output, lineaOut);
wait until Done = '1';

-- Se informa el resultado
resultado:= Convert(Result);
write(lineaOut, msj_resultado & integer'image(resultado));
writeline(output, lineaOut);
```

Figura 10. Espera y muestra del resultado en pantalla.

El código completo para la interfaz de usuario se encuentra anexado para su compilación y simulación bajo el nombre **“TestMultiplicadorIO.vhd”**. No se realizó ninguna modificación al archivo correspondiente a la entidad multiplicador.