Memoria

Sistemas Electrónicos para la Automatización

Docente: *Jorge Romero Sánchez*

Práctica: Señales Resueltas

***Jorge Benavides Macías**: jorge2@uma.es*

Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Contenido

Lista de figuras

Lista de códigos

1. Especificaciones

- Explicar en detalle en un estudio previo, cómo funciona el código suministrado (para la memoria).
- Generar un proyecto en Vivado para ejecutar la simulación.
- ¿Qué significan las líneas: “Esta línea no compilará”?.
- ¿En qué tipo de circuitos se puede utilizar la lógica resuelta con múltiples drivers?
- Elaborar un informe que ilustre el procedimiento de diseño, simulación, respondiendo a las 2 cuestiones. Podéis utilizar capturas de pantalla de los cronogramas de simulación.

2. Estudio teórico

2.1. Sobre la asignación concurrente de señales en VHDL

```
1  architecture concurrencia of memoria is
2    Signal s : Mi_Logica_Resuelta; -- tipo
3    resuelto
4    Begin
5      -- 2 asignaciones concurrentes. 2 fuentes.
6      s <= a xor b after 1 ns;
7      s <= b nor x after 2 ns;
8  end;
```

Código 1: Asignación concurrente en VHDL

La asginación concurrente permite asignar a una señal un valor desde múltiples fuentes, que a nivel de código se representa como una asignación en dos procesos distintos[1] (ver Código 1); un símil dentro de la ingeniería es la confluencia de dos señales en el mismo nodo como en la Figura 1, en cualquier caso esto plantea un problema importante y es **¿Cómo gestionamos la entrada de dos fuentes distintas?**

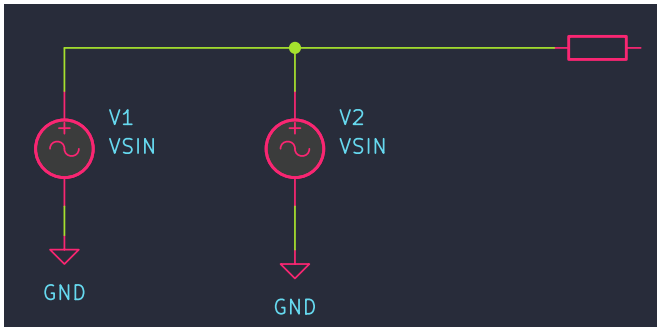

Figura 1: Circuito eléctrico con dos fuentes de señal seno.

Para solucionar este problema en electrónica analógica, se utilizan las *Leyes de Kirchhoff*. Estas leyes son una aplicación o función que permite obtener una salida para 2 o más entradas.

[1] No debe confundirse con la asignación durante el mismo proceso que es, en esencia, la modificación del driver.

Para gestionar la concurrencia en VHDL, es necesario crear una tabla o función que determine la salida de la señal. Esta función se conoce como **función de resolución**. Sin embargo, no existe una función de resolución predefinida en VHDL. Por lo tanto, es correcto hablar de lógica resuelta y no resuelta.

2.2. Lógica Resuelta y no resuelta

Una lógica resuelta es aquella que cumple las siguientes propiedades:

- **Conmutatividad**:$f(a,b) = f(b,a)$
- **Asociatividad**: $f[(a,b),c] = f[a,f(b,c)]$
- **Elemento neutro** $f(e,a) = f(a,e) = a$

Estas propiedades deben cumplirse dentro de una tabla, como la de la Figura 2, que representa la función de resolución. Esta tabla contiene 9 valores que pertenecen a la *std_logic_1164*, no obstante, la función de resolución puede ser todo lo grande o pequeña que necesitemos, depende del tipo que utilicemos (ver Código 2).

```
      ---------------------------------------------
      | 'U'  'X'  '0'  '1'  'Z'  'W'  'L'  'H'  '-' |
      ---------------------------------------------
'U' | 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U', 'U' |
'X' | 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' |
'0' | 'U', 'X', '0', 'X', '0', '0', '0', '0', 'X' |
'1' | 'U', 'X', 'X', '1', '1', '1', '1', '1', 'X' |
'Z' | 'U', 'X', '0', '1', 'Z', 'W', 'L', 'H', 'X' |
'W' | 'U', 'X', '0', '1', 'W', 'W', 'W', 'W', 'X' |
'L' | 'U', 'X', '0', '1', 'L', 'W', 'L', 'W', 'X' |
'H' | 'U', 'X', '0', '1', 'H', 'W', 'W', 'H', 'X' |
'-' | 'U', 'X', 'X', 'X', 'X', 'X', 'X', 'X', 'X' |
```

Figura 2: Tabla de lógica no resuelta

3. Diseño

3.1. Paquete Mis_Tipos

En el Código 2 se definen dos tipos nuevos:

- *Mi_Logica* contiene dos valores sintetizables (“0” y “1”) y dos no sintetizables (“X” y “U”) que para esta práctica es perfecto.
- *Mi_Logica_Vector* Este tipo es un arreglo (array) de elementos de tipo *Mi_Logica*, y su rango está determinado por un tipo subyacente “Natural” cuyo rango no está especificado.

A continuación se presenta la función “Resuelve” y definimos un nuevo subtipo. En el *body* del *package* se define el cuerpo de la función “Resuelve” que cuenta con un tipo *Tabla* y la operación que realizará la lógica resuelta. Dicha tabla cumple los requisitos expuestos en la Subsección 2.2.

Dado que realizamos una indexación recursiva en la “Tabla” operación con los valores de *Mi_Logica_Resuelta* se requiere de un valor inicial, en este caso el valor de inicialización es *U*.

```
Package Mis_Tipos is
-- En este paquete defino mis tipos de datos constantes
-- y señales globales, subprogramas, etc.

  type Mi_Logica is ('U', '0', '1', 'X');  -- La lógica base no resuelta.
  type Mi_Logica_Vector is array(Natural range <>) of Mi_Logica;

  -- La función de resolución cumple la Conmutativa, Asociativa y Elemento Neutro.
  function Resuelve(x : Mi_Logica_Vector)
    return Mi_Logica;

  subtype Mi_Logica_Resuelta is Resuelve Mi_Logica;  -- El tipo resuelto como

end Mis_Tipos;  -- subtipo del tipo base.

package body Mis_Tipos is   -- Aquí defino el cuerpo de los subprogramas.

  function Resuelve(x : Mi_Logica_Vector) return Mi_Logica is
    type Tabla is array(Mi_Logica range 'U' to 'X', Mi_Logica range 'U' to 'X')

      of Mi_Logica;
                                -- 'U'  '0'  '1'  'X'
                                   ----------------------
    constant Operacion : Tabla := (('U', '0', '1', 'X'),   --|'U'
                                   ('0', '0', 'X', 'X'),   --|'0'
                                   ('1', 'X', '1', 'X'),   --|'1'
                                   ('X', 'X', 'X', 'X'));  --|'X'
    variable Parcial : Mi_Logica := 'U';  -- El valor inicial es importante:

  begin  -- Siempre es el Elemento Neutro.

    for i in x'Range loop
      Parcial := Operacion(Parcial, x(i));  -- Recursión con propiedades:
              -- Asociativa:f(f(x,y),z)=f(x,f(y,z))
    end loop;  -- Conmutativa: f(x,y)=f(y,x)

    return Parcial;

  end Resuelve;

  end Mis_Tipos;

```

Código 2: Paquete Mis_tipos: Implementa un tipo resuelto.

3.2. Test Bench

El test bench es la puesta en práctica de las señales resueltas y de la lógica implementada en el Código 2. Definimos dos señales de tipo *Mi_Logica* y *Mi_Logica_Resuelta* y dos procesos para realizar asignaciones (como se ha explicado en la Subsección 2.1)

```
library IEEE;
use work.Mis_Tipos.all;  -- La biblioteca lógica WORK es implícita en cada diseño.

entity Mi_Logica_TB is   -- Por tanto todos mis packages pertenecen a ella.

end Mi_Logica_TB;

architecture Comportamiento of Mi_Logica_TB is

  signal s_sr     : Mi_Logica          := 'U';  -- Señales sin resolver (sr).
  signal s_r      : Mi_Logica_Resuelta := 'U';  -- Señales resueltas (r).

begin

  Bus_End_Point_1 : process  -- Fuente 1 para la señal resuelta (genera un driver).
  begin
   -- ¡Esta línea no compilará!
    s_sr <= '0' after 5 ns, '1' after 10 ns, 'X' after 12 ns;
    s_r  <= '0' after 5 ns, '1' after 10 ns, 'X' after 20 ns;
```

```
20      wait;
21    end process Bus_End_Point_1;
22
23    Bus_End_Point_2 : process  --Fuente 2 para la señal resuelta (otro driver).
24    begin
25      -- ¡Esta línea no compilará¡!
26      s_sr <= '0' after 5 ns, '1' after 12 ns, 'X' after 15 ns;
27      s_r  <= '0' after 5 ns, '1' after 12 ns, 'X' after 15 ns;
28      wait;
29    end process Bus_End_Point_2;
30
31  end Comportamiento;
```

Código 3: Test bench original

Para comprobar el correcto funcionamiento de las funciones de resolución en VHDL, se llevan a cabo dos simulaciones. La primera simulación puede no finalizar correctamente debido a un error de no resolución del tipo. La segunda simulación puede funcionar sin problemas.

4. Simulación abortada

Esta simulación no finalizará con éxito ya que el sistema no tiene la información necesaria para determinar cómo combinar las entradas para obtener la salida, es decir no está resuelta.

4.1. Test Bench

```
1   architecture Comportamiento of Mi_Logica_TB is
2
3     signal s_sr     : Mi_Logica          := 'U';  -- Señales sin resolver (sr).
4     signal s_r      : Mi_Logica_Resuelta := 'U';  -- Señales resueltas (r).
5
6   begin
7
8     Bus_End_Point_1 : process  -- Fuente 1 para la señal resuelta (genera un driver).
9     begin
10     -- ¡Esta línea no compilará!
11     s_sr <= '0' after 5 ns, '1' after 10 ns, 'X' after 12 ns;
12      s_r  <= '0' after 5 ns, '1' after 10 ns, 'X' after 20 ns;
13      wait;
14    end process Bus_End_Point_1;
15
16    Bus_End_Point_2 : process  --Fuente 2 para la señal resuelta (otro driver).
17    begin
18      -- ¡Esta línea no compilará¡!
19     s_sr <= '0' after 5 ns, '1' after 12 ns, 'X' after 15 ns;
20      s_r  <= '0' after 5 ns, '1' after 12 ns, 'X' after 15 ns;
21      wait;
22    end process Bus_End_Point_2;
23
24  end Comportamiento;
```

Código 4: Fragmento del test bench abortado

4.2. Error

El error *[XSTM 43-3249]* describe perfectamente lo que ocurre, indica la línea en la que está el “problema” y el error encontrado, que en este caso, es lo esperado en una señal no resuelta con múltiples drivers para *s_sr*.

```
ERROR: [XSIM 43-3249] File C:/Users/izana/Documents/GitHub/SEA/practical/practical.srcs/sim_1/new/Mi_Logica_TB.vhd, line 9. Unresolved signal "s_sr" is multiply driven.
INFO: [USF-XSim-69] 'elaborate' step finished in '1' seconds
INFO: [USF-XSim-99] Step results log file:'C:/Users/izana/Documents/GitHub/SEA/practical/practical.sim/sim_1/behav/xsim/elaborate.log'
ERROR: [USF-XSim-62] 'elaborate' step failed with error(s). Please check the Tcl console output or 'C:/Users/izana/Documents/GitHub/SEA/practical/practical.sim/sim_1/behav/xsim/elaborate.log' file for mor
ERROR: [Vivado 12-4473] Detected error while running simulation. Please correct the issue and retry this operation.
ERROR: [Common 17-39] 'launch_simulation' failed due to earlier errors.
```

Figura 3: Error de simulación

5. Simulación sin errores

Esta simulación es correcta ya que se ha comentado una de las dos asignaciones no resueltas, por lo tanto, hemos eliminado el problema de resolución.

5.1. Test Bench

```
1  architecture Comportamiento of Mi_Logica_TB is
2
3    signal s_sr     : Mi_Logica          := 'U';  -- Señales sin resolver (sr).
4    signal s_r      : Mi_Logica_Resuelta := 'U';  -- Señales resueltas (r).
5
6  begin
7
8    Bus_End_Point_1 : process  -- Fuente 1 para la señal resuelta (genera un driver).
9    begin
10    -- ¡Esta línea no compilará!
11    s_sr <= '0' after 5 ns, '1' after 10 ns, 'X' after 12 ns;
12     s_r  <= '0' after 5 ns, '1' after 10 ns, 'X' after 20 ns;
13     wait;
14   end process Bus_End_Point_1;
15
16   Bus_End_Point_2 : process  --Fuente 2 para la señal resuelta (otro driver).
17   begin
18     -- ¡Esta línea no compilará¡!
19   -- s_sr <= '0' after 5 ns, '1' after 12 ns, 'X' after 15 ns;
20     s_r  <= '0' after 5 ns, '1' after 12 ns, 'X' after 15 ns;
21     wait;
22   end process Bus_End_Point_2;
23
24 end Comportamiento;
```

Código 5: Fragmento del test bench simulado

5.2. Cronograma de simulación

En el cronograma de la simulación de la Figura 4, se observa como se realizan correctamente las asignaciones.

Ambas señales cuentan con un valor indefinido que va cambiado entre *0*, *1* y *X*, según la función de resolución.

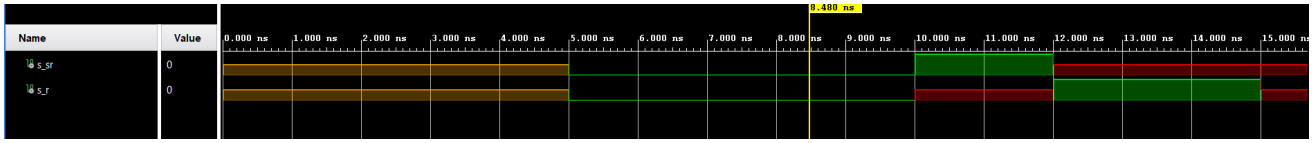

Figura 4: Cronograma de simulación

6. Cuestiones

- **¿Qué significan las líneas: “Esta línea no compilará”?**

  Estas líneas predicen el error que aparecerá cuando se realiza una asignación múltiple en una señal de lógica no resuelta, además de que la simulación abortará.

- **¿En qué tipo de circuitos se puede utilizar la lógica resuelta con múltiples drivers?**

  El uso de una lógica de resolución es necesaria en aquellos circuitos en los que se comunican distintos dispositivos simultáneamente entre ellos porque cada dispositivo conectado al bus de comunicaciones es una fuente y requiere de una gestión, por ejemplo, el bus I2C o el bus CAN.