Instituto Politécnico Nacional

Centro de Investigación en Computación

Lenguajes de descripción de hardware

Tarea 7 - FOR - GENERATE

Profesor:

M. en C. Osvaldo Espinosa Sosa

Por:

Ing. Ricardo Aldair Tirado Torres

Ciudad de México, 20 de mayo de 2024

Tabla de contenido

1. Objetivos

- Implementar la estructura *for-generate* en VHDL y Verilog, para la instanciación de hardware previamente declarado.
- Indagar acerca del uso del bloque *for-generate* en el lenguaje de Verilog, así como entender su sintaxis.
- Diferenciar la sintaxis de la estructura *for-generate* en ambos lenguajes.

2. Arreglo de compuertas OR, AND y XOR en VHDL

Actividad 1

Compilar los dos códigos vistos en clase (VHDL) y observar el resultado con el visor RTL.

La visualización RTL del arreglo de compuertas OR (primer código), descrito en VHDL, se muestra en la Figura 1. La implementación se hace instanciando 8 veces al módulo denominado "MyOr2" (debido a que las dos entradas son de 8 bits), que simplemente es una compuerta OR de 1 bit (ver Figura 2). Las simulaciones se visualizan en la Figura 3, en donde se muestra que este arreglo de compuerta OR opera de manera correcta.

En los Anexos se localiza la descripción en VHDL del primer código. Primeramente se declaran las librerías a utilizar y las señales de entrada y salida. Ahora bien, en la zona declarativa se describe un componente denominado "MyOr2", declarando sus entradas y salida. En la zona de la arquitectura, se utiliza a la estructura *for-generate* para instanciar al componente descrito anteriormente. Algunos puntos importantes son que el ciclo *generate* se denomina G0, mientras que las instancias llevan la nomenclatura U0, además, se usa una variable de iteración denominada "n" pero no se declara antes de comenzar el ciclo de instancias de hardware. Finalmente, después de declarar la arquitectura del módulo principal, se describe a la entidad "MyOr2", declarando nuevamente las librerías a utilizar, las señales de entrada y salida y el comportamiento dentro de la arquitectura.

La visualización RTL del arreglo de compuertas OR, AND y XOR (segundo código), descrito en VHDL, se muestra en la Figura 4. La implementación se hace nuevamente instanciando 8 veces al módulo denominado "MyOr2", no obstante, se agregan 4 instancias de compuertas AND para los 4 bits menos significativos de las entradas A y B, y otras 4 instancias de compuertas XOR para los 4 bits más significativos (ver Figura 5). Las simulaciones se visualizan en la Figura 6, en donde se muestra que este arreglo de compuertas OR, AND y XOR opera de manera correcta.

En los Anexos se localiza la descripción en VHDL del segundo código. Así como en el primer código se declaran las librerías a utilizar, las señales de entrada y salida y en la zona declarativa se describe un componente denominado "MyOr2", declarando sus entradas y salida. En la zona de la arquitectura, se utilizan 3 estructuras *for-generate* para instanciar al componente

descrito anteriormente, junto con las compuertas AND y XOR, según sea el caso. Algunos puntos importantes son que los ciclos *generate* se denominan G1, G2 y G3, las instancias de "MyOr2" llevan la nomenclatura U0, se usa una variable de iteración denominada "n" (no declarada antes de las iteraciones) y se utilizan dos estructuras *if* para separar a los 4 bits más y menos significativos. Finalmente, después de declarar la arquitectura del módulo principal, se describe a la entidad "MyOr2", declarando nuevamente las librerías a utilizar, las señales de entrada y salida y el comportamiento dentro de la arquitectura.

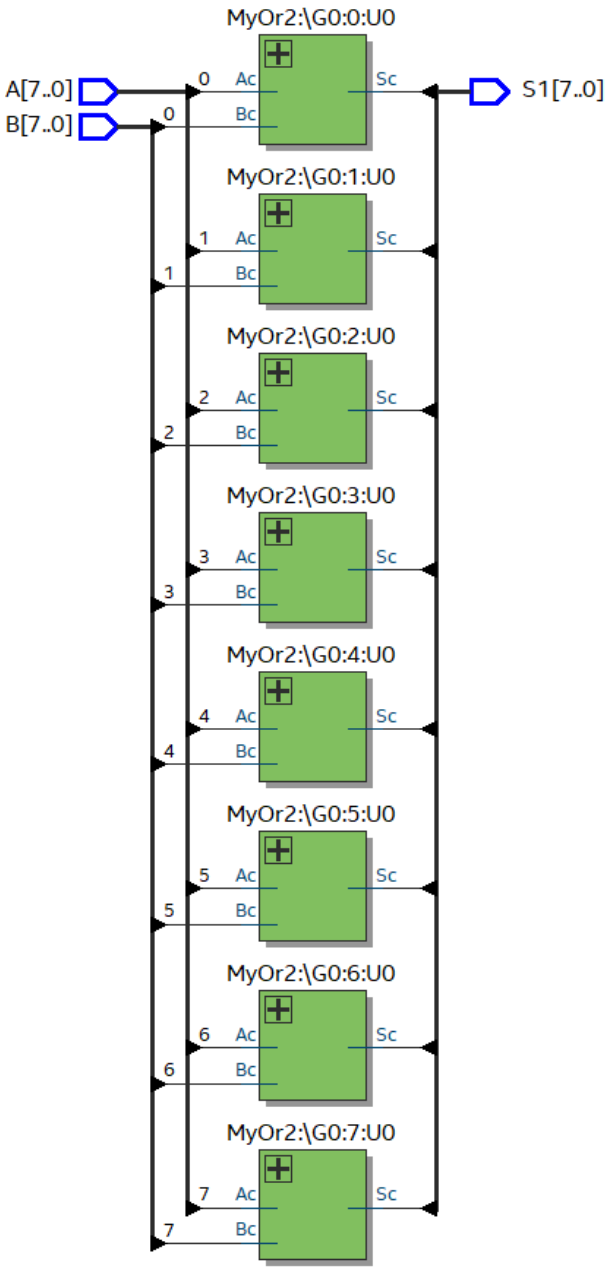

Figura 1: Diagrama RTL del arreglo de compuertas OR, descrito en VHDL.

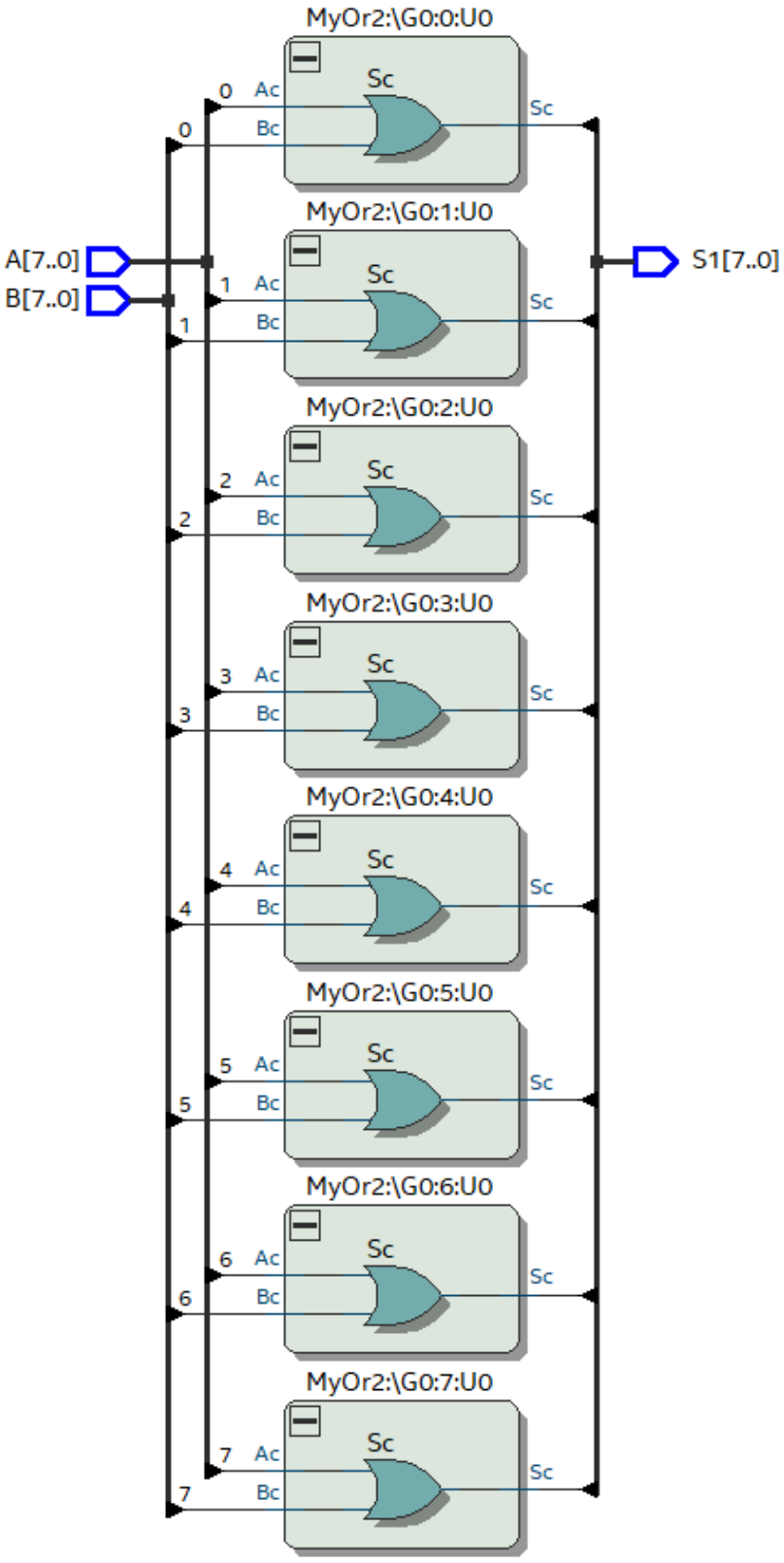

Figura 2: Diagrama RTL del arreglo de compuertas OR, descrito en VHDL (vista interna de las instancias).

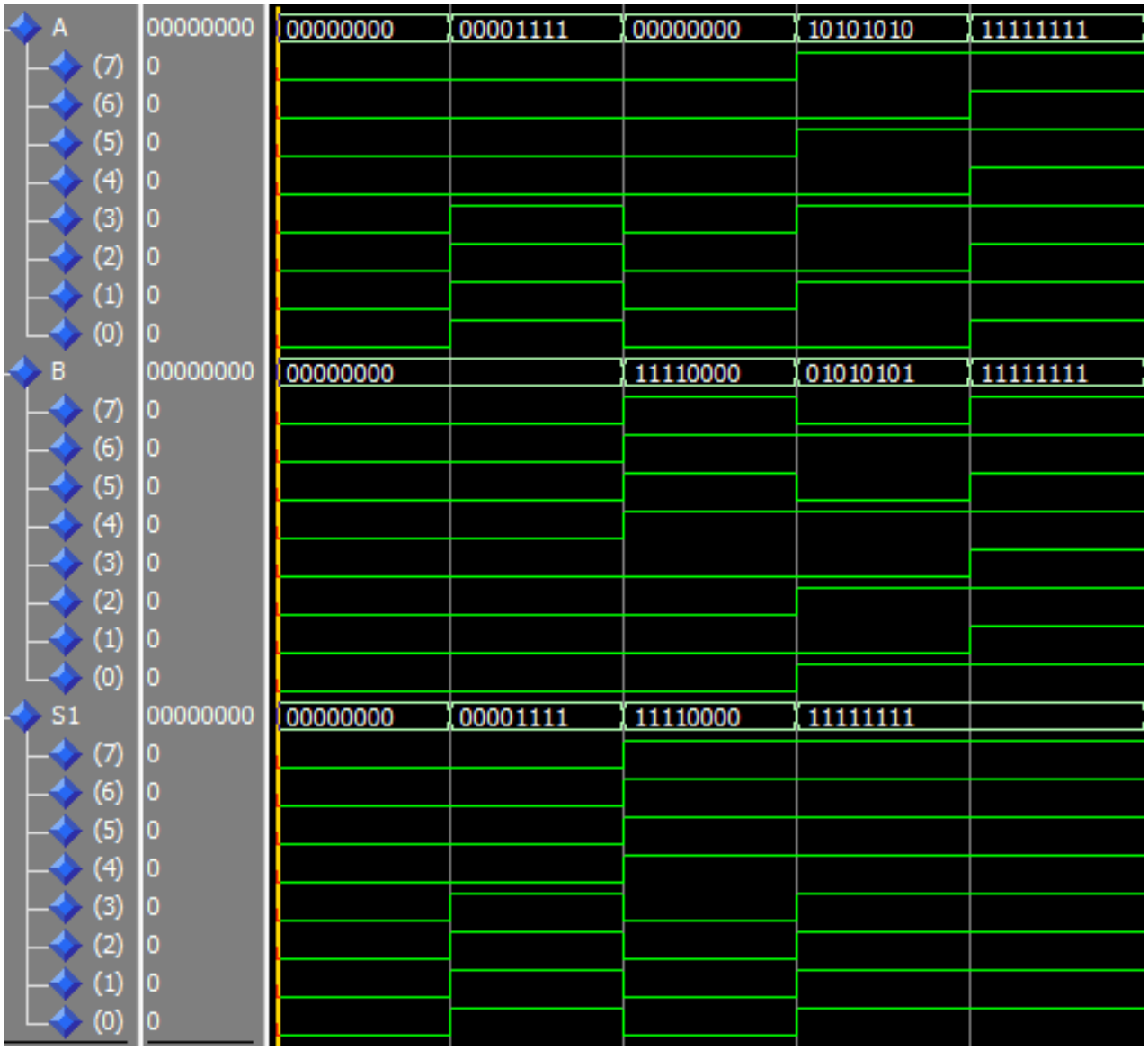

Figura 3: Simulación del arreglo de compuertas OR, descrito en VHDL, con el visor de formas de onda de ModelSim.

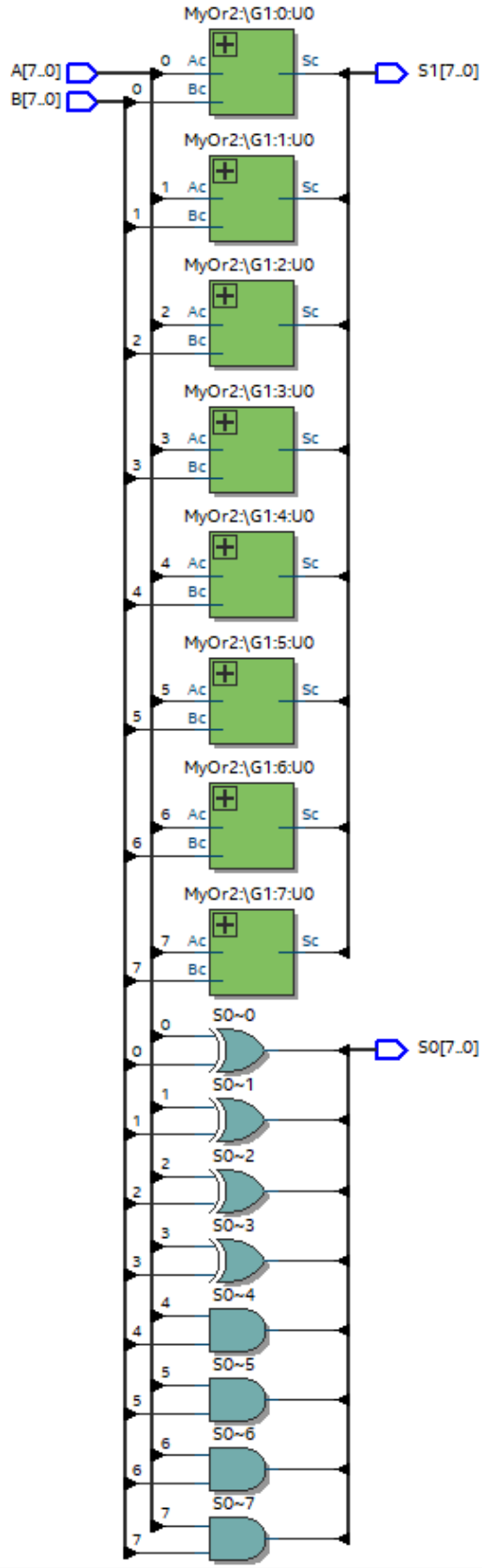

Figura 4: Diagrama RTL del arreglo de compuertas OR, AND y XOR, descrito en VHDL.

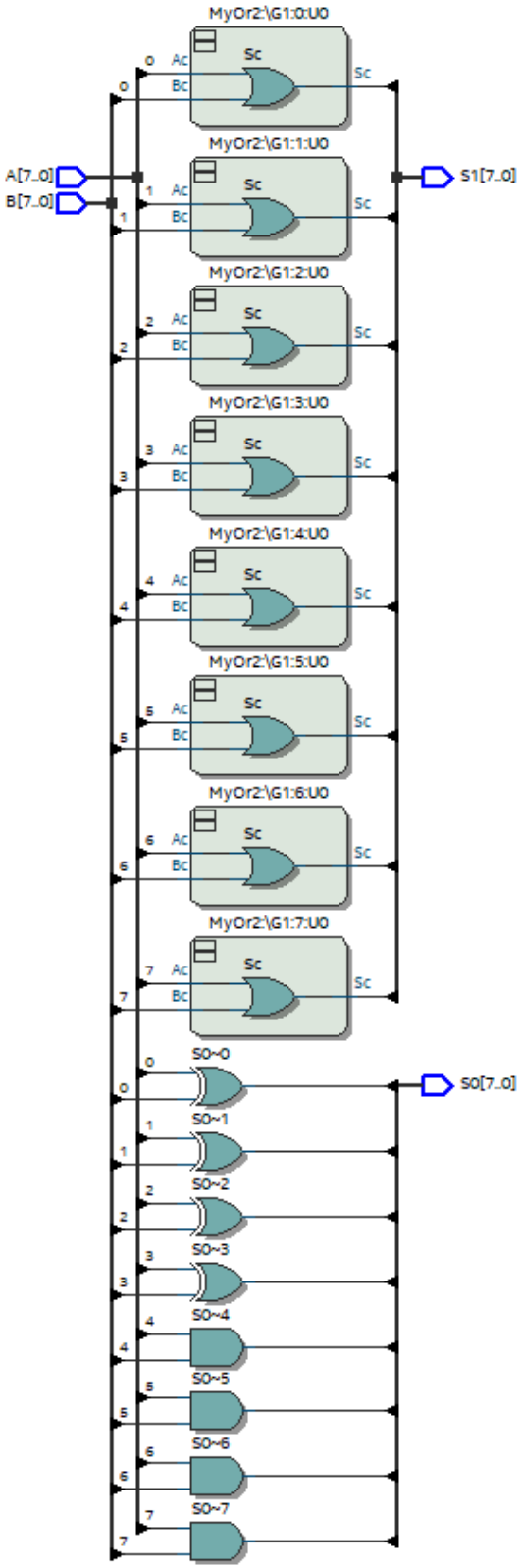

Figura 5: Diagrama RTL del arreglo de compuertas OR, AND y XOR, descrito en VHDL (vista interna de las instancias).

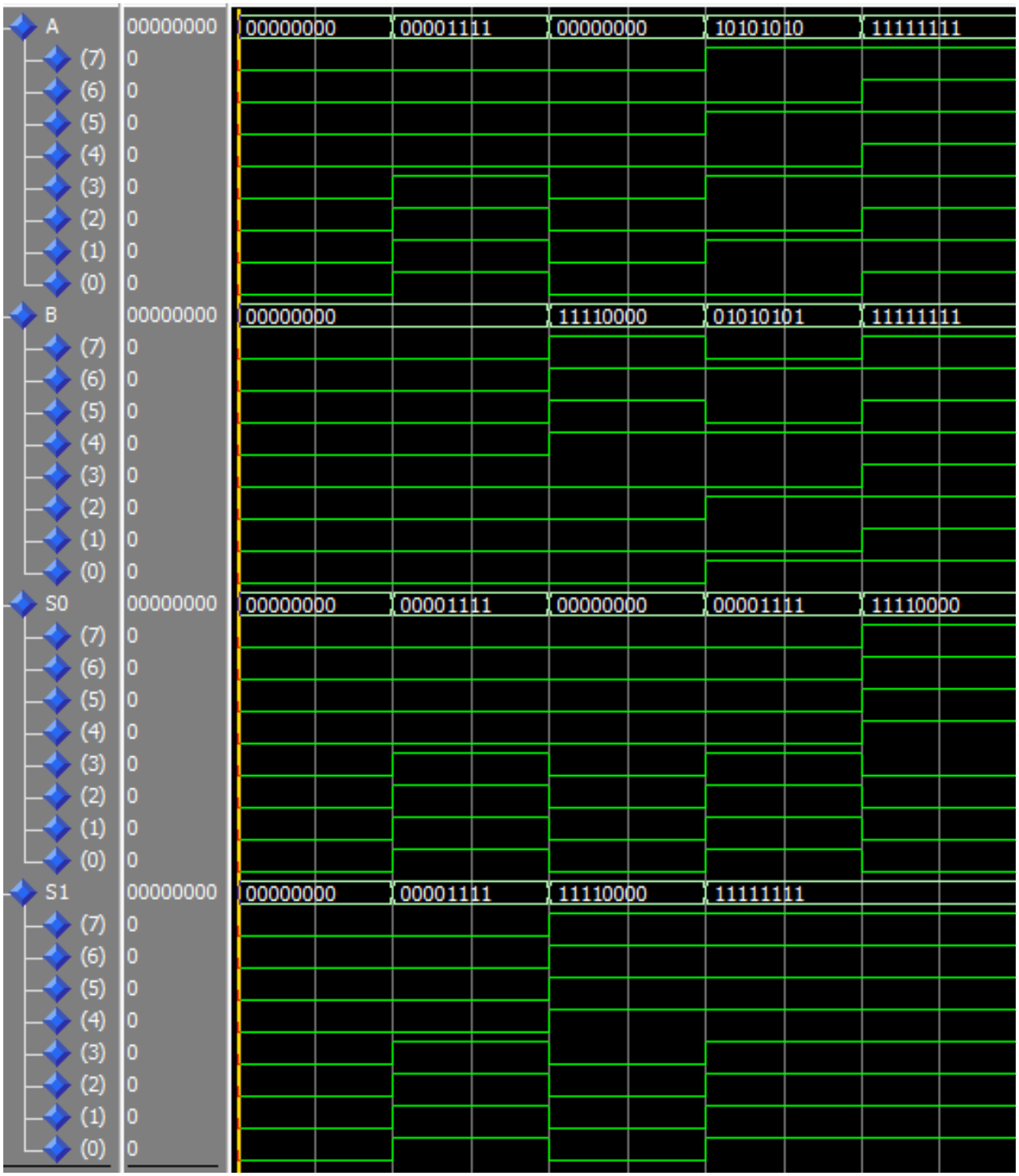

Figura 6: Simulación del arreglo de compuertas OR, AND y XOR, descrito en VHDL, con el visor de formas de onda de ModelSim.

3. *For - Generate* en Verilog

Actividad 2

Investigar cómo se usa la estructura "*generate*" en verilog.

3.1. Definición

La declaración *generate*, en Verilog, es una construcción muy útil que genera código sintetizable durante el tiempo de elaboración de forma dinámica. El simulador proporciona un código elaborado del bloque *generate* que tiene las siguientes características:

- Se generan múltiples instancias de módulo, eliminando la repetición de código.
- Se crea una instancia condicional de un bloque de código basado en un parámetro Verilog; sin embargo, el parámetro no está permitido en la declaración de generación.

Básicamente proporciona control sobre variables, funciones, tareas y declaraciones de creación de instancias. El bloque de generación se escribe dentro de las palabras clave *generate* y *endgenerate*. [1]

3.2. Sintaxis

Como se observa en el Programa 1, la estructura *for-generate* tiene los siguiente puntos importantes:

```
1 genvar i
2 generate
3   for (i = Valor_inicial; i <= Valor_final; Incremento) begin :
    nombre_bloque
4     // Codigo repetitivo que se genera para cada valor de i
5   end
6 endgenerate
```

Programa 1: Sintaxis de la estructura *for-generate*, en Verilog.

- ***genvar***: Se declara una variable de generación "i" de tipo genvar. Esta variable solo es válida dentro del bloque generate.
- **Valor_inicial**: Valor inicial de la variable "i".
- **Valor_final**: Valor final de la variable "i".
- **Incremento:** Incremento de la variable "i" en cada iteración. Si no se especifica, el incremento es de 1.
- **Código repetitivo:** El código que se desea generar para cada valor de "i". Este código puede incluir declaraciones de variables, asignaciones, instanciaciones de módulos, etc. [2]

3.3. Beneficios del *for-generate*

- **Mejora la legibilidad del código:** El código repetitivo se organiza de manera más clara y concisa, lo que facilita su comprensión y mantenimiento.
- **Reduce la redundancia:** Se elimina la necesidad de escribir el mismo código múltiples veces, lo que hace que el código sea más compacto y eficiente.
- **Aumenta la flexibilidad:** El código generado puede ser parametrizado, lo que permite adaptar el diseño a diferentes necesidades.

3.4. Consideraciones adicionales

- El *for-generate* solo se puede utilizar para generar código dentro de un bloque *generate*.
- La variable de generación "i" no se puede utilizar fuera del bloque *for-generate*.
- Es importante usar el incremento adecuado para evitar bucles infinitos.
- Se debe tener cuidado al utilizar el *for-generate* con código sensible al tiempo, como las señales de reloj. [1]

4. Arreglo de compuertas OR, AND y XOR en Verilog

Actividad 3

Obtener la versión "verilog" de los 2 códigos VHDL mostrados en esta tarea. Observar el resultado con el visor RTL.

La visualización RTL del arreglo de compuertas OR (primer código), descrito en Verilog, se muestra en la Figura 7. La implementación se hace instanciando 8 veces al módulo denominado "MyOr2" (debido a que las dos entradas son de 8 bits), que simplemente es una compuerta OR de 1 bit (ver Figura 8). Las simulaciones se visualizan en la Figura 9, en donde se muestra que este arreglo de compuerta OR opera de manera correcta.

En los Anexos se localiza la descripción en VHDL del primer código. Primeramente se crea el módulo que se va a instanciar (compuerta OR), declarando las entradas y salidas, así como la descripción de su comportamiento. Después se declara al módulo principal, con las entradas y salidas de este. Se debe declarar a la variable de iteración "n", para después utilizar a la estructura *for-generate* e instanciar 8 veces al primer módulo descrito (debido a que las entradas son de 8 bits)

La visualización RTL del arreglo de compuertas OR, AND y XOR (segundo código), descrito en Verilog, se muestra en la Figura 10. La implementación se hace nuevamente instanciando 8 veces al módulo denominado "MyOr2", no obstante, se agregan 4 instancias de compuertas AND para los 4 bits menos significativos de las entradas A y B, y otras 4 instancias de compuertas XOR para los 4 bits más significativos (ver Figura 11). Las simulaciones se visualizan en la Figura 12, en donde se muestra que este arreglo de compuertas OR, AND y XOR opera de manera correcta.

En los Anexos se localiza la descripción en VHDL del segundo código. Se realiza lo mismo que en el primer código, no obstante, se deben declarar 3 variables de iteración (una por cada ciclo) y solo es necesario emplear una estructura *for-generate* y dos estructuras *for* para separar a los 4 bits más y menos significativos e instanciar las compuertas AND y XOR.

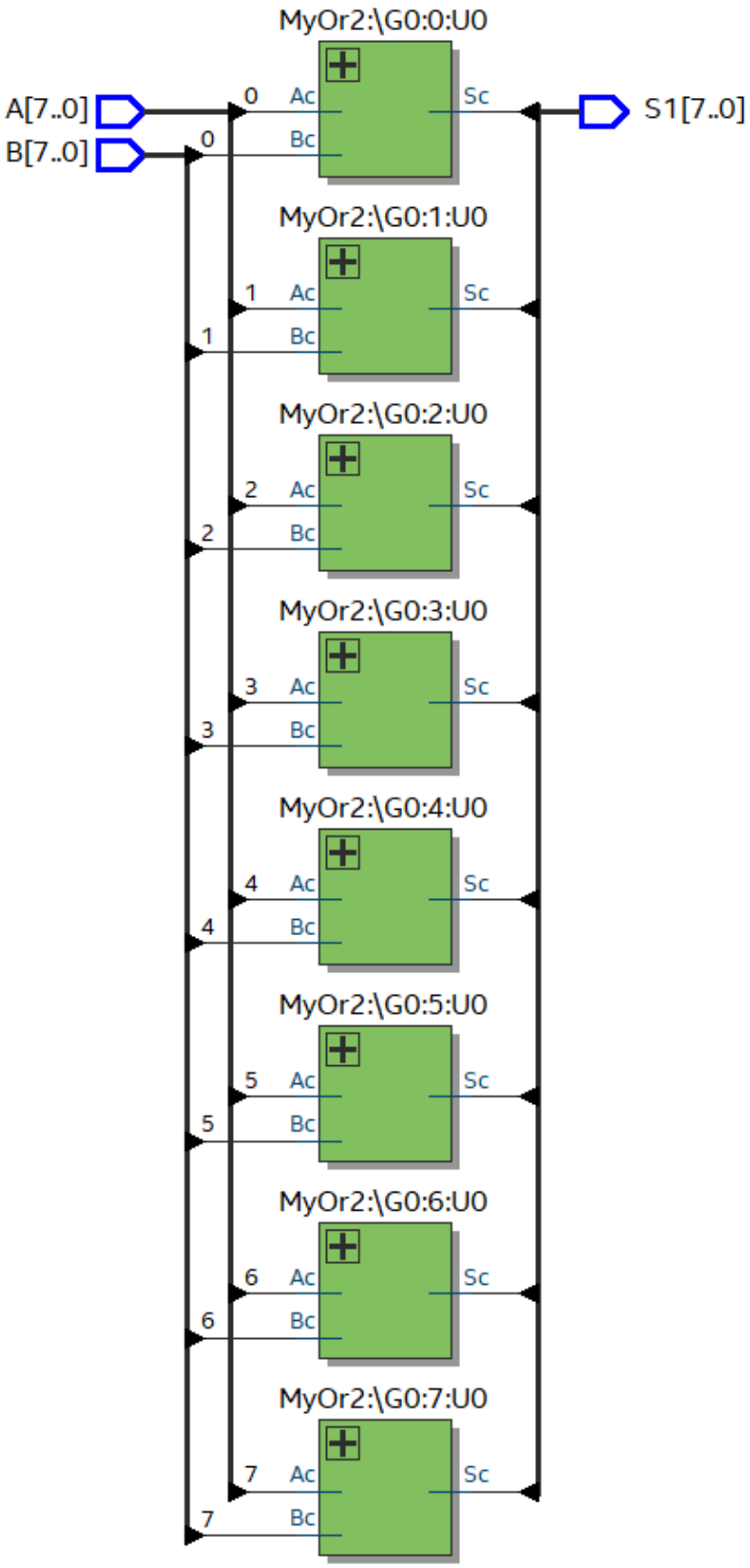

Figura 7: Diagrama RTL del arreglo de compuertas OR, descrito en VHDL.

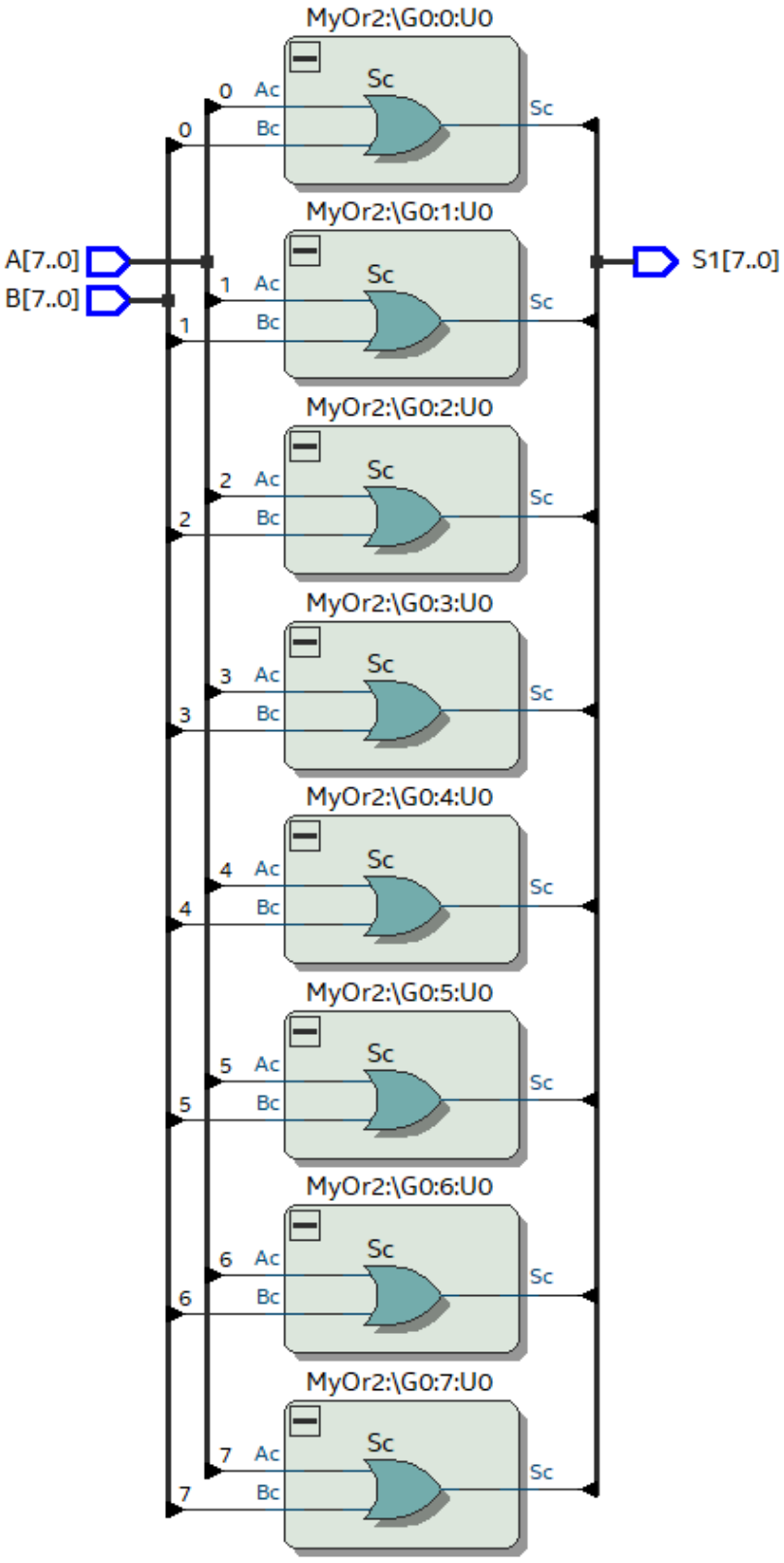

Figura 8: Diagrama RTL del arreglo de compuertas OR, descrito en VHDL (vista interna de las instancias).

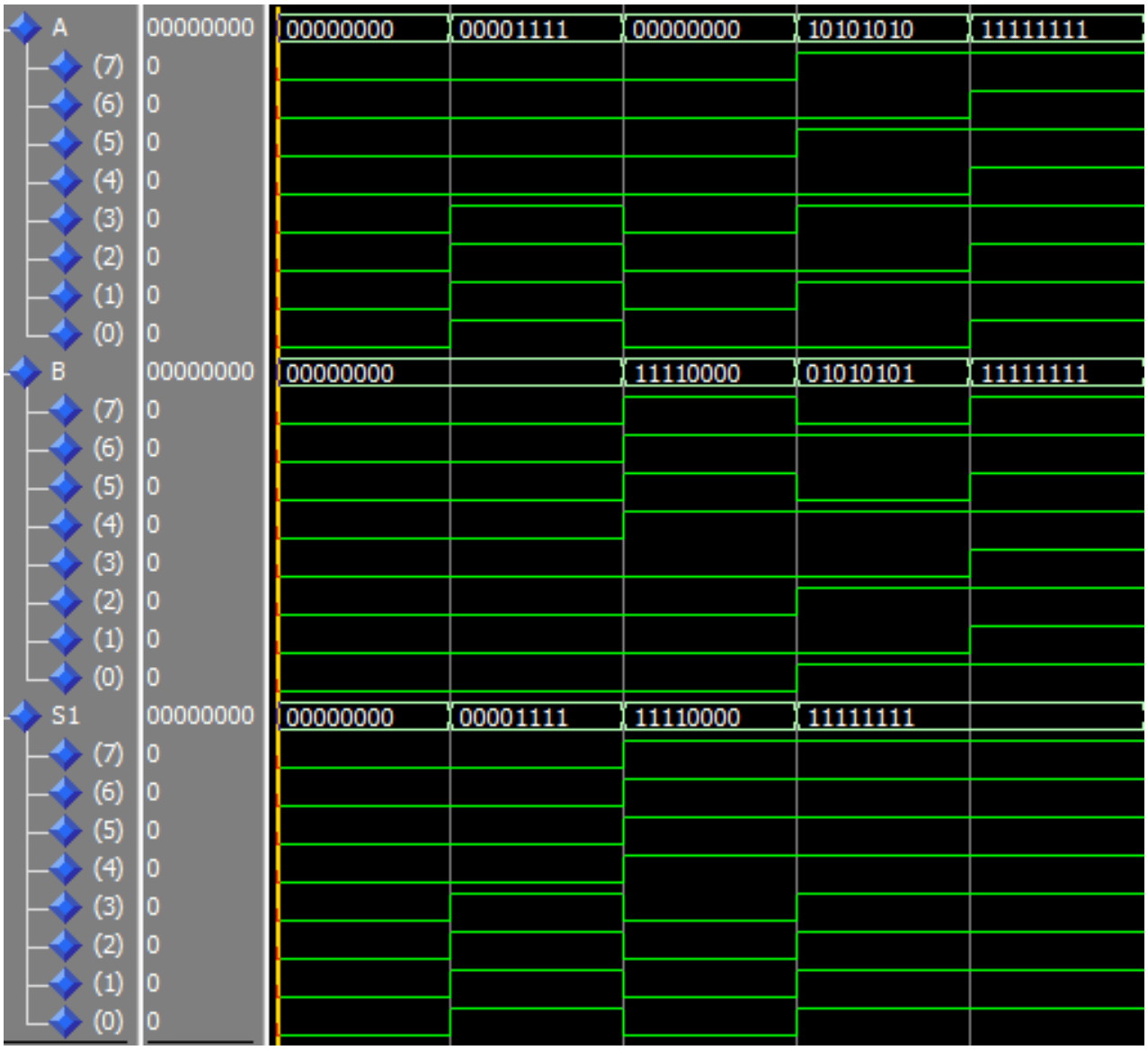

Figura 9: Simulación del arreglo de compuertas OR, descrito en VHDL, con el visor de formas de onda de ModelSim.

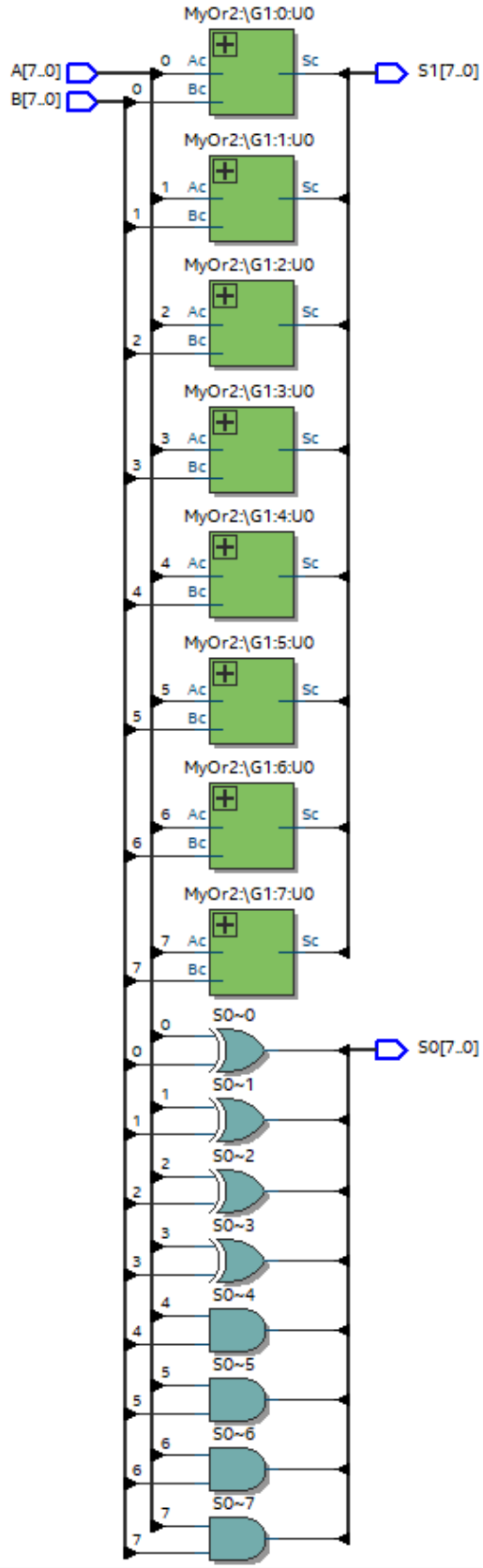

Figura 10: Diagrama RTL del arreglo de compuertas OR, AND y XOR, descrito en VHDL.

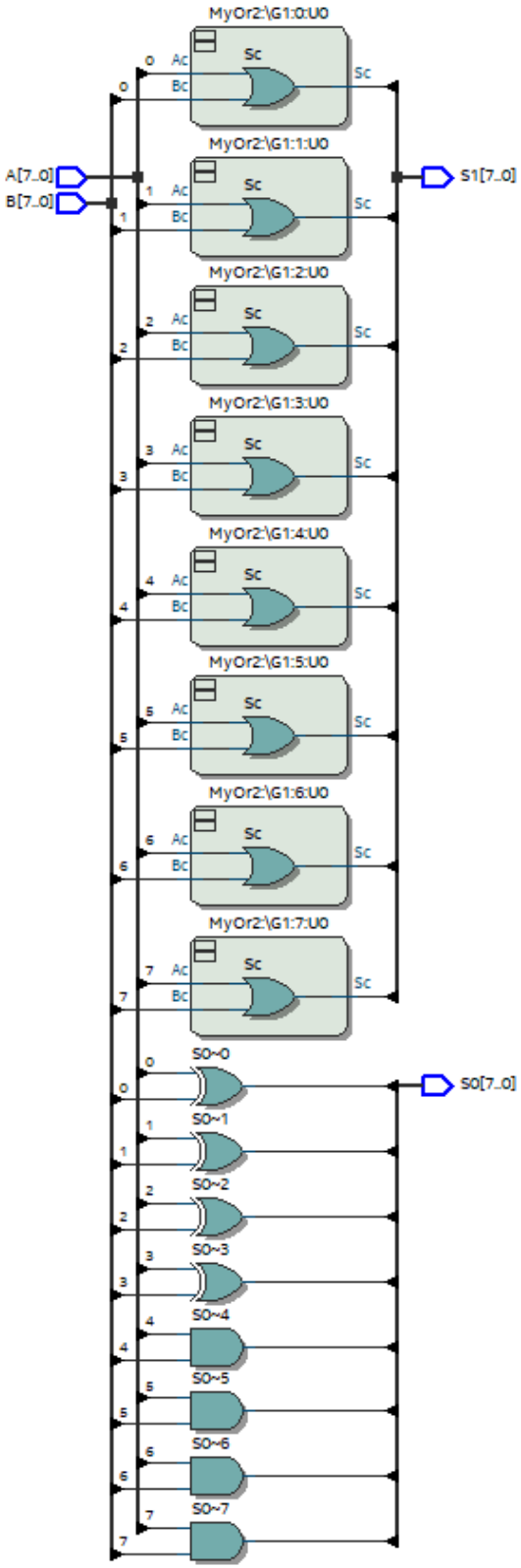

Figura 11: Diagrama RTL del arreglo de compuertas OR, AND y XOR, descrito en VHDL (vista interna de las instancias).

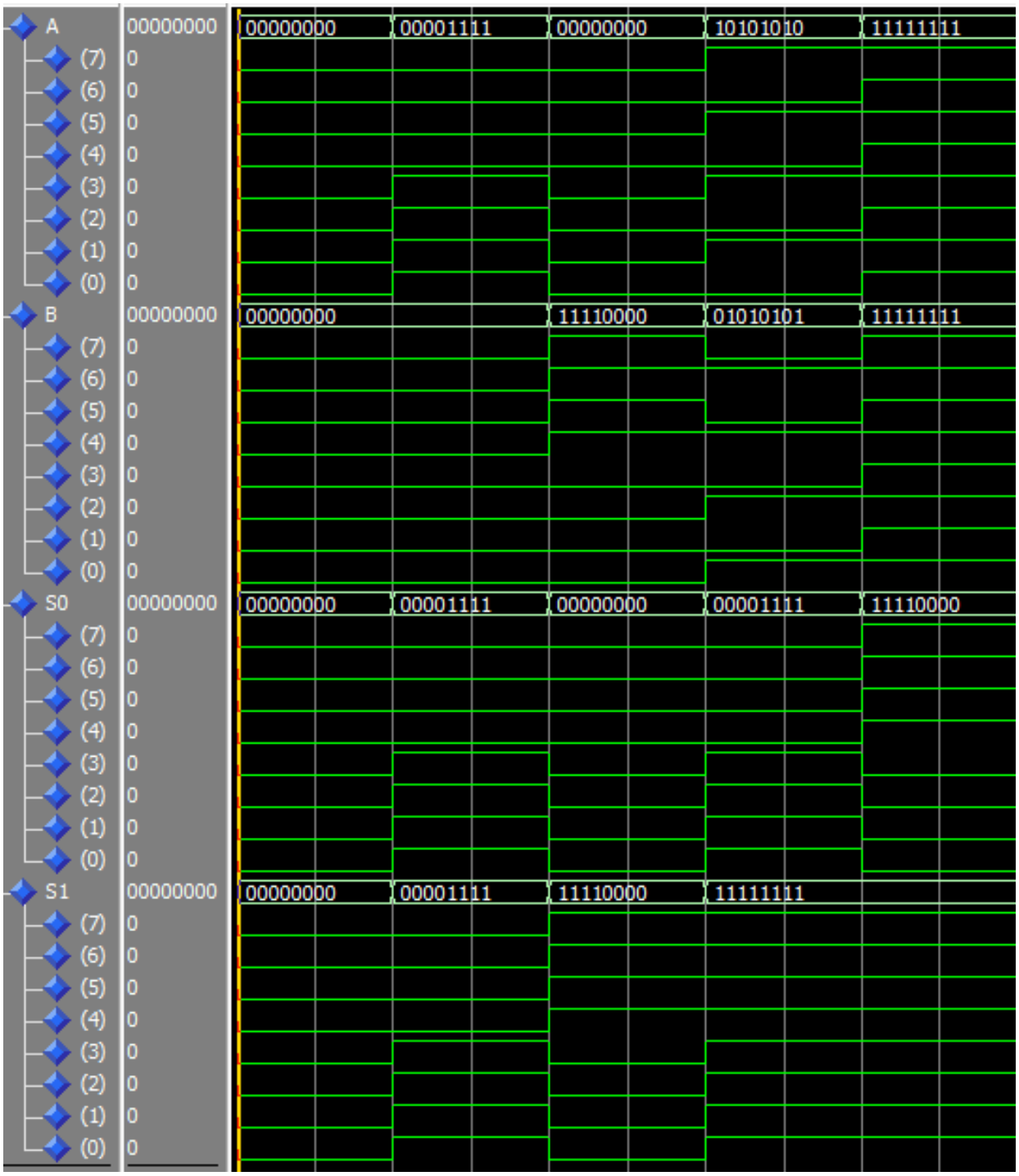

Figura 12: Simulación del arreglo de compuertas OR, AND y XOR, descrito en VHDL, con el visor de formas de onda de ModelSim.

5. Conclusiones

En conclusión, se implementaron los arreglos de compuertas lógicas, en VHDL y Verilog, de manera correcta.

Se comprendió como se utiliza al bloque *for-generate* para iterar hardware por instanciación (descripción estructural), en comparación con la estructura *for-loop* (vista anteriormente), que se utiliza para la descripción por comportamiento.

Se indagó sobre el uso del bloque *for-generate* en Verilog, considerando aspectos importantes como la declaración de una variable de generación y su importancia ante la redundancia y la legibilidad del código.

Se comprendió la sintaxis de la estructura *for-generate*, tanto en Verilog como en VHDL y se observaron algunas de sus diferencias (la declaración de la variable de generación en Verilog, por ejemplo).

En ambos casos se implementaron los arreglos de compuertas y se observó con el visor RTL a los circuitos instanciados dentro del módulo principal, y por medio de las simulaciones de forma de onda en ModelSim, se visualizó la correcta operación de los dispositivos.

En los Anexos se pueden encontrar los códigos implementados junto con sus respectivos bancos de pruebas.

Referencias

[1] VLSIVerify, “Generate blocks in verilog,” https://vlsiverify.com/verilog/generate-blocks-in-verilog/.

[2] ChipVerify, “Verilog generate block,” https://www.chipverify.com/verilog/verilog-generate-block.

6. Anexos

6.1. Descripciones del hardware

```vhdl
library ieee;
use ieee.std_logic_1164.all;

entity ForGenerate1_VHDL is
 port( A, B  : in  std_logic_vector(7 downto 0);
   S1   : out  std_logic_vector(7 downto 0));
end ForGenerate1_VHDL;

architecture behavior of ForGenerate1_VHDL is

component MyOr2 is
 port( Ac, Bc :  in  std_logic;
   Sc   :  out  std_logic);
end component MyOr2;

begin
 G0: for n in 0 to 7 generate
  U0: MyOr2 port map(A(n), B(n), S1(n));
 end generate G0;

end behavior;

library ieee;
use ieee.std_logic_1164.all;

entity MyOr2 is
 port( Ac, Bc  :  in  std_logic;
   Sc   :  out  std_logic);
end MyOr2;

architecture MyOr2_behavior of MyOr2 is
begin
 Sc <= Ac or Bc;
end MyOr2_behavior;
```

Programa 2: Descripción en VHDL del arreglo de compuertas OR.

```vhdl
library ieee;
use ieee.std_logic_1164.all;

```

```vhdl
entity ForGenerate2_VHDL is
 port( A, B  : in  std_logic_vector(7 downto 0);
   S0, S1 : out  std_logic_vector(7 downto 0));
end ForGenerate2_VHDL;

architecture behavior of ForGenerate2_VHDL is

component MyOr2 is
 port( Ac, Bc :  in  std_logic;
   Sc   :  out  std_logic);
end component MyOr2;

begin
 G1: for i in 0 to 7 generate
  U0: MyOr2 port map(A(i), B(i), S1(i));
  G2: if i < 4 generate
   S0(i) <= A(i) xor B(i);
  end generate G2;
  G3: if i >3 generate
   S0(i) <= A(i) and B(i);
  end generate G3;
 end generate G1;
end behavior;

library ieee;
use ieee.std_logic_1164.all;

entity MyOr2 is
 port( Ac, Bc  :  in  std_logic;
   Sc   :  out  std_logic);
end MyOr2;

architecture MyOr2_behavior of MyOr2 is
begin
 Sc <= Ac or Bc;
end MyOr2_behavior;
```

Programa 3: Descripción en VHDL del arreglo de compuertas OR, AND y XOR.

```verilog
module MyOr2(
 input   Ac,
 input   Bc,
 output    Sc
);
 assign Sc = Ac | Bc;
endmodule
```

```
8
9  module ForGenerate1_Verilog(
10  input  [7:0]  A,
11  input  [7:0]  B,
12  output   [7:0]  S1
13 );
14
15  genvar n;
16  generate
17   for(n = 0; n < 8; n = n + 1)
18   begin : G0
19    MyOr2 U0(
20     .Ac(A[n]),
21     .Bc(B[n]),
22     .Sc(S1[n])
23    );
24   end
25  endgenerate
26 endmodule
```

Programa 4: Descripción en Verilog del arreglo de compuertas OR.

```
1  module MyOr2(
2   input     Ac,
3   input     Bc,
4   output    Sc
5  );
6
7   assign Sc = Ac | Bc;
8
9  endmodule
10
11 module ForGenerate2_Verilog(
12     input [7:0]  A,
13     input [7:0]  B,
14     output  [7:0]  S0,
15     output  [7:0]  S1
16 );
17
18  genvar i;
19  generate
20   for(i = 0; i < 8; i = i + 1)
21   begin : G0
22    MyOr2 U0 (
23     .Ac(A[i]),
24     .Bc(B[i]),
```

```
    .Sc(S1[i])
   );
  end
  genvar j;
  for(j = 0; j < 4; j = j + 1)
  begin : G1
   assign S0[j] = A[j] ^ B[j];
  end
  for(j = 4; j < 8; j = j + 1)
  begin : G2
    assign S0[j] = A[j] & B[j];
  end
 endgenerate

endmodule
```

Programa 5: Descripción en Verilog del arreglo de compuertas OR, AND y XOR.

6.2. Bancos de pruebas (*Test Benches*)

```
library ieee;
use ieee.std_logic_1164.all;

entity ForGenerate1_VHDL_vhd_tst is
end ForGenerate1_VHDL_vhd_tst;
architecture ForGenerate1_VHDL_arch of ForGenerate1_VHDL_vhd_tst is
signal  A  :   std_logic_vector(7 downto 0);
signal  B  :   std_logic_vector(7 downto 0);
signal  S1 :   std_logic_vector(7 downto 0);

component ForGenerate1_VHDL
 port (A  : in  std_logic_vector(7 downto 0);
   B  : in  std_logic_vector(7 downto 0);
   S1 : out std_logic_vector(7 downto 0)
 );
end component;

begin
 i1 : ForGenerate1_VHDL
 port map (
   A  => A,
   B  => B,
   S1 => S1
 );

```

```
 init : process
 begin
  wait;
 end process init;

 always : process
 begin
  A <= "00000000"; B <= "00000000";
  wait for 10ns;
  A <= "00001111"; B <= "00000000";
  wait for 10ns;
  A <= "00000000"; B <= "11110000";
  wait for 10ns;
  A <= "10101010"; B <= "01010101";
  wait for 10ns;
  A <= "11111111"; B <= "11111111";
  wait for 10ns;
 end process always;

end ForGenerate1_VHDL_arch;
```

Programa 6: Banco de prueba para el Programa 2.

```
library ieee;
use ieee.std_logic_1164.all;

entity ForGenerate2_VHDL_vhd_tst is
end ForGenerate2_VHDL_vhd_tst;
architecture ForGenerate2_VHDL_arch of ForGenerate2_VHDL_vhd_tst is
signal  A  :   std_logic_vector(7 downto 0);
signal  B  :   std_logic_vector(7 downto 0);
signal  S0 :   std_logic_vector(7 downto 0);
signal  S1 :   std_logic_vector(7 downto 0);

component ForGenerate2_VHDL
 port (A  : in  std_logic_vector(7 downto 0);
   B  : in  std_logic_vector(7 downto 0);
   S0 : out std_logic_vector(7 downto 0);
   S1 : out std_logic_vector(7 downto 0)
 );
end component;

begin
 i1 : ForGenerate2_VHDL
 port map (
   A  => A,
```

```
    B  => B,
    S0 => S0,
    S1 => S1
 );

 init : process
 begin
  wait;
 end process init;

 always : process
 begin
  A <= "00000000"; B <= "00000000";
  wait for 10ns;
  A <= "00001111"; B <= "00000000";
  wait for 10ns;
  A <= "00000000"; B <= "11110000";
  wait for 10ns;
  A <= "10101010"; B <= "01010101";
  wait for 10ns;
  A <= "11111111"; B <= "11111111";
  wait for 10ns;
 end process always;

end ForGenerate2_VHDL_arch;
```

Programa 7: Banco de prueba para el Programa 3.

```
`timescale 1 ns/ 1 ps
module ForGenerate1_Verilog_vlg_tst();
 reg   [7:0]   A;
 reg   [7:0]   B;
 wire   [7:0]    S1;

 ForGenerate1_Verilog i1 (
  .A(A),
  .B(B),
  .S1(S1)
 );

 initial
 begin
  A = 8'b00000000; B = 8'b00000000;
  $display("Running testbench at CIC");
 end

```

```
 always
 begin
  #10; A = 8'b00001111; B = 8'b00000000;
  #10; A = 8'b00000000; B = 8'b11110000;
  #10; A = 8'b10101010; B = 8'b01010101;
  #10; A = 8'b11111111; B = 8'b11111111;
 end

endmodule
```

Programa 8: Banco de prueba para el Programa 4.

```
`timescale 1 ns/ 1 ps
module ForGenerate2_Verilog_vlg_tst();
 reg   [7:0]   A;
 reg   [7:0]   B;
 wire   [7:0]    S0;
 wire   [7:0]    S1;

 ForGenerate2_Verilog i1 (
  .A(A),
  .B(B),
  .S0(S0),
  .S1(S1)
 );

 initial
 begin
  A = 8'b00000000; B = 8'b00000000;
  $display("Running testbench at CIC");
 end

 always
 begin
  #10; A = 8'b00001111; B = 8'b00000000;
  #10; A = 8'b00000000; B = 8'b11110000;
  #10; A = 8'b10101010; B = 8'b01010101;
  #10; A = 8'b11111111; B = 8'b11111111;
 end

endmodule
```

Programa 9: Banco de prueba para el Programa 5.