Instituto Politécnico Nacional

Centro de Investigación en Computación

Lenguajes de descripción de hardware

Tarea 3 - Subprogramas

Profesor:

M. en C. Osvaldo Espinosa Sosa

Por:

Ing. Ricardo Aldair Tirado Torres

Ciudad de México, 26 de abril de 2024

Tabla de contenido

1. Objetivos

- Comprender el funcionamiento de los procedimientos y las funciones en VHDL, y las tareas y funciones en Verilog.
- Indagar sobre los subprogramas de Verilog y diferenciar su forma de operar así como su sintaxis.
- Implementar 3 sumadores independientes de 4 bits, utilizando los subprogramas de VHDL y Verilog, y compararlos con el visor RTL y el simulador ModelSim.

2. Subprogramas en VHDL: Funciones y procedimientos

Actividad 1

Capturar los códigos escritos en VHDL vistos en clase, compilarlos y simularlos (se proporcionan en la presentación y en el documento anexo). Ver el resultado de la síntesis con el visor RTL.

La visualización RTL de los 3 sumadores de 4 bits en VHDL se muestra en la Figura 1. Como se observa, la implementación se realiza utilizando 3 instancias de sumadores de 4 bits, independientes una de otra. Las simulaciones se visualizan en la Figura 2 en base binaria y en la Figura 3 en base decimal, en donde se muestra que la suma se hace de manera correcta en cada elemento.

En los Anexos se localiza la descripción de los 3 sumadores de 4 bits. Las 3 implementaciones comparten la misma declaración de bibliotecas y de entidad, difiriendo en la definición de la arquitectura.

- **Para el caso de la función:** Se genera el prototipo y la cabecera de la función, declarando dos parámetros que servirán como operadores de apoyo para la suma (Op1 y Op2) y una variable que devolverá la función como resultado (Total). En el cuerpo de la función se realiza la suma de los dos operandos y se asigna con ":=" el resultado a la variable que se va a retornar. Finalmente, en el cuerpo de la arquitectura se llama a la función, dándole los parámetros necesarios y asignando con "<=" el resultado a la señal correspondiente.
- **Para el caso del procedimiento (caso 1):** Se genera el prototipo y la cabecera del procedimiento, declarando tres parámetros, dos de ellos serán las entradas que sirven como operadores de apoyo para la suma (Op1 y Op2) y otra variable que será la salida que almacena el resultado (Total). En el cuerpo del procedimiento unicamente se realiza la suma, asignando con ":=" el resultado. Finalmente, en el cuerpo de la arquitectura se declaran tanto una lista sensible para las entradas como las variables en donde se guardará el resultado de cada suma. Dentro del bloque *process* se llaman a los procedimientos para después asignar a las señales de salida el valor de las variables declaradas previamente (usando "<=").

- **Para el caso del procedimiento (caso 2):** Es muy parecido al caso 1, solo que en el prototipo y cabecera del procedimiento se declara a la salida como señal. Esto provoca que en el cuerpo del procedimiento se use "<=" para asignar el resultado y en el cuerpo de la arquitectura desaparece la lista sensible y la declaración de variables, haciendo que solo se llame al procedimiento, colocando a las señales como parámetros.

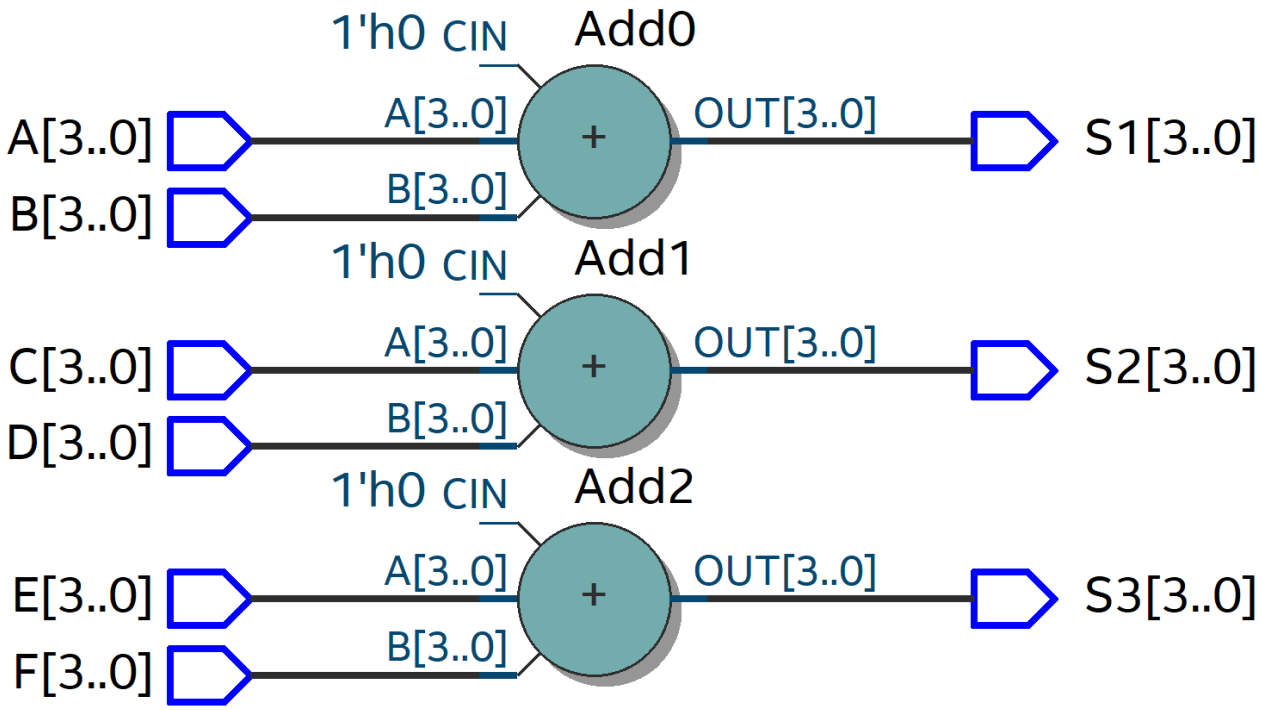

Figura 1: Diagrama RTL de los 3 sumadores de 4 bits implementados en VHDL. Se tiene el mismo resultado si se usan funciones o procedimientos.

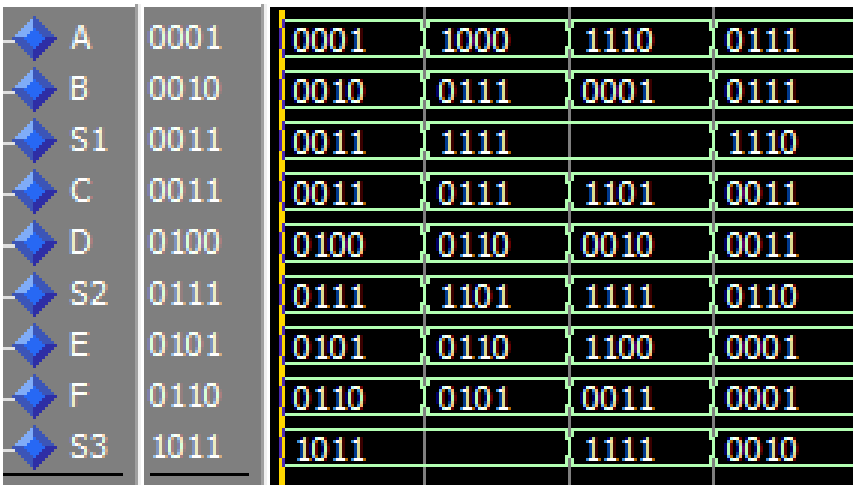

Figura 2: Simulación de los 3 sumadores de 4 bits con el visor de formas de onda de ModelSim (base binaria).

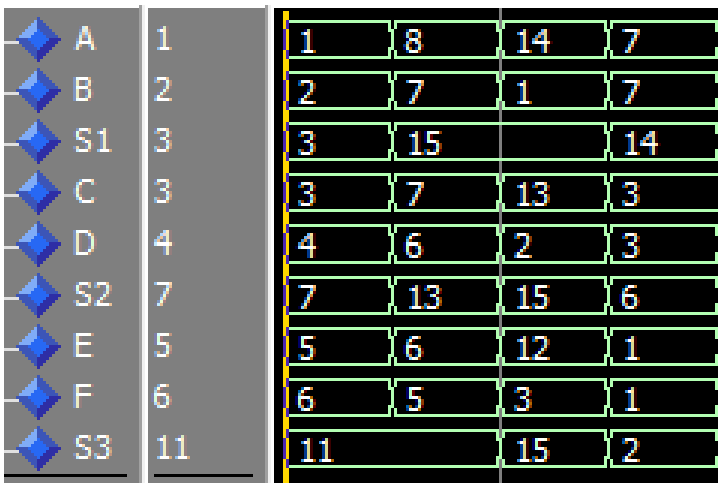

Figura 3: Simulación de los 3 sumadores de 4 bits con el visor de formas de onda de ModelSim (base decimal).

3. Funciones y tareas en Verilog

Actividad 2

En verilog existen las funciones (functions) y las tareas (tasks) como subprogramas. Hacer una breve descripción de sus características (como la que aparece al inicio de la presentación y del documento para subprogramas).

En Verilog, una función o tarea es un grupo de declaraciones que realiza alguna acción específica. Se puede llamar a ambos en varios puntos para realizar una determinada operación. También se utilizan para dividir código grande en partes más pequeñas para facilitar su lectura y depuración. [1]

3.1. Definiciones

Las tareas se utilizan en todos los lenguajes de programación, generalmente conocidas como procedimientos o subrutinas. Los datos se pasan a la tarea, se realiza el procesamiento y se devuelve el resultado. Tienen que ser llamados específicamente, con datos de entrada y salidas. Incluidos en el cuerpo principal del código, se pueden llamar muchas veces, lo que reduce la repetición del código.

Ahora bien, una función es lo mismo que una tarea, con muy pequeñas diferencias, como que la función no puede controlar más de una salida y no puede contener retrasos.

3.2. Sintaxis

La sintaxis de una tarea se observa en el Programa 1.

- Una tarea comienza con la palabra clave *task* y termina con la palabra clave *endtask*.
- Las entradas y salidas se declaran después de la palabra clave *task*.
- Las variables locales se declaran después de la declaración de entradas y salidas. [2]

```
// Style 1
task <task_name> (input <port_list>, inout <port_list>, output <port_list
    >);
 ...
endtask

// Style 2
task <task_name>;
  input <port_list>;
  inout <port_list>;
  output <port_list>;
  ...
endtask
```

Programa 1: Sintaxis de la descripción de una tarea en Verilog. [1]

La sintaxis de una función se observa en el Programa 2.

- Una función comienza con la palabra clave *function* y termina con la palabra clave *endfunction*.
- Las entradas se declaran después de la palabra clave *endfunction*. [2]

```
// Style 1
function <return_type> <function_name> (input <port_list>,inout <port_list
    >, output <port_list>);
 ...
 return <value or expression>
endfunction

// Style 2
function <return_type> <function_name>;
  input <port_list>;
  inout <port_list>;
  output <port_list>;
  ...
  return <value or expression>
endfunction
```

Programa 2: Sintaxis de la descripción de una función en Verilog. [1]

3.3. Comparación

- La tarea puede tener cero o más de un argumento. La función debe tener al menos un argumento.
- Una tarea puede tener declaraciones de retraso en su interior. Una función no puede tener una declaración de retraso. Debería devolver un valor en el mismo paso de tiempo.
- Una tarea no tiene un tipo de retorno. Sin embargo, los argumentos de salida ayudan a devolver valores. Una función tiene un tipo de retorno.
- Una tarea puede devolver más de un valor, ya que puede haber cualquier número de argumentos de salida. Una función puede devolver solo un valor ya que los argumentos de salida no se pueden usar en funciones.
- Una tarea puede llamar a otra función o tarea desde su cuerpo. Una función sólo puede llamar a otra función desde su cuerpo. No se puede llamar a una tarea porque puede consumir tiempo y no se permite que la función consuma tiempo.

3.4. Variantes

Un dato interesante es que las tareas y funciones son de 2 tipos, estáticas y automáticas.

De forma predeterminada, todas las funciones y tareas son estáticas en Verilog. Las tareas estáticas significan que las variables declaradas dentro de una tarea conservarán su valor anterior cada vez que se llame a la tarea. Por tanto, se puede decir que la memoria asignada para la función o tarea permanece igual durante cada llamada, volviéndola estática.

Mientras que en la función o tarea automática, se asigna una nueva ubicación de memoria cada vez que se llaman. Por lo tanto, cualquier variable interna presente dentro de la tarea no conservará su valor anterior. [3]

4. Subprogramas en Verilog: Funciones y tareas

Actividad 1

Codificar en verilog la versión del circuito de los tres sumadores usando primero una función (function) y después usando una tarea (task).Compilar y simular.

La visualización RTL de los 3 sumadores de 4 bits en Verilog se muestra en la Figura 4. Como se observa, la implementación se realiza utilizando 3 instancias de sumadores de 4 bits, independientes una de otra. Las simulaciones se visualizan en la Figura 5 en base binaria y en la Figura 6 en base decimal, en donde se muestra que la suma se hace de manera correcta en cada elemento.

En los Anexos se localiza la descripción de los 3 sumadores de 4 bits. Las 2 implementaciones solo comparten la misma declaración de entradas y salidas del módulo.

- **Para el caso de la función:** Se usa la palabra reservada *function* y se declara en esa misma linea el tipo de valor que va a retornar, así como el nombre de la función. Después se declaran a las entradas y en el cuerpo se describe la operación de suma. Finalmente, dentro de una lista sensible se llama a la función, asignando el valor de retorno a la señal de salida correspondiente.
- **Para el caso de la tarea:** Se usa la palabra reservada *task* y se coloca el nombre de la tarea. Después se declaran a las entradas y salidas y en el cuerpo se describe la operación de suma. Finalmente, dentro de una lista sensible se llama a la tarea, unicamente colocando a las señales como parámetros de la tarea.

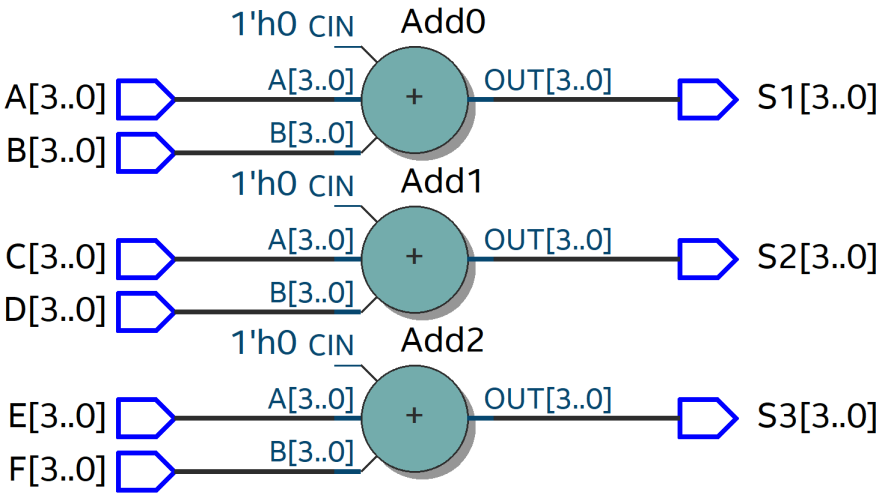

Figura 4: Diagrama RTL de los 3 sumadores de 4 bits implementados en Verilog. Se tiene el mismo resultado si se usan funciones o tareas.

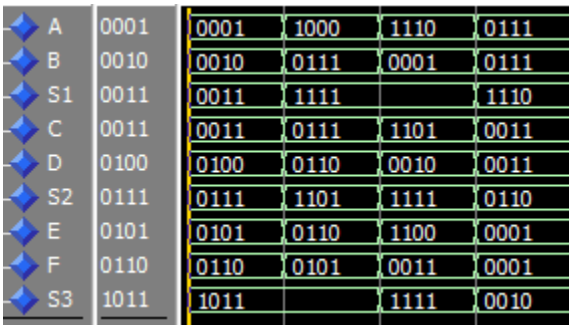

Figura 5: Simulación de los 3 sumadores de 4 bits con el visor de formas de onda de ModelSim (base binaria).

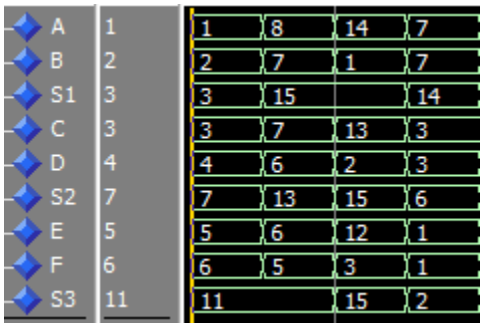

Figura 6: Simulación de los 3 sumadores de 4 bits con el visor de formas de onda de ModelSim (base decimal).

5. Conclusiones

En conclusión, se implementaron los circuitos en VHDL y en Verilog de forma correcta.

Se comprendió y se diferenció a los procedimientos y a las funciones en VHDL por medio del análisis de la sintaxis de cada uno. De igual forma, se hizo lo mismo con las tareas y funciones en Verilog.

Se investigó de manera concisa la definición, las sintaxis, las similitudes y las diferencias entre las funciones y las tareas en Verilog.

Se implementó la descripción de 3 sumadores de 4 bits empleando los subprogramas de cada lenguaje y se observó con el visor RTL que el modulo es igual en todos los casos y por medio de las simulaciones de forma de onda en ModelSim se visualizó que la operación es la misma.

En los Anexos se pueden encontrar los códigos implementados junto con sus respectivos bancos de pruebas. Cabe señalar que los *Test benches* no variaron mucho entre una implementación y otra.

Referencias

[1] VLSIVerify, "Tasks and functions in verilog," https://vlsiverify.com/verilog/tasks-and-functions/.

[2] ASICWORLD, "Task and function," https://www.asic-world.com/verilog/task-func1.html, 2014.

[3] TheOctetInstitute, "Verilog functions and tasks," https://www.theoctetinstitute.com/content/verilog/function-task/, 2021.

6. Anexos

6.1. Descripciones del hardware

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

entity Subprogram_Function1 is
 port( A, B, C, D, E, F : in  std_logic_vector(3 downto 0);
   S1, S2, S3   : out std_logic_vector(3 downto 0));
end Subprogram_Function1;

architecture behavior of Subprogram_Function1 is

function Sum(Op1, Op2: std_logic_vector) return std_logic_vector;
function Sum(Op1, Op2: std_logic_vector) return std_logic_vector is
 variable Total: std_logic_vector(3 downto 0);
 begin
  Total := Op1 + Op2;
  return Total;
end Sum;

begin
 S1 <= Sum(A, B);
 S2 <= Sum(C, D);
 S3 <= Sum(E, F);
end behavior;
```

Programa 3: Descripción en VHDL de los 3 sumadores de 4 bits usando funciones.

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

entity Subprogram_Procedure1 is
 port( A, B, C, D, E, F : in  std_logic_vector(3 downto 0);
   S1, S2, S3   : out std_logic_vector(3 downto 0));
end Subprogram_Procedure1;

architecture behavior of Subprogram_Procedure1 is

```

```
procedure Sum(Op1, Op2 : in  std_logic_vector;
      Total  : out std_logic_vector);
procedure Sum(Op1, Op2 : in  std_logic_vector;
      Total  : out std_logic_vector) is
 begin
  Total := Op1 + Op2;
end Sum;

begin
 process(A, B, C, D, E, F)
 variable Sum1, Sum2, Sum3 : std_logic_vector(3 downto 0);
 begin
  Sum(A, B, Sum1);
  Sum(C, D, Sum2);
  Sum(E, F, Sum3);
  S1 <= Sum1;
  S2 <= Sum2;
  S3 <= Sum3;
 end process;
end behavior;
```

Programa 4: Descripción en VHDL de los 3 sumadores de 4 bits usando procedimientos (caso 1).

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

entity Subprogram_Procedure2 is
 port( A, B, C, D, E, F : in  std_logic_vector(3 downto 0);
   S1, S2, S3   : out std_logic_vector(3 downto 0));
end Subprogram_Procedure2;

architecture behavior of Subprogram_Procedure2 is

procedure Sum(Op1, Op2 : in  std_logic_vector;
  signal  Total  : out std_logic_vector);
procedure Sum(Op1, Op2 : in  std_logic_vector;
  signal  Total  : out std_logic_vector) is
 begin
  Total <= Op1 + Op2;
end Sum;

begin
 Sum(A, B, S1);
```

```
 Sum(C, D, S2);
 Sum(E, F, S3);
end behavior;
```

Programa 5: Descripción en VHDL de los 3 sumadores de 4 bits usando procedimientos (caso 2).

```
module Subprogram_Function2(
 input    [3:0]  A, B, C, D, E, F,
 output reg [3:0]  S1, S2, S3
);

function [3:0] Sum;
 input    [3:0]  Op1, Op2;
 begin
  Sum = Op1 + Op2;
 end
endfunction

always @(*)
begin
 S1 = Sum(A, B);
 S2 = Sum(C, D);
 S3 = Sum(E, F);
end

endmodule
```

Programa 6: Descripción en Verilog de los 3 sumadores de 4 bits usando funciones.

```
module Subprogram_Task(
 input    [3:0]  A, B, C, D, E, F,
 output reg [3:0]  S1, S2, S3
);

task Sum;
 input    [3:0]  Op1, Op2;
 output reg [3:0]  Result;
 begin
  Result = Op1 + Op2;
 end
endtask

always @(*)
begin
 Sum(A, B, S1);
```

```
 Sum(C, D, S2);
 Sum(E, F, S3);
end

endmodule
```

Programa 7: Descripción en Verilog de los 3 sumadores de 4 bits usando tareas.

6.2. Bancos de pruebas (*Test Benches*)

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

entity Subprogram_Function1_vhd_tst is
end Subprogram_Function1_vhd_tst;
architecture Subprogram_Function1_arch of Subprogram_Function1_vhd_tst is
signal A  : std_logic_vector(3 downto 0);
signal B  : std_logic_vector(3 downto 0);
signal C  : std_logic_vector(3 downto 0);
signal D  : std_logic_vector(3 downto 0);
signal E  : std_logic_vector(3 downto 0);
signal F  : std_logic_vector(3 downto 0);
signal S1 : std_logic_vector(3 downto 0);
signal S2 : std_logic_vector(3 downto 0);
signal S3 : std_logic_vector(3 downto 0);

component Subprogram_Function1
 port ( A  : in  std_logic_vector(3 downto 0);
    B  : in  std_logic_vector(3 downto 0);
    C  : in  std_logic_vector(3 downto 0);
    D  : in  std_logic_vector(3 downto 0);
    E  : in  std_logic_vector(3 downto 0);
    F  : in  std_logic_vector(3 downto 0);
    S1  : out std_logic_vector(3 downto 0);
    S2  : out std_logic_vector(3 downto 0);
    S3  : out std_logic_vector(3 downto 0));
end component;

begin
 i1 : Subprogram_Function1
 port map (A  =>  A,
     B  =>  B,
     C  =>  C,
     D  =>  D,
     E  =>  E,
```

```
36      F  =>  F,
37      S1  =>  S1,
38      S2  =>  S2,
39      S3  =>  S3);
40  init : process
41  begin
42   wait;
43  end process init;
44  always : process
45  begin
46   A <= "0001"; B <= "0010"; C <= "0011"; D <= "0100"; E <= "0101"; F <= "
      0110"; -- 1 + 2 | 3 + 4 | 5 + 6
47   wait for 50ns;
48   A <= "1000"; B <= "0111"; C <= "0111"; D <= "0110"; E <= "0110"; F <= "
      0101"; -- 8 + 7 | 7 + 6 | 6 + 5
49   wait for 50ns;
50   A <= "1110"; B <= "0001"; C <= "1101"; D <= "0010"; E <= "1100"; F <= "
      0011"; -- 14 + 1 | 13 + 2 | 12 + 3
51   wait for 50ns;
52   A <= "0111"; B <= "0111"; C <= "0011"; D <= "0011"; E <= "0001"; F <= "
      0001"; -- 7 + 7 | 3 + 3 | 1 + 1
53   wait for 50ns;
54  end process always;
55 end Subprogram_Function1_arch;
```

Programa 8: Banco de prueba para el Programa 3 y el **??**.

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity Subprogram_Procedure1_vhd_tst is
5  end Subprogram_Procedure1_vhd_tst;
6  architecture Subprogram_Procedure1_arch of Subprogram_Procedure1_vhd_tst
       is
7  signal A  : std_logic_vector(3 downto 0);
8  signal B  : std_logic_vector(3 downto 0);
9  signal C  : std_logic_vector(3 downto 0);
10 signal D  : std_logic_vector(3 downto 0);
11 signal E  : std_logic_vector(3 downto 0);
12 signal F  : std_logic_vector(3 downto 0);
13 signal S1 : std_logic_vector(3 downto 0);
14 signal S2 : std_logic_vector(3 downto 0);
15 signal S3 : std_logic_vector(3 downto 0);
16
17 component Subprogram_Procedure1
18  port ( A  : in   std_logic_vector(3 downto 0);
```

```
    B  : in   std_logic_vector(3 downto 0);
    C  : in   std_logic_vector(3 downto 0);
    D  : in   std_logic_vector(3 downto 0);
    E  : in   std_logic_vector(3 downto 0);
    F  : in   std_logic_vector(3 downto 0);
    S1  : buffer std_logic_vector(3 downto 0);
    S2  : buffer std_logic_vector(3 downto 0);
    S3  : buffer std_logic_vector(3 downto 0));
end component;

begin
 i1 : Subprogram_Procedure1
 port map (A  =>  A,
     B  =>  B,
     C  =>  C,
     D  =>  D,
     E  =>  E,
     F  =>  F,
     S1  =>  S1,
     S2  =>  S2,
     S3  =>  S3);
 init : process
 begin
  wait;
 end process init;
 always : process
 begin
  A <= "0001"; B <= "0010"; C <= "0011"; D <= "0100"; E <= "0101"; F <= "
   0110"; -- 1 + 2 | 3 + 4 | 5 + 6
  wait for 50ns;
  A <= "1000"; B <= "0111"; C <= "0111"; D <= "0110"; E <= "0110"; F <= "
   0101"; -- 8 + 7 | 7 + 6 | 6 + 5
  wait for 50ns;
  A <= "1110"; B <= "0001"; C <= "1101"; D <= "0010"; E <= "1100"; F <= "
   0011"; -- 14 + 1 | 13 + 2 | 12 + 3
  wait for 50ns;
  A <= "0111"; B <= "0111"; C <= "0011"; D <= "0011"; E <= "0001"; F <= "
   0001"; -- 7 + 7 | 3 + 3 | 1 + 1
  wait for 50ns;
 end process always;
end Subprogram_Procedure1_arch;
```

Programa 9: Banco de prueba para el Programa 4 (Obsérvese que la S1, S2 y S3 se manejan en el componente instanciado como *buffer*).

```
`timescale 1 ns/ 1 ps
```

```
module Subprogram_Function2_vlg_tst();
 reg [3:0]  A;
 reg [3:0]  B;
 reg [3:0]  C;
 reg [3:0]  D;
 reg [3:0]  E;
 reg [3:0]  F;
 wire  [3:0]  S1;
 wire  [3:0]  S2;
 wire  [3:0]  S3;

 Subprogram_Function2 i1 (
  .A(A),
  .B(B),
  .C(C),
  .D(D),
  .E(E),
  .F(F),
  .S1(S1),
  .S2(S2),
  .S3(S3)
 );

 initial
 begin
  A = 1; B = 2; C = 3; D = 4; E = 5; F = 6;
  $display("Running testbench at CIC");
 end

 always
 begin
  #50; A = 8;  B = 7; C = 7;  D = 6; E = 6;  F = 5;
  #50; A = 14; B = 1; C = 13; D = 2; E = 12; F = 3;
  #50; A = 7;  B = 7; C = 3;  D = 3; E = 1;  F = 1;
 end

endmodule
```

Programa 10: Banco de prueba para el Programa 6 y el Programa 7.