

Asignatura: **Ingeniería de Computadores 3**

Trabajo Práctico:

- **Diseño y programación en VHDL de un circuito digital para la implementación de dos funciones lógicas dadas.**
- **Diseño y programación en VHDL de un circuito combinacional para la suma con acarreo.**

Alumno: **rsanchez628@alumno.uned.es**

## EJERCICIO 1

Se desea diseñar un circuito digital que implemente las funciones  $F$  y  $G$  cuya tabla de verdad se muestra a continuación, que dependen de las tres variables  $x$ ,  $y$  y  $z$ :

$x$	$y$	$z$	$F$	$G$
'0'	'0'	'0'	'0'	'1'
'0'	'0'	'1'	'1'	'1'
'0'	'1'	'0'	'0'	'0'
'0'	'1'	'1'	'1'	'1'
'1'	'0'	'0'	'0'	'0'
'1'	'0'	'1'	'0'	'0'
'1'	'1'	'0'	'1'	'0'
'1'	'1'	'1'	'1'	'0'

- 1.a) (0.5 puntos) Obtenga las funciones lógicas  $F$  y  $G$  a partir de la tabla de verdad. Escriba en VHDL la **entity** del circuito que implemente las dos funciones lógicas. Es decir, que tenga tres entradas  $x$ ,  $y$  y  $z$ , y dos salidas  $F$  y  $G$ .
- 1.b) (1 punto) Escriba en VHDL la **architecture** que describa el *comportamiento* del circuito.
- 1.c) (0.5 puntos) Dibuje el diagrama de un circuito que implemente estas dos funciones lógicas al nivel de puertas lógicas. No es necesario que el circuito esté simplificado. A continuación, escriba en VHDL la **entity** y la **architecture** de cada una de las puertas lógicas que componen el circuito que acaba de dibujar.
- 1.d) (1 punto) Escriba en VHDL una **architecture** que describa la *estructura* del circuito que ha dibujado, instanciando y conectando las puertas lógicas que ha diseñado anteriormente.
- 1.e) (1 punto) Escriba en VHDL un banco de pruebas que permita visualizar, para todos los posibles valores de las entradas, las salidas de los circuitos diseñados en los Apartados 1.b y 1.d. Compruebe mediante inspección visual que los dos diseños funcionan correctamente. Incluya en la memoria el cronograma obtenido al realizar la simulación del banco de pruebas.

## EJERCICIO 2

Se quiere programar en VHDL un circuito combinacional que realiza la suma con acarreo de entrada de dos número binarios con signo (representados en complemento a 2) y además proporciona tres señales de salida que informan de si ha habido desbordamiento en la suma, si el resultado es cero y el signo que debiera tener el resultado. El circuito tiene las siguientes señales de entrada: los operandos de  $n$  bits  $a$  y  $b$  y una señal de acarreo de entrada un bit llamada  $c_{in}$ . El circuito tiene las siguientes señales de salida: el resultado de  $n$  bits  $res$  y las señales de 1 bit desbordamiento,  $cero$  y  $signo$ .

El valor de la señal desbordamiento es '1' solo si la operación de suma produce desbordamiento. En cualquier otro caso su valor es '0'. Para el cálculo de esta señal ha de tener en cuenta lo siguiente:

- Si los dos operandos tienen diferentes signos, no puede existir desbordamiento ya que la suma de un número positivo y un número negativo siempre provoca una disminución de la magnitud.
- Si los dos operandos y el resultado tienen el mismo signo, no ocurre desbordamiento ya que el resultado está dentro del mismo rango.
- Si los dos operandos tienen el mismo signo pero el resultado tiene un signo diferente, está ocurriendo desbordamiento. El cambio de signo indica que el resultado va más allá del límite positivo o negativo y está, por tanto, más allá del rango.

Para el cálculo de la señal de desbordamiento ha de traducir estas tres observaciones en una expresión lógica.

El valor de la señal  $cero$  es '1' solo si el resultado de la operación tiene valor cero y, además, no existe desbordamiento. En cualquier otro caso su valor es '0'.

El valor de la señal  $signo$  es el mismo bit de signo del resultado de la suma solo si no existe desbordamiento. En caso de existir desbordamiento, el signo es el inverso al resultado de la suma.

- 2.a)** (3 puntos) Escriba en VHDL la **entity** y la **architecture** que describe el comportamiento del circuito combinacional empleando solo sentencias concurrentes. Los nombres de los puertos de la **entity** deber ser los mismos que se han especificado para las señales de entrada y salida del circuito. Emplee

el convenio de especificar en primer lugar las señales de salida del circuito y posteriormente las señales de entrada. El número de bits (n) de las señales *a*, *b* y *res* se ha de expresar como constante del tipo **generic**.

En el diseño únicamente pueden emplearse los dos siguientes paquetes de la librería IEEE:

```
IEEE.std_logic_1164  
IEEE.numeric_std
```

- 2.b)** (3 puntos) Programe en VHDL un banco de pruebas que testee todas las posibles entradas al circuito diseñado en el Apartado 2.a. El número de bits de los operandos de entrada ha de ser una constante del programa cuyo valor se pueda modificar de modo que el banco de pruebas sea válido para cualquier número de bits de los operandos de entrada.

El banco de pruebas debe comparar las salidas de la UUT con las salidas esperadas, mostrando el correspondiente mensaje de error en caso de que las salidas obtenidas de la UUT no correspondan con las esperadas. El banco de pruebas debe mostrar al final del test un mensaje con el número total de errores detectados. La forma de calcular el valor esperado de la señal de desbordamiento debe ser diferente a la realizada en el circuito que se pretende testear.

Realice la simulación para el caso en que el número de bits de los operandos sea cuatro ( $n=4$ ). Incluya en la memoria el cronograma obtenido al realizar la simulación del banco de pruebas del circuito diseñado en el Apartado 2.a.

## Solución al ejercicio 1

1.a) Obtenga las funciones lógicas a partir de la tabla de verdad:

x	y	z	F	G
0	0	0	0	1
0	0	1	1	1
0	1	0	0	0
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	1	0
1	1	1	1	0

A partir de un diagrama de Karnaugh:

F		yz			
		00	01	11	10
x	0	0	1	1	0
	1	0	0	1	1

$$F(x, y, z) = x'z + xy$$

G		yz			
		00	01	11	10
x	0	1	1	1	0
	1	0	0	0	0

$$G(x, y, z) = x'y' + x'z$$

Escriba en VHDL la **entity** del circuito que implementa las dos funciones lógicas.

```

1  -----
2  -- Entity de Funcion (x, y, z) = (F, G)
3  library IEEE;
4  use IEEE.std_logic_1164.all;
5
6  entity funcFG is
7      port ( F, G: out std_logic; x, y, z: in std_logic);
8  end entity funcFG;
9  -----

```

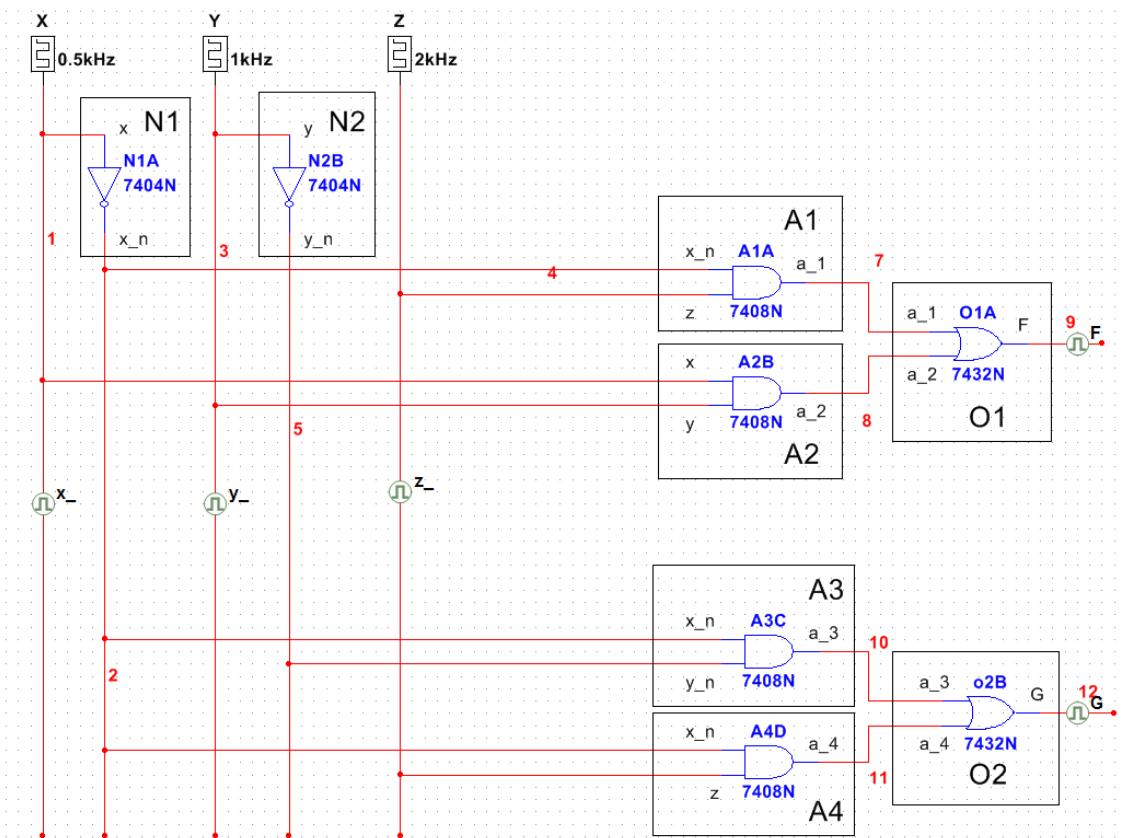
1.b) Escriba en VHDL la **architecture** que describa el *comportamiento* del circuito.

```

1   --
2   -- Architecture 1 de Funcion (x, y, z) = (F, G)
3   -- F = x'z + xy
4   -- G = x'y' + x'z
5
6   architecture arch1 of funcFG is
7     begin
8       F <= (not x and z) or (x and y);
9       G <= (not x and not y) or (not x and z);
10  end architecture arch1;
11

```

1.c) Dibuje el diagrama de un circuito que implemente estas dos funciones lógicas al nivel de puertas lógicas. No es necesario que el circuito esté simplificado.



A continuación, escriba en VHDL la **entity** y la **architecture** de cada una de las puertas lógica que componen el circuito que acaba de dibujar.

```
2 -----  
3 -- NOT  
4 library IEEE;  
5 use IEEE.std_logic_1164.all;  
6  
7 entity not1 is  
8     port ( y0 : out std_logic; x0 : in std_logic);  
9 end entity not1;  
10  
11 -----  
12  
13 architecture not1 of not1 is  
14 begin  
15     y0 <= not x0;  
16 end architecture not1;  
17 -----
```

```
2 -----  
3 -- AND  
4 library IEEE;  
5 use IEEE.std_logic_1164.all;  
6  
7 entity and2 is  
8     port ( y0 : out std_logic; x0, x1 : in std_logic);  
9 end entity and2;  
10  
11 -----  
12  
13 architecture and2 of and2 is  
14 begin  
15     y0 <= x0 and x1;  
16 end architecture and2;  
17 -----  
18 -----
```

```
2 -----  
3 -- OR  
4 library IEEE;  
5 use IEEE.std_logic_1164.all;  
6  
7 entity or2 is  
8     port ( y0 : out std_logic; x0, x1 : in std_logic);  
9 end entity or2;  
10  
11 -----  
12  
13 architecture or2 of or2 is  
14 begin  
15     y0 <= x0 or x1;  
16 end architecture or2;  
17 -----
```

1.d) Escriba en VHDL una **architecture** que describa la *estructura* del circuito que ha dibujado, instanciando y conectando las puertas lógicas que ha diseñado anteriormente.

```
2 -----  
3 -- Architecture 2 de Funcion (x, y, z) = (F, G)  
4 -- F = x'z + xy  
5 -- G = x'y' + x'z  
6 -----  
7  
8 library IEEE;  
9 use IEEE.std_logic_1164.all;  
10  
11 architecture arch2 of funcFG is  
12     signal xn, yn, a_1, a_2, a_3, a_4 : std_logic;  
13  
14     -- Declaracion de los componentes  
15  
16     component not1 is  
17         port ( y0 : out std_logic; x0 : in std_logic);  
18     end component not1;  
19  
20     component and2 is  
21         port ( y0 : out std_logic; x0, x1 : in std_logic);  
22     end component and2;  
23  
24     component or2 is  
25         port ( y0 : out std_logic; x0, x1 : in std_logic);  
26     end component or2;  
27  
28 begin  
29     -- Instanciacion y conexion de los componentes  
30     N1 : component not1 port map (xn, x);  
31     N2 : component not1 port map (yn, y);  
32     A1 : component and2 port map (a_1, xn, z);  
33     A2 : component and2 port map (a_2, x, y);  
34     A3 : component and2 port map (a_3, xn, yn);  
35     A4 : component and2 port map (a_4, xn, z);  
36     O1 : component or2 port map (F, a_1, a_2);  
37     O2 : component or2 port map (G, a_3, a_4);  
38 end architecture arch2;
```

1.e) Escriba en VHDL un banco de pruebas que permita visualizar, para todos los posibles valores de las entradas, las salidas de los circuitos diseñados en los Apartados 1.b y 1.d.

Aparatado 1.b

```

2   --
3   -- Banco de pruebas de la funcion FG
4   -- con la architecture arch1 del ejercicio 1
5   --
6
7   library IEEE;
8   use IEEE.std_logic_1164.all;
9   use IEEE.numeric_std.all;
10
11  entity bp1_funcFG is
12      constant DELAY :time := 20 ns; -- Retardo usado en el test
13  end entity bp1_funcFG;
14
15  architecture bp1_funcFG of bp1_funcFG is
16      signal F, G: std_logic;
17      signal x, y, z: std_logic;
18
19  begin
20      UUT : entity work.funcFG(arch1) port map (F, G, x, y, z);
21
22      vec_test : process is
23          variable valor : unsigned (2 downto 0);
24          begin
25              -- Genera todos los posibles valores de entrada
26              for i in 0 to 7 loop
27                  valor := to_unsigned(i, 3);
28                  x <= std_logic(valor(2));
29                  y <= std_logic(valor(1));
30                  z <= std_logic(valor(0));
31                  wait for DELAY;
32              end loop;
33              wait; -- Final de simulacion
34      end process vec_test;
35  end architecture bp1_funcFG;

```

Apartado 1.d

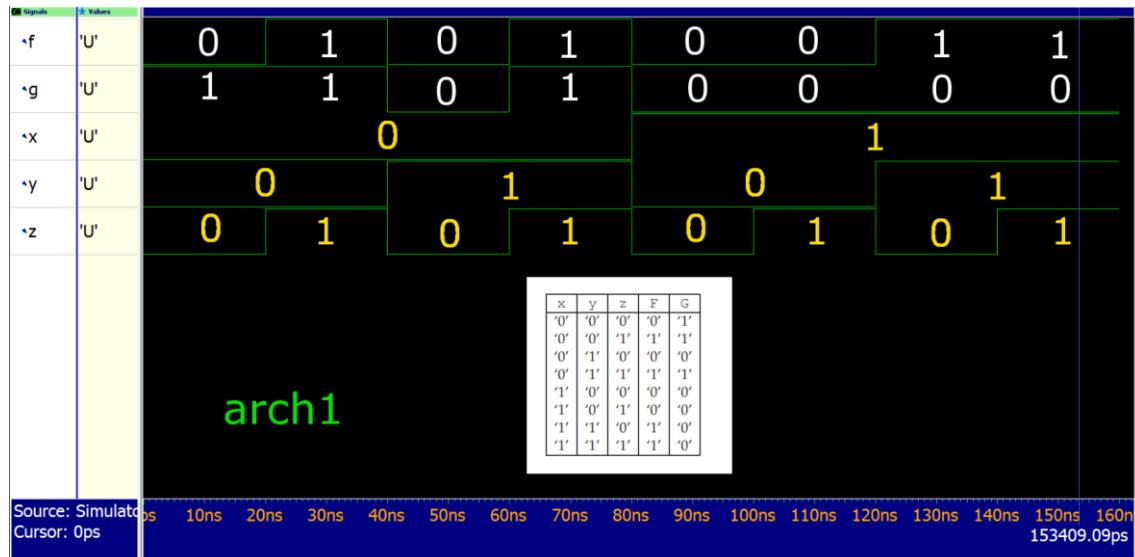
```

2   -- Banco de pruebas de la función FG
3   -- con la architecture arch2 del ejercicio 1
4
5
6
7   library IEEE;
8   use IEEE.std_logic_1164.all;
9   use IEEE.numeric_std.all;
10
11  entity bp2_funcFG is
12      constant DELAY : time := 20 ns; -- Retardo usado en el test
13  end entity bp2_funcFG;
14
15  architecture bp2_funcFG of bp2_funcFG is
16      signal F, G: std_logic;
17      signal x, y, z: std_logic;
18
19  begin
20      UUT : entity work.funcFG(arch2) port map (F, G, x, y, z);
21
22      vec_test : process is
23          variable valor : unsigned (2 downto 0);
24      begin
25          -- Genera todos los posibles valores de entrada
26          for i in 0 to 7 loop
27              valor := to_unsigned(i, 3);
28              x <= std_logic(valor(2));
29              y <= std_logic(valor(1));
30              z <= std_logic(valor(0));
31              wait for DELAY;
32          end loop;
33          wait; -- Final de simulación
34      end process vec_test;
35  end architecture bp2_funcFG;
36

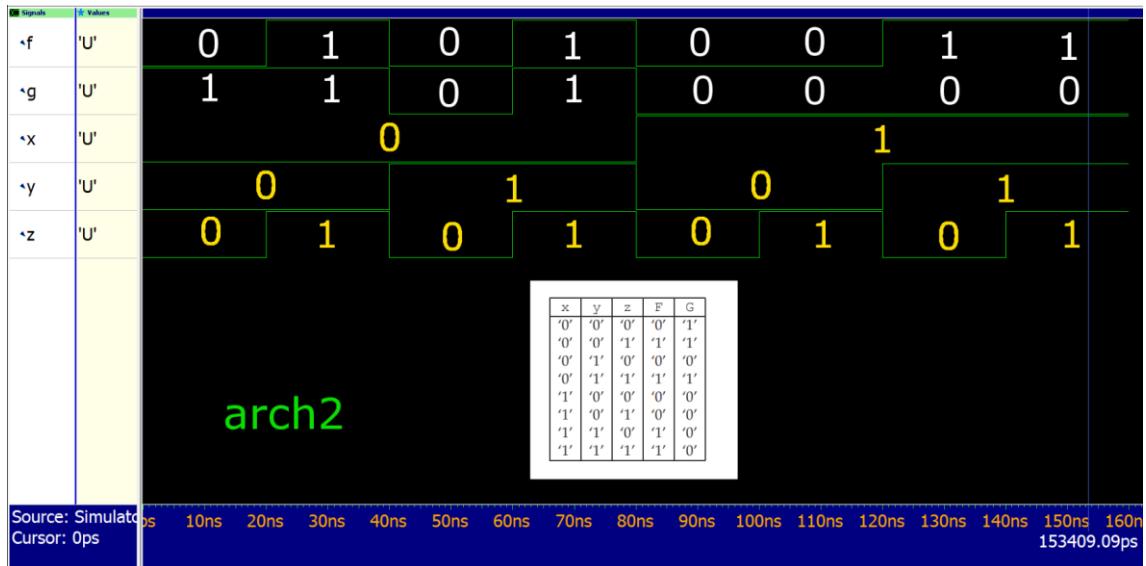
```

Incluya en la memoria el cronograma obtenido al realizar la simulación del banco de pruebas.

Apartado 1.b



Apartado 1.d



## Solución al ejercicio 2

2.a) Escriba en VHDL la **entity** y la **architecture** que describe el comportamiento del circuito combinacional empleando sólo sentencias concurrentes.

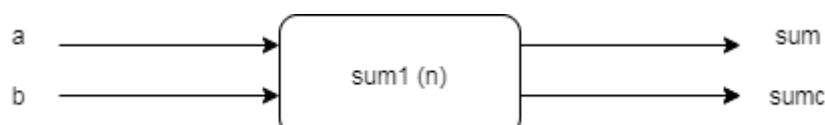
- Los nombres de los puertos de la **entity** deben ser los mismos que se han especificado para las señales de entrada y salida del circuito.
- Emplee el convenio de especificar en primer lugar las señales de salida del circuito y posteriormente las señales de entrada.
- El número de bits (n) de las señales a, b, res se ha de expresar como constante de tipo **generic**.
- En el diseño únicamente pueden emplearse los paquetes de la librería IEEE.
  - IEEE.std\_logic\_1164
  - IEEE.numeric\_std

```
1 -----  
2 --Solucion del ejercicio 2  
3 -----  
4 library ieee;  
5 use ieee.std_logic_1164.all;  
6 use ieee.numeric_std.all;  
7  
8 entity sum is  
9     generic (n : integer);  
10    port (res : out std_logic_vector (n-1 downto 0);  
11           desbordamiento : out std_logic;  
12           cero, signo : out std_logic;  
13           a, b : in std_logic_vector(n-1 downto 0);  
14           cin : in std_logic);  
15 end entity sum;  
16  
17 architecture archsum of sum is  
18     --Declaracion de constantes  
19     constant WIDTH : integer := n;  
20  
21     --Declaracion de las senales  
22     signal a1 : std_logic_vector(n-1 downto 0) := (others => '0');  
23     signal a2, a3 : std_logic := '0';  
24  
25 begin  
26     -- Instaciacion y conexion de los componentes  
27     s1 : entity work.sum2(archsum2) generic map (WIDTH) port map (a1, a, b, cin);  
28     o1 : entity work.overflow(archoverflow) generic map (WIDTH) port map (a3, a, b, a1);  
29     desbordamiento <= a3;  
30     res <= a1;  
31     ch1 : entity work.checkzero(archcheckzero) generic map (WIDTH) port map (a2, a1);  
32     cero <= a2 and not a3;  
33     sg1 : entity work.signo(archsing) generic map (WIDTH) port map (signo, a1);  
34 end architecture archsum;
```

### Pasos y componentes involucrados en la resolución del apartado 2.a

#### Sumador 1

Toma dos números de n bits en complemento a 2 y devuelve su suma con y sin acarreo.



```

1  -----
2  --sum1. Componente del ejercicio 2.
3  --Entrada: (a, b) dos números de n bits en complemento a 2
4  --Salida: (sum, sumc) Su suma con y sin acarreo.
5
6 library ieee;
7 use ieee.std_logic_1164.all;
8 use ieee.numeric_std.all;
9
10 entity sum1 is
11     generic (n:integer);
12     port (sum, sumc : out std_logic_vector (n-1 downto 0);
13           a, b : in std_logic_vector (n-1 downto 0));
14 end entity sum1;
15
16 architecture archsum1 of sum1 is
17     constant carry : std_logic_vector (n downto 0) := std_logic_vector(to_signed(1, n+1));
18     signal aux : std_logic_vector (n downto 0) := (others => '0');
19
20 begin
21     sum <= std_logic_vector(signed(a)+signed(b));
22     aux <= std_logic_vector(signed(a)+signed(b)+signed(carry));
23     sumc <= aux(n-1 downto 0);
24 end architecture archsum1;

```

### Banco de pruebas.

El código fuente de los bancos de pruebas figura al final de la memoria en el anexo: *bancos de pruebas*.

El banco de pruebas de este componente figura bajo la referencia:

*Banco de pruebas del componente sum1 del ejercicio 2*

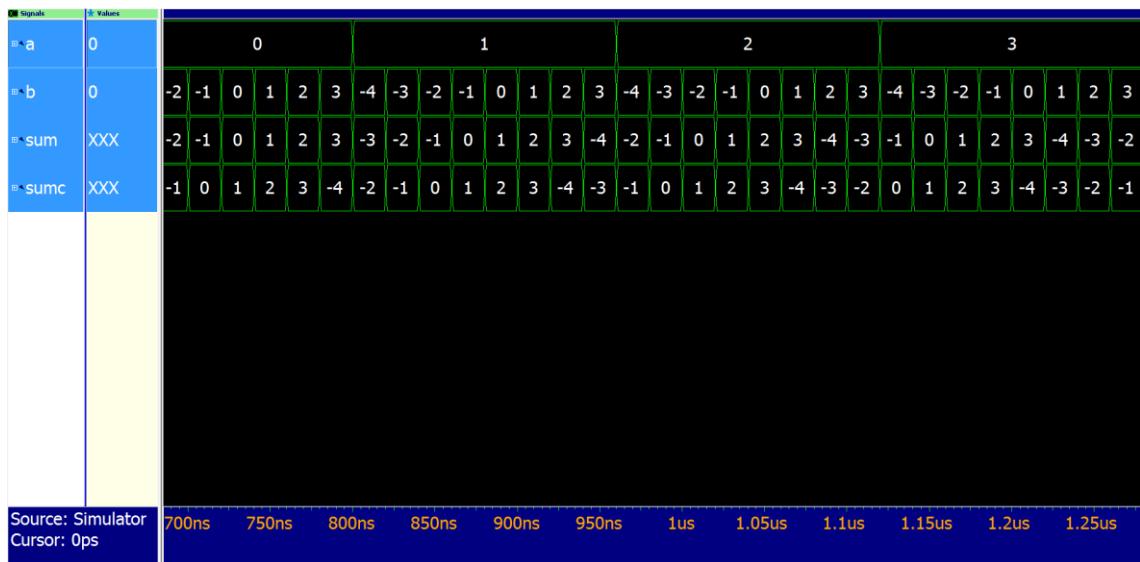
### Resultados esperados para n = 3.

		c_in = 0		c_in = 1				c_in = 0		c_in = 1				c_in = 0		c_in = 1				c_in = 0		c_in = 1	
-4 : 100		0:000	1:001	-3 : 101		2:010	-3:011	-4 : 100		2:010	3:011	-4 : 100		2:010	3:011	-4 : 100		3:011	-4 : 100	-4 : 100		3:011	-4 : 100
-3 : 101		1:001	2:010	-2 : 110		3:011	-2:110	-3 : 101		3:011	-4 : 100	-2 : 110		-4 : 100	-3 : 101	-3 : 101		-4 : 100	-3 : 101	-2 : 110		-3 : 101	-2 : 110
-2 : 110		2:010	3:011	-1 : 111		3:011	-4 : 100	-1 : 111		-4 : 100	-3 : 101	-2 : 110		-4 : 100	-3 : 101	-2 : 110		-3 : 101	-2 : 110	-1 : 111		-2 : 110	-1 : 111
-1 : 111		3:011	-4 : 100	0 : 000		-4 : 100	-3 : 101	0 : 000		-3 : 101	-2 : 110	-1 : 111		-3 : 101	-2 : 110	-1 : 111		-2 : 110	-1 : 111	0 : 000		-1 : 111	0 : 000
0 : 000		-4 : 100	-3 : 101	1 : 001		-2 : 110	-1 : 111	1 : 001		-2 : 110	-1 : 111	1 : 001		-1 : 111	0 : 000	2 : 010		0 : 000	1 : 111	2 : 010		1 : 111	2 : 010
1 : 001		-3 : 101	-2 : 110	2 : 010		-1 : 111	0 : 000	2 : 010		-1 : 111	0 : 000	2 : 010		0 : 000	1 : 111	3 : 011		0 : 000	1 : 001	0 : 000		1 : 111	2 : 010
2 : 010		-2 : 110	-1 : 111	3 : 011		-1 : 111	0 : 000	3 : 011		0 : 000	1 : 111	3 : 011		1 : 111	2 : 010	3 : 011		2 : 010	-3 : 101	2 : 010		-3 : 101	-2 : 110
3 : 011		-1 : 111	0 : 000	0 : 000		1 : 111	2 : 010	0 : 000		1 : 111	2 : 010	0 : 000		1 : 111	2 : 010	0 : 000		3 : 011	-2 : 110	0 : 000		1 : 111	-1 : 111
0 : 000		1 : 001	2 : 010	1 : 001		2 : 010	3 : 011	1 : 001		2 : 010	3 : 011	1 : 001		2 : 010	3 : 011	1 : 001		2 : 010	3 : 011	1 : 001		2 : 010	3 : 011
1 : 001		2 : 010	3 : 011	2 : 010		3 : 011	-4 : 100	2 : 010		3 : 011	-4 : 100	2 : 010		3 : 011	-4 : 100	2 : 010		3 : 011	-4 : 100	2 : 010		3 : 011	-4 : 100
2 : 010		3 : 011	-4 : 100	3 : 011		-4 : 100	-3 : 101	3 : 011		-4 : 100	-3 : 101	3 : 011		-4 : 100	-3 : 101	3 : 011		-4 : 100	-3 : 101	3 : 011		-4 : 100	-3 : 101
3 : 011		-4 : 100	-3 : 101	-4 : 100		-3 : 101	-2 : 110	-4 : 100		-3 : 101	-2 : 110	-4 : 100		-3 : 101	-2 : 110	-4 : 100		-3 : 101	-2 : 110	-4 : 100		-3 : 101	-2 : 110
-4 : 100		-3 : 101	-2 : 110	-1 : 111		-2 : 110	-1 : 111	-3 : 101		-2 : 110	-1 : 111	-4 : 100		-3 : 101	-2 : 110	-4 : 100		-3 : 101	-2 : 110	-4 : 100		-3 : 101	-2 : 110
-3 : 101		-2 : 110	-1 : 111	-2 : 110		-1 : 111	0 : 000	-3 : 101		-2 : 110	0 : 000	-4 : 100		-3 : 101	0 : 000	-4 : 100		-3 : 101	0 : 000	-4 : 100		-3 : 101	0 : 000
-2 : 110		-1 : 111	0 : 000	-1 : 111		0 : 000	1 : 111	-2 : 110		1 : 111	2 : 010	-3 : 101		1 : 111	2 : 010	-3 : 101		1 : 111	2 : 010	-3 : 101		1 : 111	2 : 010
-1 : 111		0 : 000	1 : 111	0 : 000		1 : 111	2 : 010	0 : 000		1 : 111	2 : 010	0 : 000		1 : 111	2 : 010	0 : 000		1 : 111	2 : 010	0 : 000		1 : 111	2 : 010
0 : 000		1 : 111	2 : 010	1 : 111		2 : 010	3 : 011	1 : 111		2 : 010	3 : 011	1 : 111		2 : 010	3 : 011	1 : 111		2 : 010	3 : 011	1 : 111		2 : 010	3 : 011
1 : 111		2 : 010	3 : 011	2 : 010		3 : 011	-4 : 100	2 : 010		3 : 011	-4 : 100	2 : 010		3 : 011	-4 : 100	2 : 010		3 : 011	-4 : 100	2 : 010		3 : 011	-4 : 100
2 : 010		3 : 011	-4 : 100	3 : 011		-4 : 100	-3 : 101	3 : 011		-4 : 100	-3 : 101	3 : 011		-4 : 100	-3 : 101	3 : 011		-4 : 100	-3 : 101	3 : 011		-4 : 100	-3 : 101
3 : 011		-4 : 100	-3 : 101	-4 : 100		-3 : 101	-2 : 110	-4 : 100		-3 : 101	-2 : 110	-4 : 100		-3 : 101	-2 : 110	-4 : 100		-3 : 101	-2 : 110	-4 : 100		-3 : 101	-2 : 110

### Resultados obtenidos para n = 3.

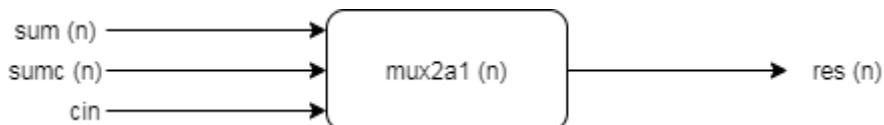
Signals	* Values
<code>a</code>	0 -4
<code>b</code>	0 -4 -3 -2 -1 0 1 2 3
<code>sum</code>	XXX 0 1 2 3 -4 -3 -2 -1 1 2 3 -4 -3 -2 -1 0 2 3 -4 -3 -2 -1 0 1 3 -4 -3 -2 -1 0 1 2
<code>sumc</code>	XXX 1 2 3 -4 -3 -2 -1 0 2 3 -4 -3 -2 -1 0 1 3 -4 -3 -2 -1 0 1 2

Source: Simulator  
Cursor: 0ps



### Multiplexor 2 a 1

Toma dos números en complemento a 2 de n bits y un bit de selección y devuelve el número de n bits en complemento a 2 que le indica el bit de selección.



```

1   --
2   -- mux2a1. Componente del ejercicio 2
3   -- Entrada: (sum, sumc) dos números de n bits en C2
4   -- Entrada: cin -> bit de selección
5   -- Salida: cin = 1 -> sumc
6   -- Salida: cin = 0 -> sum
7   --
8   library IEEE;
9   use IEEE.std_logic_1164.all;
10
11 entity mux2a1 is
12     generic (n:integer);
13     port (res : out std_logic_vector(n-1 downto 0);
14           sum, sumc : in std_logic_vector(n-1 downto 0);
15           cin : in std_logic);
16 end mux2a1;
17
18 architecture archmux of mux2a1 is
19 begin
20     res <= sumc when cin='1' else sum;
21 end archmux;

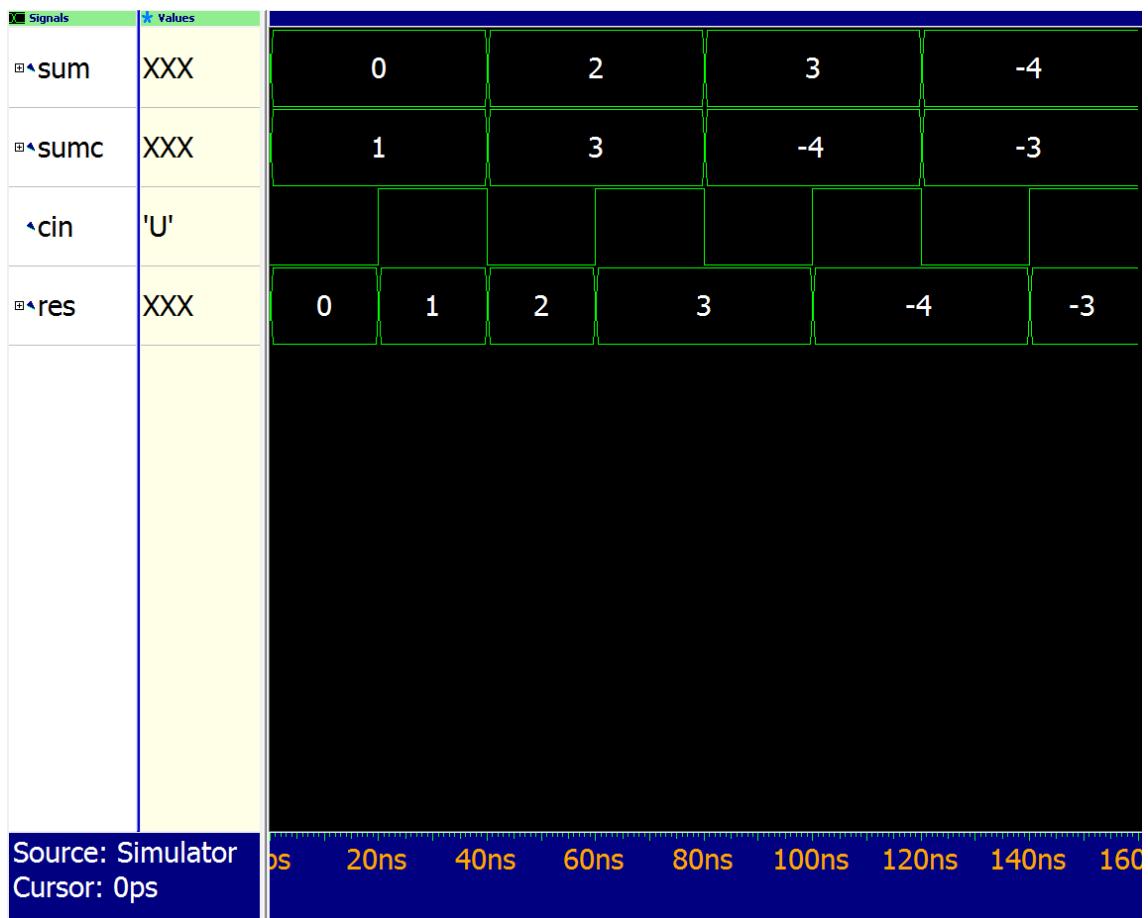
```

### Banco de pruebas.

El banco de pruebas de este componente figura en el Anexo: *Banco de Pruebas* bajo la referencia:

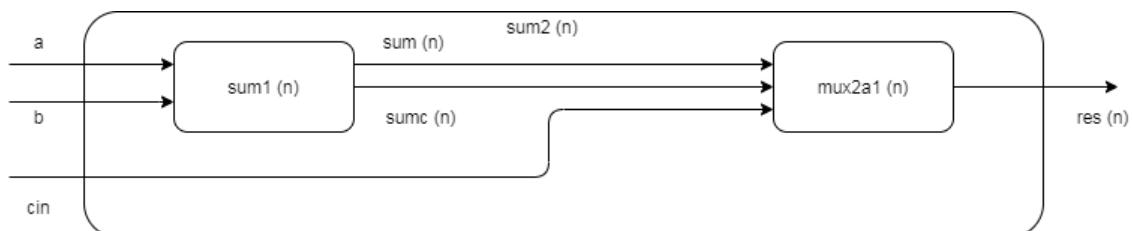
*Banco de pruebas del componente mux2a1 del ejercicio 2*

### Cronograma.



### Sumador 2

Toma dos números de  $n$  bits en complemento a 2 y un bit de acarreo y devuelve la suma considerando el acarreo.



```

1  -----
2  -- sumc. Componente del ejercicio 2.
3  -- ENTRADA: (sum, sumc) dos números de n bits en C2.
4  -- ENTRADA: cin -> acarreo.
5  -- SALIDA: res -> resultado en C2 de sumar los números
6  -- de la entrada contemplando el acarreo.
7  -----
8  library ieee;
9  use ieee.std_logic_1164.all;
10 use ieee.numeric_std.all;
11
12 entity sum2 is
13     generic (n:integer);
14     port (res : out std_logic_vector(n-1 downto 0);
15           a, b : in std_logic_vector(n-1 downto 0);
16           cin : in std_logic);
17 end entity sum2;
18
19 architecture archsum2 of sum2 is
20     -- Declaracion de constantes
21     constant WIDTH : integer := n;
22
23     -- Declaración de señales
24     signal sum, sumc : std_logic_vector(n-1 downto 0) := (others => '0');
25
26     begin
27         --Instanciación y conexión de los componentes
28         s : entity work.sum1(archsum1) generic map (WIDTH) port map (sum, sumc, a, b);
29         m : entity work.mux2a1(archmux) generic map (WIDTH) port map (res, sum, sumc, cin);
30     end architecture archsum2;

```

### Banco de pruebas.

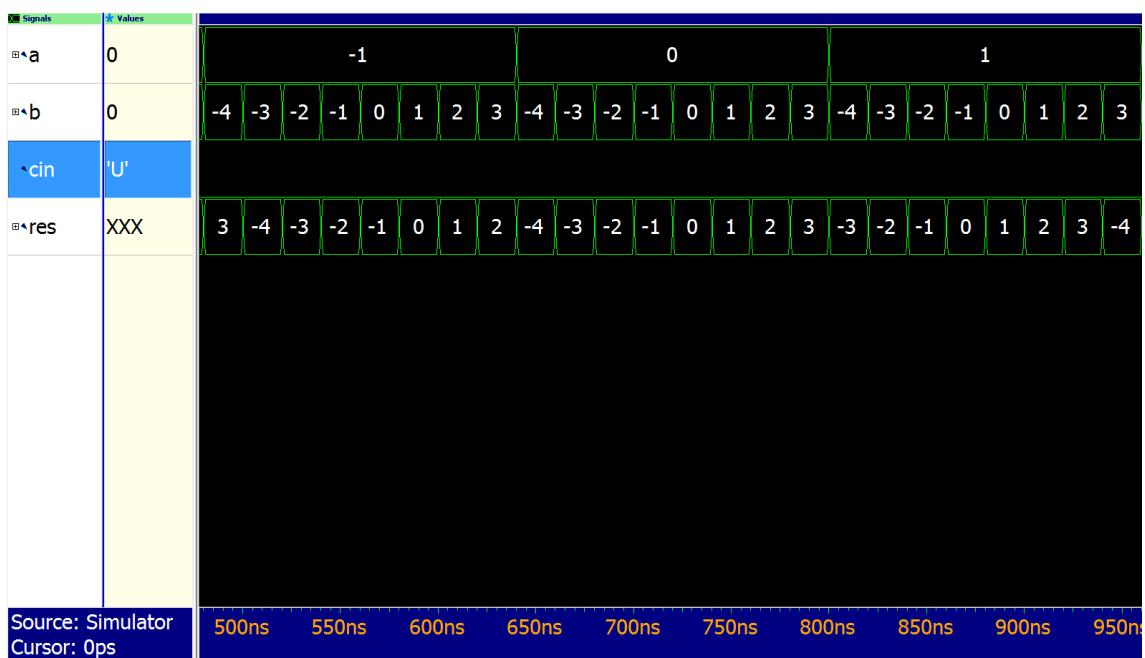
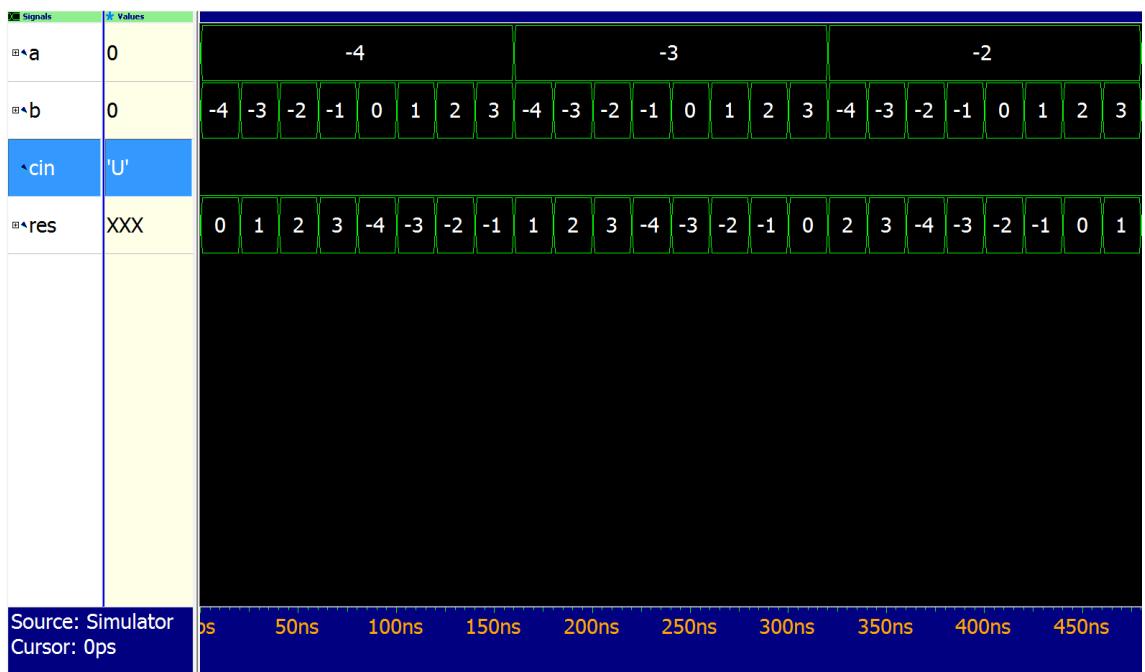
El banco de pruebas de este componente figura en el Anexo: *Banco de Pruebas* bajo la referencia:

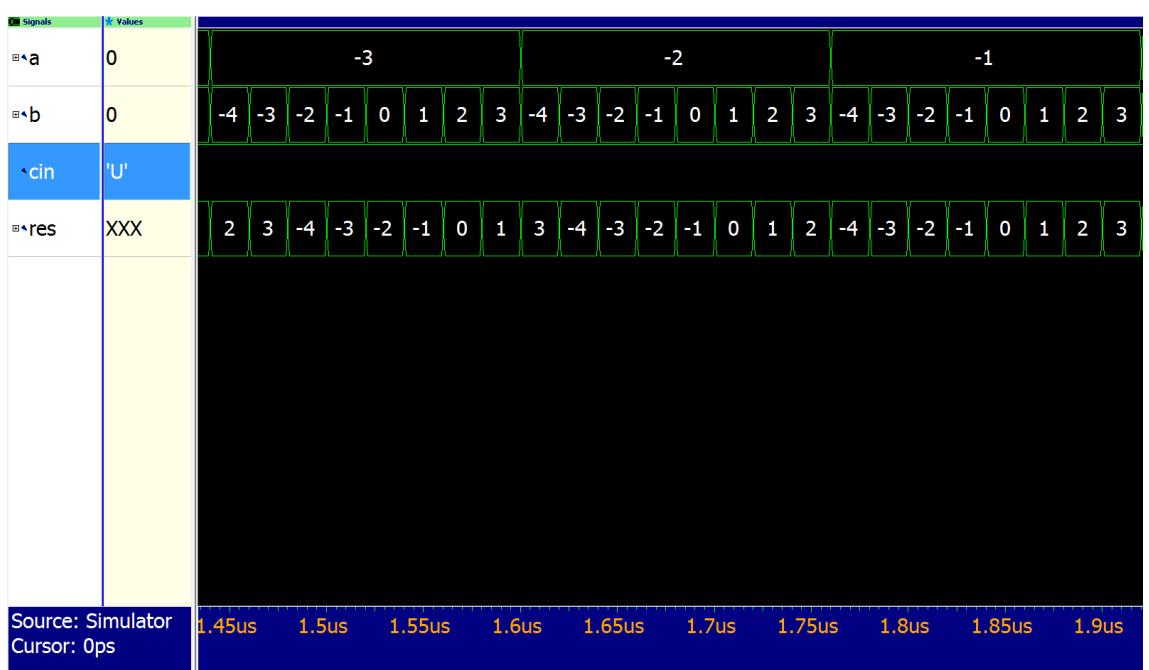
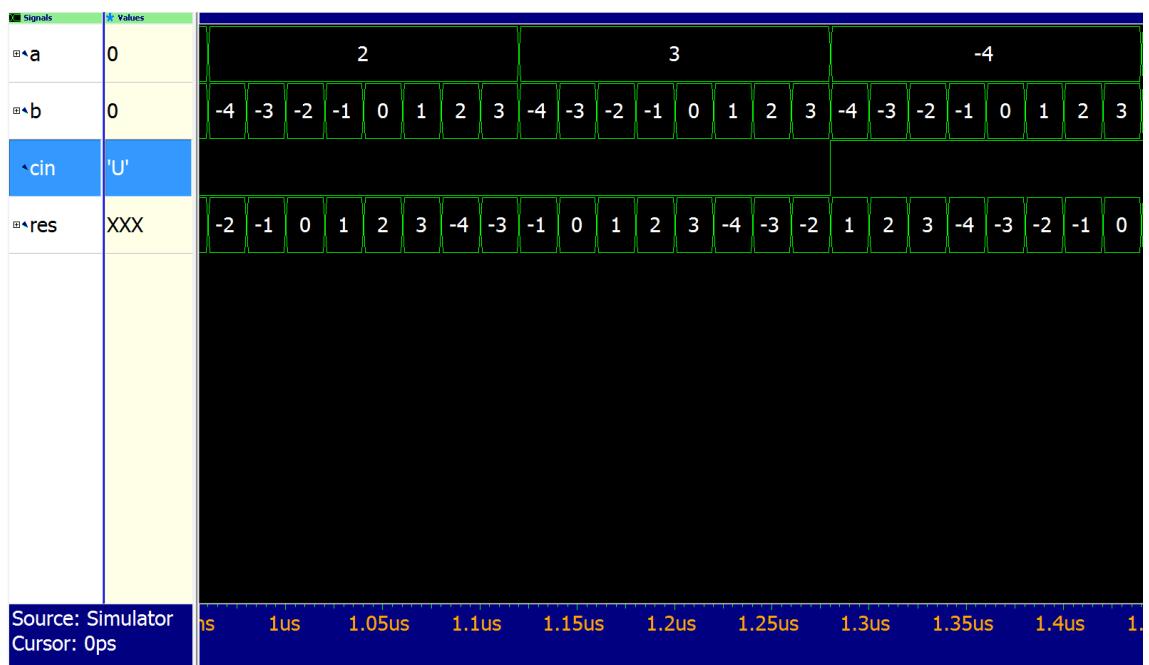
*Banco de pruebas del componente sum2 del ejercicio 2*

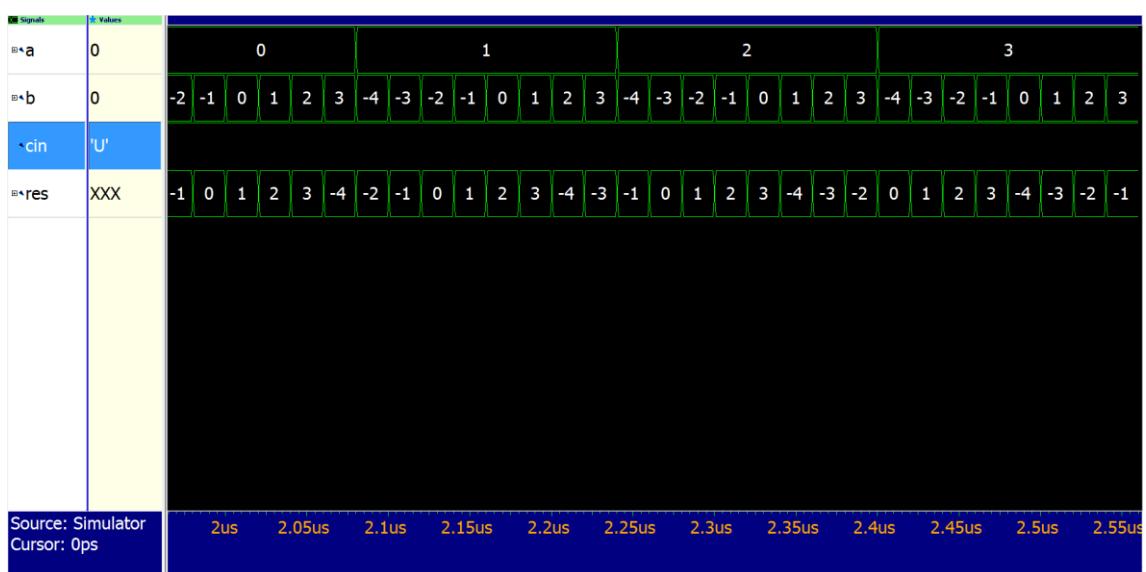
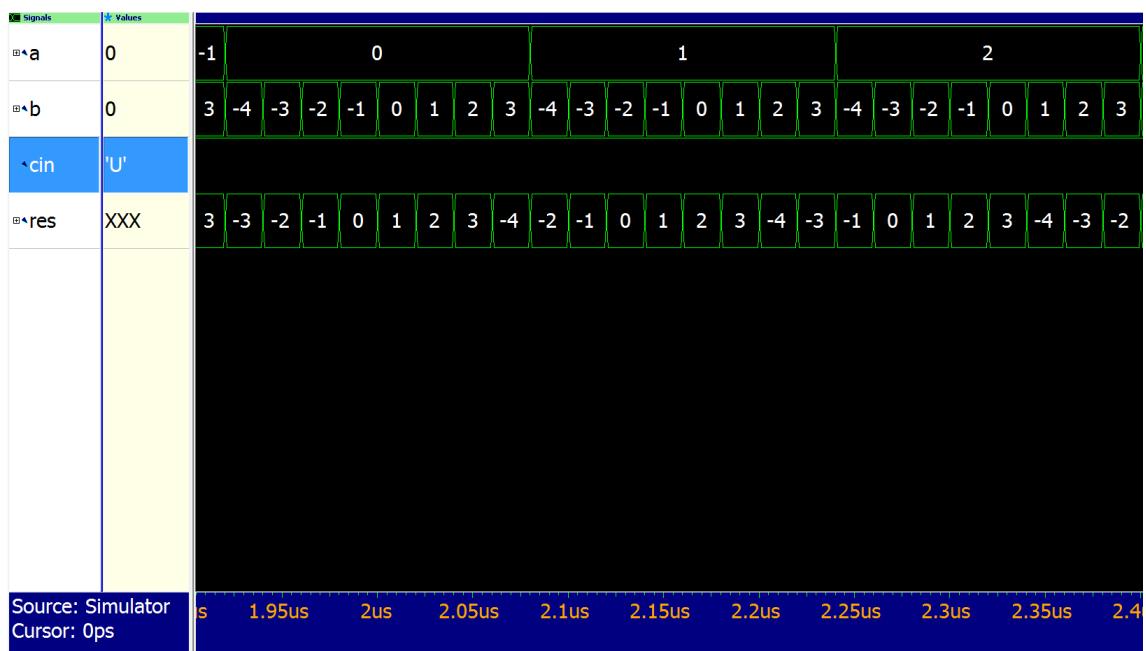
### Resultados esperados para n = 3.

		c_in = 0		c_in = 1		c_in = 0		c_in = 1		c_in = 0		c_in = 1		c_in = 0		c_in = 1		
-4:100	-4:100	0:000	1:001	-4:100	1:001	2:010	-4:100	2:010	3:011	-4:100	2:010	3:011	-4:100	3:011	-4:100	3:011	-4:100	
	-3:101	1:001	2:010	-3:101	2:010	3:011	-3:101	2:010	3:011	-3:101	2:010	3:011	-3:101	2:010	-3:101	-4:100	-3:101	
	-2:110	2:010	3:011	-2:110	3:011	-4:100	-2:110	3:011	-4:100	-2:110	3:011	-4:100	-2:110	3:011	-2:110	-3:101	-2:110	
	-1:111	3:011	-4:100	-1:111	-4:100	-3:101	-1:111	-4:100	-3:101	-1:111	-3:101	-2:110	-1:111	-3:101	-2:110	-1:111	-1:111	
	0:000	-4:100	-3:101	0:000	-3:101	-2:110	0:000	-3:101	-2:110	0:000	-2:110	-1:111	0:000	-1:111	0:000	0:000	0:000	
	1:001	-3:101	-2:110	1:001	-2:110	-1:111	1:001	-2:110	-1:111	1:001	-1:111	0:000	1:001	0:000	1:111	1:111	1:111	
	2:010	-2:110	-1:111	2:010	-1:111	0:000	2:010	-1:111	0:000	2:010	0:000	1:111	2:010	1:111	2:010	1:111	2:010	
	3:011	-1:111	0:000	3:011	0:000	1:111	3:011	0:000	1:111	3:011	1:111	2:010	3:011	2:010	3:011	2:010	3:011	
		c_in = 0		c_in = 1		c_in = 0		c_in = 1		c_in = 0		c_in = 1		c_in = 0		c_in = 1		
0:000	-4:100	-4:100	-3:101	-4:100	-3:101	-2:110	-4:100	-2:110	-1:111	-4:100	-2:110	-1:111	-4:100	-1:111	0:000	-4:100	-1:111	0:000
	-3:101	-3:101	-2:110	-3:101	-2:110	-1:111	-3:101	-2:110	-1:111	-3:101	-2:110	-1:111	-3:101	-1:111	0:000	-3:101	0:000	1:111
	-2:110	-2:110	-1:111	-2:110	-1:111	0:000	-2:110	-1:111	0:000	-2:110	0:000	1:111	-2:110	1:111	2:010	-2:110	1:111	2:010
	-1:111	-1:111	0:000	-1:111	0:000	1:111	-1:111	0:000	1:111	-1:111	0:000	1:111	-1:111	2:010	3:011	0:000	3:011	4:100
	0:000	0:000	1:111	0:000	1:111	2:010	0:000	1:111	2:010	0:000	1:111	2:010	0:000	3:011	4:100	1:001	-4:100	-3:101
	1:001	1:111	2:010	1:001	1:111	2:010	3:011	1:001	1:111	2:010	3:011	4:100	1:001	3:011	-4:100	1:001	-4:100	-3:101
	2:010	2:010	3:011	2:010	3:011	4:100	2:010	3:011	4:100	2:010	3:011	4:100	2:010	3:011	-3:101	2:010	-3:101	-2:110
	3:011	3:011	4:100	3:011	4:100	3:011	4:100	3:011	4:100	3:011	4:100	3:011	4:100	3:011	4:100	3:011	4:100	3:011

### Resultados obtenidos para n = 3.

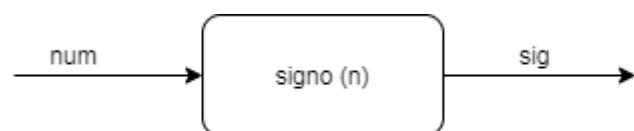






## Signo

Toma un número de n bits en complemento a 2 y devuelve el signo del número.



```

1  -- signo. Componente del ejercicio 2.
2  -- Entrada: num -> numero en C2 de n bits.
3  -- Salida: sig = 1 -> num < 0.
4  -- Salida: sig = 0 -> sig >= 0.
5
6
7  library IEEE;
8  use IEEE.std_logic_1164.all;
9
10 entity signo is
11     generic (n:integer);
12     port (sig: out std_logic; num : in std_logic_vector (n-1 downto 0));
13 end entity signo;
14
15 architecture archsigno of signo is
16 begin
17     sig <= num(n-1);
18 end architecture archsigno;

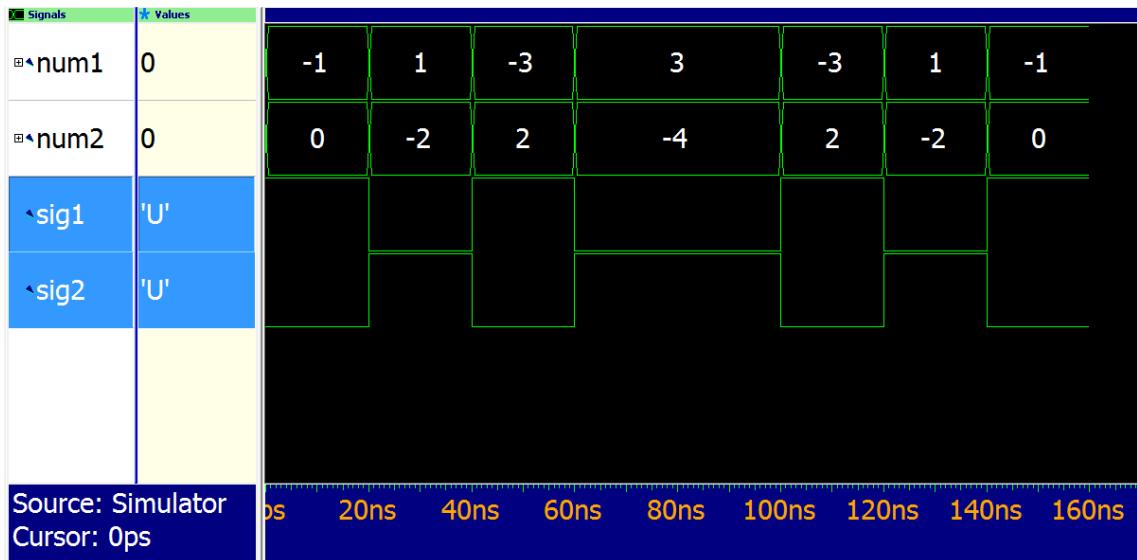
```

### Banco de pruebas.

El banco de pruebas de este componente figura en el Anexo: *Banco de Pruebas* bajo la referencia:

*Banco de pruebas del componente signo del ejercicio 2*

### Cronograma.



### **Componente fof**

Tomando del enunciado las condiciones para el cálculo del bit de desbordamiento:

- Si los dos operandos tienen distintos signos, no puede existir desbordamiento.
- Si los dos operandos y el resultado tienen el mismo signo, no ocurre desbordamiento.
- Si los dos operandos tienen el mismo signo, pero el resultado tiene un signo diferente, está ocurriendo desbordamiento.

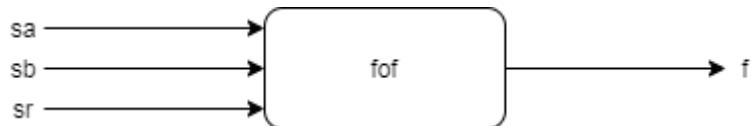
En forma de tabla:

Signo (a)	Signo (b)	Signo (res)	desbordamiento
0	0	0	0
0	0	1	1

0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

El componente fof toma 3 bits y evalúa la función lógica:  $f = sa \times sb \times sr' + sa' \times sb' \times sr$ .

Se trata de una función auxiliar para el cálculo del desbordamiento evaluando los signos de los sumandos y el resultado.



```

1  -----
2  -- fof. Componente del ejercicio 2.
3  -- funcion auxiliar para el cálculo del desbordamiento
4  -- Entrada: sa, sb, sr -> 3 bits
5  -- Salida: fof =sa x sb x sr' + sa' x sb' x sr
6  -----
7
8  library IEEE;
9  use IEEE.std_logic_1164.all;
10
11 entity fof is
12     port ( f : out std_logic; n1, n2, r: in std_logic);
13 end entity fof;
14
15
16 architecture archfof of fof is
17 begin
18     f <= (n1 and n2 and (not r)) or ((not n1) and (not n2) and r);
19 end architecture archfof;

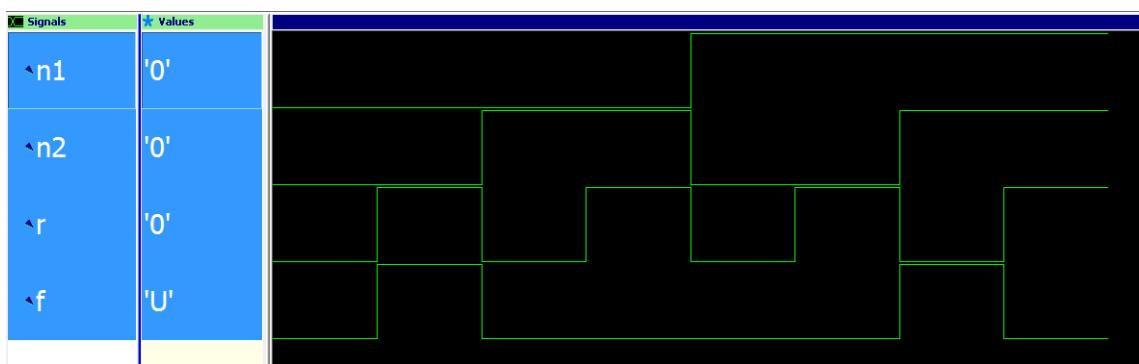
```

#### Banco de pruebas.

El banco de pruebas de este componente figura en el Anexo: *Banco de Pruebas* bajo la referencia:

*Banco de pruebas del componente fof del ejercicio 2*

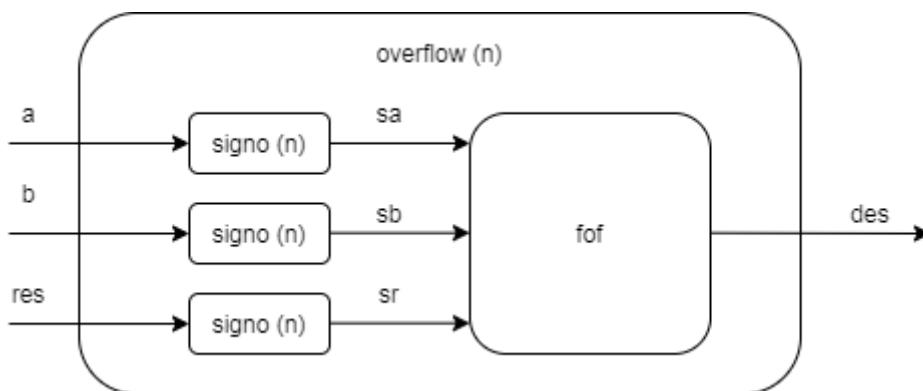
#### Cronograma.



Sa	Sb	Sr	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

## Overflow

Toma dos números de n bits en complemento a 2 y el resultado de su suma y evalúa si se produce desbordamiento.



```

1  --
2  -- overflow. Componente del ejercicio 2
3  -- ENTRADA: (a, b, res) -> 3 numeros de n bits en C2
4  -- SALIDA: des. bit de desbordamiento.
5  -- SALIDA: des = 1 -> hay desbordamiento
6  -- SALIDA: des = 0 -> no hay desbordamiento
7  --
8  library ieee;
9  use ieee.std_logic_1164.all;
10 use ieee.numeric_std.all;
11
12 entity overflow is
13     generic (n:integer);
14     port (des : out std_logic;
15           a, b, res : in std_logic_vector (n-1 downto 0));
16 end entity overflow;
17
18 architecture archoverflow of overflow is
19     -- Declaracion de las constantes
20     constant WIDTH : integer := n;
21
22     -- Declaracion de las señales
23     signal sa, sb, sr: std_logic := '0';
24
25 begin
26     -- Instanciacion y conexion de los componentes
27     S1 : entity work.signo(archsigno) generic map (WIDTH) port map (sa, a);
28     S2 : entity work.signo(archsigno) generic map (WIDTH) port map (sb, b);
29     S3 : entity work.signo(archsigno) generic map (WIDTH) port map (sr, res);
30     F0 : entity work.of(archof) port map (des, sa, sb, sr);
31 end architecture archoverflow;

```

Banco de pruebas.

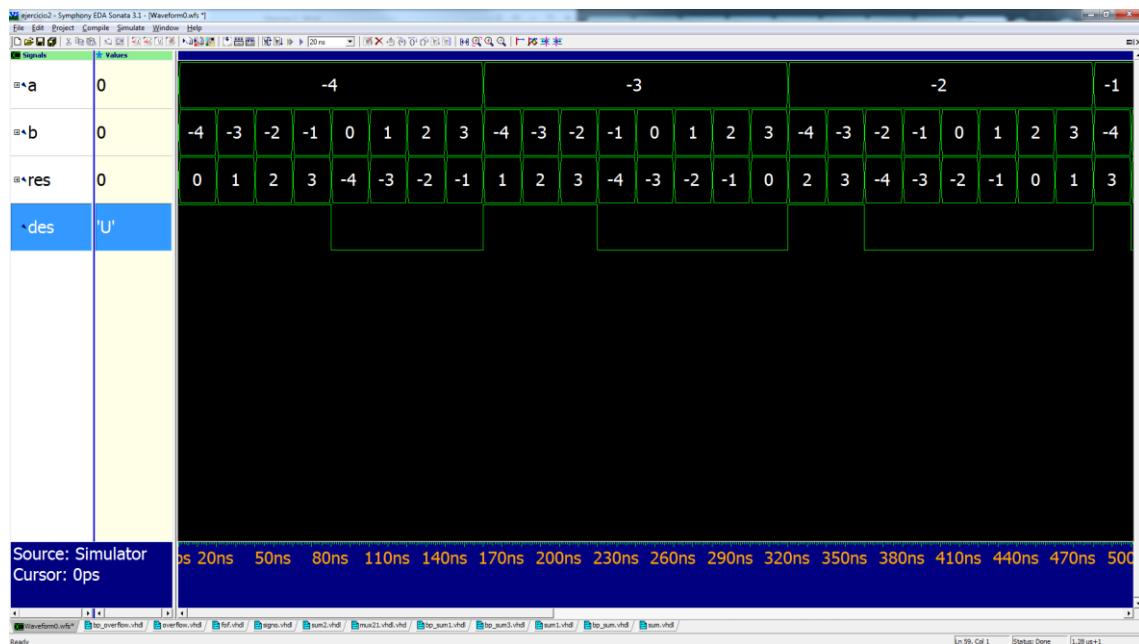
El banco de pruebas de este componente figura en el Anexo: *Banco de Pruebas* bajo la referencia:

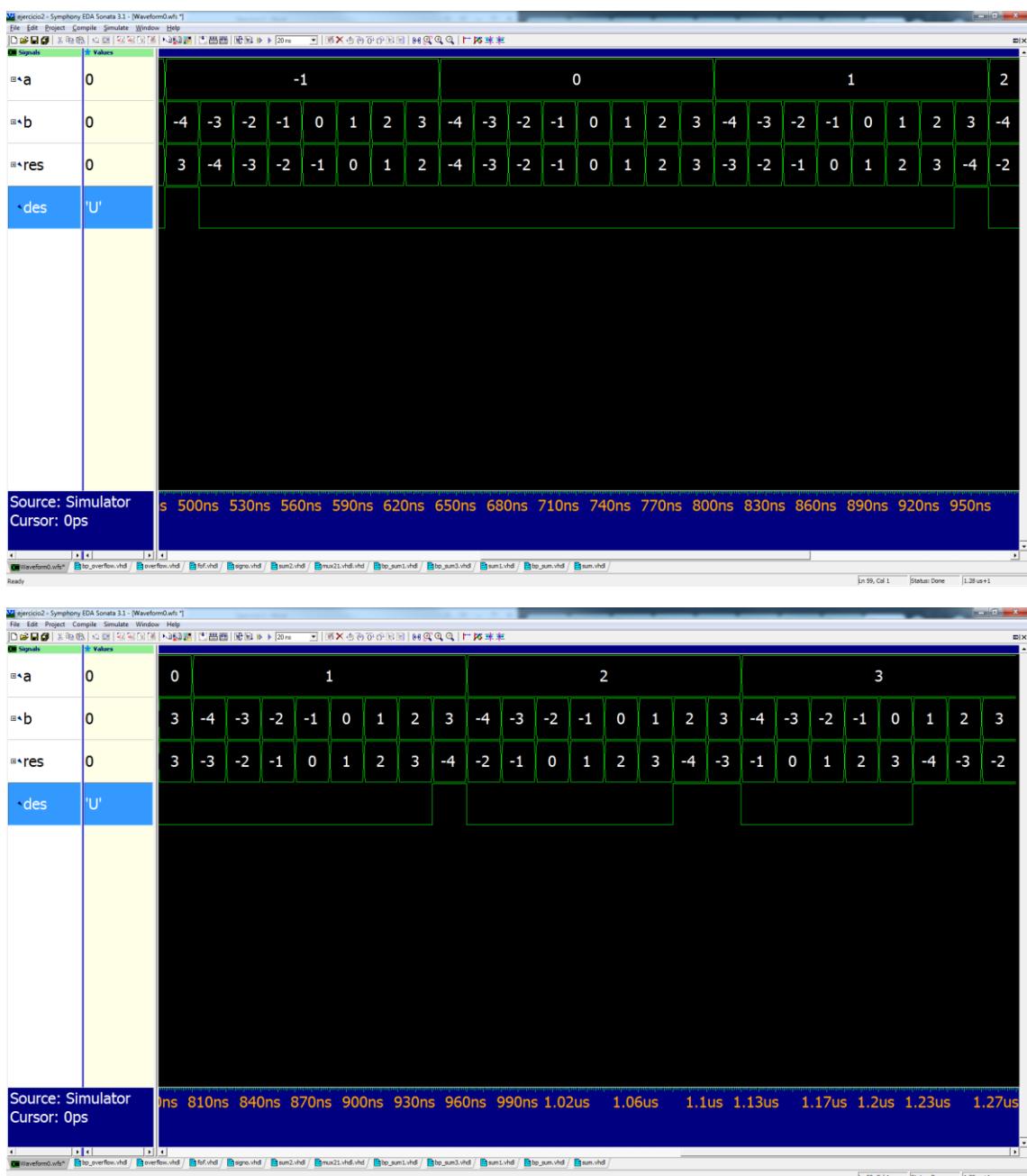
### *Banco de pruebas del componente overflow del ejercicio 2*

Resultados esperados para n = 3.

		c_in = 0		c_in = 1				c_in = 0		c_in = 1				c_in = 0		c_in = 1				c_in = 0		c_in = 1		
-4 : 100		-4 : 100	0 : 000	1 : 001		-4 : 100		1 : 001	2 : 010		-4 : 100		2 : 010	3 : 011		-4 : 100		3 : 011		-4 : 100		3 : 011	-4 : 100	
-3 : 101		-3 : 101	1 : 001	2 : 010		-3 : 101		2 : 010	3 : 011		-3 : 101		3 : 011	-4 : 100		-3 : 101		-4 : 100		-3 : 101		-4 : 100		
-2 : 110		-2 : 110	2 : 010	3 : 011		-2 : 110		3 : 011	-4 : 100		-2 : 110		-4 : 100	-3 : 101		-2 : 110		-3 : 101		-2 : 110		-3 : 101		
-1 : 111		-1 : 111	3 : 011	-4 : 100		-1 : 111		-4 : 100	-3 : 101		-1 : 111		-3 : 101	-2 : 110		-1 : 111		-2 : 110		-1 : 111		-2 : 110		
0 : 000		0 : 000	-4 : 100	-3 : 101		0 : 000		-3 : 101	-2 : 110		0 : 000		-2 : 110	-1 : 111		0 : 000		-1 : 111		0 : 000		-1 : 111		
1 : 001		1 : 001	-3 : 101	-2 : 110		1 : 001		-2 : 110	-1 : 111		1 : 001		-1 : 111	0 : 000		1 : 001		-1 : 111		1 : 001		0 : 000		
2 : 010		2 : 010	-2 : 110	-1 : 111		2 : 010		-1 : 111	0 : 000		2 : 010		0 : 000	1 : 111		2 : 010		1 : 111		2 : 010		2 : 010		
3 : 011		3 : 011	-1 : 111	0 : 000		3 : 011		0 : 000	1 : 111		3 : 011		1 : 111	2 : 010		3 : 011		2 : 010		3 : 011		3 : 011		
			c_in = 0	c_in = 1				c_in = 0	c_in = 1				c_in = 0	c_in = 1				c_in = 0	c_in = 1				c_in = 0	c_in = 1
0:000		-4 : 100	-3 : 101	-2 : 110		-3 : 101		-2 : 110	-1 : 111		-3 : 101		-2 : 110	-1 : 111		-3 : 101		-2 : 110	-1 : 111		-3 : 101		0 : 000	
1:001		1 : 111	2 : 010	3 : 011		1 : 001		2 : 010	3 : 011		1 : 001		2 : 010	-4 : 100		1 : 001		3 : 011	-4 : 100		1 : 001		-4 : 100	
2:010		2 : 010	3 : 011		2 : 010		3 : 011	-4 : 100		2 : 010		-4 : 100	-3 : 101		2 : 010		-3 : 101	-2 : 110		2 : 010		-3 : 101		
3:011		3 : 011	-4 : 100		3 : 011		-4 : 100		3 : 011		-3 : 101	-2 : 110		3 : 011		-2 : 110	-1 : 111		3 : 011		-2 : 110			

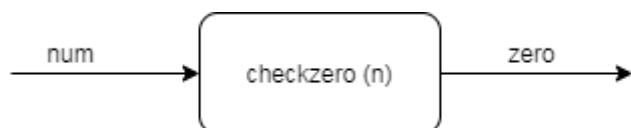
Resultados obtenidos para n = 3.





## Checkzero

Toma un número de n bits en complemento a 2 y comprueba si es igual a 0.



```

1  --
2  -- checkzero. Componente del ejercicio 2
3  -- Entrada: num -> numero binario de n bits.
4  -- Salida: zero = 1 -> num = 0.
5  -- Salida: zero = 0 -> numero de entrada distinto de cero
6  -----
7  library ieee;
8  use ieee.std_logic_1164.all;
9  use ieee.numeric_std.all;
10
11 entity checkzero is
12     generic (n: integer);
13     port(zero : out std_logic; num : in std_logic_vector(n-1 downto 0));
14 end entity checkzero;
15
16 architecture archcheckzero of checkzero is
17
18 begin
19     zero <= '1' when to_integer(unsigned(num))=0 else '0';
20 end architecture archcheckzero;

```

### Banco de pruebas.

El banco de pruebas de este componente figura en el Anexo: *Banco de Pruebas* bajo la referencia:

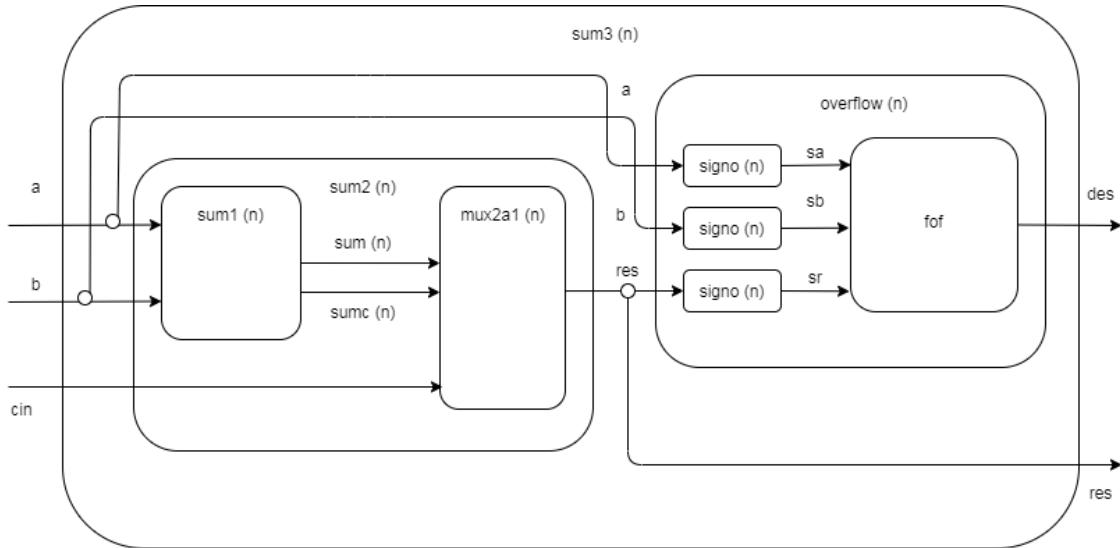
*Banco de pruebas del componente check0 del ejercicio 2*

### Cronograma.

Signals	Values
num	0
zero	'U'

### **Sumador 3**

Toma dos números de n bits en complemento a 2 y un bit de acarreo y devuelve el resultado así como un bit que señala si se ha producido desbordamiento.



```

1  -- sum3. Componente del ejercicio2.
2  -- ENTRADA: (a, b) dos n鷖eros de n bits en C2.
3  -- ENTRADA: cin. bit de acarreo.
4  -- SALIDA: res. resultado a+b
5  -- SALIDA: des = 1 -> hay desbordamiento
6  -- SALIDA: des = 0 -> no hay desbordamiento
7  -----
8
9  library ieee;
10 use ieee.std_logic_1164.all;
11 use ieee.numeric_std.all;
12
13 entity sum3 is
14     generic (n : integer);
15     port (res : out std_logic_vector (n-1 downto 0);
16           des : out std_logic;
17           a, b : in std_logic_vector(n-1 downto 0);
18           cin : in std_logic);
19 end entity sum3;
20
21 architecture archsum3 of sum3 is
22     --Declaracion de constantes
23     constant WIDTH : integer := n;
24
25     --Declaracion de las señales
26     signal a1 : std_logic_vector(n-1 downto 0) := (others => '0');
27
28 begin
29     -- Instaciacion y conexion de los componentes
30     s1 : entity work.sum2(archsum2) generic map (WIDTH) port map (a1, a, b, cin);
31     o1 : entity work.overflow(archoverflow) generic map (WIDTH) port map (des, a, b, a1);
32     res <= a1;
33 end architecture archsum3;

```

### Banco de pruebas.

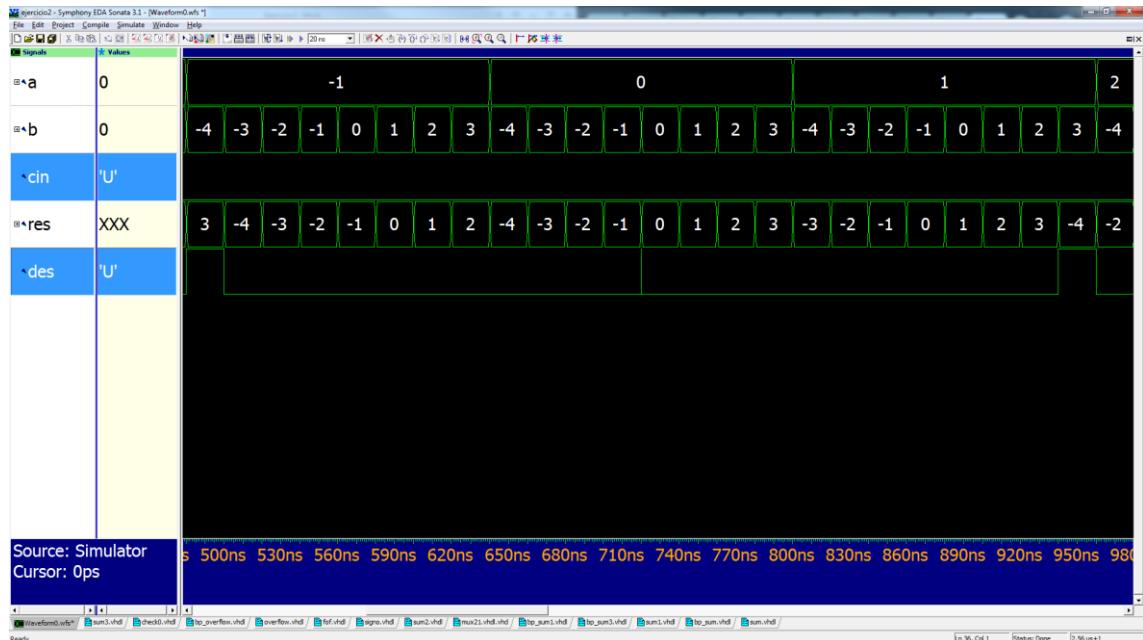
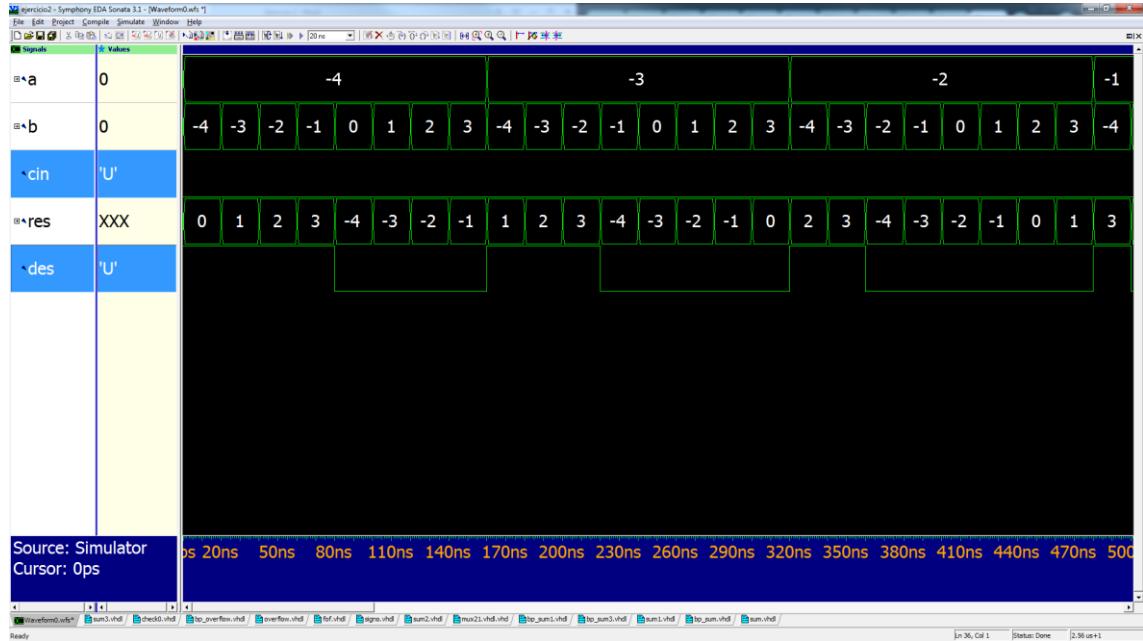
El banco de pruebas de este componente figura en el Anexo: *Banco de Pruebas* bajo la referencia:

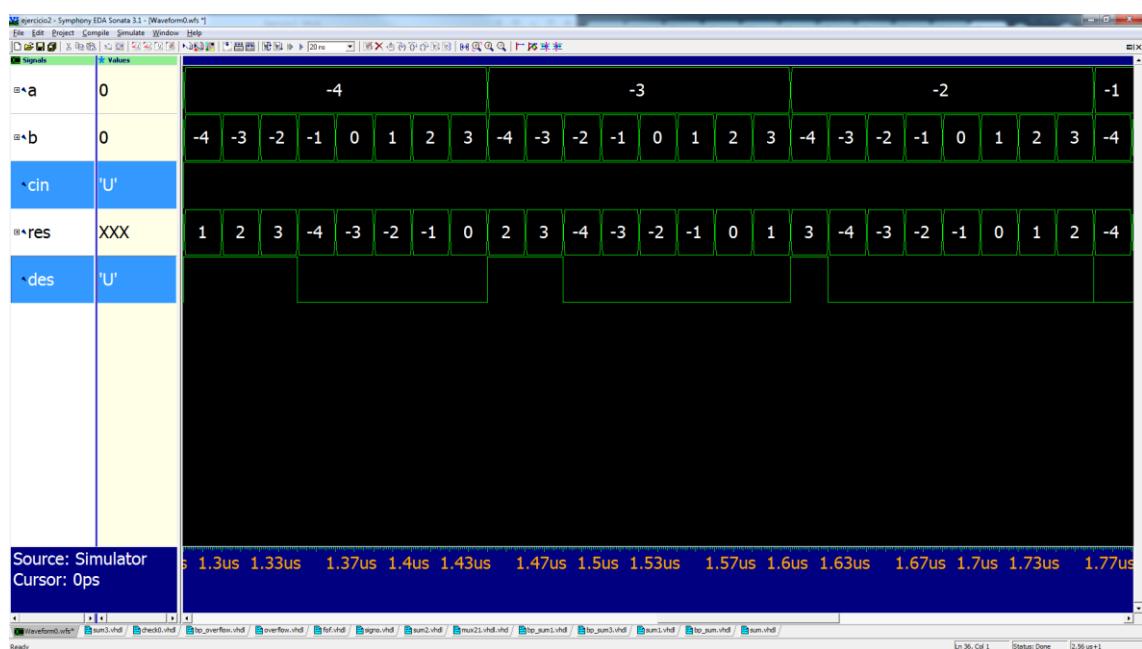
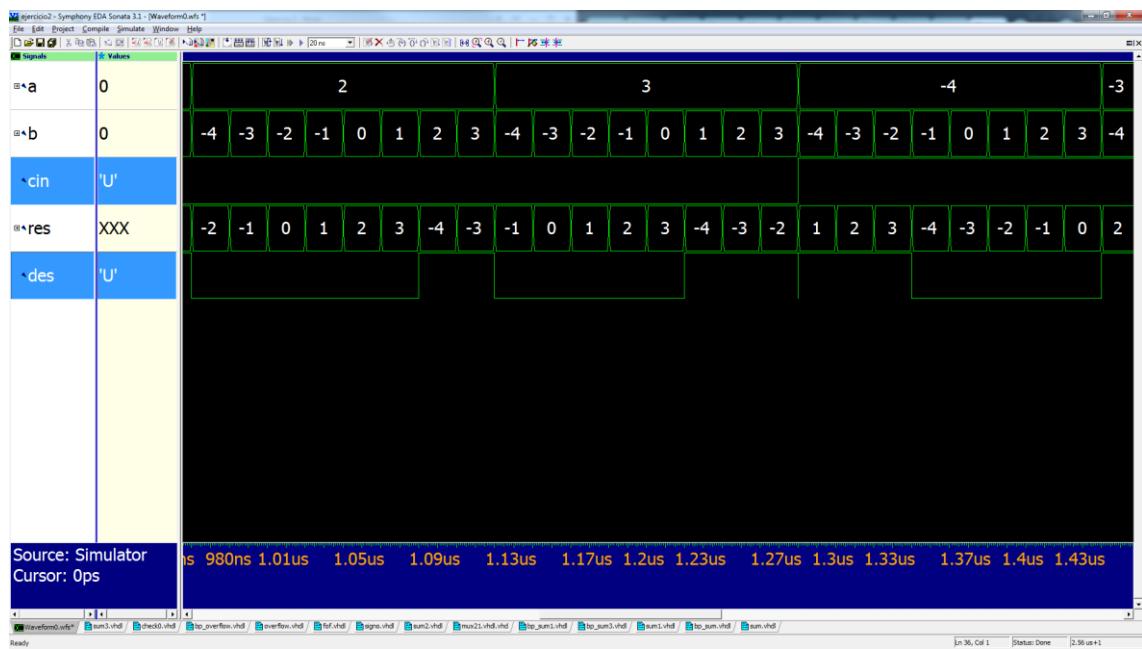
*Banco de pruebas del componente sum3 del ejercicio 2*

Resultados esperados para n = 3.

	c_in = 0	c_in = 1								
-4 : 100	-4 : 100	0 : 000	1 : 001	-4 : 100	1 : 001	2 : 010	-4 : 100	2 : 010	3 : 011	-4 : 100
	-3 : 101	1 : 001	2 : 010	-3 : 101	2 : 010	3 : 011	-3 : 101	3 : 011	-4 : 100	-3 : 101
	-2 : 110	2 : 010	3 : 011	-2 : 110	3 : 011	-4 : 100	-2 : 110	-4 : 100	-3 : 101	-2 : 110
	-1 : 111	3 : 011	-4 : 100	-1 : 111	4 : 100	-3 : 101	-1 : 111	-3 : 101	-2 : 110	-1 : 111
	0 : 000	-4 : 100	-3 : 101	0 : 000	-3 : 101	-2 : 110	0 : 000	-2 : 110	-1 : 111	0 : 000
	1 : 001	-3 : 101	-2 : 110	1 : 001	-2 : 110	-1 : 111	1 : 001	-1 : 111	0 : 000	1 : 001
	2 : 010	-2 : 110	-1 : 111	2 : 010	-1 : 111	0 : 000	2 : 010	0 : 000	1 : 111	2 : 010
	3 : 011	-1 : 111	0 : 000	3 : 011	0 : 000	1 : 111	3 : 011	1 : 111	2 : 010	3 : 011
	c_in = 0	c_in = 1								
	-4 : 100	-4 : 100	-3 : 101	-4 : 100	-3 : 101	-2 : 110	-4 : 100	-2 : 110	-1 : 111	-4 : 100
0 : 000	-3 : 101	-3 : 101	-2 : 110	-3 : 101	-2 : 110	-1 : 111	-3 : 101	-1 : 111	0 : 000	-3 : 101
	-2 : 110	-2 : 110	-1 : 111	-2 : 110	-1 : 111	0 : 000	-2 : 110	0 : 000	1 : 111	-2 : 110
	-1 : 111	-1 : 111	0 : 000	-1 : 111	0 : 000	1 : 111	-1 : 111	1 : 111	2 : 010	-1 : 111
	0 : 000	0 : 000	1 : 111	0 : 000	1 : 111	2 : 010	0 : 000	2 : 010	3 : 011	0 : 000
	1 : 001	1 : 111	2 : 010	1 : 001	2 : 010	3 : 011	1 : 001	3 : 011	-4 : 100	1 : 001
	2 : 010	2 : 010	3 : 011	2 : 010	3 : 011	-4 : 100	2 : 010	-4 : 100	-3 : 101	2 : 010
	3 : 011	3 : 011	-4 : 100	3 : 011	-4 : 100	-3 : 101	3 : 011	-3 : 101	-2 : 110	3 : 011
	c_in = 0	c_in = 1								
	-4 : 100	-4 : 100	-3 : 101	-4 : 100	-3 : 101	-2 : 110	-4 : 100	-2 : 110	-1 : 111	-4 : 100
	-3 : 101	-3 : 101	-2 : 110	-3 : 101	-2 : 110	-1 : 111	-3 : 101	-1 : 111	0 : 000	-3 : 101

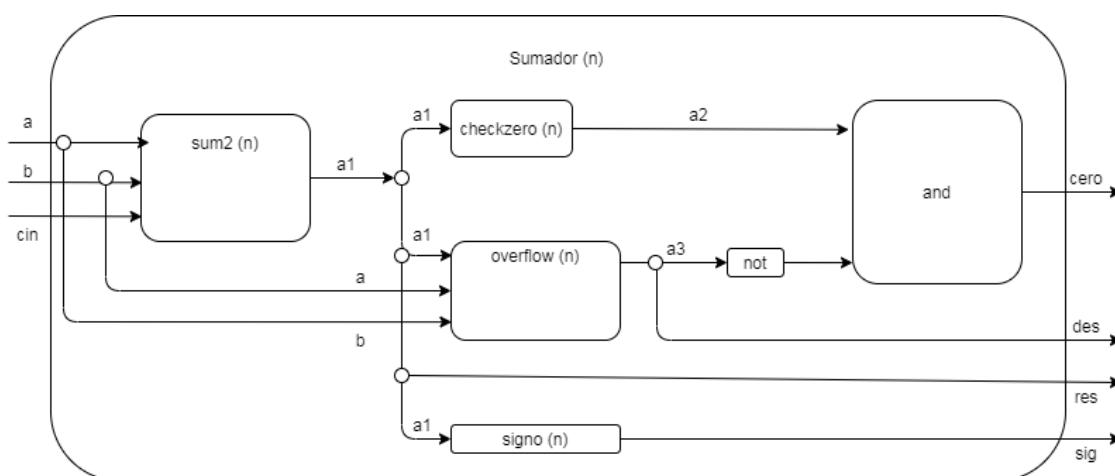
## Resultados obtenidos para n = 3.







Finalmente se llega el sumador que se da como solución.



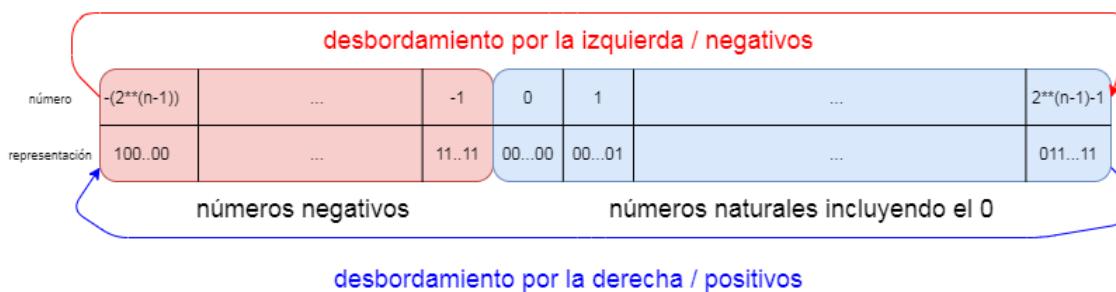
2.b) Programe en VHDL un banco de pruebas que testeé todas las posibles entradas al circuito diseñado en el Apartado 2.a.

1. El número de bits de los operandos de entrada a de ser una constante del programa cuyo valor se pueda modificar de modo que el banco de pruebas sea válido para cualquier número de bits de los operandos de entrada.
2. El banco de pruebas debe comparar las salidas UUT con las salidas esperadas, mostrando el correspondiente mensaje de error en el caso de que las salidas obtenidas de la UUT no se correspondan con las esperadas.
3. El banco de pruebas debe mostrar al final del test un mensaje con el número total de errores detectados.
4. La forma de calcular el valor esperado de la señal de desbordamiento debe ser diferente a la realizada en el circuito que se pretende testear.

#### **Generación de señales de entrada.**

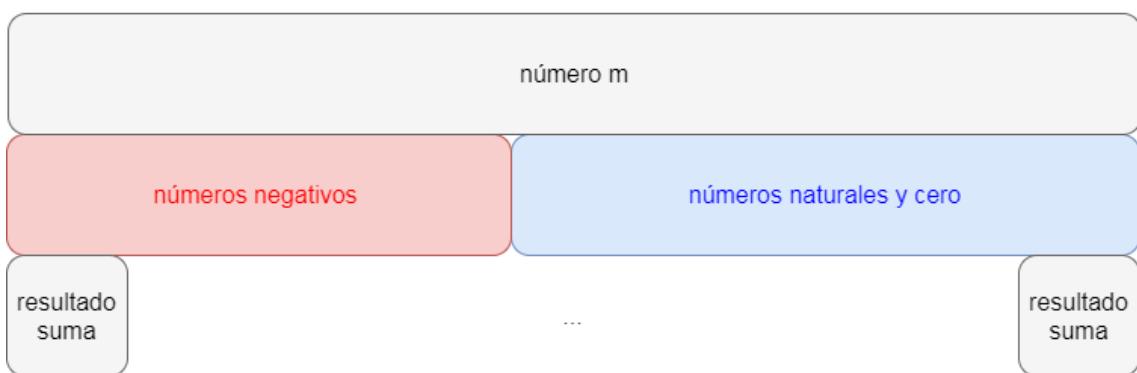
La representación en complemento a 2 es de la forma:

#### **Representación en Complemento a 2 para n bits**

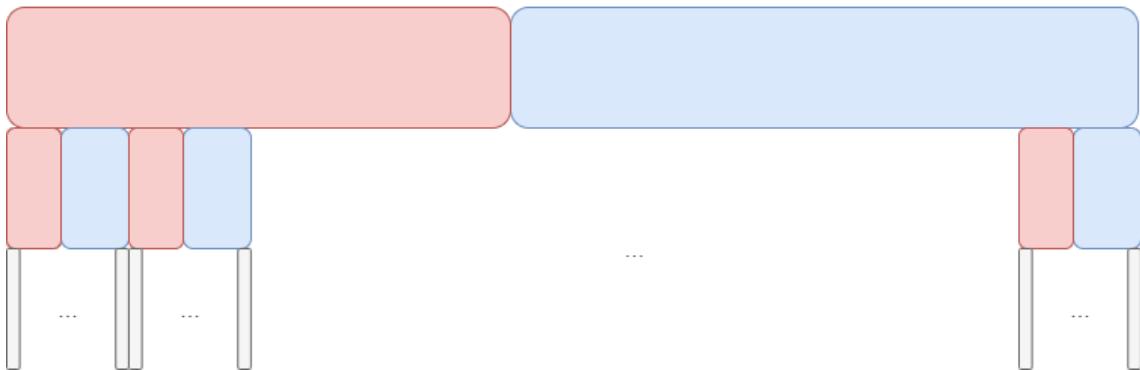


En el banco de pruebas se testeán todas las posibles entradas al componente, en este caso todas las sumas entre todos los números representados por n bits en complemento a 2, con todos los números representados por n bits en complemento a 2.

Número m sumado con todas las posibles representaciones en complemento a 2 de n bits

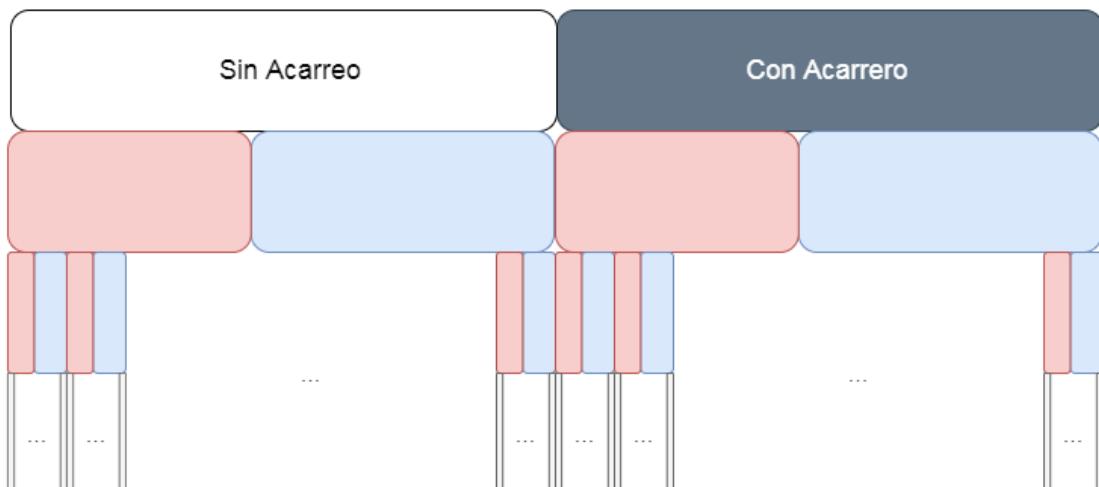


Todas las posibles representaciones en complemento a 2 de n bits sumadas con todas las posibles representaciones en complemento a 2 de n bits



Al tener el sumador 3 entradas, se deben hacer las sumas considerando la presencia de acarreo a la entrada y sin considerarlo.

Todas las posibles representaciones en complemento a 2 de n bits sumadas con todas las posibles representaciones en complemento a 2 de n bits considerando acarreo a la entrada.



### Implementación.

Para generar todos los números de n bit en complemento a 2, se crean dos bucles:

- El primer bucle genera los números negativos.
- El segundo bucle genera los números naturales y el cero.

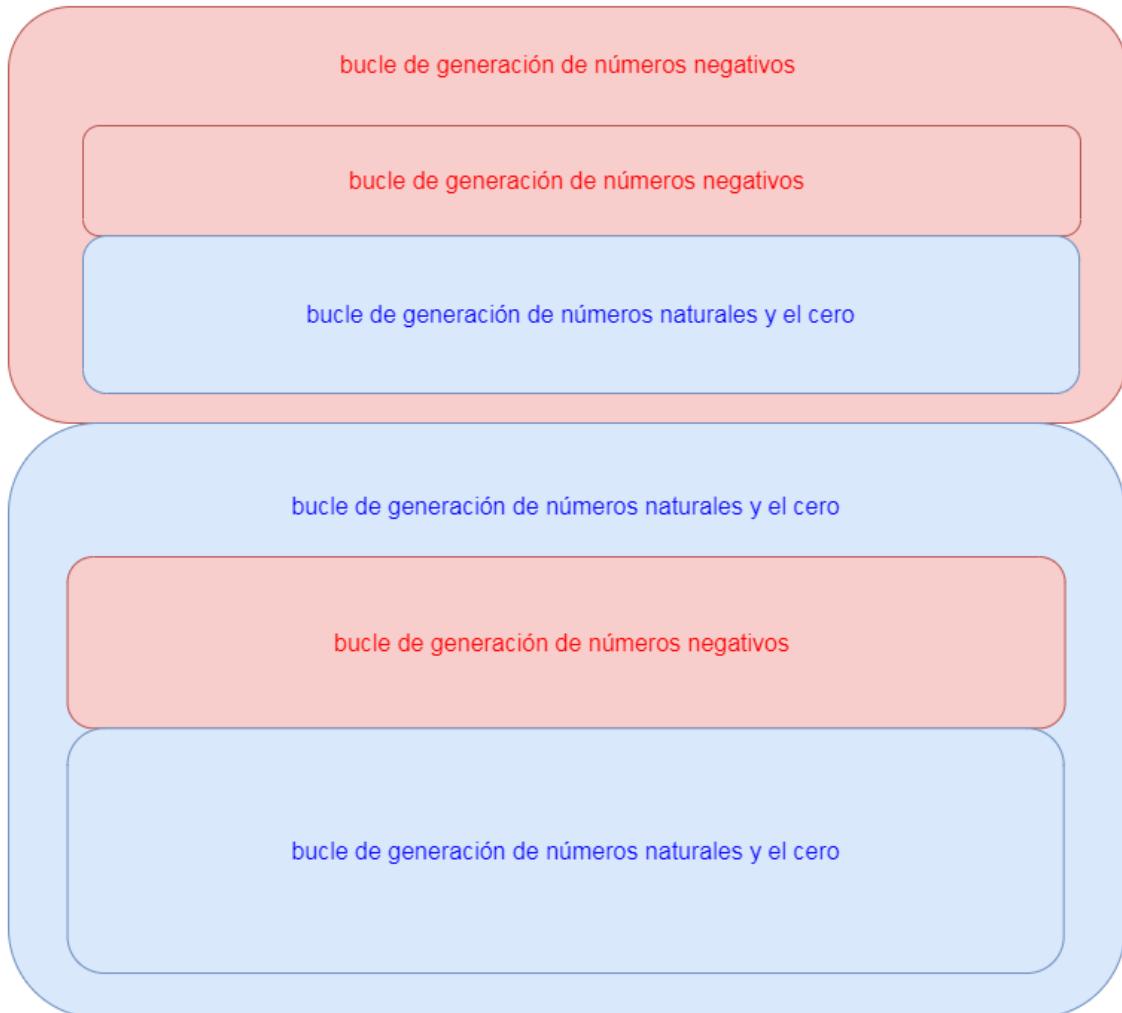
## Generación de los números representados por n bits en complemento a 2

bucle de generación de números negativos

bucle de generación de números naturales y el cero

Para hacer todas las posibles sumas con los números representados por n bits en complemento a 2, se deben anidar los bucles de forma que todos los números se vuelvan a generar para cada uno de los números generados.

## Generación de las sumas por los números representados por n bits en complemento a 2



Por último, para considerar los valores de la entrada de acarreo, se utiliza un tercer bucle en el que se anidan los anteriores resultando toda la estructura con 3 niveles de bucles anidados.

## Bucle generación Acarreo

bucle de generación de números negativos

bucle de generación de números negativos

bucle de generación de números naturales y el cero

bucle de generación de números naturales y el cero

bucle de generación de números negativos

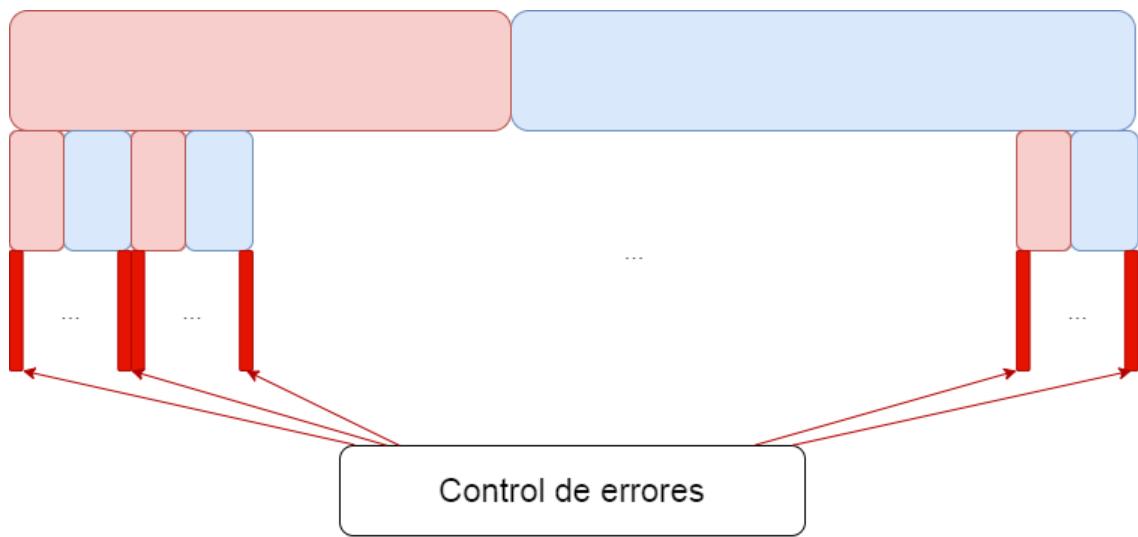
bucle de generación de números naturales y el cero

### Comprobación de resultados.

Para comprobar las señales de salida del componente, se siguen 2 pasos:

1. Se genera el resultado que debería dar el componente con las señales de entrada que se le proporcionaron.
2. Se comprueban las señales de salida.
  - a. Cada vez que se comprueba una señal de salida obtenida con la esperada, en caso de producirse un error, se registra en un contador de errores.

La comprobación de resultados se hará para cada suma.



Implementación

## Bloque de control de errores

### Generación

Generación Resultado Esperado

Condición de Signo Esperada

Condición de Cero Esperada

Generación Condición Desbordamiento

### Chequeo

Chequeo Errores Suma

Contador Errores

Chequeo Errores  
Desbordamiento

Contador Errores

Chequeo Cero

Contador Errores

Chequeo Signo

Contador Errores

## Bucle generación Acarreo



La forma de calcular el valor esperado de la señal de desbordamiento debe ser diferente a la realizada en el circuito que se pretende testear.

Para el cálculo de la condición de desbordamiento en el banco de pruebas, se toman los dos números naturales ( $n_a, n_b$ ) que representan las señales de entrada al sumador (a, b), el acarreo de entrada  $c_{in}$  y se calcula su suma:  $S = n_a + n_b + c_{in}$ .

Siendo  $n$  el número de bits de las señales de entrada,

- si  $S \in [-2^{n-1}, 2^{n-1}-1]$ , no se produce condición de desbordamiento.
- Si  $S \notin [-2^{n-1}, 2^{n-1}-1]$ , se produce condición de desbordamiento.

Código fuente.

El código fuente del banco de pruebas se encuentra en el anexo *Anexo: Banco de pruebas* bajo la referencia.

```

1  ----- Banco de pruebas del ejercicio 2 -----
2
3
4  library IEEE;
5  use IEEE.std_logic_1164.all;
6  use IEEE.numeric_std.all;
7
8  entity bp_sum is
9    constant DELAY : time := 20 ns; -- Retardo usado en el test
10 end entity bp_sum;
11
12 architecture bp_sum of bp_sum is
13   constant WIDTH : integer := 4;
14   signal a, b : std_logic_vector(WIDTH-1 downto 0) := (others => '0');
15   signal res : std_logic_vector(WIDTH-1 downto 0);
16   signal cin, des, zero, sig : std_logic;
17
18 procedure gen_ref (total, WIDTH : in integer;
19                     ref_des, ref_zero, ref_signo : out std_logic) is
20
21 begin
22   ----- Genera el bit referencia para la comparacion de acarreo -----
23   if total>2**WIDTH-1 or total<-1*2**WIDTH then
24     ref_des := '1';
25   else
26     ref_des := '0';
27   end if;
28   ----- Genera el bit referencia para la comparacion de cero-----
29   if total = 0 then
30     ref_zero := '1';
31   else
32     ref_zero := '0';
33   end if;
34   ----- Genera el bit referencia para la comparacion de signo -----
35   if total < 0 then
36     ref_signo := '1';
37   else
38     ref_signo := '0';
39   end if;
40 end procedure gen_ref;
41
42 procedure check_error(zero, ref_zero, sig, ref_signo, ref_des : in std_logic;
43                      num_errores : inout integer) is
44
45 begin
46   -----Chequeo de cero -----
47   assert zero = ref_zero report "ERROR EN EL CERO. Obtenido/Esperado: "
48   & std_logic'image(zero) & std_logic'image(ref_zero);
49   -- contador de errores
50   if zero /= ref_zero then
51     num_errores := num_errores + 1;
52   end if;
53   -----Chequeo de signo -----
54   if ref_des = '0' then
55     assert sig = ref_signo report "ERROR EN EL SIGNO. Obtenido/Esperado: "
56     & std_logic'image(sig) & std_logic'image(ref_signo);
57     -- contador de errores
58     if sig /= ref_signo then
59       num_errores := num_errores + 1;
60     end if;
61   end if;
62 end procedure check_error;
63
63 begin
64   UUT : entity work.sum(archsum) generic map (WIDTH) port map (res, des, zero, sig, a, b, cin);
65
66   vect_test : process is
67     variable valorA : signed (WIDTH-1 downto 0);
68     variable valorB : signed (WIDTH-1 downto 0);
69     variable total : integer;
70     variable vec_total : std_logic_vector(2*WIDTH-1 downto 0);
71     variable ref_des : std_logic;
72     variable ref_zero : std_logic;
73     variable ref_signo : std_logic;
74     variable num_errores : integer := 0;
75
76   begin
77     for k in 0 to 1 loop
78       -- Genera los resultados sin acarreo
79       if k=0 then
80         cin <= '0';

```

```

80      -- Genera todos los posibles valores de entrada
81      -- Genera los negativos y el cero del primer sumando
82      for i in 0 to 2**(WIDTH-1)-1 loop
83          valorA := to_signed((-1**2** (WIDTH-1))+i, WIDTH);
84          a <= std_logic_vector(valorA);
85          -- Genera los negativos y el cero del segundo sumando
86          for j in 0 to 2** (WIDTH-1)-1 loop
87              valorB := to_signed((-1**2** (WIDTH-1))+j, WIDTH);
88              b <= std_logic_vector(valorB);
89              -- GENERACION DE REFERENCIAS
90              -- Genera el resultado referencia para chequeo de errores
91              total := -1*2** (WIDTH)+i+j;
92              vec_total := std_logic_vector(to_signed(total, 2*WIDTH));
93              wait for DELAY;
94              -- Genera el resto de referencias
95              gen_ref(total, WIDTH, ref_des, ref_zero, ref_signo);
96              -- CHEQUEO
97              -----Bucle para chequear el resultado de la suma
98              for m in 0 to WIDTH-1 loop
99                  assert res(m)=vec_total(m) report "ERROR EN LA SUMA. Obtenido/Esperado: "
100                     & std_logic'image(res(m)) & std_logic'image(vec_total(m));
101                  -- contador de errores
102                  if res(m) /= vec_total(m) then
103                      num_errores := num_errores + 1;
104                  end if;
105              end loop;
106              ----- Chequeo del desbordamiento -----
107              assert des = ref_des report "ERROR EN EL DESBORDAMIENTO. Obtenido/Esperado: "
108                 & std_logic'image(des) & std_logic'image(ref_des);
109              -- contador de errores
110              if des /= ref_des then
111                  num_errores := num_errores + 1;
112              end if;
113              ----- Chequeo el resto de senales de salida y contador errores -----
114              check_error(zero, ref_zero, sig, ref_signo, ref_des, num_errores);
115          end loop;
116          -----Final de negativos y cero del segundo sumando-----

117          --Genera los positivos del segundo sumando-----
118          for j in 0 to 2** (WIDTH-1)-1 loop
119              valorB := to_signed(j, WIDTH);
120              b <= std_logic_vector(valorB);
121              -- GENERACION DE REFERENCIAS
122              -- Genera el resultado referencia para el chequeo de errores
123              total := -1*2** (WIDTH-1)+i+j;
124              vec_total := std_logic_vector(to_signed(total, 2*WIDTH));
125              wait for DELAY;
126              ----- Genera el resto de referencias
127              gen_ref(total, WIDTH, ref_des, ref_zero, ref_signo);
128              -- CHEQUEO
129              -----Bucle para chequear el resultado de la suma
130              for m in 0 to WIDTH-1 loop
131                  assert res(m)=vec_total(m) report "ERROR EN LA SUMA. Obtenido/Esperado: "
132                     & std_logic'image(res(m)) & std_logic'image(vec_total(m));
133                  -- contador errores
134                  if res(m) /= vec_total(m) then
135                      num_errores := num_errores + 1;
136                  end if;
137              end loop;
138              ----- Chequeo del desbordamiento -----
139              assert des = ref_des report "ERROR EN EL DESBORDAMIENTO. Obtenido/Esperado: "
140                 & std_logic'image(des) & std_logic'image(ref_des);
141              -- contador errores
142              if des /= ref_des then
143                  num_errores := num_errores + 1;
144              end if;
145              ----- Chequeo del resto de senales de salida y contador de errores -----
146              check_error(zero, ref_zero, sig, ref_signo, ref_des, num_errores);
147          end loop;
148          -----FINAL DE LOS NEGATIVOS PRIMER SUMANDO-----

150          -- Genera los positivos del primer sumando-----
151          for i in 0 to 2** (WIDTH-1)-1 loop
152              valorA := to_signed(i, WIDTH);
153              a <= std_logic_vector(valorA);
154              -- GENERACION DE REFERENCIAS
155              -- Genera los negativos y cero del segundo sumando-----
156              for j in 0 to 2** (WIDTH-1)-1 loop
157                  valorB := to_signed((-1*2** (WIDTH-1))+j, WIDTH);
158                  b <= std_logic_vector(valorB);
159                  -- GENERACION DE REFERENCIAS
160                  -- Genera el resultado referencia para chequeo de errores
161                  total := -1*2** (WIDTH-1)+i+j;
162                  vec_total := std_logic_vector(to_signed(total, 2*WIDTH));
163                  wait for DELAY;
164                  ----- Genera el resto de referencias -----
165                  gen_ref(total, WIDTH, ref_des, ref_zero, ref_signo);
166                  -- CHEQUEO
167                  -----Bucle para chequear el resultado de la suma
168                  for m in 0 to WIDTH-1 loop
169                      assert res(m)=vec_total(m) report "ERROR EN LA SUMA. Obtenido/Esperado: "
170                         & std_logic'image(res(m)) & std_logic'image(vec_total(m));
171                      -- contador errores
172                      if res(m) /= vec_total(m) then
173                          num_errores := num_errores + 1;
174                      end if;
175                  end loop;
176                  ----- Chequeo del desbordamiento -----
177                  assert des = ref_des report "ERROR EN EL DESBORDAMIENTO. Obtenido/Esperado: "
178                     & std_logic'image(des) & std_logic'image(ref_des);
179                  -- contador errores
180                  if des /= ref_des then
181                      num_errores := num_errores + 1;
182                  end if;
183                  ----- Chequeo el resto de senales de salida y contador errores
184                  check_error(zero, ref_zero, sig, ref_signo, ref_des, num_errores);
end loop;
```

```

185      -----genera los positivos del segundo sumando-----
186      for j in 0 to 2**WIDTH-1 loop
187          valorB := to_signed(j, WIDTH);
188          b <= std_logic_vector(valorB);
189          -- GENERACION DE REFERENCIAS
190          -- Genera el resultado referencia para el chequeo de errores
191          total := i+j;
192          vec_total := std_logic_vector(to_signed(total, 2*WIDTH));
193          wait for DELAY;
194          ----- Genera el resto de referencias
195          gen_ref(total, WIDTH, ref_des, ref_zero, ref_signo);
196          -- CHEQUEO
197          ----- Bucle para chequear el resultado de la suma
198          for m in 0 to WIDTH-1 loop
199              assert res(m)=vec_total(m) report "ERROR EN LA SUMA. Obtenido/Esperado: "
200                  & std_logic'image(res(m)) & std_logic'image(vec_total(m));
201              -- contador de errores
202              if res(m) /= vec_total(m) then
203                  num_errores := num_errores + 1;
204              end if;
205          end loop;
206          ----- Chequeo del desbordamiento -----
207          assert des = ref_des report "ERROR EN EL DESBORDAMIENTO. Obtenido/Esperado: "
208              & std_logic'image(des) & std_logic'image(ref_des);
209          -- contador de errores
210          if des /= ref_des then
211              num_errores := num_errores + 1;
212          end if;
213          ----- Chequeo el resto de senales de salida y contador de errores -----
214          check_error(zero, ref_zero, sig, ref_signo, ref_des, num_errores);
215      end loop;
216  end loop;

217      ----- Genera los resultados con acarreo
218      else
219          cin <= '1';
220          -- Genera todos los posibles valores de entrada
221          -- Genera los negativos y el cero del primer sumando
222          for i in 0 to 2**WIDTH-1 loop
223              valorA := to_signed((-1*2**WIDTH)+i, WIDTH);
224              a <= std_logic_vector(valorA);
225              -- Genera los negativos y el cero del segundo sumando
226              for j in 0 to 2**WIDTH-1 loop
227                  valorB := to_signed((-1*2**WIDTH)+j, WIDTH);
228                  b <= std_logic_vector(valorB);
229                  -- GENERACION DE REFERENCIAS
230                  -- Genera el resultado referencia para chequeo de errores
231                  total := -1*2**WIDTH+i+j+1;
232                  vec_total := std_logic_vector(to_signed(total, 2*WIDTH));
233                  wait for DELAY;
234                  ----- Genera el resto de referencias
235                  gen_ref(total, WIDTH, ref_des, ref_zero, ref_signo);
236                  --CHEQUEO
237                  ----- Bucle para chequear el resultado de la suma
238                  for m in 0 to WIDTH-1 loop
239                      assert res(m)=vec_total(m) report "ERROR EN LA SUMA. Obtenido/Esperado: "
240                          & std_logic'image(res(m)) & std_logic'image(vec_total(m));
241                      -- contador de errores
242                      if res(m) /= vec_total(m) then
243                          num_errores := num_errores + 1;
244                      end if;
245                  end loop;
246                  ----- Chequeo del desbordamiento -----
247                  assert des = ref_des report "ERROR EN EL DESBORDAMIENTO. Obtenido/Esperado: "
248                      & std_logic'image(des) & std_logic'image(ref_des);
249                  -- contador de errores
250                  if des /= ref_des then
251                      num_errores := num_errores + 1;
252                  end if;
253                  ----- Chequeo del resto de senales de salida y contador de errores -----
254                  check_error(zero, ref_zero, sig, ref_signo, ref_des, num_errores);
255              end loop;
256          -----Final de negativos y cero del segundo sumando-----

257          -----Genera los positivos del segundo sumando-----
258          for j in 0 to 2**WIDTH-1 loop
259              valorB := to_signed(j, WIDTH);
260              b <= std_logic_vector(valorB);
261              -- GENERACION DE REFERENCIAS
262              -- Genera el resultado referencia para el chequeo de errores
263              total := -1*2**WIDTH+i+j+1;
264              vec_total := std_logic_vector(to_signed(total, 2*WIDTH));
265              wait for DELAY;
266              ----- Genera el resto de referencias
267              gen_ref(total, WIDTH, ref_des, ref_zero, ref_signo);
268              -- CHEQUEO
269              ----- Bucle para chequear el resultado de la suma
270              for m in 0 to WIDTH-1 loop
271                  assert res(m)=vec_total(m) report "ERROR EN LA SUMA. Obtenido/Esperado: "
272                      & std_logic'image(res(m)) & std_logic'image(vec_total(m));
273                  -- contador de errores
274                  if res(m) /= vec_total(m) then
275                      num_errores := num_errores + 1;
276                  end if;
277              end loop;
278              ----- Chequeo del desbordamiento -----
279              assert des = ref_des report "ERROR EN EL DESBORDAMIENTO. Obtenido/Esperado: "
280                  & std_logic'image(des) & std_logic'image(ref_des);
281              -- contador de errores
282              if des /= ref_des then
283                  num_errores := num_errores + 1;
284              end if;
285              ----- Chequeo el resto de senales de salida y contador de errores -----
286              check_error(zero, ref_zero, sig, ref_signo, ref_des, num_errores);
287          end loop;
288      end loop;
289  -----FINAL DE LOS NEGATIVOS PRIMER SUMANDO-----

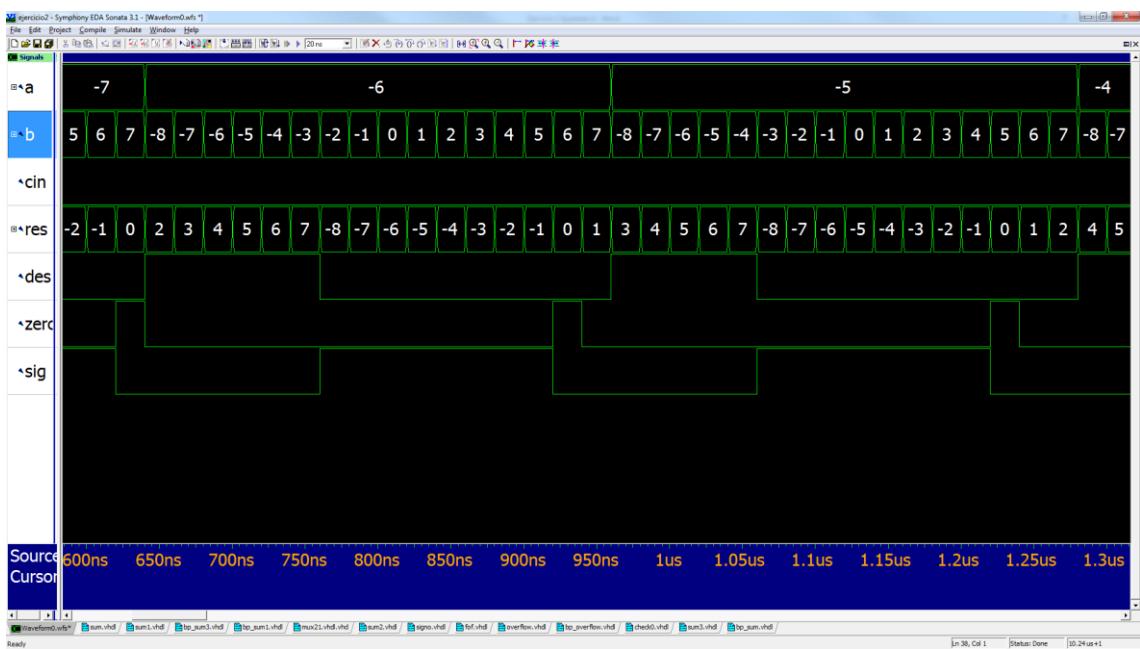
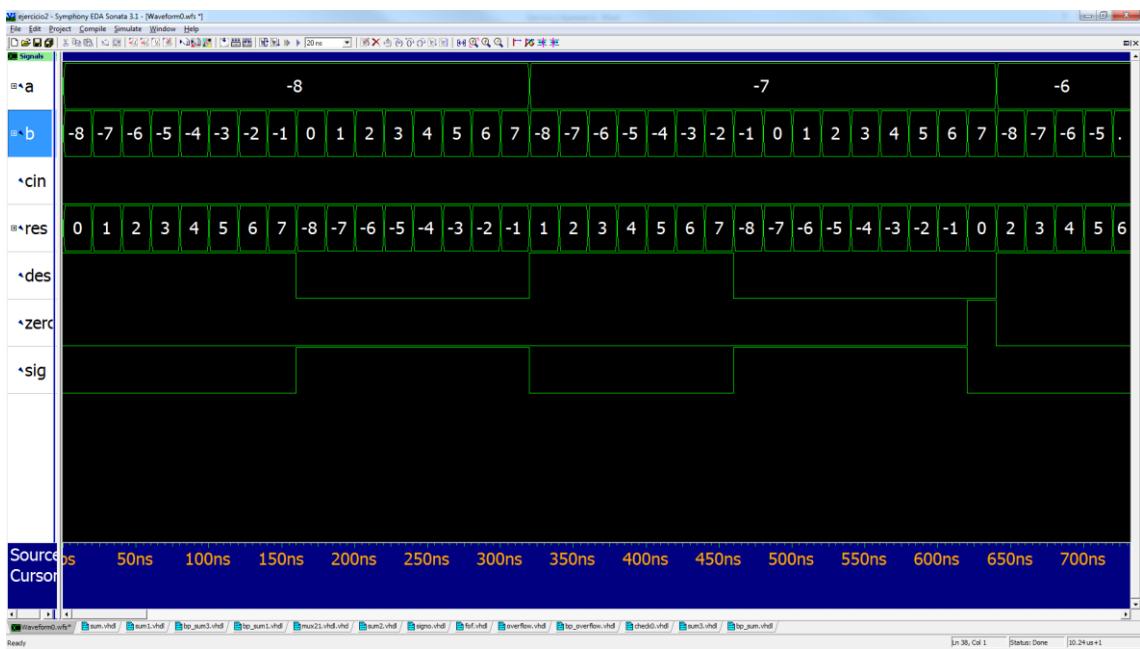
```

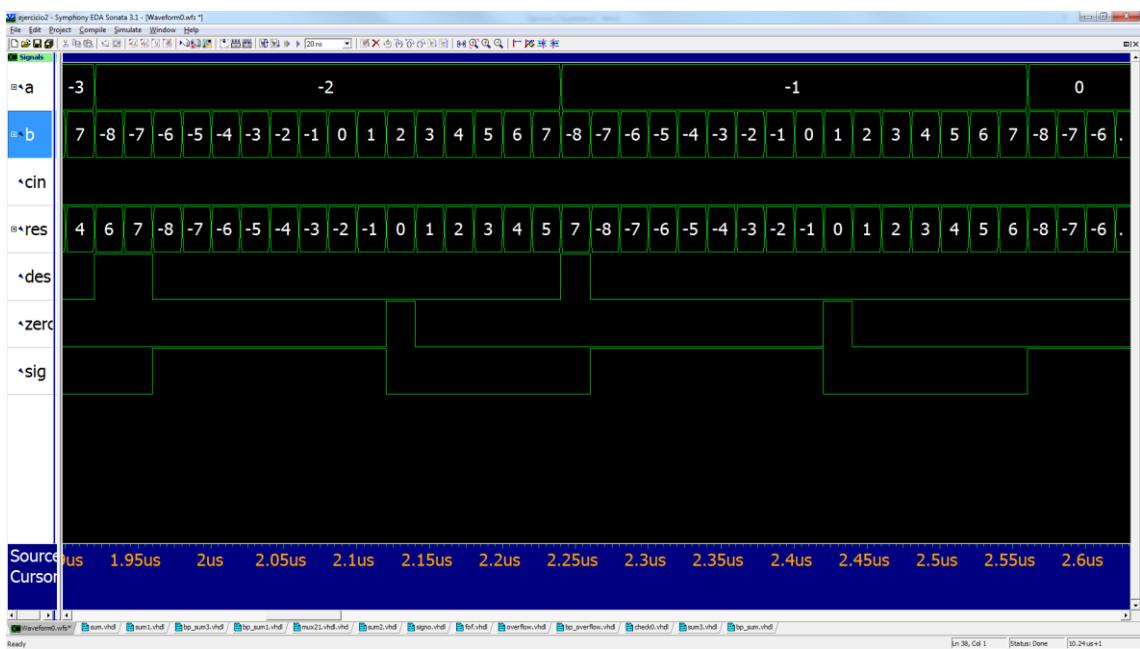
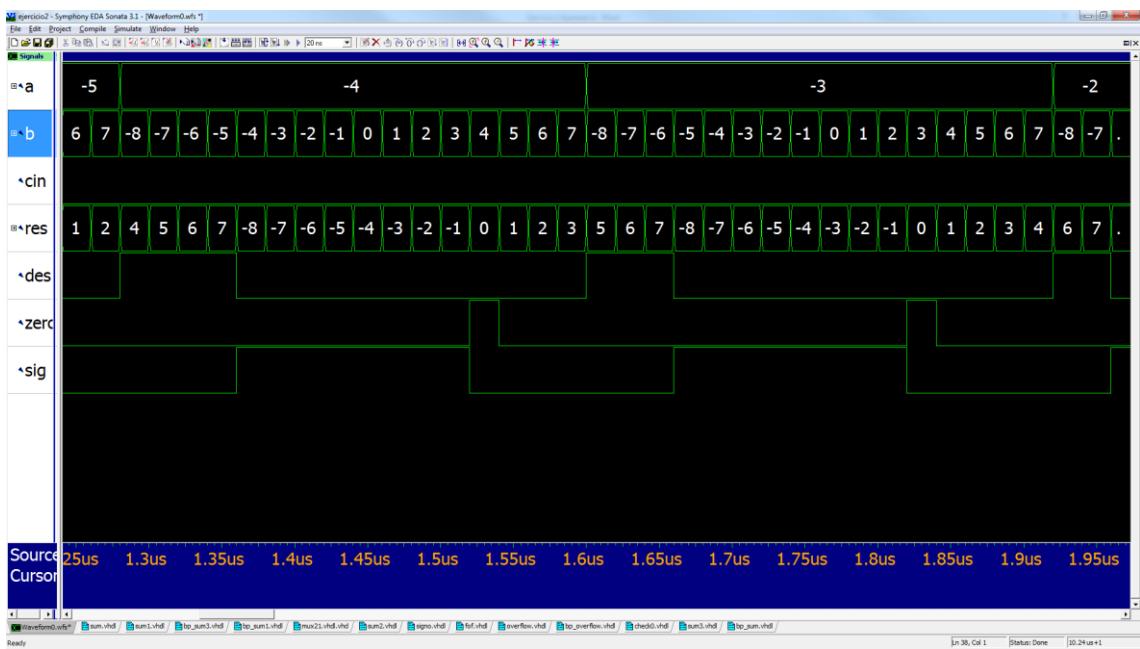
```

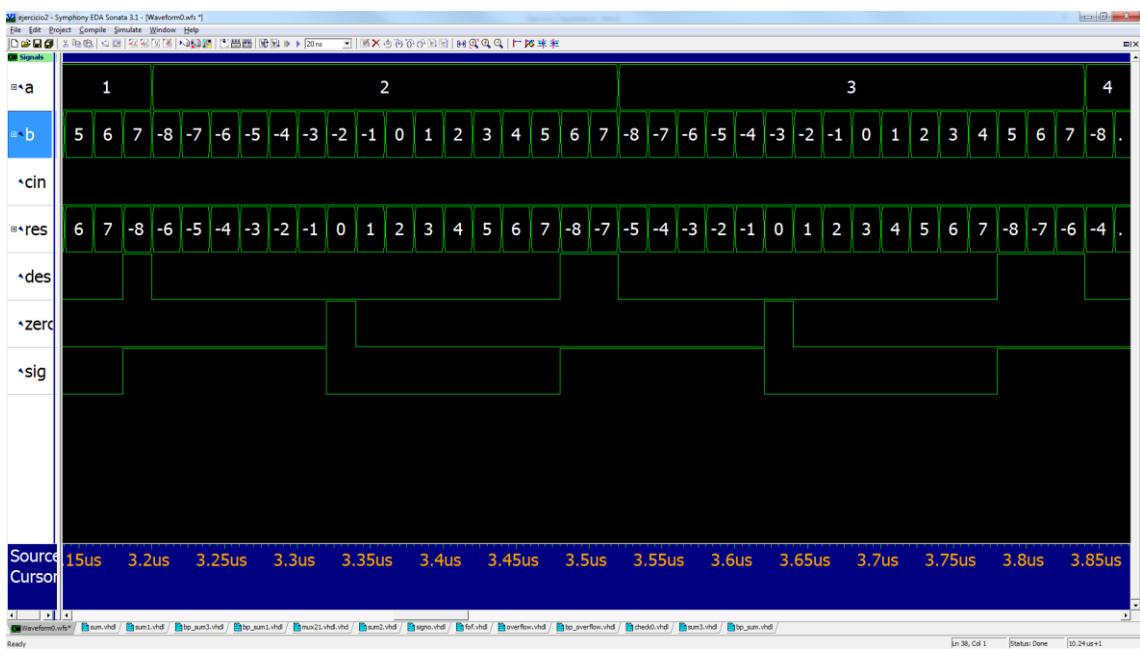
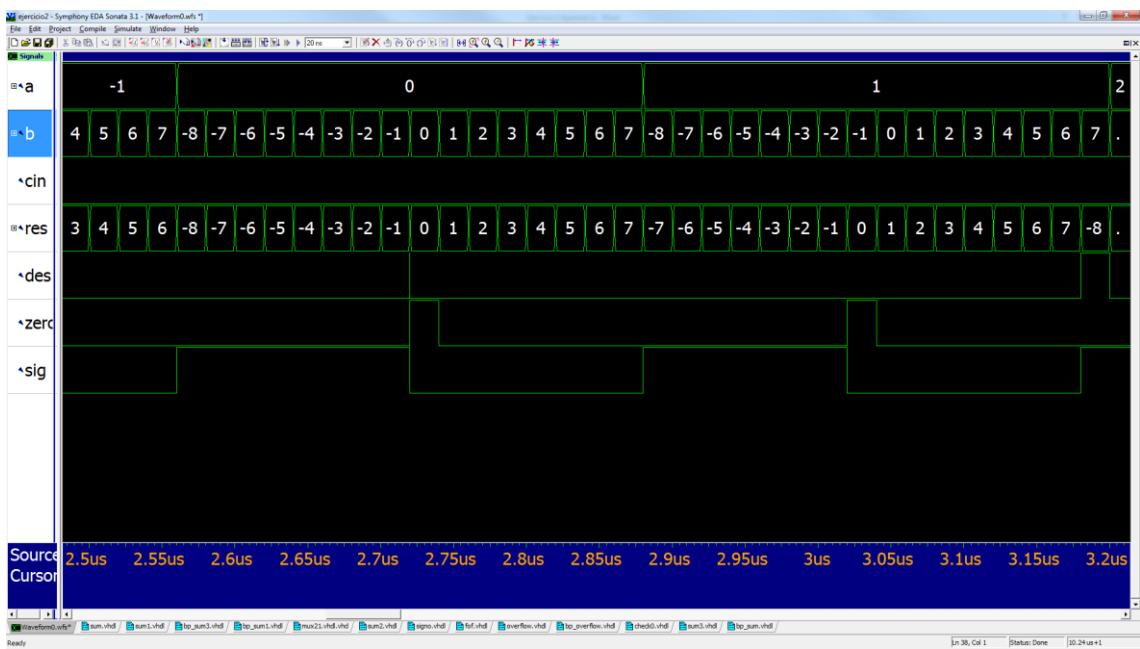
290      -- Genera los positivos del primer sumando-----
291      for i in 0 to 2**WIDTH-1 loop
292          valorA := to_signed(i, WIDTH);
293          a <= std_logic_vector(valorA);
294      end loop;
295      -- Genera los negativos y cero del segundo sumando-----
296      for j in 0 to 2**WIDTH-1 loop
297          valorB := to_signed((-1**2**WIDTH)+j, WIDTH);
298          b <= std_logic_vector(valorB);
299          -- GENERACION DE REFERENCIAS
300          -- Genera el resultado referencia para chequeo de errores
301          total := -1**2**WIDTH+i+j;
302          vec_total := std_logic_vector(to_signed(total, 2*WIDTH));
303          wait for DELAY;
304          -- Genera el resto de referencias
305          gen_ref(total, WIDTH, ref_des, ref_zero, ref_signo);
306          -- CHEQUEO
307          -- Bucle para chequear el resultado de la suma
308          for m in 0 to WIDTH-1 loop
309              assert res(m)=vec_total(m) report "ERROR EN LA SUMA. Obtenido/Esperado: "
310                  & std_logic'image(res(m)) & std_logic'image(vec_total(m));
311              -- contador errores
312              if res(m) /= vec_total(m) then
313                  num_errores := num_errores + 1;
314              end if;
315          end loop;
316          -- Chequeo del desbordamiento -----
317          assert des = ref_des report "ERROR EN EL DESBORDAMIENTO. Obtenido/Esperado: "
318              & std_logic'image(des) & std_logic'image(ref_des);
319          -- contador de errores
320          if des /= ref_des then
321              num_errores := num_errores + 1;
322          end if;
323          -- Chequeo el resto de senales de salida y contador errores -----
324          check_error(zero, ref_zero, sig, ref_signo, ref_des, num_errores);
325      end loop;
326
327      --genera los positivos del segundo sumando-----
328      for j in 0 to 2**WIDTH-1 loop
329          valorB := to_signed(j, WIDTH);
330          b <= std_logic_vector(valorB);
331          -- GENERACION DE REFERENCIAS
332          -- Genera el resultado de referencia para el chequeo de errores
333          total := i+j;
334          vec_total := std_logic_vector(to_signed(total, 2*WIDTH));
335          wait for DELAY;
336          -- Genera el resto de referencias
337          gen_ref(total, WIDTH, ref_des, ref_zero, ref_signo);
338          -- CHEQUEO
339          -- Bucle para chequear el resultado de la suma
340          for m in 0 to WIDTH-1 loop
341              assert res(m)=vec_total(m) report "ERROR EN LA SUMA. Obtenido/Esperado: "
342                  & std_logic'image(res(m)) & std_logic'image(vec_total(m));
343              -- contador de errores
344              if res(m) /= vec_total(m) then
345                  num_errores := num_errores + 1;
346              end if;
347          end loop;
348          -- Chequeo del desbordamiento -----
349          assert des = ref_des report "ERROR EN EL DESBORDAMIENTO. Obtenido/Esperado: "
350              & std_logic'image(des) & std_logic'image(ref_des);
351          -- contador de errores
352          if des /= ref_des then
353              num_errores := num_errores + 1;
354          end if;
355          -- Chequeo del resto de senales de salida y contador errores -----
356          check_error(zero, ref_zero, sig, ref_signo, ref_des, num_errores);
357      end loop;
358      end if;
359      --Salida por consola del numero de errores
360      --condicion 3 del enunciado
361      report "TEST COMPLETO. HAY " & integer'image(num_errores) & " ERRORES.";
362      wait; -- Final de simulacion
363  end process vect_test;
364
365 end architecture bp_sum;

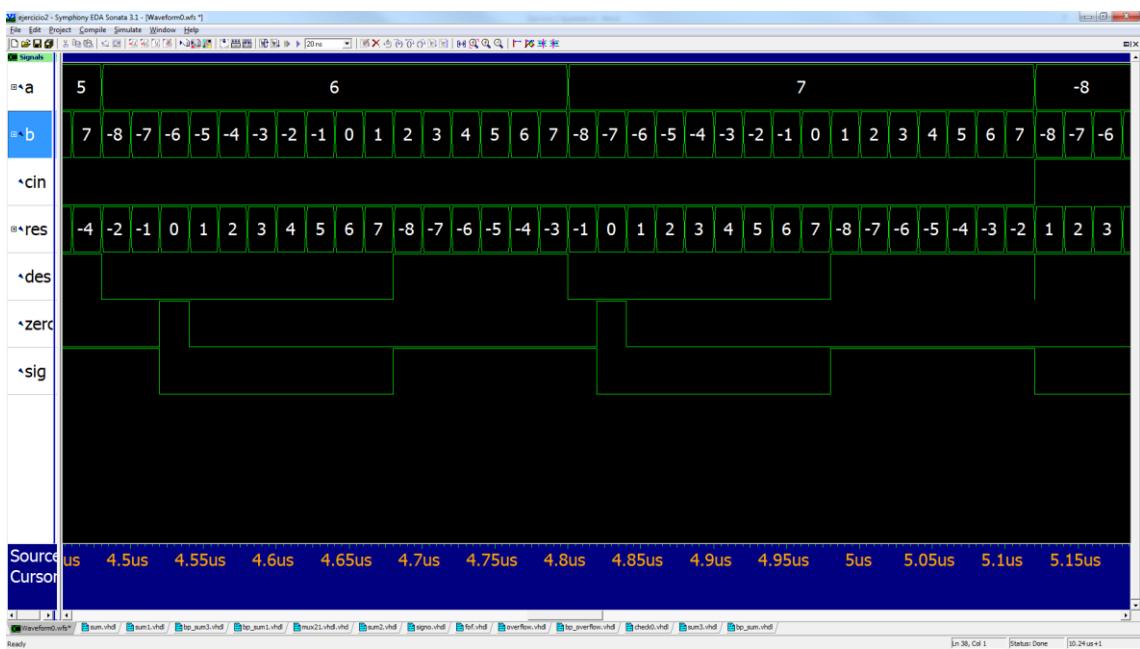
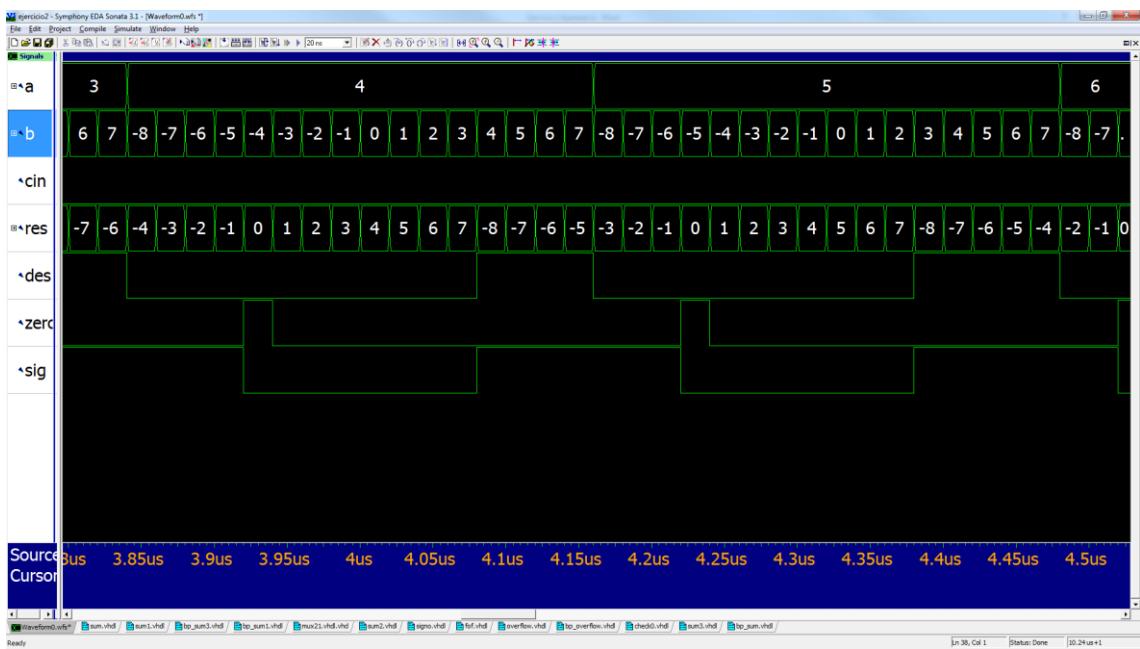
```

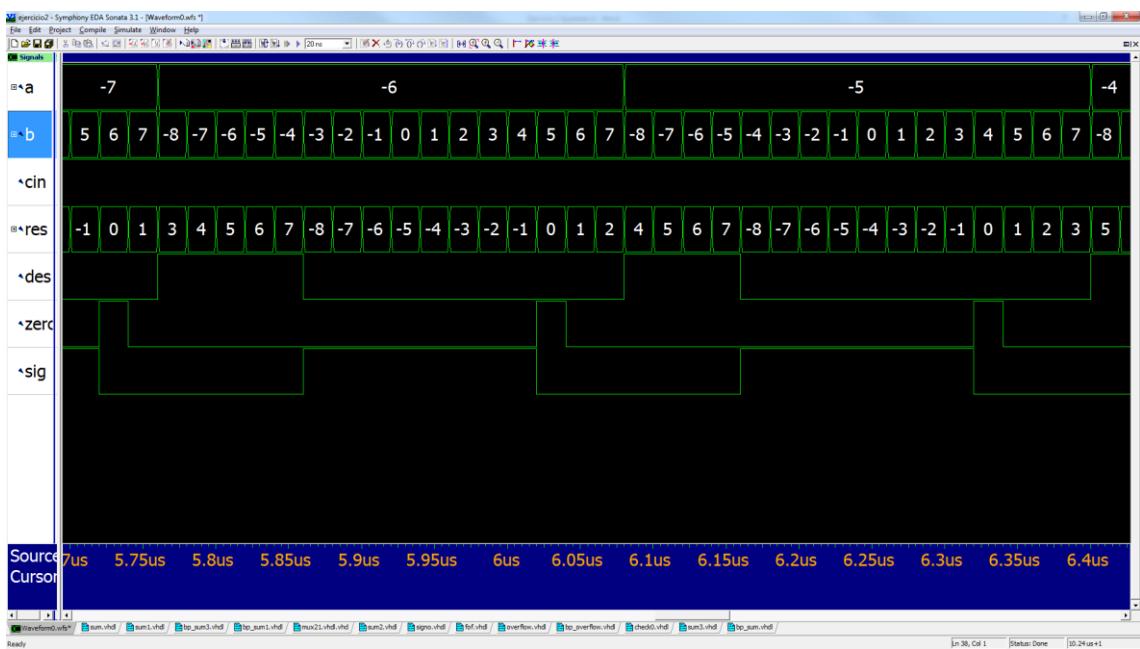
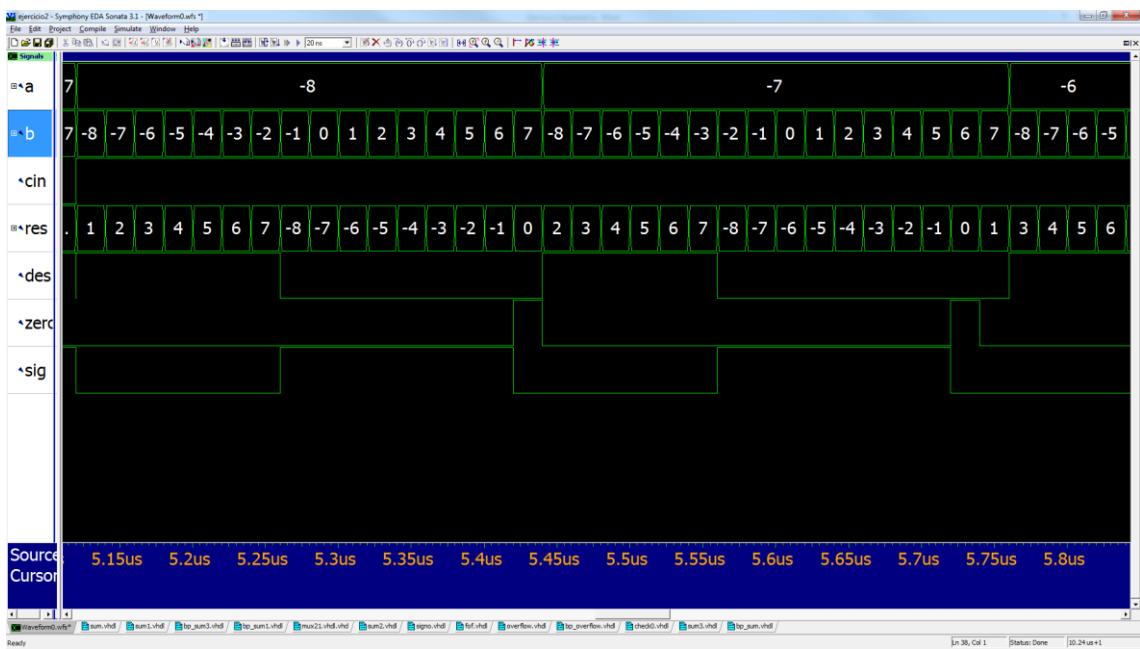
Realice la simulación para el caso en que el número de bits de los operandos sea (n=4).

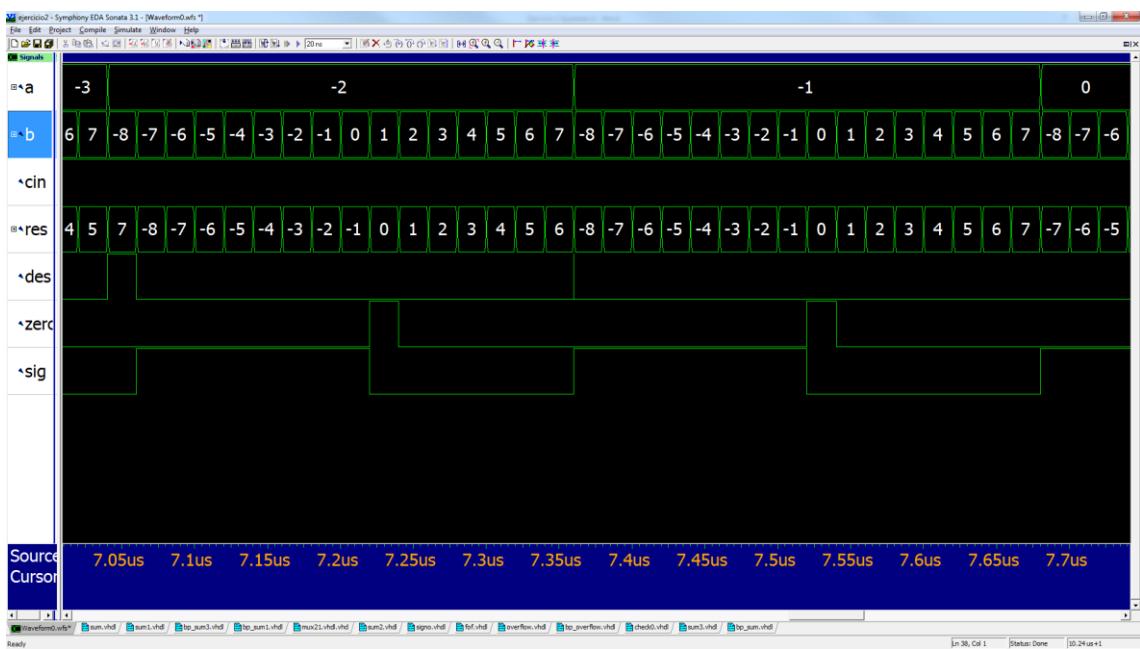
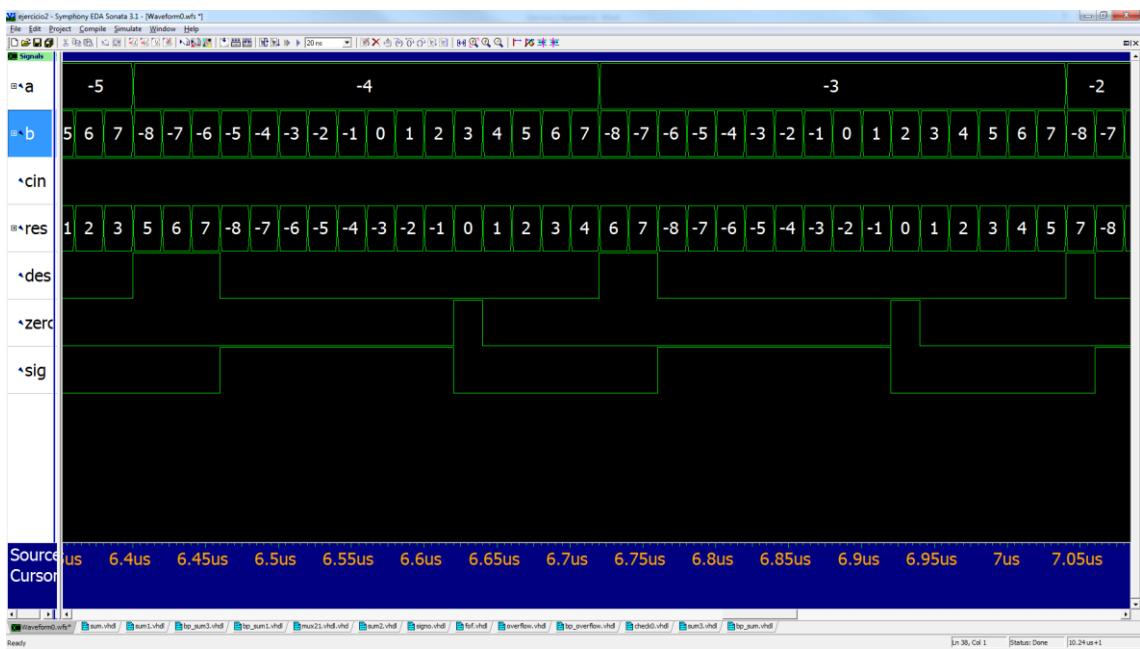


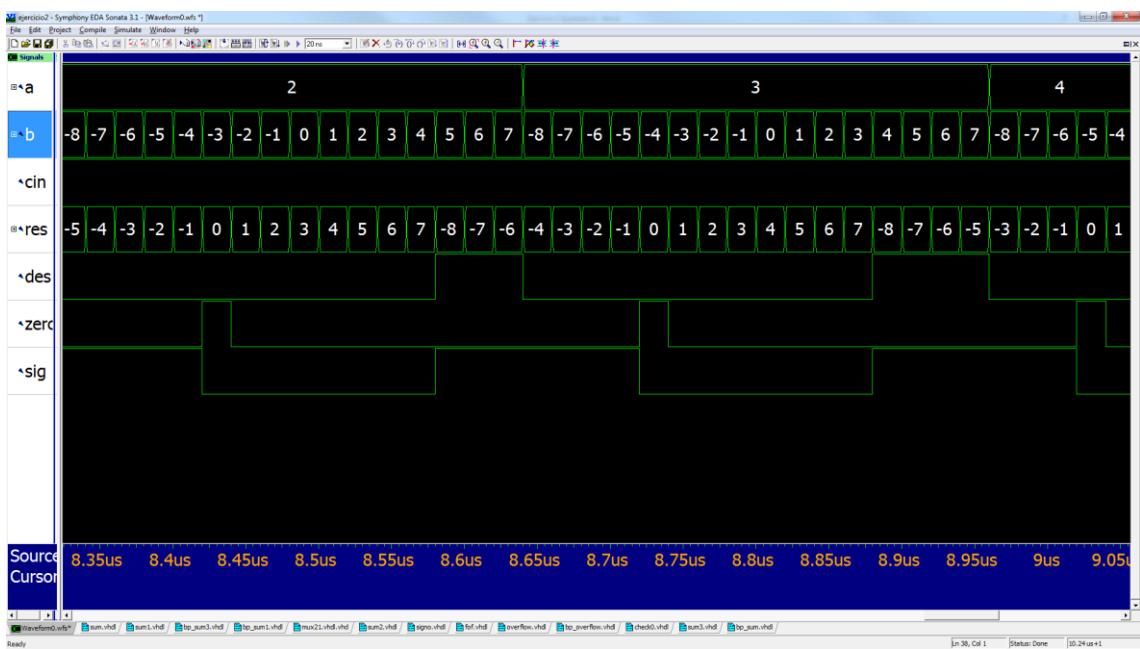
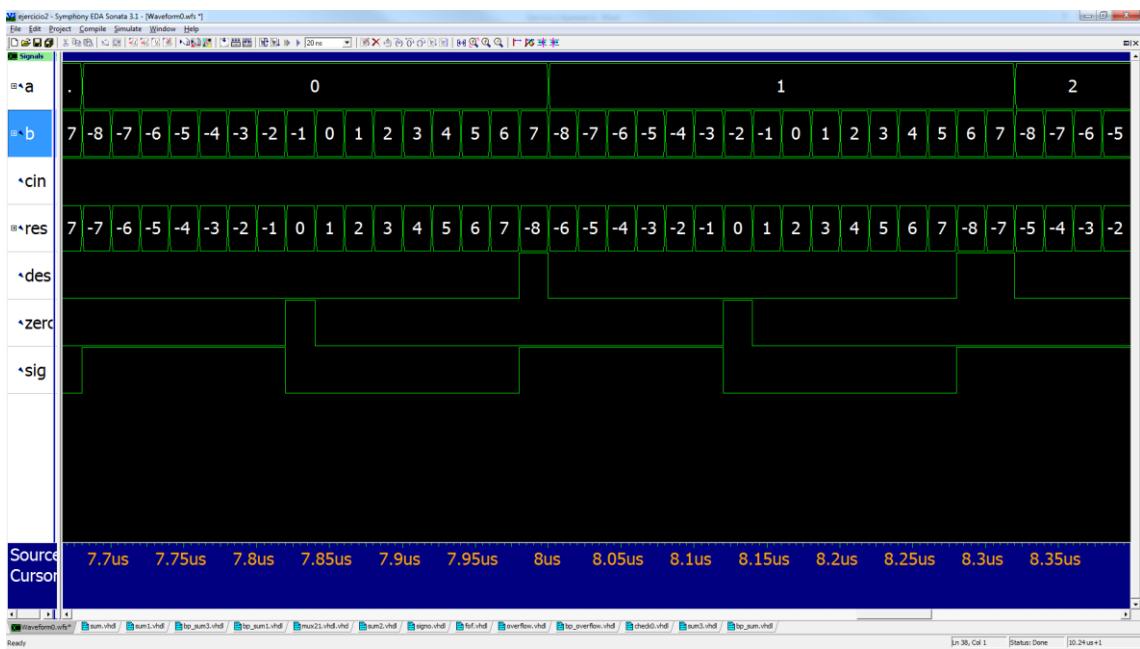


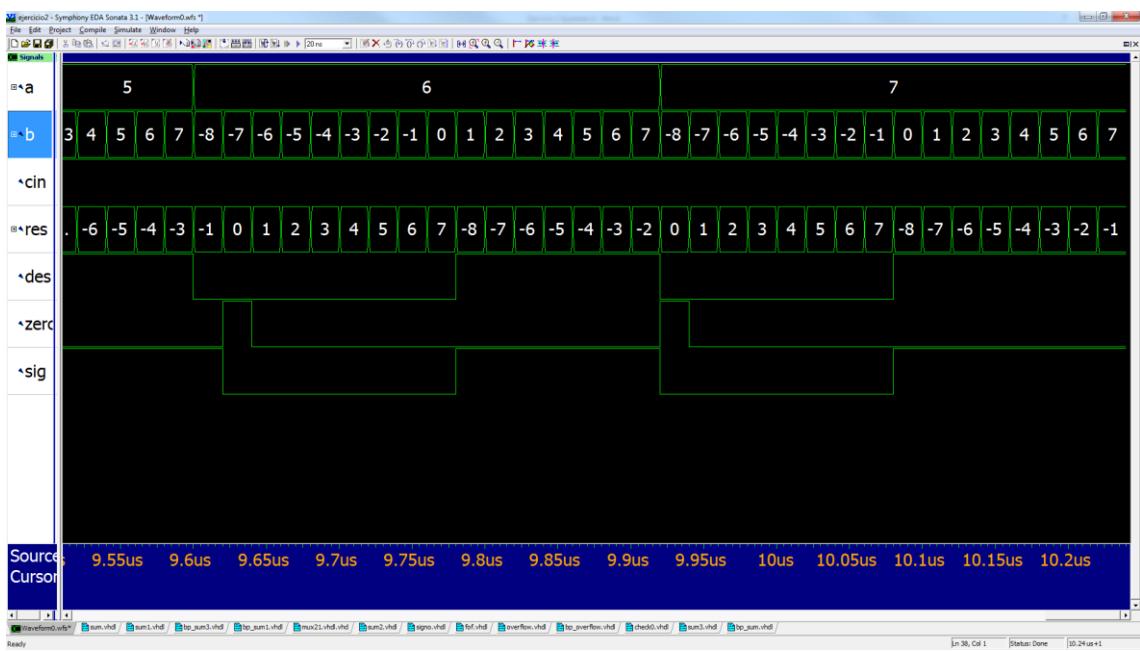
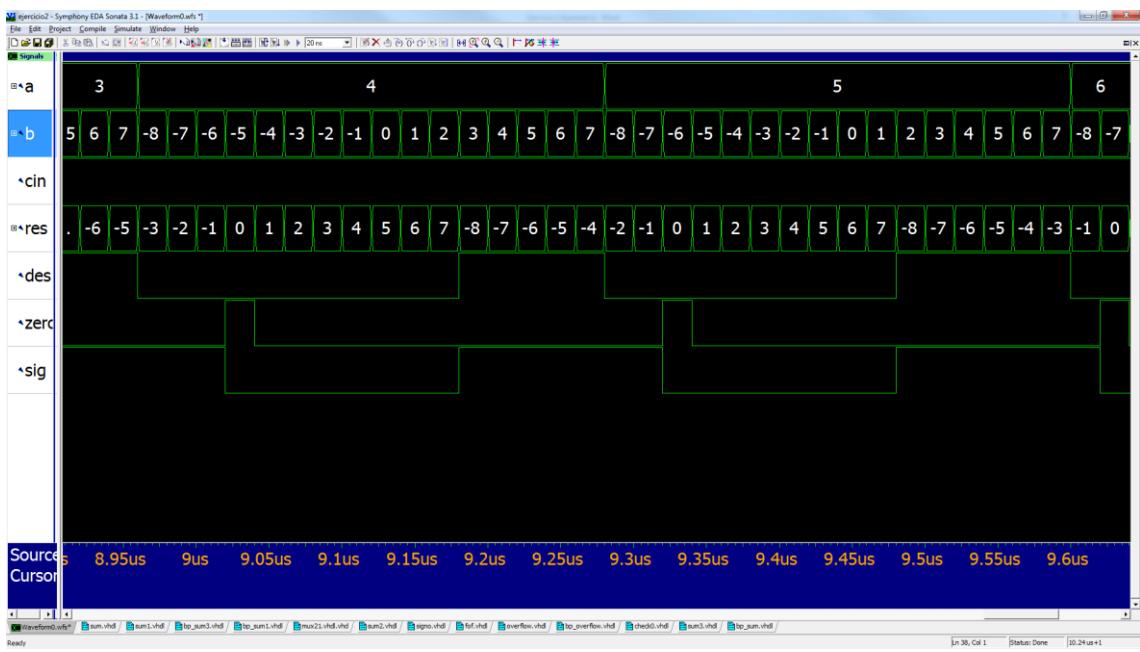












# Anexo: Banco de Pruebas

```

1 --- Banco de pruebas del componente sum1 del ejercicio 2
2
3
4 library IEEE;
5 use IEEE.std_logic_1164.all;
6 use IEEE.numeric_std.all;
7
8 entity bp_sum1 is
9     constant DELAY : time := 20 ns; -- Retardo usado en el test
10 end entity bp_sum1;
11
12 architecture bp_sum1 of bp_sum1 is
13     constant WIDTH : integer := 3;
14     signal a, b : std_logic_vector(WIDTH-1 downto 0) := (others => '0');
15     signal sum, sumc : std_logic_vector(WIDTH-1 downto 0);
16
17 begin
18     UUT : entity work.sum1(archsum1) generic map (WIDTH) port map (sum,
19     sumc, a , b);
20
21     vect_test : process is
22         variable valorA : signed (WIDTH-1 downto 0);
23         variable valorB : signed (WIDTH-1 downto 0);
24         begin
25             -- Genera todos los posibles valores de entrada
26             -----
27             -- Primer sumando. negativos y el cero
28             -----
29             for i in 0 to 2**WIDTH-1 loop
30                 valorA := to_signed((-1*2**WIDTH)+i, WIDTH);
31                 a ≤ std_logic_vector(valorA);
32             -- Primer sumando. negativos y el cero. Segundo sumando
33             -- negativos y cero
34             for j in 0 to 2**WIDTH-1 loop
35                 valorB := to_signed((-1*2**WIDTH)+j, WIDTH);
36                 b ≤ std_logic_vector(valorB);
37                 wait for DELAY;
38             end loop;
39             -- Primer NEGATIVOS-CERO. Segundo POSITIVOS
40             for j in 0 to 2**WIDTH-1 loop
41                 valorB := to_signed(j, WIDTH);
42                 b ≤ std_logic_vector(valorB);
43                 wait for DELAY;
44             end loop;
45         end loop;
46             -----
47             -- Primer POSITIVOS
48             -----
49             for i in 0 to 2**WIDTH-1 loop
50                 valorA := to_signed(i, WIDTH);
51                 a ≤ std_logic_vector(valorA);
52             -- Primer POSITIVOS. Segundo NEGATIVOS Y CERO
53             for j in 0 to 2**WIDTH-1 loop
54                 valorB := to_signed((-1*2**WIDTH)+j, WIDTH);
55                 b ≤ std_logic_vector(valorB);
56                 wait for DELAY;
57             end loop;
58             -- Primer POSITIVOS. Segundo POSITIVOS.
59             for j in 0 to 2**WIDTH-1 loop

```

```
58         valorB := to_signed(j, WIDTH);
59         b ≤ std_logic_vector(valorB);
60         wait for DELAY;
61     end loop;
62 end loop;
63 wait; -- Final de simulacion
64 end process vect_test;
65
66 end architecture bp_sum1;
67
68
69
70
71
```

```

1-----+
2-- Banco de pruebas del componente mux2a1 del ejercicio 2
3-----+
4
5library ieee;
6use ieee.std_logic_1164.all;
7use ieee.numeric_std.all;
8
9entity bp_mux is
10    constant DELAY : time := 20 ns; -- Retardo usado en el test
11end entity bp_mux;
12
13architecture bp_mux of bp_mux is
14    constant WIDTH : integer := 3;
15    signal res, sum, sumc :std_logic_vector(WIDTH-1 downto 0);
16    signal cin : std_logic;
17    signal aux : std_logic_vector (WIDTH downto 0) := (others => '0');
18    signal auxc : std_logic_vector (WIDTH downto 0) := (0 => '1', others =>
19        '0');
20
21begin
22    UUT : entity work.mux2a1(archmux) generic map (WIDTH) port map (res,
23    sum, sumc, cin);
24
25    vect_test : process is
26        variable valor1 : unsigned (WIDTH downto 0);
27        variable valor2 : unsigned (WIDTH downto 0);
28
29        begin
30            for i in 1+2 to (2** (WIDTH-1))+2 loop
31                valor1 := to_unsigned(i-1, WIDTH+1);
32                valor2 := to_unsigned(i, WIDTH+1);
33                aux ≤ std_logic_vector(valor1);
34                auxc ≤ std_logic_vector(valor2);
35                sum ≤ aux (WIDTH-1 downto 0);
36                sumc ≤ auxc (WIDTH-1 downto 0);
37                cin ≤ '0';
38                wait for DELAY;
39                cin ≤ '1';
40                wait for DELAY;
41            end loop;
42            wait; -- Final de la simulacion
43        end process vect_test;
44
45end architecture bp_mux;

```

```

1 -----
2 -- Banco de pruebas del componente sum2 del ejercicio 2
3 -----
4 library IEEE;
5 use IEEE.std_logic_1164.all;
6 use IEEE.numeric_std.all;
7
8 entity bp_sum2 is
9     constant DELAY : time := 20 ns; -- Retardo usado en el test
10 end entity bp_sum2;
11
12 architecture bp_sum2 of bp_sum2 is
13     constant WIDTH : integer := 3;
14     signal a, b : std_logic_vector(WIDTH-1 downto 0) := (others => '0');
15     signal res : std_logic_vector(WIDTH-1 downto 0);
16     signal cin : std_logic;
17
18 begin
19     UUT : entity work.sum2(archsum2) generic map (WIDTH) port map (res, a,
b, cin);
20
21     vect_test : process is
22         variable valorA : signed (WIDTH-1 downto 0);
23         variable valorB : signed (WIDTH-1 downto 0);
24     begin
25         for k in 0 to 1 loop
26             -- Genera los resultados sin acarreo
27             if k=0 then
28                 cin ≤ '0';
29             -- Genera todos los posibles valores de entrada
30             -- Genera los negativos y el cero
31             for i in 0 to 2**WIDTH-1 loop
32                 valorA := to_signed((-1*2**WIDTH)+i, WIDTH);
33                 a ≤ std_logic_vector(valorA);
34                 for j in 0 to 2**WIDTH-1 loop
35                     valorB := to_signed((-1*2**WIDTH)+j, WIDTH);
36                     b ≤ std_logic_vector(valorB);
37                     wait for DELAY;
38                 end loop;
39                 for j in 0 to 2**WIDTH-1 loop
40                     valorB := to_signed(j, WIDTH);
41                     b ≤ std_logic_vector(valorB);
42                     wait for DELAY;
43                 end loop;
44             end loop;
45             -- Genera los positivos
46             for i in 0 to 2**WIDTH-1 loop
47                 valorA := to_signed(i, WIDTH);
48                 a ≤ std_logic_vector(valorA);
49                 for j in 0 to 2**WIDTH-1 loop
50                     valorB := to_signed((-1*2**WIDTH)+j, WIDTH);
51                     b ≤ std_logic_vector(valorB);
52                     wait for DELAY;
53                 end loop;
54                 for j in 0 to 2**WIDTH-1 loop
55                     valorB := to_signed(j, WIDTH);
56                     b ≤ std_logic_vector(valorB);
57                     wait for DELAY;
58                 end loop;

```

```

59      end loop;
60      -- Genera los resultados con acarreo
61  else
62      cin <= '1';
63      -- Genera todos los posibles valores de entrada
64      -- Genera los negativos y el cero
65      for i in 0 to 2**(WIDTH-1)-1 loop
66          valorA := to_signed((-1*2**(WIDTH-1))+i, WIDTH);
67          a <= std_logic_vector(valorA);
68          for j in 0 to 2**(WIDTH-1)-1 loop
69              valorB := to_signed((-1*2**(WIDTH-1))+j,
70 WIDTH);
71                  b <= std_logic_vector(valorB);
72                  wait for DELAY;
73          end loop;
74          for j in 0 to 2**(WIDTH-1)-1 loop
75              valorB := to_signed(j, WIDTH);
76              b <= std_logic_vector(valorB);
77              wait for DELAY;
78          end loop;
79          -- Genera los positivos
80          for i in 0 to 2**(WIDTH-1)-1 loop
81              valorA := to_signed(i, WIDTH);
82              a <= std_logic_vector(valorA);
83              for j in 0 to 2**(WIDTH-1)-1 loop
84                  valorB := to_signed((-1*2**(WIDTH-1))+j,
85 WIDTH);
86                  b <= std_logic_vector(valorB);
87                  wait for DELAY;
88          end loop;
89          for j in 0 to 2**(WIDTH-1)-1 loop
90              valorB := to_signed(j, WIDTH);
91              b <= std_logic_vector(valorB);
92              wait for DELAY;
93          end loop;
94      end if;
95  end loop;
96  wait; -- Final de simulacion
97 end process vect_test;
98
99 end architecture bp_sum2;
100
101

```

```

1-----+
2 -- Banco de pruebas del componente signo del ejercicio 2
3-----+
4
5 library ieee;
6 use ieee.std_logic_1164.all;
7 use ieee.numeric_std.all;
8
9 entity bp_signo is
10    constant DELAY : time := 20 ns; -- Retardo usado en el test
11 end entity bp_signo;
12
13 architecture bp_signo of bp_signo is
14    constant WIDTH : integer := 3;
15    signal num1, num2 :std_logic_vector(WIDTH-1 downto 0):= (others => '0');
16    signal sig1, sig2 : std_logic;
17
18 begin
19 UTT1 : entity work.signo(archsigno) generic map (WIDTH) port map (sig1, num1);
20 UTT2 : entity work.signo(archsigno) generic map (WIDTH) port map (sig2, num2);
21
22     vect_test : process is
23         variable valor1 : unsigned (WIDTH-1 downto 0);
24         variable valor2 : unsigned (WIDTH-1 downto 0);
25         begin
26             for i in 0 to (2**WIDTH)-1 loop
27                 if i mod 2 = 0 then
28                     valor1 := to_unsigned((2**WIDTH)-1-i, WIDTH);
29                     valor2 := to_unsigned(i, WIDTH);
30                 else
31                     valor2 := to_unsigned((2**WIDTH)-1-i, WIDTH);
32                     valor1 := to_unsigned(i, WIDTH);
33                 end if;
34                 num1 ≤ std_logic_vector(valor1);
35                 num2 ≤ std_logic_vector(valor2);
36                 wait for DELAY;
37             end loop;
38             wait; -- Final de la simulacion
39         end process vect_test;
40
41 end architecture bp_signo;
42
43
44
```

```
1-----  
2-- Banco de pruebas del componente fof del ejercicio 2  
3-----  
4library IEEE;  
5use IEEE.std_logic_1164.all;  
6use IEEE.numeric_std.all;  
7  
8entity bp_fof is  
9    constant DELAY :time := 20 ns; -- Retardo usado en el test  
10end entity bp_fof;  
11  
12architecture bp_fof of bp_fof is  
13    signal n1, n2, r : std_logic := '0';  
14    signal f : std_logic;  
15  
16begin  
17    UUT : entity work.fof(archfof) port map (f, n1, n2, r);  
18  
19    vec_test : process is  
20        variable valor : unsigned (2 downto 0);  
21        begin  
22            -- Genera todos los posibles valores de entrada  
23            for i in 0 to 7 loop  
24                valor := to_unsigned(i, 3);  
25                r ≤ std_logic(valor(0));  
26                n2 ≤ std_logic(valor(1));  
27                n1 ≤ std_logic(valor(2));  
28                wait for DELAY;  
29            end loop;  
30            wait; -- Final de simulacion  
31        end process vec_test;  
32    end architecture bp_fof;  
33
```

```

1-----+
2 -- Banco de pruebas del componente overflow del ejercicio 2
3-----+
4
5 library IEEE;
6 use IEEE.std_logic_1164.all;
7 use IEEE.numeric_std.all;
8
9 entity bp_overflow is
10    constant DELAY : time := 20 ns; -- Retardo usado en el test
11 end entity bp_overflow;
12
13 architecture bp_overflow of bp_overflow is
14    constant WIDTH : integer := 3;
15    signal a, b, res : std_logic_vector(WIDTH-1 downto 0):= (others => '0');
16    signal des : std_logic;
17
18 begin
19    UUT : entity work.overflow(archoverflow) generic map (WIDTH) port map
20        (des, a, b, res);
21
22    vect_test : process is
23        variable valorA : signed (WIDTH-1 downto 0);
24        variable valorB : signed (WIDTH-1 downto 0);
25        begin
26            -- Genera todos los posibles valores de entrada
27            -- Genera los negativos y el cero
28            for i in 0 to 2**(WIDTH-1)-1 loop
29                valorA := to_signed((-1*2**(WIDTH-1))+i, WIDTH);
30                a ≤ std_logic_vector(valorA);
31                for j in 0 to 2**(WIDTH-1)-1 loop
32                    valorB := to_signed((-1*2**(WIDTH-1))+j, WIDTH);
33                    b ≤ std_logic_vector(valorB);
34                    res ≤ std_logic_vector(valorA+valorB);
35                    wait for DELAY;
36                end loop;
37                for j in 0 to 2**(WIDTH-1)-1 loop
38                    valorB := to_signed(j, WIDTH);
39                    b ≤ std_logic_vector(valorB);
40                    res ≤ std_logic_vector(valorA+valorB);
41                    wait for DELAY;
42                end loop;
43            -- Genera los positivos
44            for i in 0 to 2**(WIDTH-1)-1 loop
45                valorA := to_signed(i, WIDTH);
46                a ≤ std_logic_vector(valorA);
47                for j in 0 to 2**(WIDTH-1)-1 loop
48                    valorB := to_signed((-1*2**(WIDTH-1))+j, WIDTH);
49                    b ≤ std_logic_vector(valorB);
50                    res ≤ std_logic_vector(valorA+valorB);
51                    wait for DELAY;
52                end loop;
53                for j in 0 to 2**(WIDTH-1)-1 loop
54                    valorB := to_signed(j, WIDTH);
55                    b ≤ std_logic_vector(valorB);
56                    res ≤ std_logic_vector(valorA+valorB);
57                    wait for DELAY;
58                end loop;

```

```
59          end loop;
60          wait; -- Final de simulacion
61      end process vect_test;
62
63 end architecture bp_overflow;
64
65
66
67
```

```

1 -----
2 --banco de pruebas del componente check0 del ejercicio 2
3 -----
4 library ieee;
5 use ieee.std_logic_1164.all;
6 use ieee.numeric_std.all;
7
8 entity bp_checkzero is
9     constant DELAY : time := 20 ns; --Retardo usado en el test
10 end entity bp_checkzero;
11
12 architecture bp_checkzero of bp_checkzero is
13     constant WIDTH : integer := 3;
14     signal num : std_logic_vector(WIDTH-1 downto 0) := (others => '0');
15     signal zero : std_logic;
16
17 begin
18     UUT : entity work.checkzero(archcheckzero) generic map (WIDTH) port map
19     (zero, num);
20
21     vect_test : process is
22         variable valor : signed(WIDTH-1 downto 0);
23
24         begin
25             --Genera todos los posibles valores de entrada
26             --Genera los negativos y el cero
27             for i in 0 to 2**WIDTH-1 loop
28                 valor := to_signed((-1*2**WIDTH)+i, WIDTH);
29                 num <= std_logic_vector(valor);
30                 wait for DELAY;
31             end loop;
32             --Genera los negativos
33             for i in 0 to 2**WIDTH-1 loop
34                 valor := to_signed(i, WIDTH);
35                 num <= std_logic_vector(valor);
36                 wait for DELAY;
37             end loop;
38             wait;
39         end process vect_test;
40     end architecture bp_checkzero;
41
42

```

```

1 -----
2 -- Banco de pruebas del componente sum3 del ejercicio 2
3 -----
4 library IEEE;
5 use IEEE.std_logic_1164.all;
6 use IEEE.numeric_std.all;
7
8 entity bp_sum3 is
9     constant DELAY : time := 20 ns; -- Retardo usado en el test
10 end entity bp_sum3;
11
12 architecture bp_sum3 of bp_sum3 is
13     constant WIDTH : integer := 3;
14     signal a, b : std_logic_vector(WIDTH-1 downto 0) := (others => '0');
15     signal res : std_logic_vector(WIDTH-1 downto 0);
16     signal cin, des : std_logic;
17
18 begin
19     UUT : entity work.sum3(archsum3) generic map (WIDTH) port map (res,
des, a, b, cin);
20
21     vect_test : process is
22         variable valorA : signed (WIDTH-1 downto 0);
23         variable valorB : signed (WIDTH-1 downto 0);
24         variable total : integer;
25         variable vec_total : std_logic_vector(2*WIDTH-1 downto 0);
26         variable ref_des : std_logic;
27     begin
28         for k in 0 to 1 loop
29             -- Genera los resultados sin acarreo
30             if k=0 then
31                 cin ≤ '0';
32             -- Genera todos los posibles valores de entrada
33             -- Genera los negativos y el cero del primer sumando
34             for i in 0 to 2**WIDTH-1 loop
35                 valorA := to_signed((-1*2**WIDTH+1)+i, WIDTH);
36                 a ≤ std_logic_vector(valorA);
37                 -- Genera los negativos y el cero del segundo sumando
38                 for j in 0 to 2**WIDTH-1 loop
39                     valorB := to_signed((-1*2**WIDTH+j), WIDTH);
40                     b ≤ std_logic_vector(valorB);
41                     total := -1*2**WIDTH+i+j;
42                     vec_total := std_logic_vector(to_signed(total,
2*WIDTH));
43                     wait for DELAY;
44                     ----- Genera el bit referencia para la
comparacion de acarrero -----
45                     if total>2**WIDTH-1 or total<1*2**WIDTH-1
then
46                         ref_des := '1';
47                         else
48                             ref_des := '0';
49                         end if;
45                     -----Bucle para chequear el
resultado de la suma
50                         for m in 0 to WIDTH-1 loop

```









```

219                               b <= std_logic_vector(valorB);
220                               total := i+j+1;
221                               vec_total := std_logic_vector(to_signed(total,
222                                         2*WIDTH));
223                               wait for DELAY;
224                               ----- Genera el bit referencia para la
225                               comparacion de acarrero -----
226                               if total>2**WIDTH-1 or total<1*2**WIDTH-1
227 then
228                               ref_des := '1';
229                               else
230                               ref_des := '0';
231                               end if;
232                               -----Bucle para chequear el
233                               resultado de la suma
234                               for m in 0 to WIDTH-1 loop
235                               assert res(m)=vec_total(m) report "ERROR EN LA
236                               SUMA. Obtenido/Esperado: "
237                               std_logic'image(vec_total(m));
238                               & std_logic'image(res(m)) &
239                               & std_logic'image(vec_total(m)) & integer'image(total);
240                               end loop;
241                               ----- Chequeo del desbordamiento -----
242                               assert des = ref_des report "ERROR EN EL
243                               DESBORDAMIENTO. Obtenido/Esperado: "
244                               & std_logic'image(des) & std_logic'image(ref_des) &
245                               integer'image(i) & integer'image(j);
246                               end loop;
247                               end loop;
248                               end if;
249                               end loop;
250                               wait; -- Final de simulacion
251 end process vect_test;
252
253 end architecture bp_sum3;

```

```

1 -----
2 -- Banco de pruebas del ejercicio 2
3 -----
4 library IEEE;
5 use IEEE.std_logic_1164.all;
6 use IEEE.numeric_std.all;
7
8 entity bp_sum is
9     constant DELAY : time := 20 ns; -- Retardo usado en el test
10 end entity bp_sum;
11
12 architecture bp_sum of bp_sum is
13     constant WIDTH : integer := 4;
14     signal a, b : std_logic_vector(WIDTH-1 downto 0) := (others => '0');
15     signal res : std_logic_vector(WIDTH-1 downto 0);
16     signal cin, des, zero, sig : std_logic;
17
18     procedure gen_ref (total, WIDTH : in integer;
19                         ref_des, ref_zero, ref_signo : out std_logic) is
20
21         begin
22             ----- Genera el bit referencia para la comparacion de acarrero -----
23
24             if total>2**WIDTH-1 or total<1*2**WIDTH-1 then
25                 ref_des := '1';
26                 else
27                     ref_des := '0';
28                 end if;
29             ----- Genera el bit referencia para la comparacion de cero-----
30
31             if total = 0 then
32                 ref_zero := '1';
33                 else
34                     ref_zero := '0';
35                 end if;
36             ----- Genera el bit referencia para la comparacion de signo -----
37
38             if total < 0 then
39                 ref_signo := '1';
40                 else
41                     ref_signo := '0';
42                 end if;
43             end procedure gen_ref;
44
45             procedure check_error(zero, ref_zero, sig, ref_signo, ref_des : in
46 std_logic;
47                         numErrores : inout integer) is
48         begin
49             -----Chequeo de cero -----
50
51             assert zero = ref_zero report "ERROR EN EL CERO. Obtenido/Esperado: "
52             & std_logic'image(zero) & std_logic'image(ref_zero);
53             -- contador de errores
54             if zero /= ref_zero then
55                 numErrores := numErrores + 1;
56             end if;

```

```

52          -----Chequeo de signo -----
53
54      if ref_des = '0' then
55          assert sig = ref_signo report "ERROR EN EL SIGNO.
56
57  Obtido/Esperado: "
58      & std_logic'image(sig) & std_logic'image(ref_signo);
59      -- contador de errores
60      if sig /= ref_signo then
61          num_errores := num_errores + 1;
62          end if;
63      end if;
64  end procedure check_error;
65
66 begin
67     UUT : entity work.sum(archsum) generic map (WIDTH) port map (res, des,
68 zero, sig, a, b, cin);
69
70     vect_test : process is
71         variable valorA : signed (WIDTH-1 downto 0);
72         variable valorB : signed (WIDTH-1 downto 0);
73         variable total : integer;
74         variable vec_total : std_logic_vector(2*WIDTH-1 downto 0);
75         variable ref_des : std_logic;
76         variable ref_zero : std_logic;
77         variable ref_signo : std_logic;
78         variable num_errores : integer := 0;
79     begin
80         for k in 0 to 1 loop
81             -- Genera los resultados sin acarreo
82             if k=0 then
83                 cin ≤ '0';
84             -- Genera todos los posibles valores de entrada
85             -- Genera los negativos y el cero del primer sumando
86             for i in 0 to 2**WIDTH-1 loop
87                 valorA := to_signed((-1*2**WIDTH)+i, WIDTH);
88                 a ≤ std_logic_vector(valorA);
89                 -- Genera los negativos y el cero del segundo sumando
90                 for j in 0 to 2**WIDTH-1 loop
91                     valorB := to_signed((-1*2**WIDTH)+j, WIDTH);
92                     b ≤ std_logic_vector(valorB);
93                     -- GENERACION DE REFERENCIAS
94                     -- Genera el resultado referencia para chequeo de
95                     errores
96                     total := -1*2**WIDTH+i+j;
97                     vec_total := std_logic_vector(to_signed(total,
98 2*WIDTH));
99                     wait for DELAY;
100                    -- Genera el resto de referencias
101                    gen_ref(total, WIDTH, ref_des, ref_zero,
102 ref_signo);
103                    -- CHEQUEO
104                    -----Bucle para chequear el
105 resultado de la suma
106                    for m in 0 to WIDTH-1 loop
107                        assert res(m)=vec_total(m) report "ERROR EN LA
108 SUMA. Obtido/Esperado: "
109                     & std_logic'image(vec_total(m)) &
110 std_logic'image(vec_total(m));
111                     -- contador de errores
112                     if res(m) /= vec_total(m) then

```

```

103                     num_errores := num_errores + 1;
104                 end if;
105             end loop;
106             ----- Chequeo del desbordamiento -----
107
107             assert des = ref_des report "ERROR EN EL
108             DESBORDAMIENTO. Obtenido/Esperado: "
109             & std_logic'image(des) & std_logic'image(ref_des);
110             -- contador de errores
111             if des /= ref_des then
112                 num_errores := num_errores + 1;
113             end if;
114             ----- Chequeo el resto de senales de
114             salida y contador errores -----
115             check_error(zero, ref_zero, sig, ref_signo,
116             ref_des, num_errores);
115         end loop;
116         -----Final de negativos y cero
117         del segundo sumando-----
117         --Genera los positivos del segundo sumando-----
118
118         for j in 0 to 2**WIDTH-1 loop
119             valorB := to_signed(j, WIDTH);
120             b ≤ std_logic_vector(valorB);
121             -- GENERACIÓN DE REFERENCIAS
122             -- Genera el resultado referencia para el chequeo
122             de errores
123             total := -1*2**WIDTH+i+j;
124             vec_total := std_logic_vector(to_signed(total,
124             2*WIDTH));
125
125             wait for DELAY;
126             ----- Genera el resto de referencias
127             gen_ref(total, WIDTH, ref_des, ref_zero,
127             ref_signo);
128             -- CHEQUEO
129             -----Bucle para chequear el
129             resultado de la suma
130             for m in 0 to WIDTH-1 loop
131                 assert res(m)=vec_total(m) report "ERROR EN LA
131                 SUMA. Obtenido/Esperado: "
132                 & std_logic'image(vec_total(m));
133
133                 -- contador errores
134                 if res(m) /= vec_total(m) then
135                     num_errores := num_errores +1;
136                 end if;
137             end loop;
138             ----- Chequeo del desbordamiento -----
139
139             assert des = ref_des report "ERROR EN EL
139             DESBORDAMIENTO. Obtenido/Esperado: "
140             & std_logic'image(des) & std_logic'image(ref_des);
141             -- contador errores
142             if des /= ref_des then
143                 num_errores := num_errores + 1;
144             end if;
145             ----- Chequeo del resto de senales de salida y
145             contador de errores -----
146             check_error(zero, ref_zero, sig, ref_signo,
146             ref_des, num_errores);

```

```

147                     end loop;
148                 end loop;
149             -----FINAL DE LOS
150             NEGATIVOS PRIMER SUMANDO-----  

151             -- Genera los positivos del primer sumando-----  

152
153             for i in 0 to 2**(WIDTH-1)-1 loop
154                 valorA := to_signed(i, WIDTH);
155                 a ≤ std_logic_vector(valorA);
156                 -----Genera los negativos y cero
157                 del segundo sumando-----  

158                 for j in 0 to 2**(WIDTH-1)-1 loop
159                     valorB := to_signed((-1*2**((WIDTH-1))+j, WIDTH);
160                     b ≤ std_logic_vector(valorB);
161                     -- GENERACION DE REFERENCIAS
162                     -- Genera el resultado referencia para chequeo de
163                     errores
164                     total := -1*2**((WIDTH-1)+i+j;
165                     vec_total := std_logic_vector(to_signed(total,
166                         2*WIDTH));
167                     wait for DELAY;
168                     ----- Genera el resto de referencias -----
169
170                     gen_ref(total, WIDTH, ref_des, ref_zero,
171                     ref_signo);
172                     -- CHEQUEO
173                     -----Bucle para chequear el
174                     resultado de la suma
175                     for m in 0 to WIDTH-1 loop
176                         assert res(m)=vec_total(m) report "ERROR EN LA
177                         SUMA. Obtenido/Esperado: "
178                         std_logic'image(vec_total(m));
179                         & std_logic'image(res(m)) &
180                         -- contador errores
181                         if res(m) /= vec_total(m) then
182                             num_errores := num_errores + 1;
183                         end if;
184                     end loop;
185                     ----- Chequeo del desbordamiento -----
186
187                     assert des = ref_des report "ERROR EN EL
188                     DESBORDAMIENTO. Obtenido/Esperado: "
189                     & std_logic'image(des) & std_logic'image(ref_des);
190                     -- contador errores
191                     if des /= ref_des then
192                         num_errores := num_errores + 1;
193                     end if;
194                     ----- Chequeo el resto de senales de
195                     salida y contador errores
196                     check_error(zero, ref_zero, sig, ref_signo,
197                     ref_des, num_errores);
198                     end loop;
199                     -----genera los
200                     positivos del segundo sumando-----  

201                     for j in 0 to 2**((WIDTH-1)-1) loop
202                         valorB := to_signed(j, WIDTH);
203                         b ≤ std_logic_vector(valorB);
204                         -- GENERACION DE REFERENCIAS
205                         -- Genera el resultado referencia para el chequeo
206                         de errores

```

```

191                               total := i+j;
192                               vec_total := std_logic_vector(to_signed(total,
193                                         2*WIDTH));
194
195                               wait for DELAY;
196                               ----- Genera el resto de referencias
197                               gen_ref(total, WIDTH, ref_des, ref_zero,
198                               ref_signo);
199                               -- CHEQUEO
200                               -----Bucle para chequear el
201 resultado de la suma
202
203                               SUMA. Obtenido/Esperado: "
204                               std_logic'image(vec_total(m));
205                               & std_logic'image(res(m)) &
206                               -- contador de errores
207                               if res(m) /= vec_total(m) then
208                                   num_errores := num_errores + 1;
209                               end if;
210                               end loop;
211                               ----- Chequeo del desbordamiento -----
212
213                               assert des = ref_des report "ERROR EN EL
214 DESBORDAMIENTO. Obtenido/Esperado: "
215                               & std_logic'image(des) & std_logic'image(ref_des);
216                               -- contador de errores
217                               if des /= ref_des then
218                                   num_errores := num_errores + 1;
219                               end if;
220                               ----- Chequeo el resto de senales de salida y
221 contador de errores -----
222                               check_error(zero, ref_zero, sig, ref_signo,
223                               ref_des, num_errores);
224                               end loop;
225                               end loop;
226                               -- Genera los resultados con acarreo
227                               else
228                                   cin <= '1';
229                                   -- Genera todos los posibles valores de entrada
230                                   -- Genera los negativos y el cero del primer sumando
231                                   for i in 0 to 2**(WIDTH-1)-1 loop
232                                       valorA := to_signed((-1*2**(WIDTH-1))+i, WIDTH);
233                                       a <= std_logic_vector(valorA);
234                                       -- Genera los negativos y el cero del segundo sumando
235                                       for j in 0 to 2**(WIDTH-1)-1 loop
236                                           valorB := to_signed((-1*2**(WIDTH-1))+j, WIDTH);
237                                           b <= std_logic_vector(valorB);
238                                           -- GENERACION DE REFERENCIAS
239                                           -- Genera el resultado referencia para chequeo de
240 errores
241
242                               total := -1*2**(WIDTH)+i+j+1;
243                               vec_total := std_logic_vector(to_signed(total,
244                                         2*WIDTH));
245
246                               wait for DELAY;
247                               ----- Genera el resto de referencias
248                               gen_ref(total, WIDTH, ref_des, ref_zero,
249                               ref_signo);
250                               --CHEQUEO
251                               -----Bucle para chequear el
252 resultado de la suma

```

```

238                     for m in 0 to WIDTH-1 loop
239                         assert res(m)=vec_total(m) report "ERROR EN LA
240                         SUMA. Obtenido/Esperado: "
241                         & std_logic'image(vec_total(m)) &
242                         -- contador de errores
243                         if res(m) /= vec_total(m) then
244                             num_errores := num_errores + 1;
245                         end if;
246                     end loop;
247                         ----- Chequeo del desbordamiento -----
248
249                     assert des = ref_des report "ERROR EN EL
250                     DESBORDAMIENTO. Obtenido/Esperado: "
251                     & std_logic'image(des) & std_logic'image(ref_des);
252                     -- contador de errores
253                     if des /= ref_des then
254                         num_errores := num_errores + 1;
255                     end if;
256                         ----- Chequeo del resto de senales de salida y
257                     contador de errores -----
258                     check_error(zero, ref_zero, sig, ref_signo,
259                     ref_des, num_errores);
260                     end loop;
261                         -----Final de negativos y cero
262                     del segundo sumando-----
263                     --Genera los positivos del segundo sumando-----
264
265                     for j in 0 to 2**WIDTH-1 loop
266                         valorB := to_signed(j, WIDTH);
267                         b ≤ std_logic_vector(valorB);
268                         -- GENERACION DE REFERENCIAS
269                         -- Genera el resultado referencia para el chequeo
270                         de errores
271                         total := -1*2**WIDTH+i+j+1;
272                         vec_total := std_logic_vector(to_signed(total,
273                         2*WIDTH));
274                         wait for DELAY;
275                         ----- Genera el resto de referencias
276                         gen_ref(total, WIDTH, ref_des, ref_zero,
277                         ref_signo);
278                         -- CHEQUEO
279                         -----Bucle para chequear el
280                         resultado de la suma
281                         for m in 0 to WIDTH-1 loop
282                             assert res(m)=vec_total(m) report "ERROR EN LA
283                             SUMA. Obtenido/Esperado: "
284                             & std_logic'image(vec_total(m)) &
285                             -- contador de errores
286                             if res(m) /= vec_total(m) then
287                                 num_errores := num_errores + 1;
288                             end if;
289                         end loop;
290                         ----- Chequeo del desbordamiento -----
291
292                     assert des = ref_des report "ERROR EN EL
293                     DESBORDAMIENTO. Obtenido/Esperado: "
294                     & std_logic'image(des) & std_logic'image(ref_des);
295                     --contador de errores

```



```

326         for j in 0 to 2**WIDTH-1 loop
327             valorB := to_signed(j, WIDTH);
328             b ≤ std_logic_vector(valorB);
329             -- GENERACION DE REFERENCIAS
330             -- Genera el resultado de referencia para el
331             -- chequeo de errores
332             total := i+j+1;
333             vec_total := std_logic_vector(to_signed(total,
334             2*WIDTH));
335
336             ref_signo);
337
338             resultado de la suma
339             SUMA. Obtenido/Esperado: "
340             std_logic'image(vec_total(m));
341
342             -- contador de errores
343             if res(m) /= vec_total(m) then
344                 num_errores := num_errores + 1;
345             end if;
346             end loop;
347             ----- Chequeo del desbordamiento -----
348             assert des = ref_des report "ERROR EN EL
349             DESBORDAMIENTO. Obtenido/Esperado: "
350             & std_logic'image(des) & std_logic'image(ref_des);
351             -- contador de errores
352             if des /= ref_des then
353                 num_errores := num_errores + 1;
354             end if;
355             ----- Chequeo del resto de senales de salida y
356             contador errores -----
357             check_error(zero, ref_zero, sig, ref_signo,
358             ref_des, num_errores);
359             end loop;
360             end if;
361             end loop;
362             --Salida por consola del numero de errores
363             --condicion 3 del enunciado
364             report "TEST COMPLETO. HAY " & integer'image(num_errores) &
365             "ERRORES.";
366             wait; -- Final de simulacion
367         end process vect_test;
368
369 end architecture bp_sum;

```