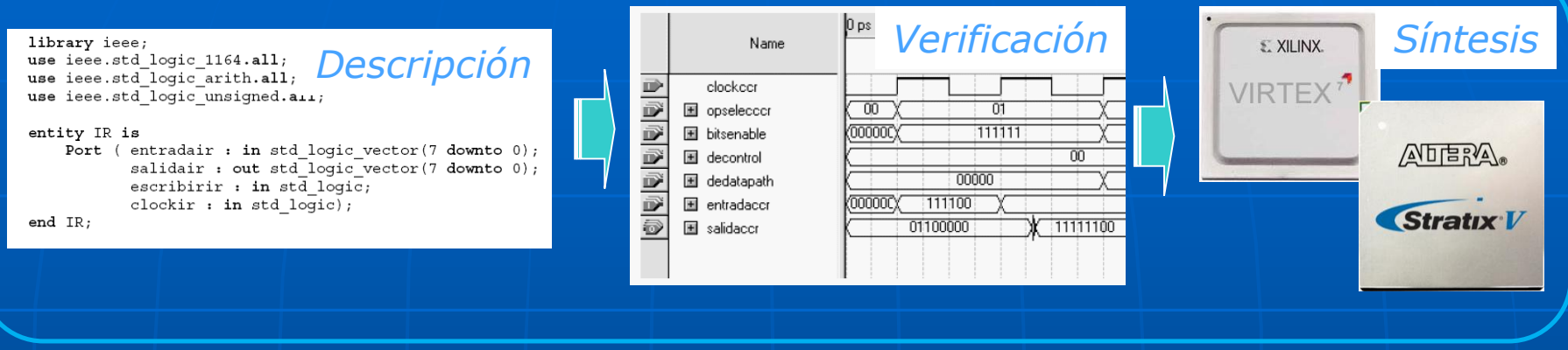


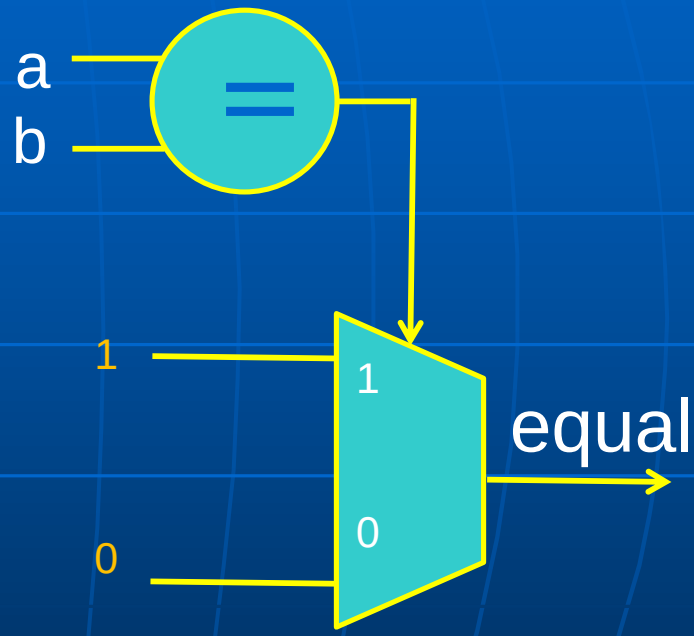

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;

entity IR is
    Port ( entradair : in std_logic_vector(7 downto 0);
           salidair : out std_logic_vector(7 downto 0);
           escribirir : in std_logic;
           clockir : in std_logic);
end IR;
```

Introducción al diseño lógico con VHDL

Parte 3 (Circuitos Aritméticos)

Ejemplo 2: DESCRIPCIÓN COMBINACIONAL CON PROCESS y señales

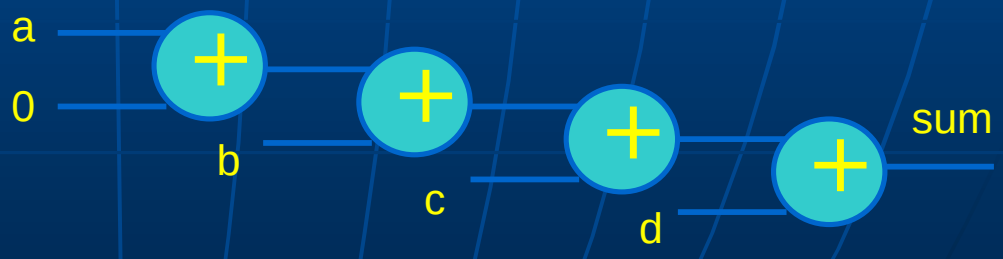
```
library ieee;
use ieee.std_logic_1164.all,
ieee.numeric_std.all;
entity compare is
   port (a, b : in unsigned (7 downto 0);
   equal : out std_logic);
end;
architecture behaviour of compare is
  begin
    process (a, b)
     begin
        if a = b then
           equal <= '1';
        else
           equal <= '0';
        end if;
     end process;
end;
```



Ejemplo 3: DESCRIPCIÓN COMBINACIONAL CON PROCESS y variables

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
entity adder is
    port (a, b, c, d : in signed(7 downto 0);
          result : out signed(7 downto 0));
end entity;

architecture behaviour of adder is
   begin
      four_add: process (a, b, c, d)
         variable sum : signed(7 downto 0);
      begin
         sum := a;
         sum := sum + b;
         sum := sum + c;
         sum := sum + d;
         result <= sum;
      end process;
end;
```

Cada vez que cambia alguna señal de entrada, se dispara el proceso y se recalcula la suma de a+b+c+d.



EJEMPLO de uso de LOOP (asignación secuencial)

Diseño de un sumador ripple-carry de 4 bits

```
LIBRARY ieee; USE ieee.std_logic_1164.ALL;
ENTITY sumador IS
  GENERIC (n_bits : INTEGER :=4);
  PORT (a : IN std_logic_vector (n_bits DOWNTO 1);
          b : IN std_logic_vector (n_bits DOWNTO 1);
          c_in : IN std_logic;
          c_out : OUT std_logic;
          suma : std_logic_vector (n_bits DOWNTO 1));
END ENTITY sumador;

ARCHITECTURE sumador_rc OF sumador IS
BEGIN
   PROCESS (a, b, c_in)
      VARIABLE vsuma : std_logic_vector (n_bits DOWNTO 1);
      VARIABLE carry : std_logic;
    BEGIN
        carry := c_in;
          FOR i IN 1 TO n_bits LOOP
            vsuma(i) := a(i) XOR b(i) XOR carry;
            carry := (a(i) AND b(i)) OR (carry AND (a(i) OR b(i)));
          END LOOP;
          suma <= vsuma; c_out <= carry;
    END PROCESS;
END ARCHITECTURE sumador_rd;
```

EJEMPLO de uso de GENERATE

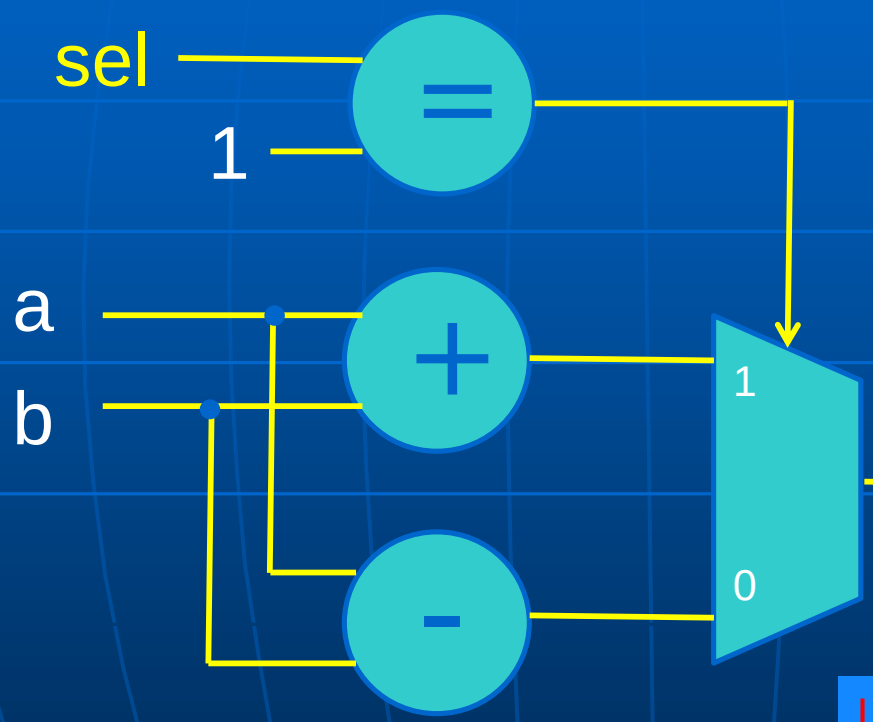
Diseño de un sumador ripple-carry de 4 bits

```
1  ENTITY sumador2 IS
2     PORT ( a,b : IN BIT_VECTOR (4 DOWNTO 1);
3            sum : OUT BIT_VECTOR(4 DOWNTO 1);
4            cout  : OUT BIT );
5
6  END ENTITY sumador2;
7
8  ARCHITECTURE sum_con_gen OF sumador2 IS
9
10 SIGNAL c: BIT_VECTOR (5 DOWNTO 1);
11
12    BEGIN
13       c(1) <= '0';
14       adders: FOR i IN 1 TO 4 GENERATE
15          sum(i) <= a(i) XOR b(i) XOR c(i);
16          c(i+1) <= (a(i) AND b(i)) OR (a(i) AND b(i))
17          OR (b(i) AND c(i));
18       END GENERATE;
19
20       cout <= c(5);
21
22 END ARCHITECTURE sum_con_gen;
```

Generate es una sentencia concurrente empleada usualmente para describir estructuras que tienen un patrón repetitivo en su diseño.

Resultado posible de una descripción antes de PLACE and ROUTE

```
sum <= a + b when sel = '1' else a - b;
```



Posible circuito basado en una pseudosíntesis con bloques genéricos

Una simulación a este nivel sólo dará información de si el circuito cumple con lo especificado funcionalmente

Luego del PLACE and ROUTE

EN BASE A LOS RECURSOS QUE TENGA LA FPGA, A LA INTELIGENCIA DEL COMPILADOR Y EVENTUALES DIRECTIVAS DEL DISEÑADOR, SERÁ EL HARDWARE FINALMENTE IMPLEMENTADO. Aquí SI se tendrá información confiable del TIMING.

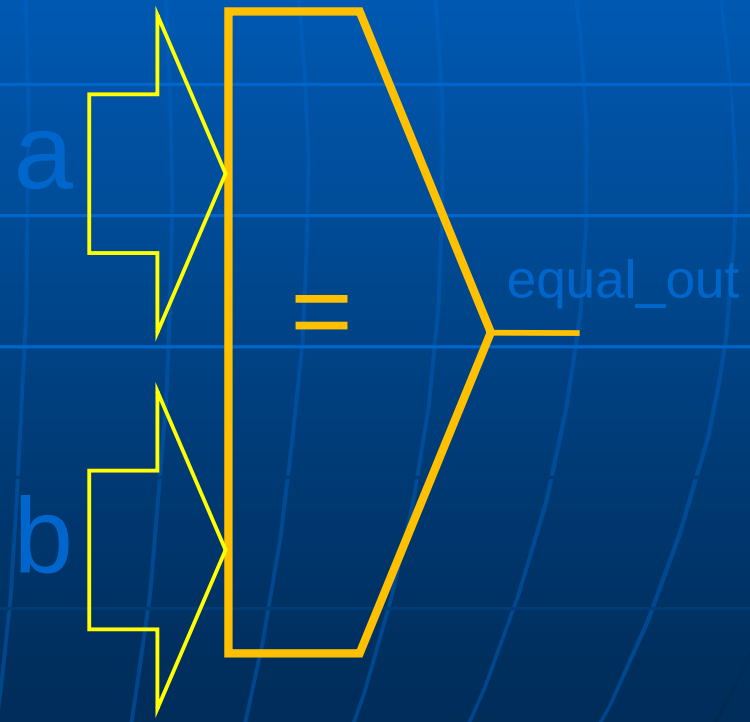
PROCESS: USO DE SENTENCIA FOR … LOOP

EJEMPLO: COMPARADOR DE IGUALDAD BIT A BIT

Circuito Combinatorio

```
library ieee;
use ieee.std_logic_1164.all;
entity match_bits is
port (a, b : in std_logic_vector(7 downto 0);
equal_out : out std_logic_vector(7 downto 0));
end;
```

```
architecture behaviour of match_bits is
   begin
     process (a, b)
       begin
          for i in 7 downto 0 loop
            equal_out(i) <= a(i) xnor b(i);
          end loop;
     end process;
end;
```



Otra manera de expresar el lazo:

```
for i in integer range 7 downto 0 loop
```

PROCESS: USO DE SENTENCIA FOR … LOOP

EJEMPLO: CONTADOR DE UNOS.

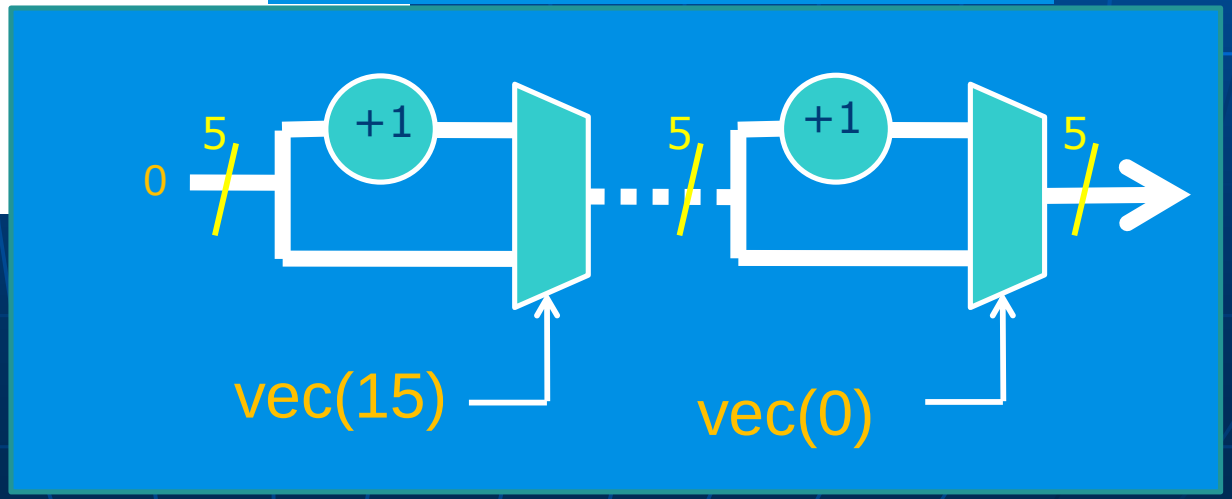
Circuito Combinatorio

contador_de_unos.vhd* | Compilation Report

```
1  library ieee;
2  use ieee.std_logic_1164.all, ieee.numeric_std.all;
3  entity contador_de_unos is
4    port (vec : in std_logic_vector(15 downto 0);
5          count : out unsigned(4 downto 0));
6  end;
7  architecture behaviour of contador_de_unos is
8    begin
9      process (vec)
10       variable result : unsigned(4 downto 0);
11       begin
12          result := "00000";
13          for i in 15 downto 0 loop
14             if vec(i) = '1' then
15                result := result + 1;
16             else result := result;
17             end if;
18          end loop;
19       count <= result;
20     end process;
21 end;
22
```

Inicialización de VARIABLE

POSIBLE SÍNTESIS DEL CIRCUITO



PROCESS: USO DE SENTENCIA FOR … LOOP

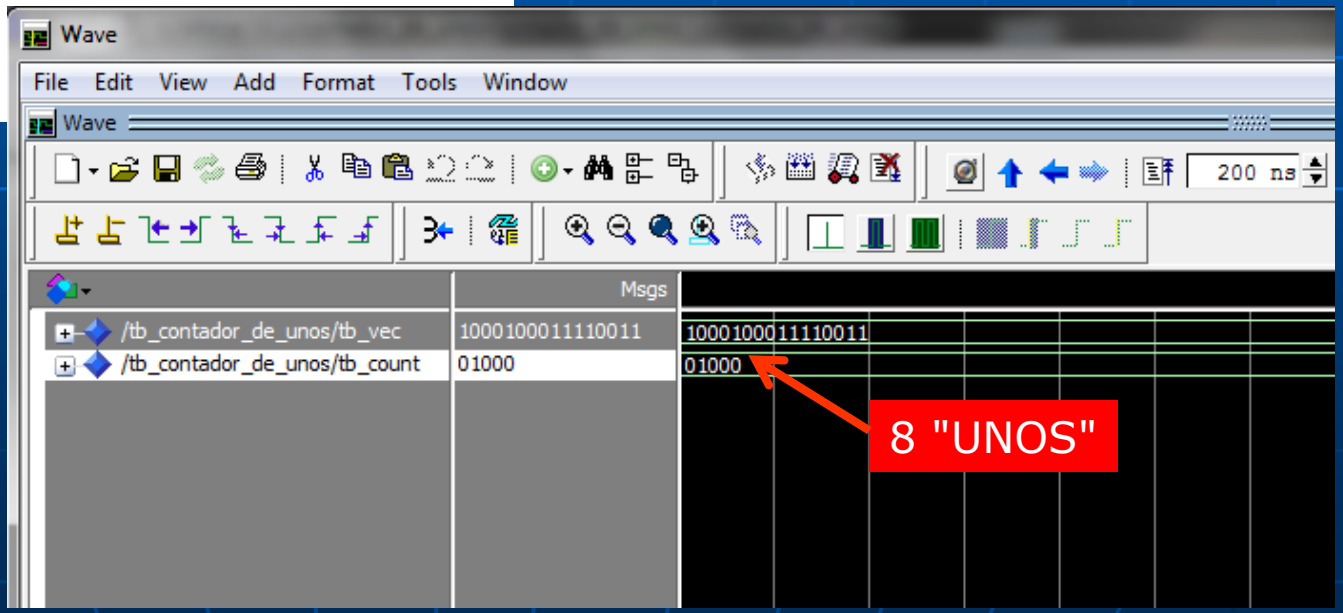
EJEMPLO: CONTADOR DE UNOS.

contador_de_unos.vhd* | Compilation Report | tb_contador_de_unos.vhd

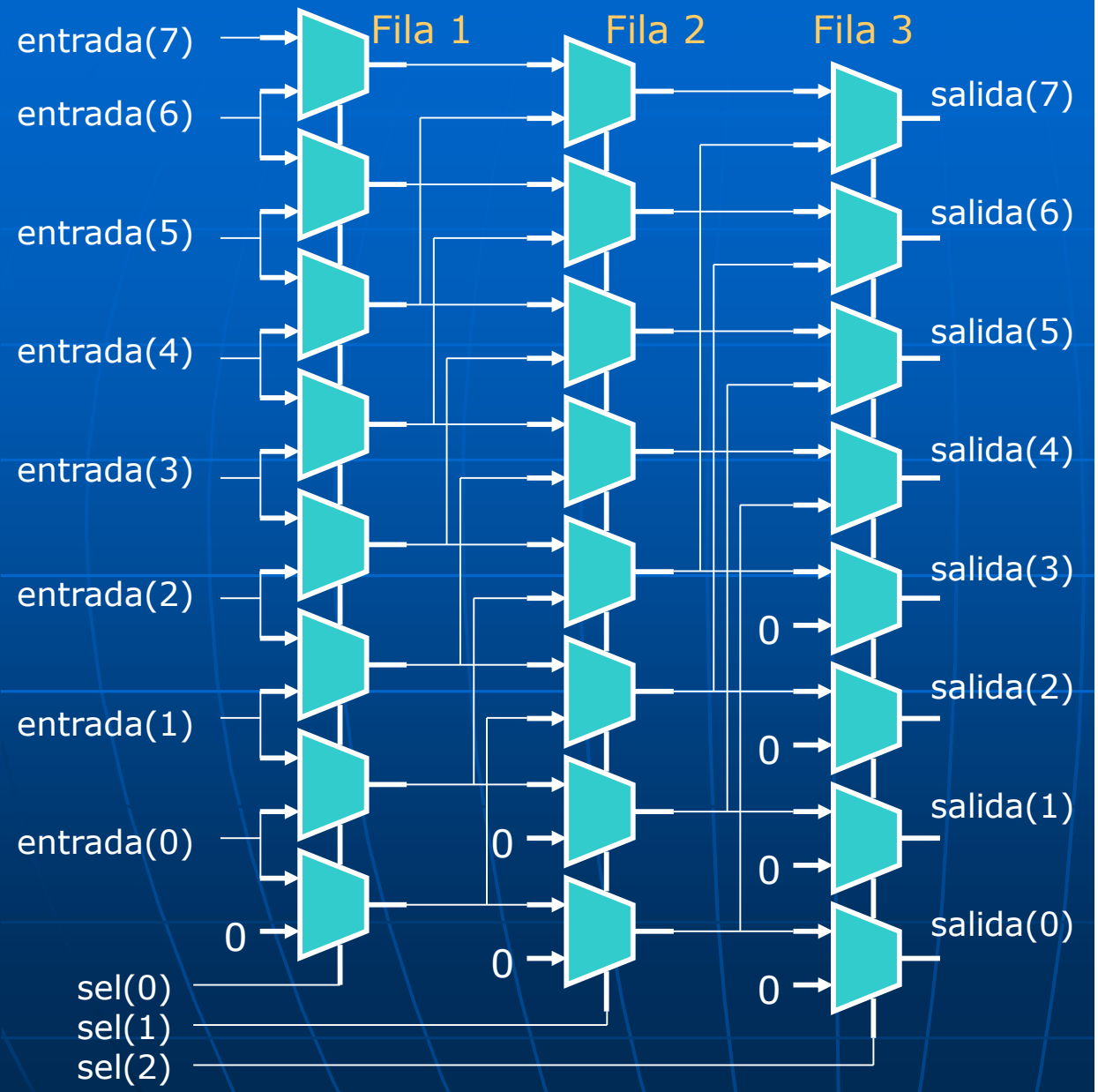
```
1  library ieee;
2  use ieee.std_logic_1164.all, ieee.numeric_std.all;
3
4  entity tb_contador_de_unos is
5  end tb_contador_de_unos;
6
7  architecture test of tb_contador_de_unos is
8
9     component contador_de_unos
10       Port (    vec : in std_logic_vector(15 downto 0);
11                 count : out unsigned(4 downto 0));
12    end component;
13
14 signal tb_vec       : std_logic_vector(15 downto 0);
15 signal tb_count     : unsigned(4 downto 0);
16
17 begin
18
19   uut: contador_de_unos port map (  tb_vec, tb_count);
20
21    estimulos : process
22    begin
23                  tb_vec <= "1000100011110011";
24
25                  wait for 200 ns;
26
27               end process estimulos;
28
29   end test;
```

TEST-BENCH

VALOR INICIAL DE LA ENTRADA PARA CORRER LA SIMULACIÓN



Diseño de Barrel Shifters

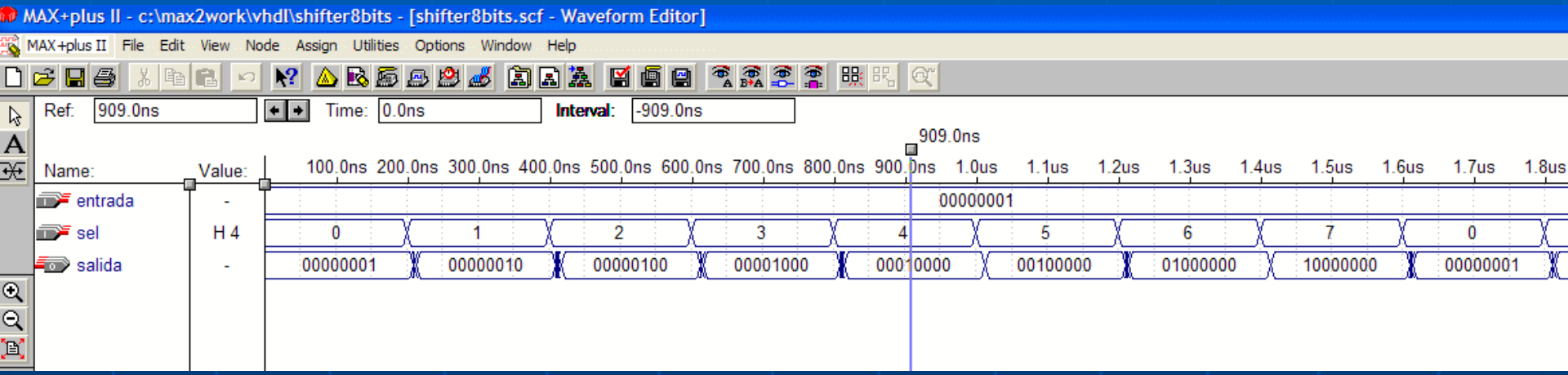



```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY shifter8bits IS
    PORT(
        entrada: IN STD_LOGIC_VECTOR (7 downto 0);
        sel: IN STD_LOGIC_VECTOR (2 downto 0);
        salida: OUT STD_LOGIC_VECTOR (7 downto 0));
END shifter8bits;

ARCHITECTURE barrel OF shifter8bits IS
BEGIN
PROCESS (entrada, sel)
    VARIABLE temp1: STD_LOGIC_VECTOR(7 downto 0);
    VARIABLE temp2: STD_LOGIC_VECTOR(7 downto 0);
BEGIN
--Primera fila
IF (sel(0)='0') THEN
    temp1 := entrada;
ELSE
    temp1(0) := '0';
    FOR i IN 1 TO entrada'HIGH LOOP
        temp1(i) := entrada(i-1);
    END LOOP;
END IF;
--Segunda fila
IF (sel(1)='0') THEN
    temp2 := temp1;
ELSE
    FOR i IN 0 TO 1 LOOP
        temp2(i) := '0';
    END LOOP;
    FOR i IN 2 TO entrada'HIGH LOOP
        temp2(i) := temp1(i-2);
    END LOOP;
END IF;
--Tercera fila
IF (sel(2)='0') THEN
    salida <= temp2;
ELSE
    FOR i IN 0 TO 3 LOOP
        salida(i) <= '0';
    END LOOP;
    FOR i IN 4 TO entrada'HIGH LOOP
        salida(i) <= temp2(i-4);
    END LOOP;
END IF;
END PROCESS;
END barrel;
```

Diseño de Barrel Shifters

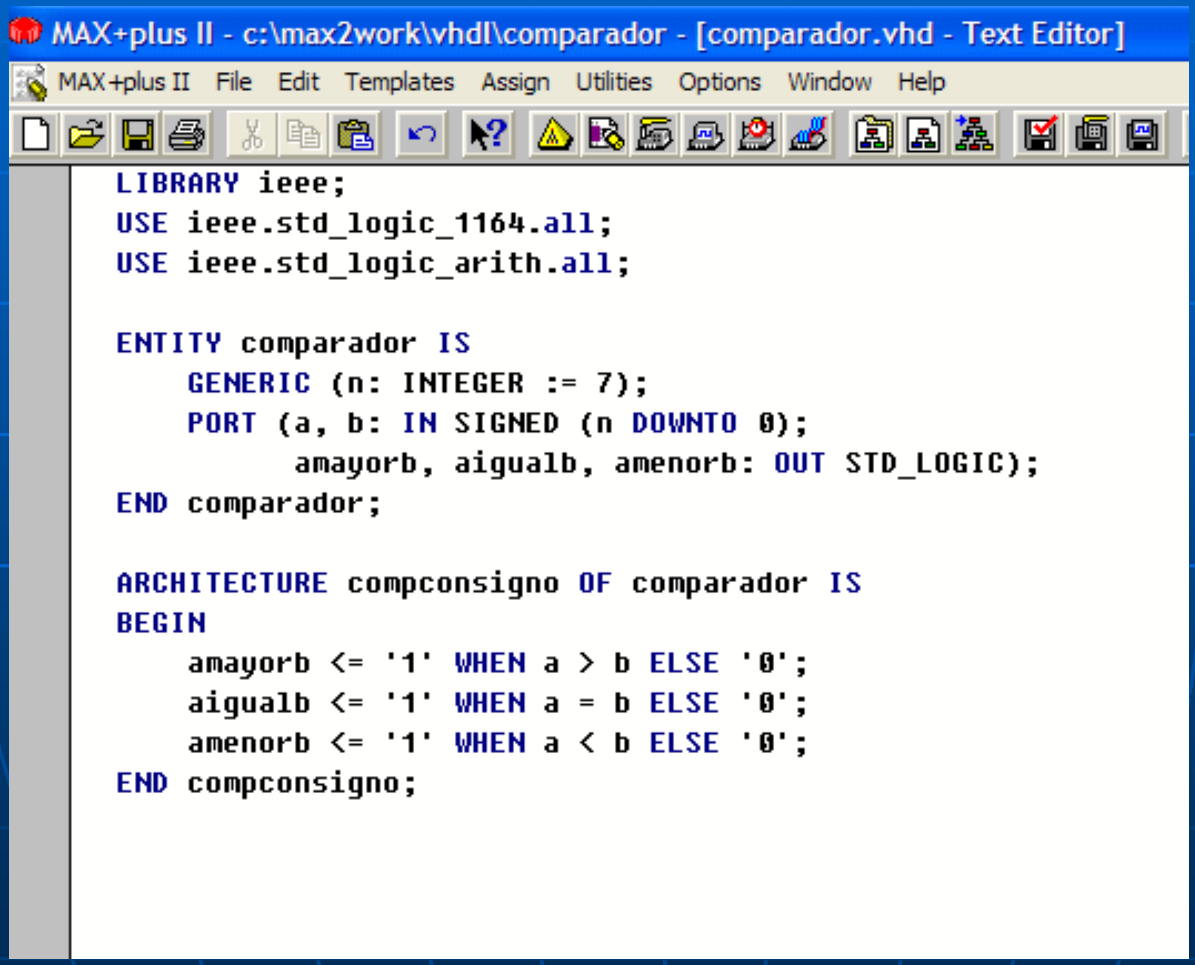
Ejemplo: Diseño de Shifter aritmético de 8 bits a izquierda.

Simulación



Diseño de comparadores

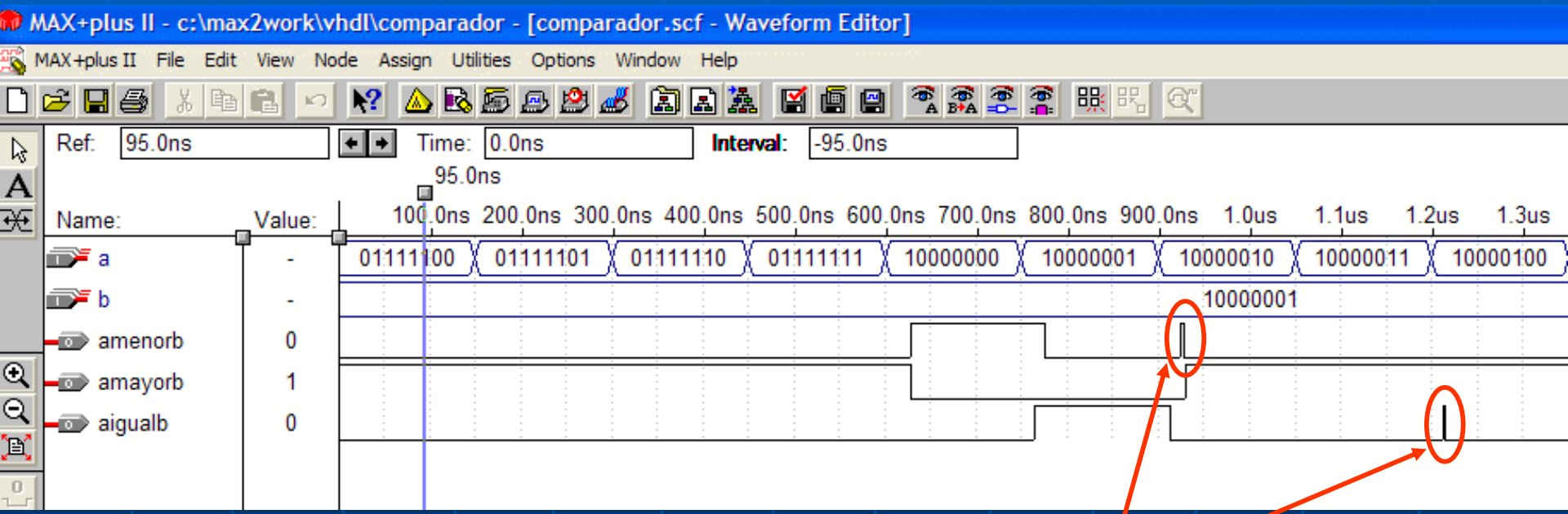
Ejemplo: Diseño de comparador binario con signo en punto fijo de dos números “a” y “b” de 8 bits de longitud de palabra



Diseño de comparadores

Ejemplo: Diseño de comparador binario con signo en punto fijo de dos números “a” y “b” de 8 bits de longitud de palabra

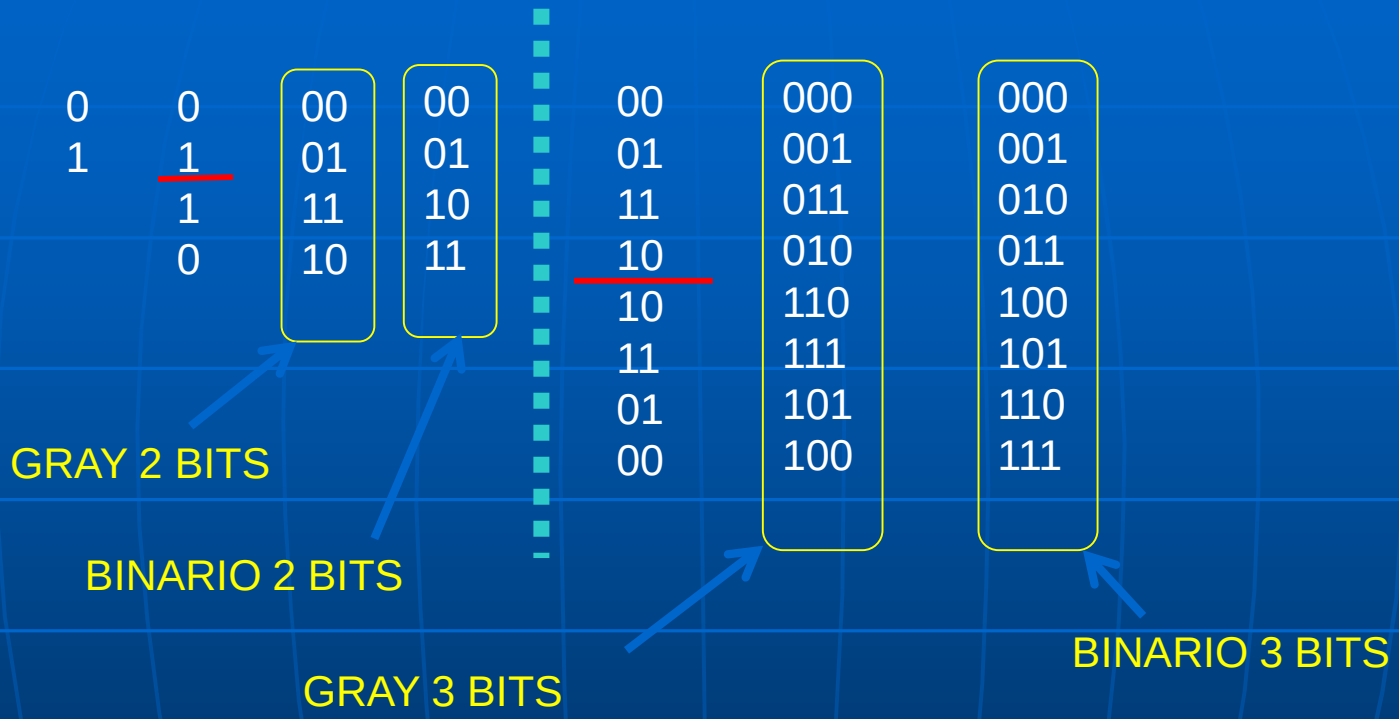
Simulación



Porqué los glitches...???

Cómo se soluciona...???

CONVERSIÓN BINARIO A GRAY



CONVERSIÓN BINARIO A GRAY EN 3 BITS

CONCEPTO DE MEMORIA ROM (sólo lectura)

bin_a_gray_3bits.vhd | tb_bin_a_gray_3bits.vhd

```
1  --Conversor binario a gray en formato paralelo
2  --Noriega ISLD 2016
3  library ieee;
4  use ieee.std_logic_1164.all;
5  use ieee.numeric_std.all;
6  entity bin_a_gray_3bits is
7  port (bin      : in unsigned(2 downto 0);
8        gray     : out std_logic_vector(2 downto 0));
9  end;
10
11 architecture behaviour of bin_a_gray_3bits is
12 type memory_type is array (0 to 7) of
13 std_logic_vector(2 downto 0);
14
15 constant memory : memory_type :=
16 ("000", "001", "011", "010", "110", "111", "101", "100");
17
18 begin
19      gray <= memory(to_integer(bin));
20 end;
21
```

BIN => GRAY

BIN	GRAY
000	000
001	001
010	011
011	010
100	110
101	111
110	101
111	100

Función «to_integer»
Se necesita pasar a INTEGER para indexar en un ARRAY

Desde el punto de vista funcional, ‘memory’ es una memoria de sólo lectura (ROM) la cual se direcciona por ‘bin’ para leer la información de uno de los 8 datos almacenados en ella.

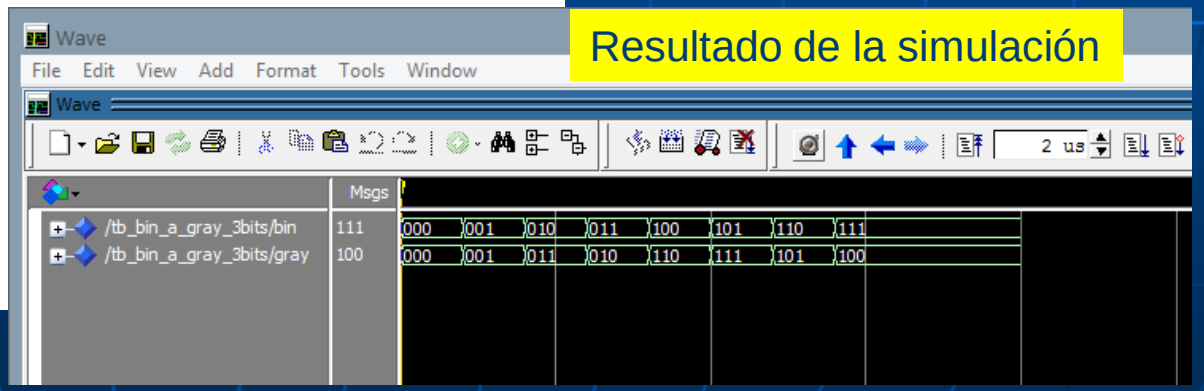
CONVERSIÓN BINARIO A GRAY EN 3 BITS

bin_a_gray_3bits.vhd | tb_bin_a_gray_3bits.vhd

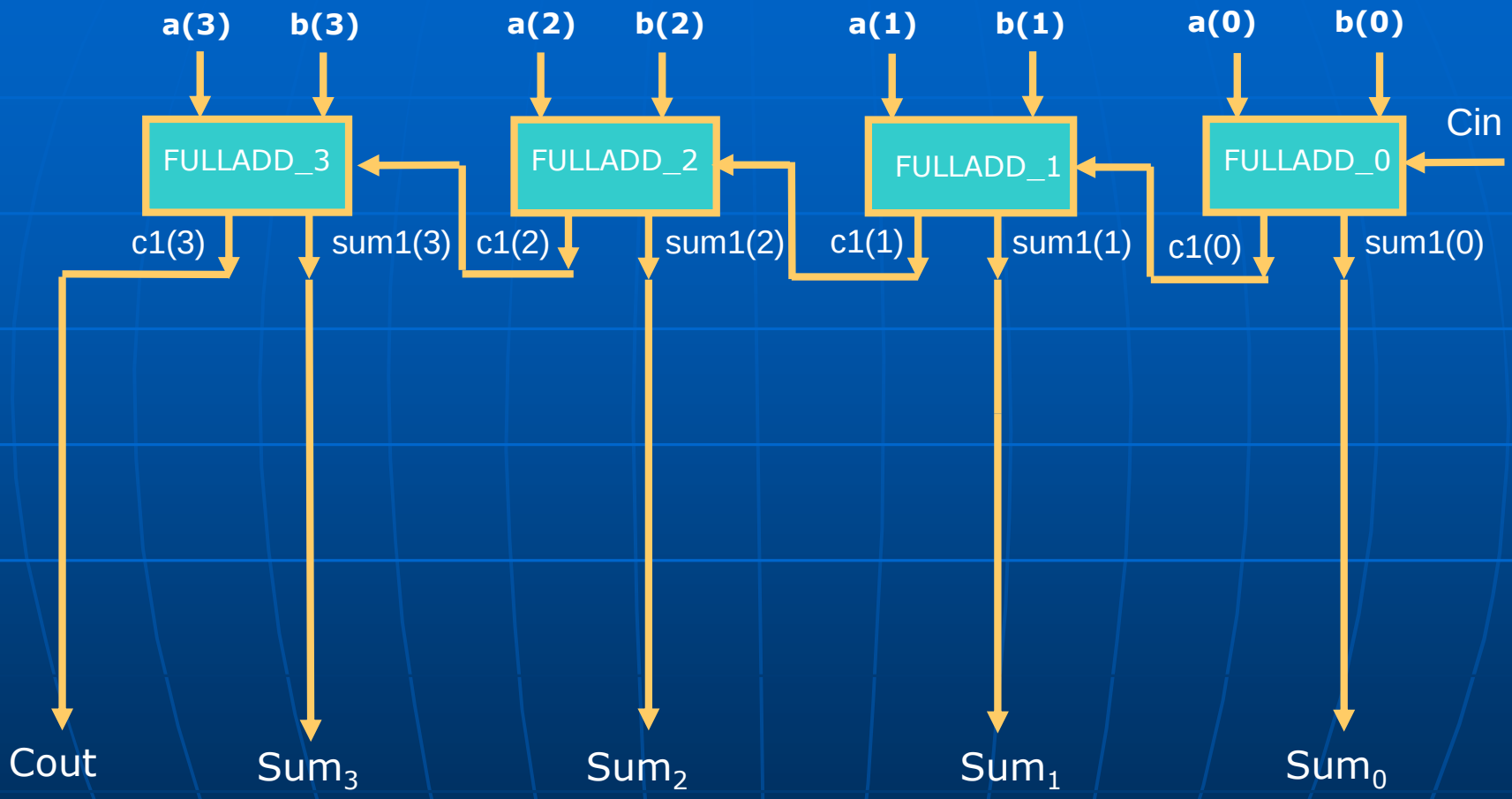
Test bench en VHDL

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  use ieee.numeric_std.all;
4
5  ENTITY tb_bin_a_gray_3bits IS
6  END tb_bin_a_gray_3bits;
7
8  ARCHITECTURE behavior OF tb_bin_a_gray_3bits IS
9
10    COMPONENT bin_a_gray_3bits is
11       port (bin     : in unsigned(2 downto 0);
12             gray    : out std_logic_vector(2 downto 0)
13             );
14    end COMPONENT;
15
16    signal bin    : unsigned(2 downto 0);
17    signal gray   : std_logic_vector(2 downto 0);
18
19 BEGIN
20
21    uut: bin_a_gray_3bits PORT MAP (bin, gray);
22
23    estimulo_proc: process
24       begin
25          for i in 0 to 7 loop
26             bin <= to_unsigned (i,3);
27             wait for 200 ns;
28          end loop;
29          wait;
30    end process;
31 END;
```

Resultado de la simulación



Diseño de un sumador tipo RIPPLE-CARRY de 2 operandos de 4bits



Diseño de un sumador tipo RIPPLE-CARRY de 2 operandos de 4bits

SUMADOR FULL-ADDER DE UN BIT

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity full_adder_1bit is
5       port (  x    : in std_logic;
6               y    : in std_logic;
7               ci   : in std_logic;
8               s    : out std_logic;
9               co   : out std_logic
10          );
11  end full_adder_1bit;
12
13  architecture behavior of full_adder_1bit is
14
15  begin
16
17  s  <= x xor y xor ci;
18  co <= (x and y) or (ci and (x xor y));
19
20  end;
```

Diseño de un sumador tipo RIPPLE-CARRY de 2 operandos de 4bits

Descripción en VHDL

```vhdl
--Sumador de 2 operandos de 4 bits sin signo.
--Tipo: Ripple-Carry en formato Paralelo.
--Se emplea el metodo de descripcion ESTRUCTURAL para describir el
--sumador Ripple-Carry formado por sumadores completos de un bit.
--Se emplea descripcion de COMPORTAMIENTO para describir al sumador
--de un bit.
--SERGIO NORIEGA - ISLD - 2016
--Nombre del archivo TOP LEVEL: sum_ripple_carry_4bits.
--Este archivo se emplea junto con el denominado "full_adder_1bit"
--el cual describe un sumador completo de 1 bit y en este proyecto
--sera invocado 4 veces.


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity sum_ripple_carry_4bits is
    Port ( a    : in  unsigned (3 downto 0);
           b    : in  unsigned (3 downto 0);
           cin  : in std_logic;
           cout : out std_logic;
           sum  : out unsigned (3 downto 0)
              );
end sum_ripple_carry_4bits;

architecture Behavioral of sum_ripple_carry_4bits is

component full_adder_1bit is
    port (  x    : in std_logic;
            y    : in std_logic;
            ci   : in std_logic;
            s    : out std_logic;
            co   : out std_logic
         );
end component;

signal c1,sum1,c2 : unsigned (3 downto 0) := (others => '0');

begin

fulladd_0 : full_adder_1bit port map(a(0),b(0),cin,sum1(0),c1(0));
fulladd_1 : full_adder_1bit port map(a(1),b(1),c1(0),sum1(1),c1(1));
fulladd_2 : full_adder_1bit port map(a(2),b(2),c1(1),sum1(2),c1(2));
fulladd_3 : full_adder_1bit port map(a(3),b(3),c1(2),sum1(3),c1(3));

sum(0) <= sum1(0);
sum(1) <= sum1(1);
sum(2) <= sum1(2);
sum(3) <= sum1(3);
cout <= c1(3);


end Behavioral;
```

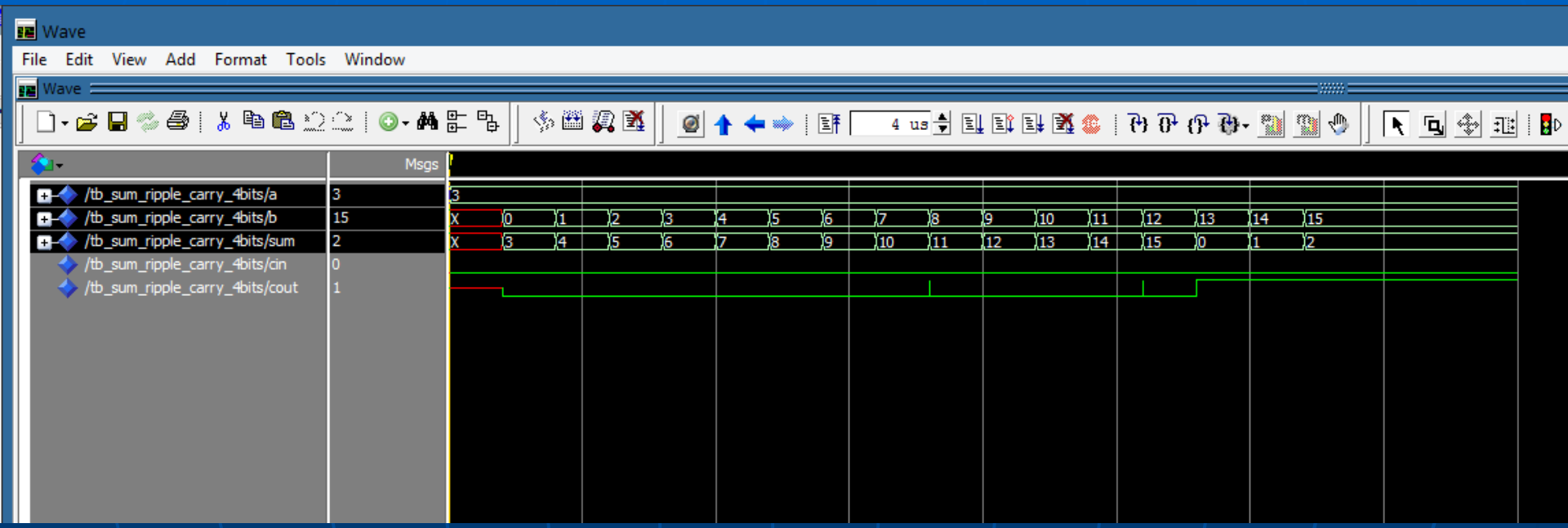
Diseño de un sumador tipo RIPPLE-CARRY de 2 operandos de 4bits

Test bench en VHDL

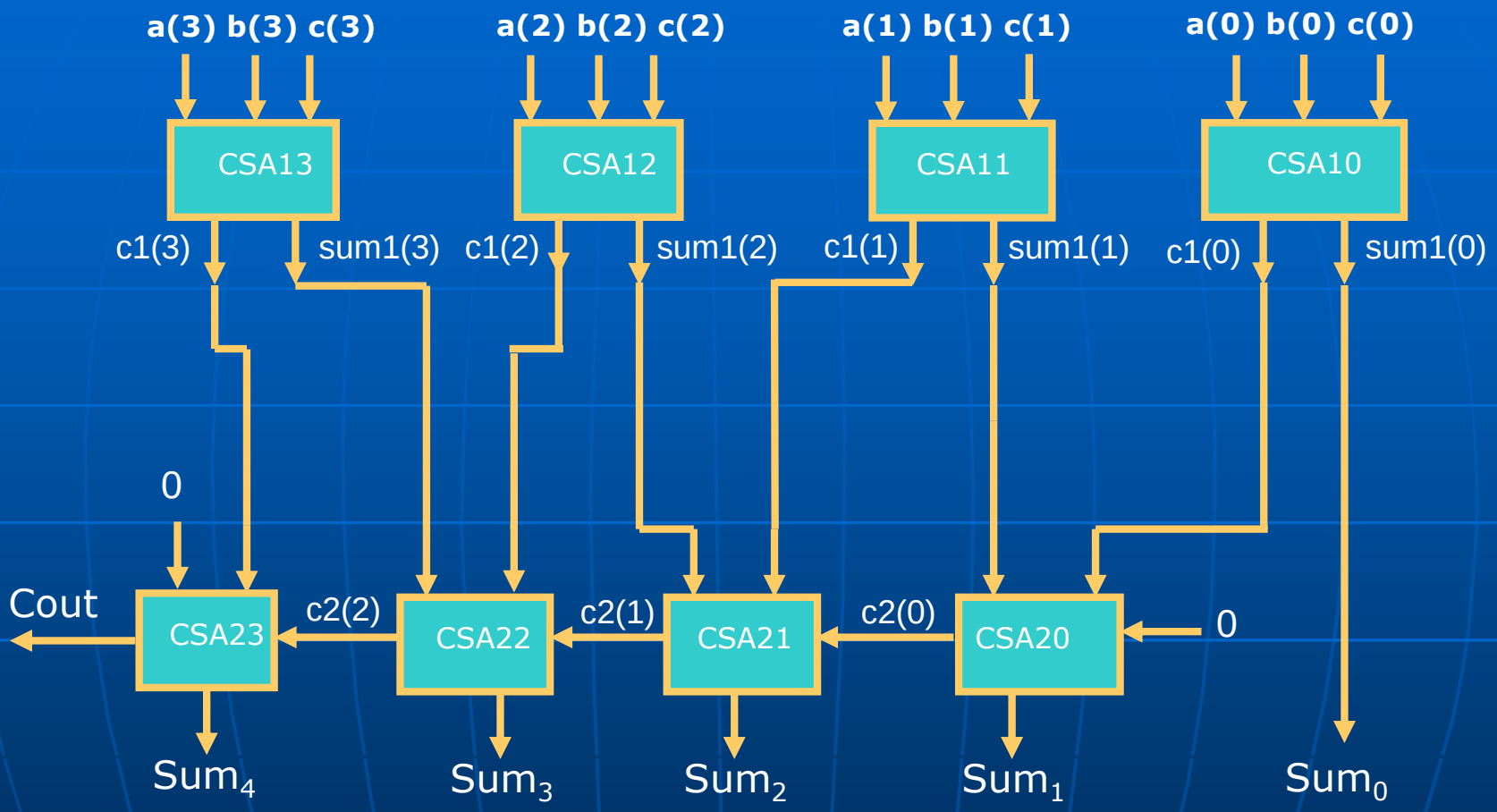
```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  use ieee.numeric_std.all;
4
5  ENTITY tb_sum_ripple_carry_4bits IS
6  END tb_sum_ripple_carry_4bits;
7
8  ARCHITECTURE behavior OF tb_sum_ripple_carry_4bits IS
9
10     COMPONENT sum_ripple_carry_4bits is
11     Port ( a    : in  unsigned (3 downto 0);
12            b    : in  unsigned (3 downto 0);
13            cin  : in std_logic;
14            cout : out std_logic;
15            sum  : out unsigned (3 downto 0)
16               );
17         end COMPONENT;
18
19     signal a, b : unsigned(3 downto 0);
20     signal sum  : unsigned(3 downto 0) := (others => '0');
21     signal cin  : std_logic;
22     signal cout : std_logic := '0';
23
24 BEGIN
25
26    uut: sum_ripple_carry_4bits PORT MAP (a, b, cin, cout, sum);
27
28    estimulo_proc: process
29       begin
30          cin <= '0';
31          a <= to_unsigned (3,4);
32          for i in 0 to 15 loop
33             wait for 200 ns;
34             b <= to_unsigned(i,4);
35          end loop;
36          wait;
37    end process;
38
39 END;
```

Diseño de un sumador tipo RIPPLE-CARRY de 2 operandos de 4bits

Resultado de la simulación



Diseño de un sumador tipo CARRY-SAVE de 3 operandos de 4bits



Diseño de un sumador de 3 operandos tipo carry-save

Descripción en VHDL

sum_carry_save_4bits.vhd | tb_sum_carry_save_4bits.vhd

```vhdl
--Sumador de 3 operandos de 4 bits sin signo.
--Tipo: Carry-Save en formato Paralelo.
--Se emplea el metodo de descripcion ESTRUCTURAL para describir el
--sumador Carry-Save formado por sumadores completos de un bit.
--Se emplea descripcion de COMPORTAMIENTO para describir al sumador
--de un bit.
--SERGIO NORIEGA - ISLD - 2016
--Nombre del archivo TOP LEVEL: sum_carry_save_4bits.
--Este archivo se emplea junto con el denominado "full_adder_1bit"
--el cual describe un sumador completo de 1 bit y en este proyecto
--sera invocado 8 veces.


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity sum_carry_save_4bits is
    Port ( a    : in  unsigned (3 downto 0);
           b    : in  unsigned (3 downto 0);
           c    : in  unsigned (3 downto 0);
           cout : out std_logic;
           sum  : out unsigned (4 downto 0)
             );
end sum_carry_save_4bits;

architecture Behavioral of sum_carry_save_4bits is

component full_adder_1bit is
    port (  x    : in std_logic;
            y    : in std_logic;
            ci   : in std_logic;
            s    : out std_logic;
            co   : out std_logic
         );
end component;

signal c1,sum1,c2 : unsigned (3 downto 0) := (others => '0');

begin

csa_10 : full_adder_1bit port map(a(0),b(0),c(0),sum1(0),c1(0));
csa_11 : full_adder_1bit port map(a(1),b(1),c(1),sum1(1),c1(1));
csa_12 : full_adder_1bit port map(a(2),b(2),c(2),sum1(2),c1(2));
csa_13 : full_adder_1bit port map(a(3),b(3),c(3),sum1(3),c1(3));

csa_20 : full_adder_1bit port map(sum1(1),c1(0),'0',sum(1),c2(1));
csa_21 : full_adder_1bit port map(sum1(2),c1(1),c2(1),sum(2),c2(2));
csa_22 : full_adder_1bit port map(sum1(3),c1(2),c2(2),sum(3),c2(3));
csa_23 : full_adder_1bit port map('0',c1(3),c2(3),sum(4),cout);

sum(0) <= sum1(0);

end Behavioral;
```

Diseño de un sumador de 3 operandos tipo carry-save

SUMADOR FULL-ADDER DE UN BIT

```
1   library IEEE;
2   use IEEE.STD_LOGIC_1164.ALL;
3
4   entity full_adder_1bit is
5       port (  x    : in std_logic;
6               y    : in std_logic;
7               ci   : in std_logic;
8               s    : out std_logic;
9               co   : out std_logic
10          );
11  end full_adder_1bit;
12
13  architecture behavior of full_adder_1bit is
14
15  begin
16
17  s  <= x xor y xor ci;
18  co <= (x and y) or (ci and (x xor y));
19
20  end;
```

Diseño de un sumador de 3 operandos tipo carry-save

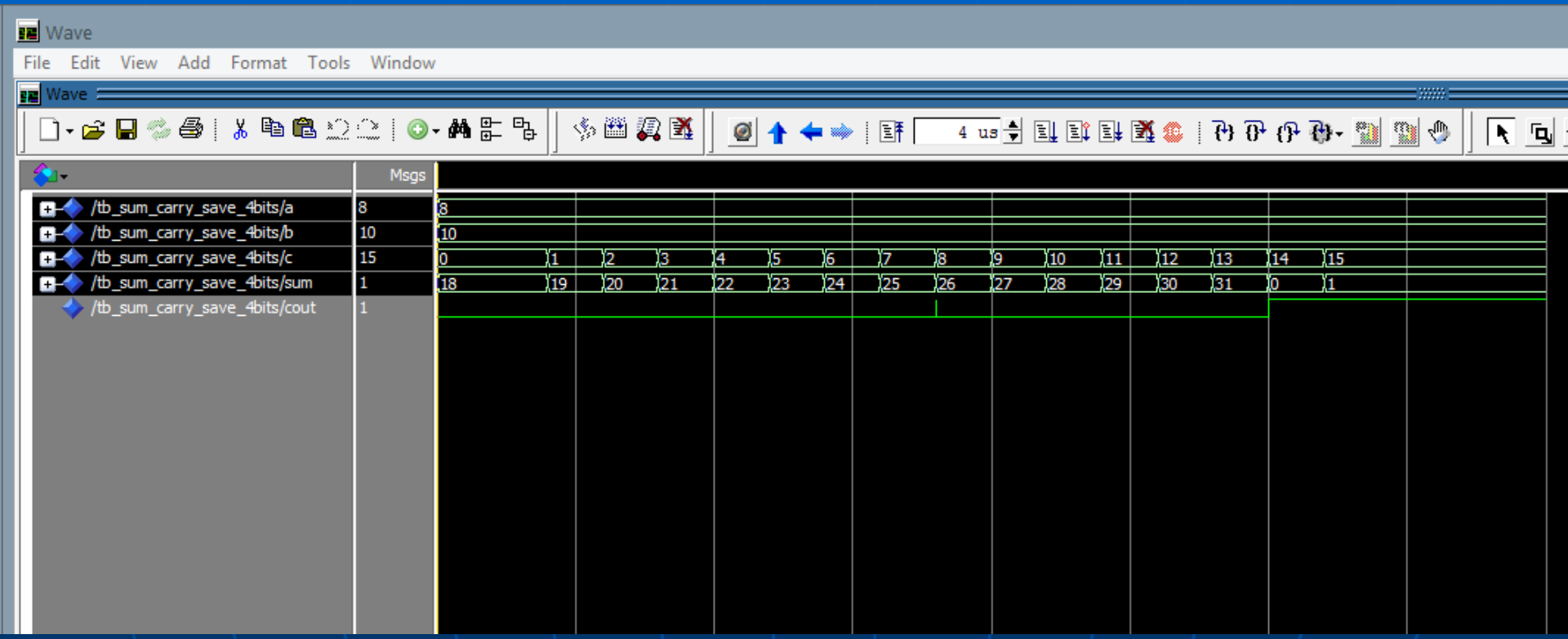
Test bench en VHDL

sum_carry_save_4bits.vhd | tb_sum_carry_save_4bits.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  use ieee.numeric_std.all;
4
5  ENTITY tb_sum_carry_save_4bits IS
6  END tb_sum_carry_save_4bits;
7
8  ARCHITECTURE behavior OF tb_sum_carry_save_4bits IS
9
10     COMPONENT sum_carry_save_4bits is
11     Port ( a    : in  unsigned (3 downto 0);
12            b    : in  unsigned (3 downto 0);
13            c    : in  unsigned (3 downto 0);
14            cout : out std_logic;
15            sum  : out unsigned (4 downto 0)
16               );
17         end COMPONENT;
18
19     signal a, b : unsigned(3 downto 0);
20     signal c    : unsigned(3 downto 0) := (others => '0');
21     signal sum  : unsigned(4 downto 0) := (others => '0');
22     signal cout : std_logic := '0';
23
24 BEGIN
25
26    uut: sum_carry_save_4bits PORT MAP (a, b, c, cout, sum);
27
28    estimulo_proc: process
29       begin
30          a <= to_unsigned (8,4);
31          b <= to_unsigned (10,4);
32          for i in 0 to 15 loop
33             wait for 200 ns;
34             c <= to_unsigned(i,4);
35          end loop;
36          wait;
37    end process;
38
39 END;
```

Diseño de un sumador de 3 operandos tipo carry-save

Resultado de la simulación



Diseño de un complementador a 2 (CA2) tipo serie

Descripción en VHDL

El chequear el valor de din después de procesar a data_out dentro del proceso, hace que se modifique data_out recién en el próximo ciclo de reloj, condición necesaria para ejecutar el algoritmo de complemento.

complementador_a2_serie.vhd | Compilation Report

```
1  library ieee;
2  use ieee.std_logic_1164.all;
3
4  entity complementador_a2_serie is
5     Port ( clock   : in std_logic;
6            reset   : in std_logic;
7            din     : in std_logic_vector(7 downto 0);
8            dout    : out std_logic_vector(7 downto 0)
9          );
10 end complementador_a2_serie;
11
12 architecture Comportamiento of complementador_a2_serie is
13
14    signal data_out, data_in  : std_logic_vector (7 downto 0);
15    signal lock               : integer range 0 to 1:= 0;
16    signal var                : integer range 0 to 8:= 0;
17
18    begin
19
20       data_in <= din;
21
22       process (reset, clock)
23          begin
24             if reset = '1' then
25             var <= 0;
26             lock <= 0;
27             elsif (clock'event and clock = '1') then
28                if var < 8 then
29                   if lock = 0 then data_out(var) <= din(var);
30                   else data_out(var) <= not din(var);
31                   end if;
32                   if din(var) = '1' then lock <= 1;
33                   end if;
34                   var <= var + 1;
35                end if;
36             end if;
37       end process;
38                   dout <= data_out;
39
40    end Comportamiento;
```

Diseño de un complementador a 2 (CA2) tipo serie

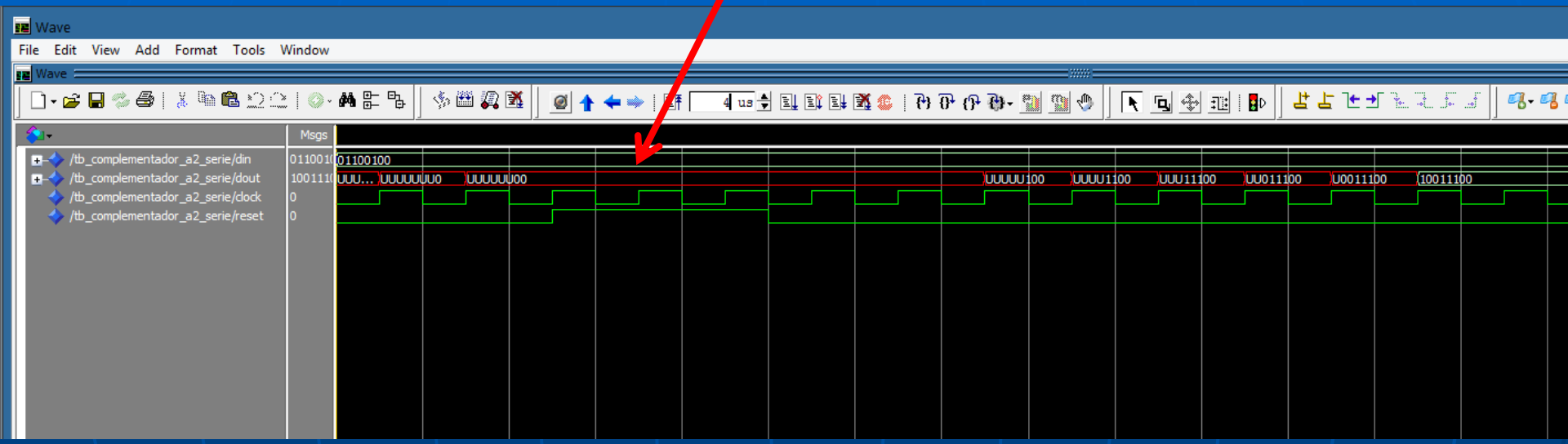
Test bench en VHDL

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.all;
3  use IEEE.STD_LOGIC_ARITH.ALL;
4  use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6  entity tb_complementador_a2_serie is
7  end tb_complementador_a2_serie;
8
9  architecture test of tb_complementador_a2_serie is
10
11
12    component complementador_a2_serie
13       Port (   clock        : in std_logic;
14                reset        : in std_logic;
15                din    :in std_logic_vector (7 downto 0);
16                dout   : out std_logic_vector (7 downto 0)
17
18            );
19    end component;
20
21   signal din, dout        : STD_LOGIC_VECTOR( 7 downto 0);
22   signal clock, reset     : STD_LOGIC;
23
24
25 begin
26
27   uut: complementador_a2_serie port map    ( clock => clock, reset => reset,
28                                               din => din, dout => dout
29                                 );
30
31    gen_reloj : process  -- Reloj de 200 ns de periodo y 50% de ciclo de trabajo
32       begin
33          clock <= '0';
34          wait for 100 ns;
35          clock <= '1';
36          wait for 100 ns;
37    end process gen_reloj;
38
39    estimulos : process
40
41    begin
42             din <= "01100100";
43             reset    <= '0';
44             wait for 500 ns;
45             reset    <= '1';
46             wait for 500 ns;
47             reset    <= '0';
48             wait for 4 us;
49
50   end process estimulos;
51
52 end test;
```

Diseño de un complementador a 2 (CA2) tipo serie

Resultado de la simulación

"U" indica "no definido"



Diseño de un sumador tipo RIPPLE-CARRY de 2 operandos SERIE

Descripción en VHDL

sum_ripple_carry_serie.vhd | Compilation Report | tb_sum_ripple_carry_serie.vhd

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

entity sum_ripple_carry_serie is
   Port ( clock   : in std_logic;
          reset   : in std_logic;
          a       : in std_logic_vector (7 downto 0);
          b       : std_logic_vector(7 downto 0);
          cin     : in std_logic;
          sum     : out std_logic_vector(7 downto 0);
          cout    : out std_logic
        );
end sum_ripple_carry_serie;

architecture Comportamiento of sum_ripple_carry_serie is

   signal suma, a_in, b_in : std_logic_vector(7 downto 0);
   signal cin_in, carry    : std_logic;

   begin

      a_in <= a;
      b_in <= b;
      cin_in <= cin;

      process (reset, clock)
         variable i      : integer range 0 to 8:= 0;
         begin
            if reset = '1' then
            i := 0;
            suma <= x"00";
            carry <= cin_in;
            elsif (clock'event and clock = '1') then
               if i < 8 then
                     suma(i) <= a_in(i) XOR b_in(i)XOR carry;
                     carry <= (a_in(i) AND b_in(i)) OR (carry AND (a_in(i) OR b_in(i)));
                     i := i + 1;
               end if;
            end if;
      end process;
      cout <= carry;
      sum <= suma;

   end Comportamiento;
```

Diseño de un sumador tipo RIPPLE-CARRY de 2 operandos SERIE

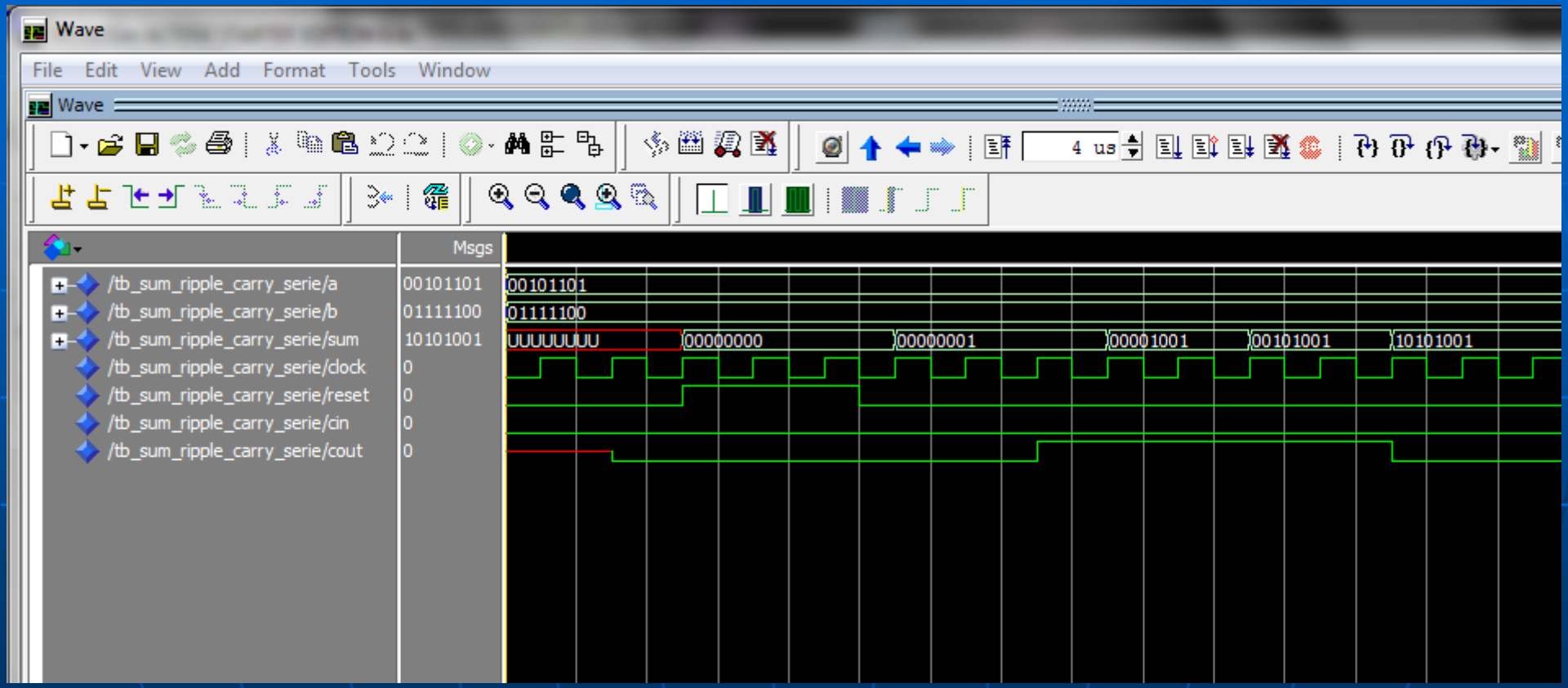
Test bench en VHDL

sum_ripple_carry_serie.vhd | Compilation Report | tb_sum_ripple_carry_serie.vhd

```
 6  entity tb_sum_ripple_carry_serie is
 7  end tb_sum_ripple_carry_serie;
 8
 9  architecture test of tb_sum_ripple_carry_serie is
10
11
12    component sum_ripple_carry_serie
13       Port (   clock : in std_logic;
14                reset : in std_logic;
15                a     : in std_logic_vector (7 downto 0);
16                b     : in std_logic_vector (7 downto 0);
17                cin   : in std_logic;
18                sum   : out std_logic_vector (7 downto 0);
19                cout  : out std_logic
20             );
21    end component;
22
23    signal a, b, sum                : std_logic_vector( 7 downto 0);
24    signal clock, reset, cin, cout  : std_logic;
25
26
27  begin
28
29    uut: sum_ripple_carry_serie port map     ( clock => clock, reset => reset,
30                                               a => a, b => b, cin => cin,
31                                               sum =>  sum, cout => cout
32                                    );
33
34    gen_reloj : process  -- Reloj de 200 ns de periodo y 50% de ciclo de trabajo
35       begin
36          clock <= '0';
37          wait for 100 ns;
38          clock <= '1';
39          wait for 100 ns;
40    end process gen_reloj;
41
42    estimulos : process
43
44    begin
45             cin <= '0';
46             a <= "00101101";
47             b <= "01111100";
48             reset    <= '0';
49             wait for 500 ns;
50             reset    <= '1';
51             wait for 500 ns;
52             reset    <= '0';
53             wait for 4 us;
54
55    end process estimulos;
56
57  end test;
```

Diseño de un sumador tipo RIPPLE-CARRY de 2 operandos SERIE

Resultado de la simulación



Diseño de un multiplicador sin signo paralelo con «template»

Descripción en VHDL

Usando el template se obtiene una plantilla de inicio, la cual puede ser modificada a conveniencia.

En este caso se cargó un multiplicador sin signo con longitud de palabra genérica ajustada inicialmente a 8 bits de entrada de datos.

multiplicador1.vhd | tb_multiplicador1.vhd

```
1  -- Quartus II VHDL Template
2  -- Unsigned Multiply
3
4  library ieee;
5  use ieee.std_logic_1164.all;
6  use ieee.numeric_std.all;
7
8  entity multiplicador1 is
9
10    generic
11    (
12       DATA_WIDTH : natural := 8
13    );
14
15    port
16    (
17       a        : in unsigned ((DATA_WIDTH-1) downto 0);
18       b        : in unsigned ((DATA_WIDTH-1) downto 0);
19       result   : out unsigned ((2*DATA_WIDTH-1) downto 0)
20    );
21
22 end entity;
23
24 architecture rtl of multiplicador1 is
25 begin
26
27    result <= a * b;
28
29 end rtl;
30
```

Diseño de un multiplicador sin signo paralelo con «template»

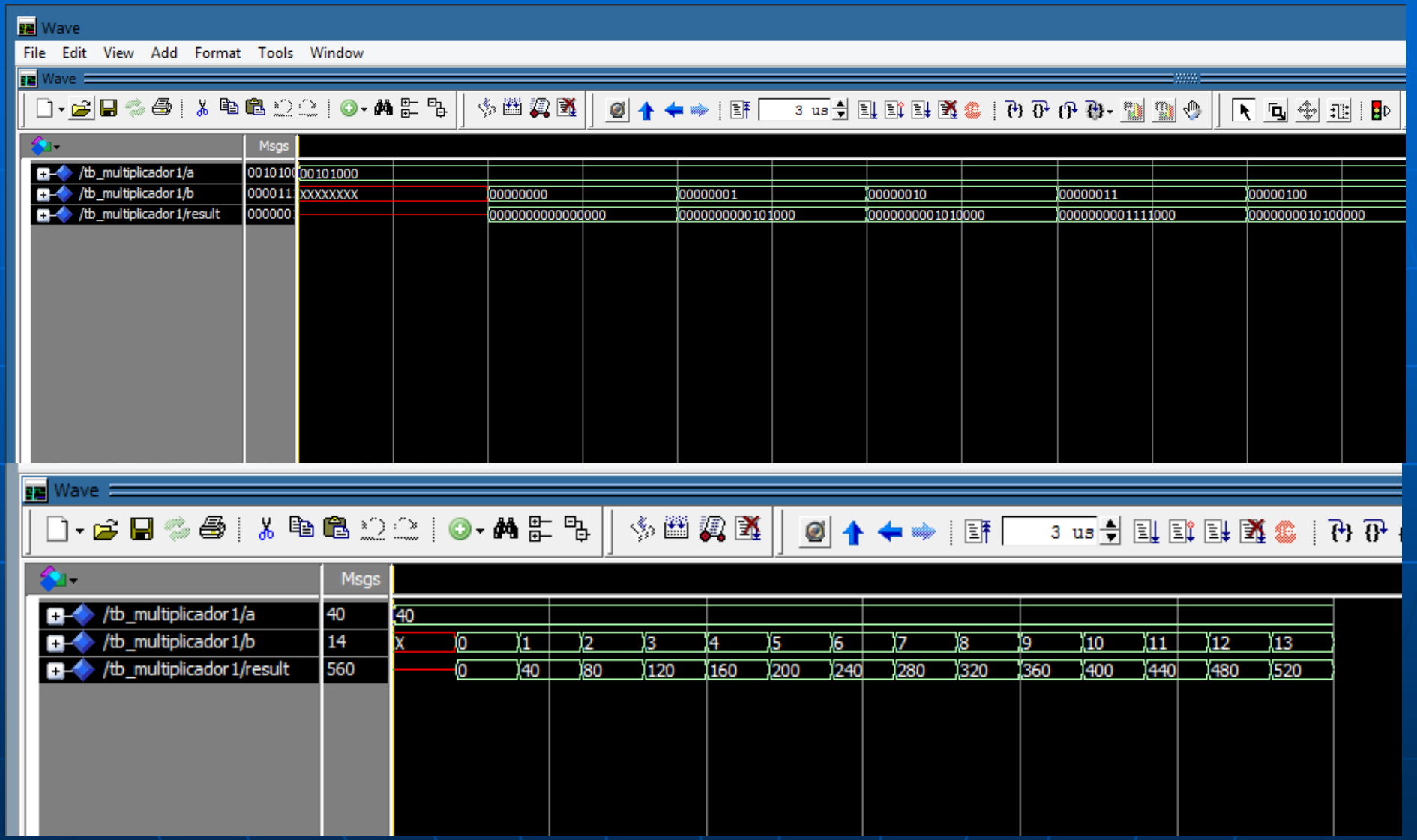
Test bench en VHDL

multiplicador1.vhd | tb_multiplicador1.vhd

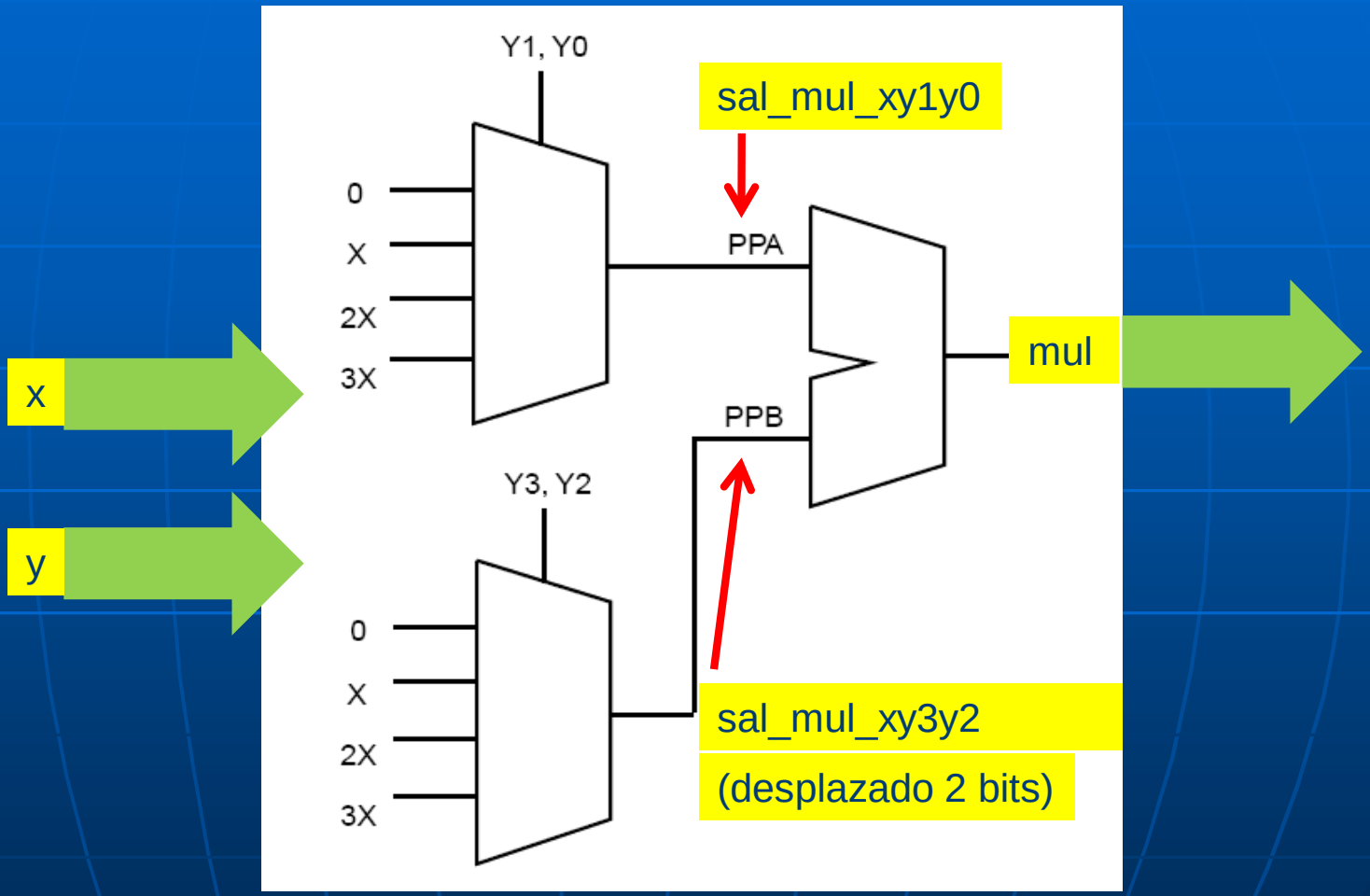
```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  use ieee.numeric_std.all;
4
5  ENTITY tb_multiplicador1 IS
6
7     generic
8     (
9        DATA_WIDTH : natural := 8
10    );
11
12
13 END tb_multiplicador1;
14
15 ARCHITECTURE behavior OF tb_multiplicador1 IS
16
17    COMPONENT multiplicador1 is
18    Port ( a        : in unsigned ((DATA_WIDTH-1) downto 0);
19           b        : in unsigned ((DATA_WIDTH-1) downto 0);
20           result   : out unsigned ((2*DATA_WIDTH-1) downto 0)
21              );
22    end COMPONENT;
23
24    signal a, b    : unsigned((DATA_WIDTH-1) downto 0);
25    signal result  : unsigned((2*DATA_WIDTH-1)downto 0) := (others => '0');
26
27 BEGIN
28
29    uut: multiplicador1 PORT MAP (a, b, result);
30
31    estimulo_proc: process
32       begin
33          a <= to_unsigned(40,DATA_WIDTH);
34          for i in 0 to 15 loop
35             wait for 200 ns;
36             b <= to_unsigned(i,DATA_WIDTH);
37          end loop;
38          wait;
39    end process;
40
41 END;
```

Diseño de un multiplicador sin signo paralelo con «template»

Resultado de la simulación



Diseño de un multiplicador paralelo con algoritmo de Booth de 4 bits



Diseño de un multiplicador paralelo con algoritmo de Booth de 4 bits

Descripción en VHDL

```
6   library IEEE;
7   use IEEE.STD_LOGIC_1164.ALL;
8   use IEEE.NUMERIC_STD.ALL;
9
10  entity mul_booth_4bits is
11      Port ( x       : in  unsigned (3 downto 0);
12             y       : in  unsigned (3 downto 0);
13             s_xy1y0 : out unsigned (7 downto 0);
14             s_xy3y2 : out unsigned (7 downto 0);
15             mul     : out unsigned (7 downto 0)
16                );
17  end mul_booth_4bits;
18
19  architecture Behavioral of mul_booth_4bits is
20
21  signal sal_mul_x_y1y0, sal_mul_x_y3y2,xr,yr : unsigned (7 downto 0);
22  signal sal : unsigned (7 downto 0);
23
24  begin
25
26           xr <= resize(x,8);
27           yr <= resize(y,8);
28
29  mul_x_y1y0 : process (xr, yr)
30              begin
31                 if ((not yr(1) and not yr(0)) = '1')  then sal_mul_x_y1y0 <= "00000000";
32                 elsif ((not yr(1) and yr(0))  = '1')  then sal_mul_x_y1y0 <= xr;
33                 elsif ((yr(1) and not yr(0))  = '1')  then sal_mul_x_y1y0 <= shift_left(xr,1);
34                 elsif ((yr(1) and yr(0))  = '1')      then sal_mul_x_y1y0 <= xr + shift_left(xr,1);
35                 end if;
36           end process;
37
38  mul_x_y3y2 : process (xr, yr)
39              begin
40                 if ((not yr(3) and not yr(2)) = '1' ) then sal_mul_x_y3y2 <= "00000000";
41                 elsif ((not yr(3) and yr(2)) = '1' ) then sal_mul_x_y3y2 <= xr;
42                 elsif ((yr(3) and not yr(2)) = '1' ) then sal_mul_x_y3y2 <= shift_left(xr,1);
43                 elsif ((yr(3) and yr(2)) = '1' ) then sal_mul_x_y3y2 <= xr + shift_left(xr,1);
44                 end if;
45           end process;
46
47  --suma  : process (sal_mul_x_y1y0, sal_mul_x_y3y2)
48  --          begin
49                sal <= sal_mul_x_y1y0 + shift_left(sal_mul_x_y3y2,2);
50  --        end process;
51                mul <= sal;
52                s_xy1y0 <= sal_mul_x_y1y0;
53                s_xy3y2 <= sal_mul_x_y3y2;
54  end Behavioral;
```

Diseño de un multiplicador paralelo con algoritmo de Booth de 4 bits

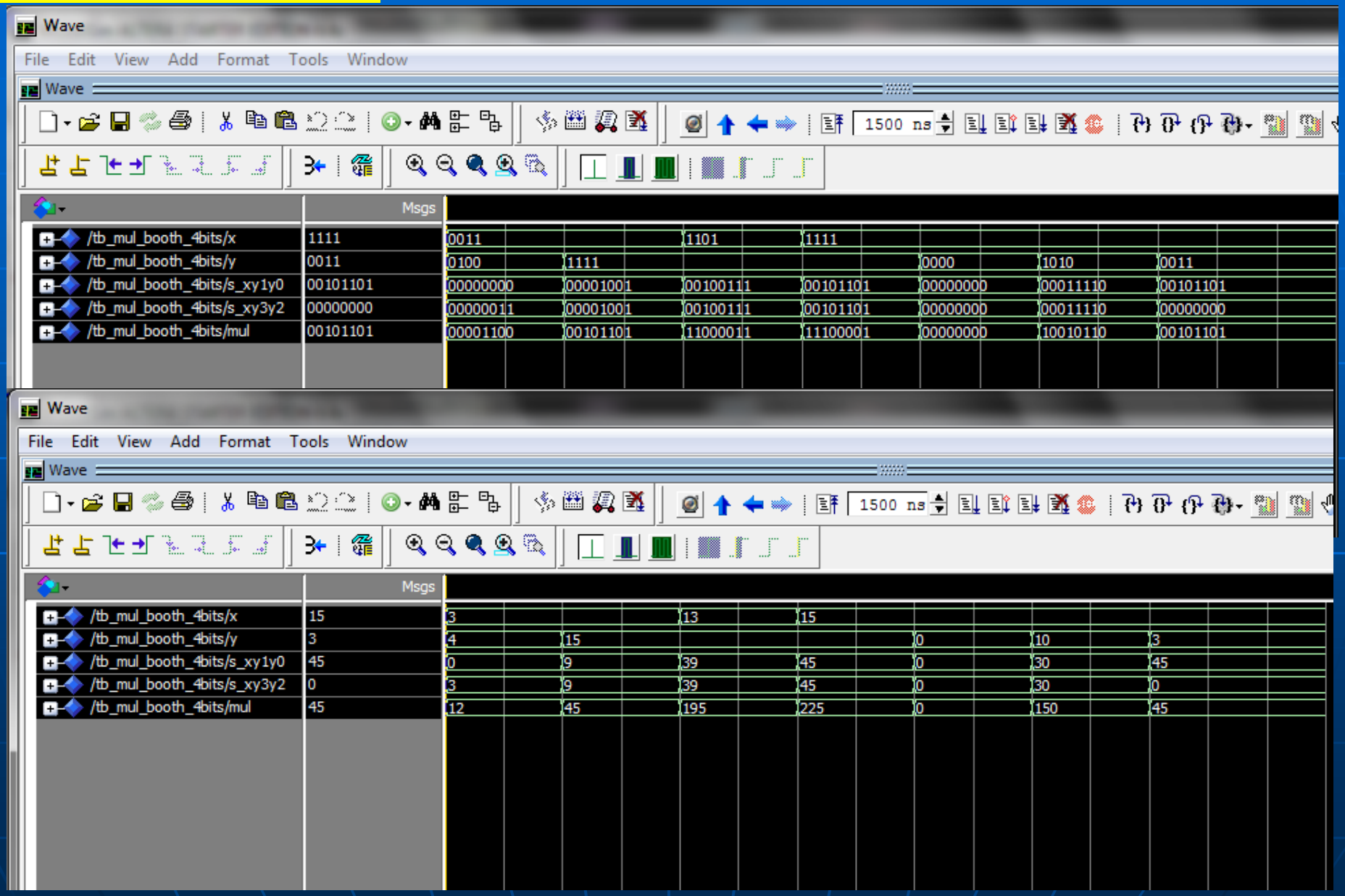
tb_mul_booth_4bits.vhd

Test bench en VHDL

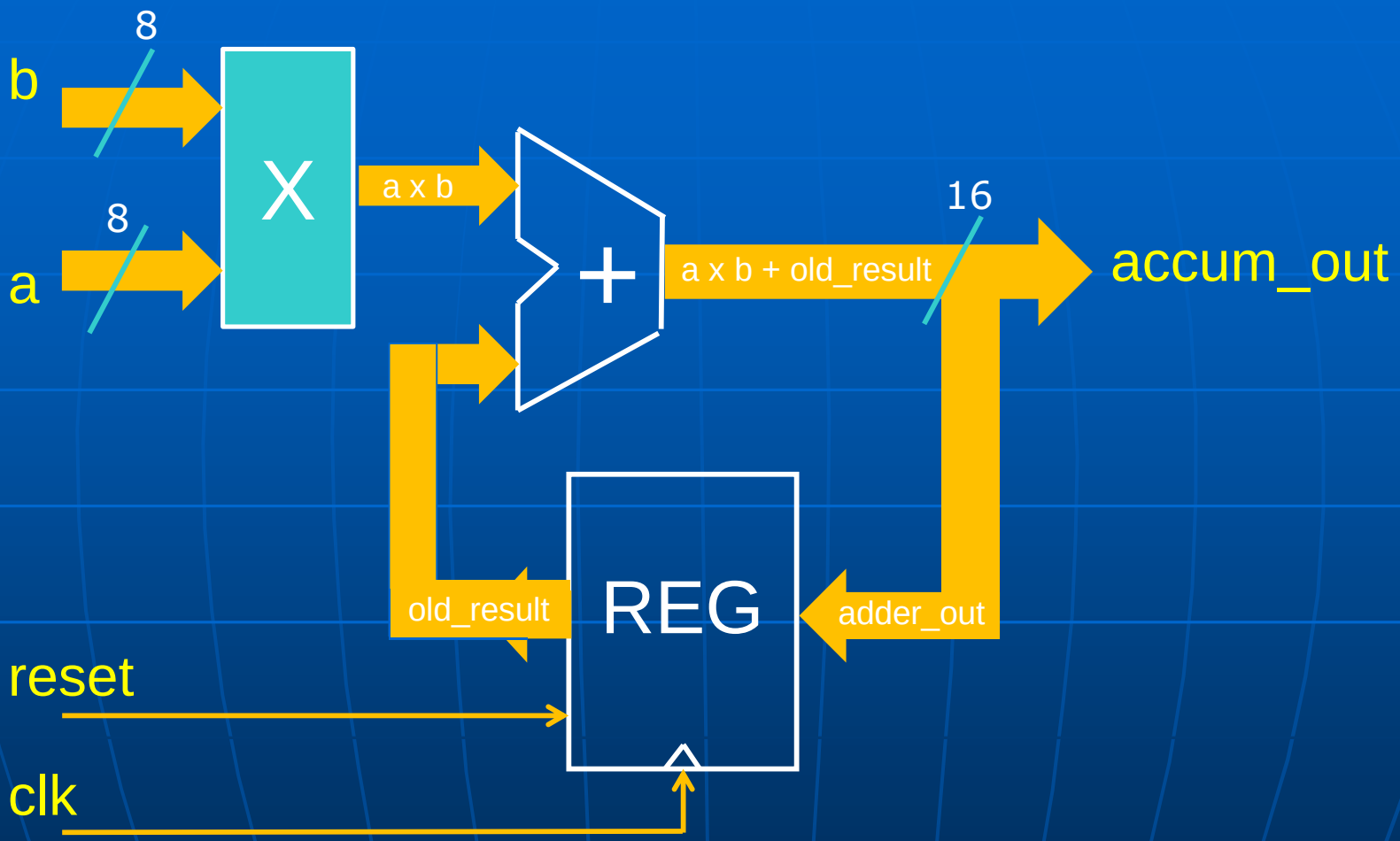
```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  use ieee.numeric_std.all;
4
5  ENTITY tb_mul_booth_4bits IS
6  END tb_mul_booth_4bits;
7
8  ARCHITECTURE behavior OF tb_mul_booth_4bits IS
9
10     COMPONENT mul_booth_4bits is
11     Port ( x       : in  unsigned (3 downto 0);
12            y       : in  unsigned (3 downto 0);
13            s_xy1y0 : out unsigned (7 downto 0);
14            s_xy3y2 : out unsigned (7 downto 0);
15            mul     : out unsigned (7 downto 0)
16               );
17         end COMPONENT;
18
19     signal x, y             : unsigned(3 downto 0);
20     signal s_xy1y0, s_xy3y2 : unsigned(7 downto 0);
21     signal mul              : unsigned(7 downto 0);
22
23 BEGIN
24
25    uut: mul_booth_4bits PORT MAP (x, y, s_xy1y0, s_xy3y2, mul);
26
27    estimulo_proc: process
28       begin
29          x <= "0011";
30          y <= "0100";
31          wait for 200 ns;
32          y <= "1111";
33          wait for 200 ns;
34          x <= "1101";
35          wait for 200 ns;
36          x <= "1111";
37          wait for 200 ns;
38          y <= "0000";
39          wait for 200 ns;
40          y <= "1010";
41          wait for 200 ns;
42          y <= "0011";
43          wait;
44    end process;
45
46 END;
```

Diseño de un multiplicador paralelo con algoritmo de Booth de 4 bits

Resultado de la simulación



Diseño de un multiplicador – acumulador paralelo con signo de 8 bits



Diseño de un multiplicador – acumulador paralelo con signo de 8 bits

Descripción en VHDL

mult_acum_con_signo.vhd | tb_mult_acum_con_signo.vhd

```
1  --Ejemplo de multiplicador mas acumulador
2  -- Sergio Noriega ISLD 2016
3  library ieee;
4  use ieee.std_logic_1164.all;
5  use ieee.numeric_std.all;
6
7  entity mult_acum_con_signo is
8     port
9     (
10       a           : in signed(7 downto 0);
11       b           : in signed (7 downto 0);
12       clk         : in std_logic;
13       reset       : in std_logic;
14       accum_out   : out signed (15 downto 0)
15    );
16
17 end entity;
18
19 architecture rtl of mult_acum_con_signo is
20
21    signal a_reg : signed (7 downto 0);
22    signal b_reg : signed (7 downto 0);
23    signal mult_reg : signed (15 downto 0);
24    signal adder_out : signed (15 downto 0);
25    signal old_result : signed (15 downto 0);
26
27 begin
28          mult_reg <= a_reg * b_reg;
29          adder_out <= old_result + mult_reg;
30
31          a_reg <= a;
32          b_reg <= b;
33
34    process (clk, reset)
35    begin
36       if (reset = '1') then
37          old_result <= (others => '0');
38       elsif (rising_edge(clk)) then
39       old_result <= adder_out;
40       accum_out <= adder_out;
41       end if;
42
43    end process;
44
45 end rtl;
46
```

Diseño de un multiplicador – acumulador paralelo con signo de 8 bits

Test bench en VHDL

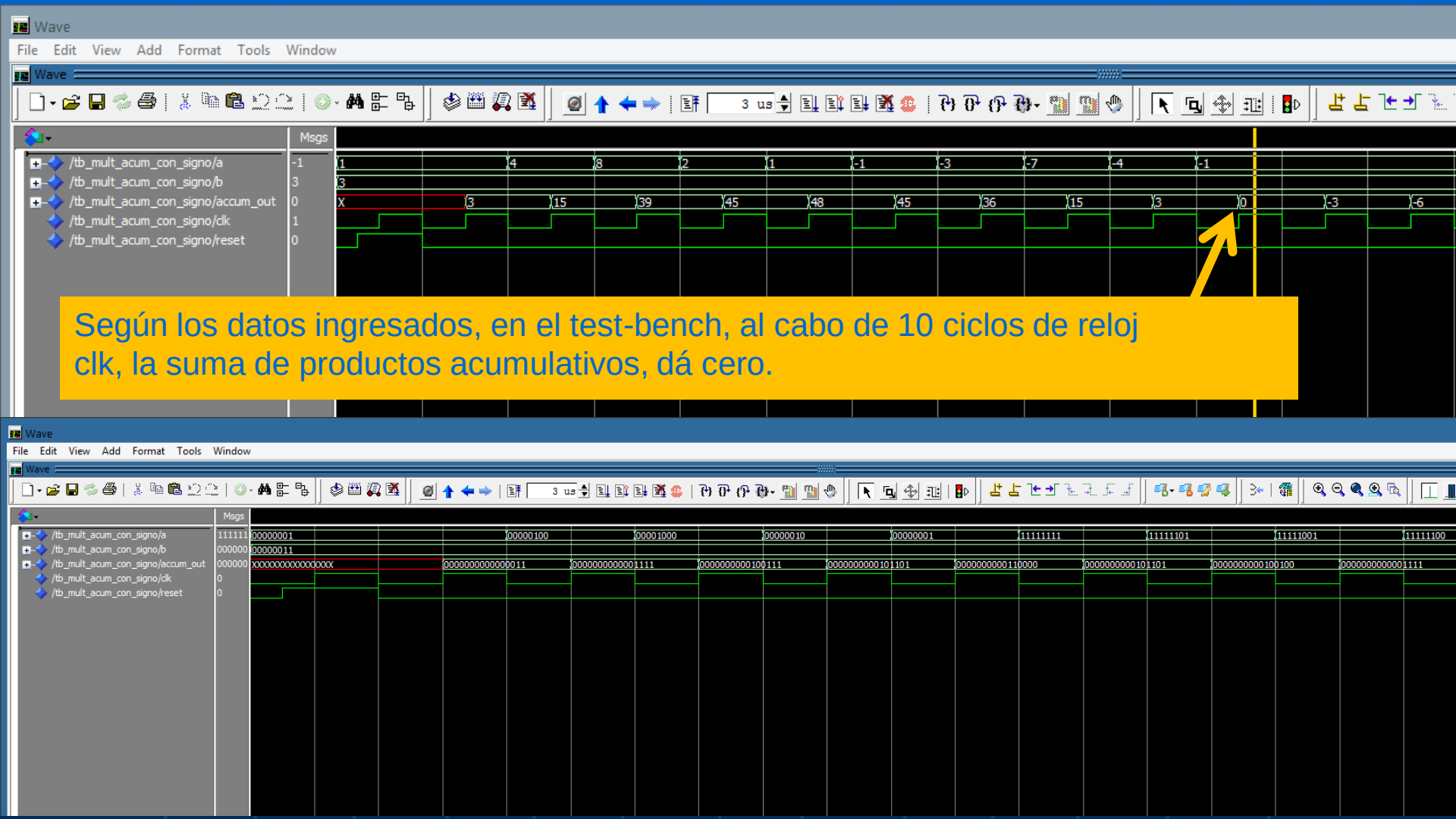
En este test-bench se mantiene constante la entrada ‘b’ en el valor +3 y se entran 10 valores de ‘a’ en sincronía con el reloj: +1, +4, +8, +2, +1, -1, -3, -7, -4, -1.

mult_acum_con_signo.vhd | tb_mult_acum_con_signo.vhd

```
1  LIBRARY ieee; USE ieee.std_logic_1164.ALL; use ieee.numeric_std.all;
2
3  ENTITY tb_mult_acum_con_signo IS
4  END tb_mult_acum_con_signo;
5
6  ARCHITECTURE behavior OF tb_mult_acum_con_signo IS
7
8      COMPONENT mult_acum_con_signo is
9      port (
10      a           : in signed(7 downto 0);
11      b           : in signed (7 downto 0);
12      clk         : in std_logic;
13      reset       : in std_logic;
14      accum_out   : out signed (15 downto 0));
15   end COMPONENT;
16
17    signal a, b             : signed(7 downto 0);
18    signal accum_out        : signed(15 downto 0);
19    signal clk, reset       : std_logic;
20
21 type datos is array (integer range 0 to 9) of signed(7 downto 0);
22
23 BEGIN
24
25  uut: mult_acum_con_signo PORT MAP (a, b, clk, reset, accum_out);
26
27    gen_reloj : process -- Reloj de 200 ns de periodo
28       begin
29          clk <= '0';
30          wait for 100 ns;
31          clk <= '1';
32          wait for 100 ns;
33    end process gen_reloj;
34
35    estimulo_proc: process
36    variable muestras: datos;
37             begin
38          muestras := ("00000001", "00000100", "00001000","00000010","00000001",
39                       "11111111", "11111101", "11111001","11111100","11111111");
40                       --(1, 4, 8, 2, 1, -1, -3, -7, -4, -1)
41          a <= muestras (0);
42          b <= "00000011";
43          reset <= '0';
44          wait for 50 ns;
45          reset <= '1';
46          wait for 150 ns;
47          reset <= '0';
48          wait for 200 ns;
49          for i in 1 to 9 loop
50             a <= muestras(i);
51             wait for 200 ns;
52          end loop;
53          wait;
54    end process;
55 END;
```

Diseño de un multiplicador – acumulador paralelo con signo de 8 bits

Resultado de la simulación



Diseño de un multiplicador en Punto Flotante Simple Precisión IEEE-754

Single-Precision Format

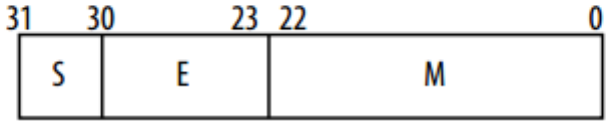
The single-precision format contains the following binary patterns:

- The MSB holds the sign bit.
- The next 8 bits hold the exponent bits.
- 23 LSBs hold the mantissa.

The total width of a floating-point number in the single-precision format is 32 bits. The bias for the single-precision format is 127.

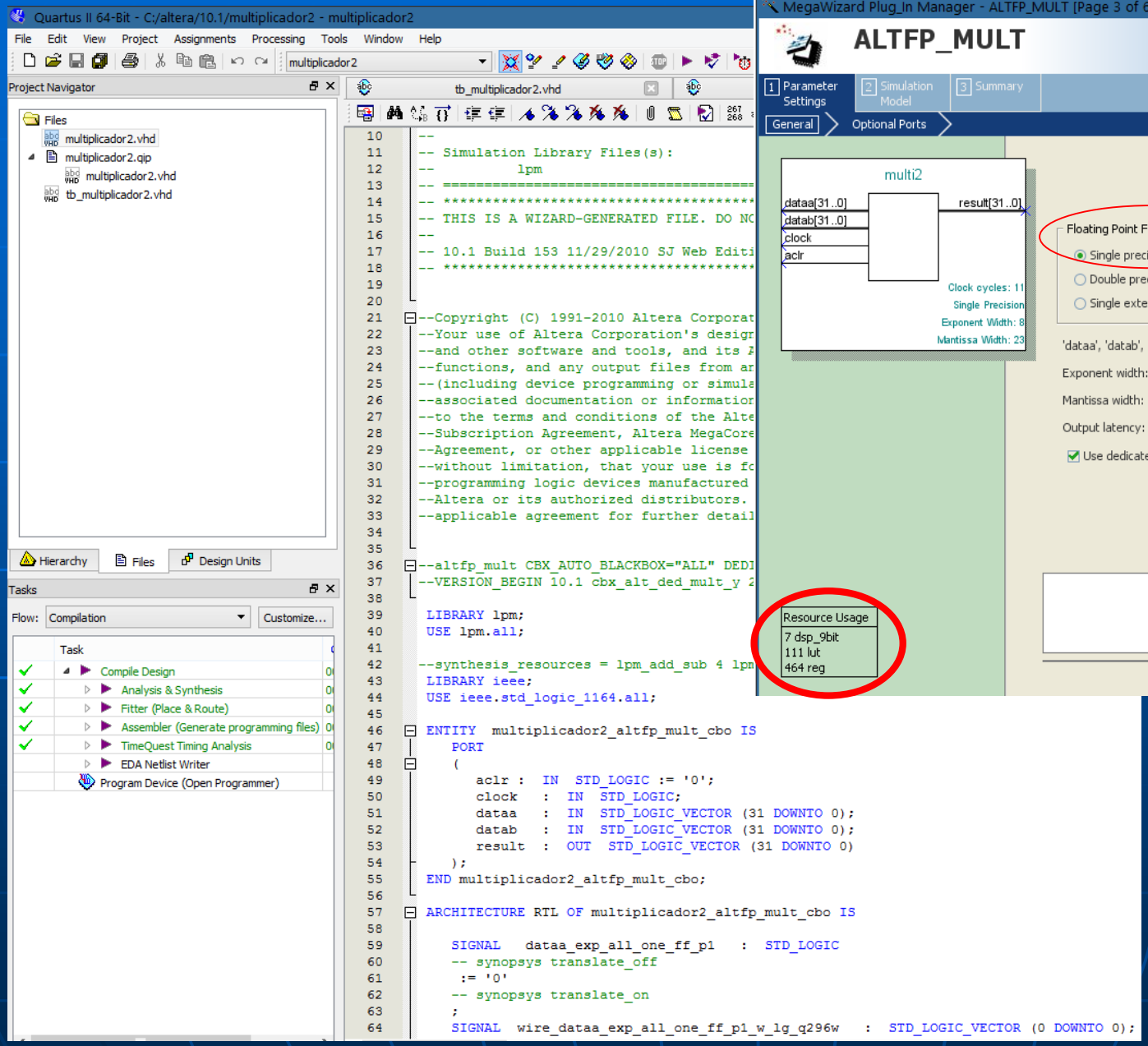
Figure 1-10: Single-Precision Representation

This figure shows a single-precision representation.



Referencia: Nota de aplicación de Altera 'Floating-Point IP Cores User Guide': 'ug_altfp_mfug.pdf' año 2015.

Diseño de un multiplicador en Punto Flotante Simple Precisión IEEE-754



Diseño de un multiplicador en Punto Flotante Simple Precisión IEEE-754

tb_multiplicador2.vhd | multiplicador2.qip | multiplicador2.vhd

```
1  LIBRARY ieee;
2  USE ieee.std_logic_1164.ALL;
3  use ieee.numeric_std.all;
4  LIBRARY lpm;
5  USE lpm.all;
6
7  ENTITY tb_multiplicador2 IS
8  END tb_multiplicador2;
9
10 ARCHITECTURE behavior OF tb_multiplicador2 IS
11
12
13     COMPONENT multiplicador2 is
14     PORT
15     (
16        aclr :  IN  STD_LOGIC := '0';
17        clock   :  IN  STD_LOGIC;
18        dataa   :  IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
19        datab   :  IN  STD_LOGIC_VECTOR (31 DOWNTO 0);
20        result  :  OUT  STD_LOGIC_VECTOR (31 DOWNTO 0)
21     );
22   END COMPONENT;
23
24
25
26     signal dataa, datab, result   : STD_LOGIC_VECTOR (31 DOWNTO 0);
27     signal aclr, clock : STD_LOGIC;
28
29 BEGIN
30
31    uut: multiplicador2 PORT MAP (aclr, clock, dataa, datab, result);
32
33    gen_reloj : process -- Reloj de 20 ns de periodo y 50% de ciclo de trabajo
34       begin
35          clock <= '0';
36          wait for 100 ns;
37          clock <= '1';
38          wait for 100 ns;
39    end process gen_reloj;
40
41    estimulo_proc: process
42       begin
43          dataa <= "00111111111000000000000000000000";
44          datab <= "11000000010000000000000000000000";
45          aclr <= '0';
46          wait for 300 ns;
47          aclr <= '1';
48          wait for 150 ns;
49          aclr <= '0';
50          wait for 5 us;
51    end process;
52
53 END;
```

Test bench en VHDL

Positivo

equivale a ‘1,11’

dataa = 00111111111000000……0

equivale a ’2^0’

$+1,75_{10}$

Negativo

equivale a ‘1,10’

datab = 11000000011000000……0

-3_{10}

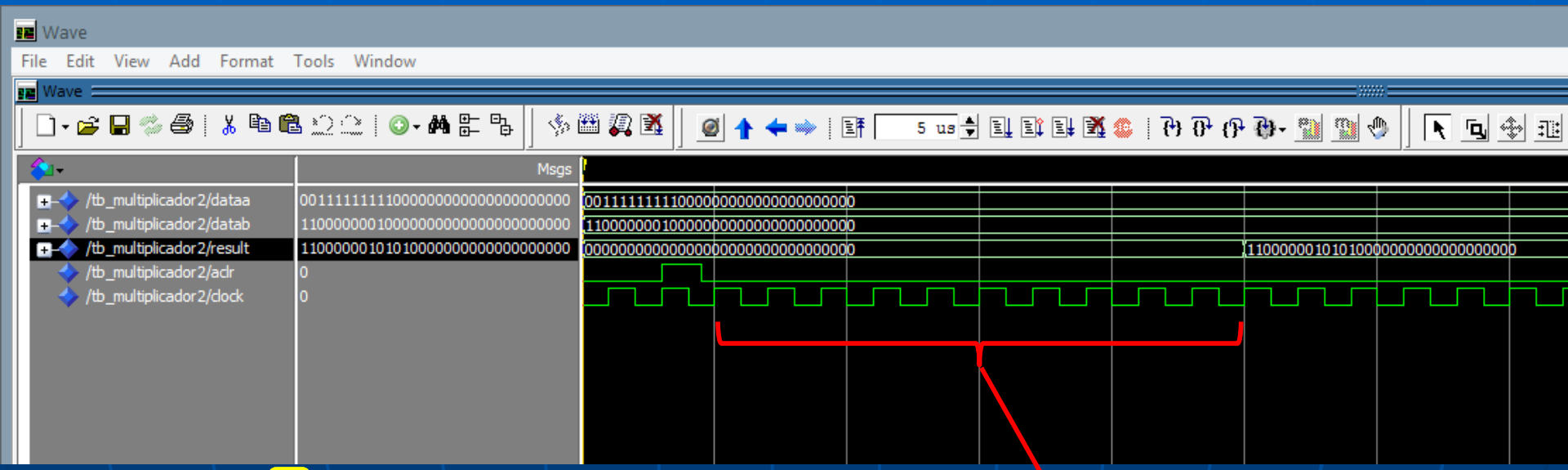
equivale a ’2^1’

Diseño de un multiplicador en Punto Flotante Simple Precisión IEEE-754

Resultado de la simulación

$(1,75) \times (-3) = -5,25_{10}$

dataa datab result



result : 1 10000001 01010.....0

equivale a '2^{2}' equivale a '1,0101'

(-) 101,01

11 ciclos de reloj de latencia para generar el resultado de la multiplicación.

Diseño de Unidad Aritmético-Lógica (ALU)

Ejemplo: Diseño de ALU de 8 bits con operaciones aritméticas sin signo

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_unsigned.all;

ENTITY alu3 IS
    PORT (a, b: IN STD_LOGIC_VECTOR (7 DOWNTO 0);
          sel: IN STD_LOGIC_VECTOR (4 DOWNTO 0);
          cin: IN STD_LOGIC;
          sal: OUT STD_LOGIC_VECTOR (7 DOWNTO 0));
END alu3;

ARCHITECTURE alucomp3 OF alu3 IS
    SIGNAL aritme, logica, swap_a: STD_LOGIC_VECTOR (7 DOWNTO 0);
BEGIN
    swap_a(7 DOWNTO 4) <= a(3 DOWNTO 0);
    swap_a(3 DOWNTO 0) <= a(7 DOWNTO 4);
--Sección de operaciones aritméticas (sin signo)
    WITH sel(3 DOWNTO 0) SELECT
        aritme <= a WHEN "0000",        --Transferencia del operando "a"
                  a+1 WHEN "0001",      --Incremento del operando "a"
                  a-1 WHEN "0010",      --Decremento del operando "a"
                  b WHEN "0011",        --Transferencia del operando "b"
                  b+1 WHEN "0100",      --Incremento del operando "b"
                  b-1 WHEN "0101",      --Decremento del operando "b"
                  a+b WHEN "0110",      --Suma sin signo
                  a+b+cin WHEN "0111", --Suma sin signo con carry
                  a-b WHEN "1000",      --Resta sin signo
                a-b-cin WHEN "1001",    --Resta sin signo con borrow
                a*b WHEN "1010",        --Multiplicación sin signo
                a WHEN OTHERS;          --Evita generación de lógica extra
```

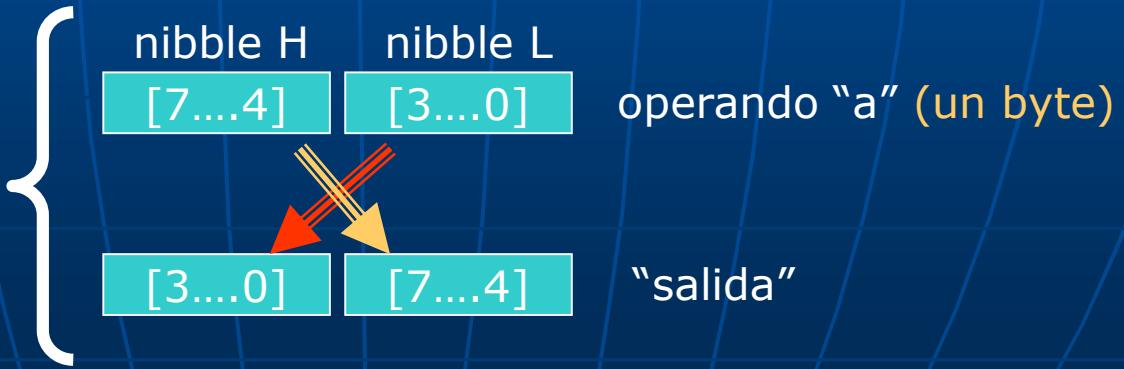
sentencias para implementar la operación “swap” del operando “a”

Diseño de Unidad Aritmético-Lógica (ALU)

Ejemplo: Diseño de ALU de 8 bits con operaciones aritméticas sin signo (continuación)

```
--Sección de operaciones lógicas (operaciones bit a bit)
    WITH sel(3 DOWNTO 0) SELECT
        logica <= NOT a WHEN "0000",   --Negación del operando "a"
                  NOT b WHEN "0001",   --Negación del operando "b"
                  a AND b WHEN "0010", --Operación AND e/ "a" y "b"
                  a OR b WHEN "0011",  --Operación OR e/ "a" y "b"
                  a NAND b WHEN "0100",--Operación NAND e/ "a" y "b"
                  a NOR b WHEN "0101", --Operación NOR e/ "a" y "b"
                  a XOR b WHEN "0110", --Operación XOR e/ "a" y "b"
                  a XNOR b WHEN "0111",--Operación NOT(XOR) e/ "a" y "b"
                  swap_a WHEN "1000",  --Intercambio de nibles en "a"
                  a WHEN OTHERS;       --Evita generación de lógica extra
    WITH sel(4) SELECT
        sal <= aritme WHEN '0',
               logica WHEN OTHERS;
END alucomp3;
```

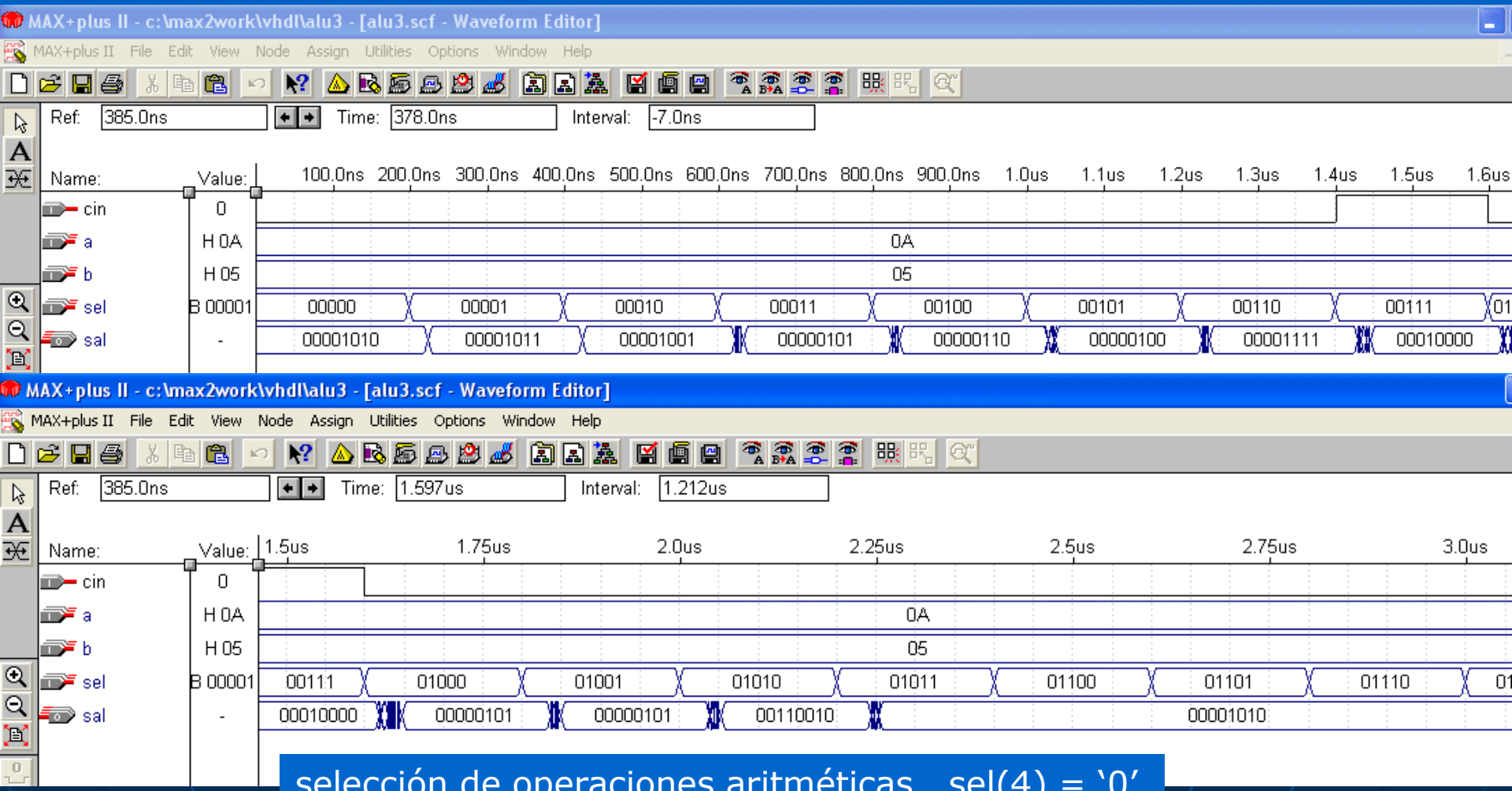
Ejemplo de operación swap para el operando "a" ("sel" = "11000")



Diseño de Unidad Aritmético-Lógica (ALU)

Diseño de ALU de 8 bits con operaciones aritméticas sin signo (continuación)

Simulación

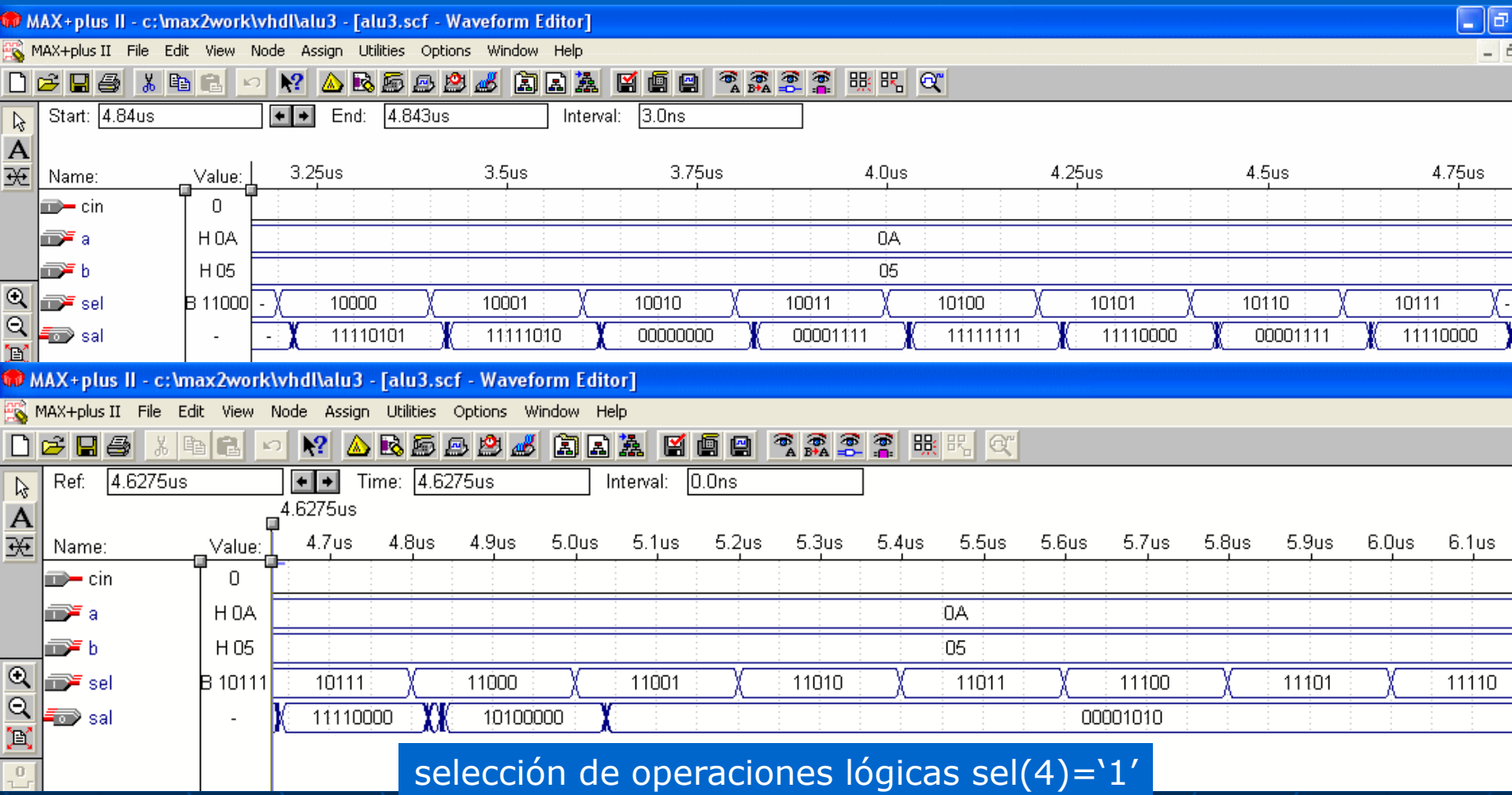


selección de operaciones aritméticas sel(4) = ‘0’

Diseño de Unidad Aritmético-Lógica (ALU)

Diseño de ALU de 8 bits con operaciones aritméticas sin signo (continuación)

Simulación



selección de operaciones lógicas sel(4)='1'