VHDL ET TEST BENCH

2024-2025

additionneur 1 bit

C:/intelFPGA/18.1/ADDI.vhd (/tb_add4/uut/inst_add3) - Default

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

entity add1 is
    port (
        a, b, cin : in std_logic;
        s, cout   : out std_logic
    );
end add1;

architecture comportement of add1 is
begin
    cout <= (a and b) or (a and cin) or (b and cin);
    s <= a xor b xor cin;
end comportement;
```

Cette image présente un code VHDL implémentant un additionneur complet (Full Adder).
Le module, nommé add1, reçoit trois entrées (a, b, cin) et génère deux sorties : la somme (s) et la retenue (cout).
L'architecture comportement décrit la logique combinatoire selon les équations standards du Full Adder en utilisant des opérations logiques (xor, and, or

Test bench

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

entity tb_add1 is
end tb_add1;

architecture test of tb_add1 is
    signal a, b, cin : std_logic;
    signal s, cout   : std_logic;
begin
    -- Instanciation du composant
    UUT: entity work.add1
        port map (
            a => a,
            b => b,
            cin => cin,
            s => s,
            cout => cout
        );

    -- Processus de test
    process
    begin
        -- Cas 1: 0 + 0 + 0 = 0
        a <= '0'; b <= '0'; cin <= '0'; wait for 10 ns;
        report "Test 000 => s=" & std_logic'image(s) & " cout=" & std_logic'image(cout);

        -- Cas 2: 0 + 0 + 1 = 1
        a <= '0'; b <= '0'; cin <= '1'; wait for 10 ns;
        report "Test 001 => s=" & std_logic'image(s) & " cout=" & std_logic'image(cout);

        -- Cas 3: 0 + 1 + 0 = 1
        a <= '0'; b <= '1'; cin <= '0'; wait for 10 ns;
        report "Test 010 => s=" & std_logic'image(s) & " cout=" & std_logic'image(cout);
```

```
35
36          -- Cas 4: 0 + 1 + 1 = 0, cout=1
37          a <= '0'; b <= '1'; cin <= '1'; wait for 10 ns;
38          report "Test 011 => s=" & std_logic'image(s) & " cout=" & std_logic'image(cout);
39
40          -- Cas 5: 1 + 0 + 0 = 1
41          a <= '1'; b <= '0'; cin <= '0'; wait for 10 ns;
42          report "Test 100 => s=" & std_logic'image(s) & " cout=" & std_logic'image(cout);
43
44          -- Cas 6: 1 + 0 + 1 = 0, cout=1
45          a <= '1'; b <= '0'; cin <= '1'; wait for 10 ns;
46          report "Test 101 => s=" & std_logic'image(s) & " cout=" & std_logic'image(cout);
47
48          -- Cas 7: 1 + 1 + 0 = 0, cout=1
49          a <= '1'; b <= '1'; cin <= '0'; wait for 10 ns;
50          report "Test 110 => s=" & std_logic'image(s) & " cout=" & std_logic'image(cout);
51
52          -- Cas 8: 1 + 1 + 1 = 1, cout=1
53          a <= '1'; b <= '1'; cin <= '1'; wait for 10 ns;
54          report "Test 111 => s=" & std_logic'image(s) & " cout=" & std_logic'image(cout);
55
56          wait; -- Fin du test
57      end process;
58  end test;
59
```

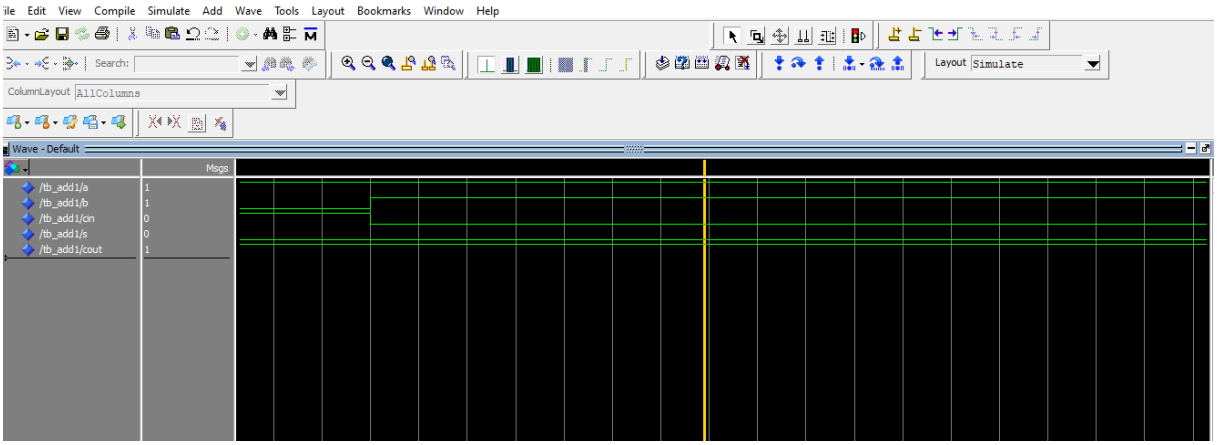
Ces deux images montrent un banc de test (testbench) en VHDL pour valider le fonctionnement d'un additionneur complet (add1).
Le composant est instancié, puis testé avec les huit combinaisons possibles des entrées (a, b, cin) via des signaux internes.
Chaque cas de test vérifie les sorties s et cout, affichées avec la procédure report pour évaluer la justesse du circuit

résultats ModelSim

```
# ** Note: Test 000 => s='0' cout='0'
#    Time: 10 ns  Iteration: 0  Instance: /tb_add1
# ** Note: Test 001 => s='1' cout='0'
#    Time: 20 ns  Iteration: 0  Instance: /tb_add1
# ** Note: Test 010 => s='1' cout='0'
#    Time: 30 ns  Iteration: 0  Instance: /tb_add1
# ** Note: Test 011 => s='0' cout='1'
#    Time: 40 ns  Iteration: 0  Instance: /tb_add1
# ** Note: Test 100 => s='1' cout='0'
#    Time: 50 ns  Iteration: 0  Instance: /tb_add1
# ** Note: Test 101 => s='0' cout='1'
#    Time: 60 ns  Iteration: 0  Instance: /tb_add1
# ** Note: Test 110 => s='0' cout='1'
#    Time: 70 ns  Iteration: 0  Instance: /tb_add1
# ** Note: Test 111 => s='1' cout='1'
#    Time: 80 ns  Iteration: 0  Instance: /tb_add1
```

La fenêtre **Transcript** de ModelSim affiche les résultats de la simulation du testbench. Pour chaque combinaison d'entrées (a, b, cin), elle indique le **temps de simulation**, la **valeur de la somme s**, ainsi que le **bit de retenue cout**. Ces

résultats permettent de vérifier le bon fonctionnement de l'additionneur 1 bit en comparant les sorties obtenues avec les valeurs attendues pour chaque cas de test.



L’image représente la **fenêtre de visualisation des signaux (Waveform Viewer)** dans ModelSim après la simulation du testbench. Elle montre l’évolution temporelle des signaux a, b, cin, s, et cout de l’additionneur 1 bit (tb_add1).

Chaque ligne horizontale correspond à un signal, tandis que les transitions verticales indiquent les **changements d’état** (de 0 à 1 ou inversement) au cours du temps. Cette représentation permet de **vérifier visuellement** le comportement du circuit simulé et de confirmer que les **résultats de sortie (s et cout)** sont corrects en fonction des **entrées appliquées (a, b, cin)**.

additionneur 4 bits

```
Ln#
 1  library IEEE;
 2  use IEEE.std_logic_1164.all;
 3  use IEEE.numeric_std.all;
 4
 5  entity add4 is
 6      port (
 7          a, b  : in  unsigned(3 downto 0);
 8          cin   : in  std_logic;
 9          s     : out unsigned(3 downto 0);
10          r     : out std_logic
11      );
12  end add4;
13
14  architecture struct of add4 is
15      component addl is
16          port (
17              a, b, cin : in std_logic;
18              s, cout   : out std_logic
19          );
20      end component;
21
22      signal c : std_logic_vector(2 downto 0);
23  begin
24      inst_add0 : addl port map (a(0), b(0), cin,    s(0), c(0));
25      inst_addl : addl port map (a(1), b(1), c(0),   s(1), c(1));
26      inst_add2 : addl port map (a(2), b(2), c(1),   s(2), c(2));
27      inst_add3 : addl port map (a(3), b(3), c(2),   s(3), r);
28  end struct;
```

code VHDL décrit un additionneur 4 bits (add4) construit à partir de 4 instances d'un composant Ce code VHDL décrit un additionneur 4 bits (add4) construit à partir de 4 instances d'un composant élémentaire add1. Les bits de retenue intermédiaires sont transmis via un signal c. Le module prend en entrée deux vecteurs a, b et un cin, et génère la somme s et le report final r.

Test bench

```vhdl
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;

entity tb_add4 is
end tb_add4;

architecture test of tb_add4 is

    -- Signaux d'entrée/sortie
    signal a, b : unsigned(3 downto 0);
    signal s    : unsigned(3 downto 0);
    signal cin  : std_logic;
    signal r    : std_logic;

begin
    -- Instanciation du composant à tester (Unit Under Test)
    uut: entity work.add4
        port map (
            a   => a,
            b   => b,
            cin => cin,
            s   => s,
            r   => r
        );

    -- Processus de stimulation
    process
        variable total : integer;
        variable i, j : integer;
        variable c    : integer;
        variable r_val : unsigned(4 downto 0);
    begin
        for i in 0 to 15 loop
```

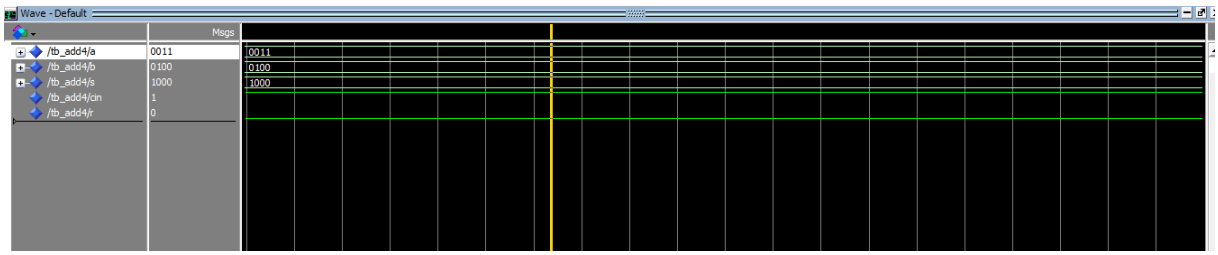
```vhdl
        variable r_val : unsigned(4 downto 0);
    begin
        for i in 0 to 15 loop
            for j in 0 to 15 loop
                for c in 0 to 1 loop
                    a <= to_unsigned(i, 4);
                    b <= to_unsigned(j, 4);
                    if c = 0 then
                        cin <= '0';
                    else
                        cin <= '1';
                    end if;

                    wait for 10 ns;

                    -- Étendre la retenue r à 5 bits pour la somme complète
                    r_val := (4 => r, 3 downto 0 => '0');

                    -- Calculer la somme totale : s + retenue
                    total := to_integer(s) + to_integer(r_val);

                    report "Test: " & integer'image(i) & " + " & integer'image(j) & " + " & integer'image(c) &
                           " = " & integer'image(total) &
                           " | s=" & integer'image(to_integer(s)) & " r='" & std_logic'image(r) & "'";
                end loop;
            end loop;
        end loop;

        wait;
    end process;

end test;
```

Ce testbench vérifie le fonctionnement de l'additionneur 4 bits add4 en testant toutes les combinaisons possibles des entrées a, b (0 à 15) et cin (0 ou 1), soit 512 cas. Il applique chaque combinaison, attend 10 ns, puis compare la somme obtenue (s + r) avec la valeur attendue. Les résultats sont affichés via des messages report pour faciliter le débogage. Cette méthode garantit une validation exhaustive et automatisée du circuit

résultats ModelSim

```
SIM 6> run -all
** Note: Test: 0 + 0 + 0 = 0 | s=0 r='0'
   Time: 10 ns  Iteration: 0  Instance: /tb_add4
** Note: Test: 0 + 0 + 1 = 1 | s=1 r='0'
   Time: 20 ns  Iteration: 0  Instance: /tb_add4
** Note: Test: 0 + 1 + 0 = 1 | s=1 r='0'
   Time: 30 ns  Iteration: 0  Instance: /tb_add4
** Note: Test: 0 + 1 + 1 = 2 | s=2 r='0'
   Time: 40 ns  Iteration: 0  Instance: /tb_add4
** Note: Test: 0 + 2 + 0 = 2 | s=2 r='0'
   Time: 50 ns  Iteration: 0  Instance: /tb_add4
** Note: Test: 0 + 2 + 1 = 3 | s=3 r='0'
   Time: 60 ns  Iteration: 0  Instance: /tb_add4
```

L'image représente une portion des résultats de simulation d'un circuit numérique, probablement un additionneur 1 bit. On y voit plusieurs cas de test avec les entrées binaires, les sorties (s et cout), et le temps associé. Bien que les tests soient présentés de manière séquentielle et structurée, l'image ne couvre pas l'ensemble des combinaisons possibles. Elle constitue donc un **extrait partiel** d'une vérification plus complète du circuit



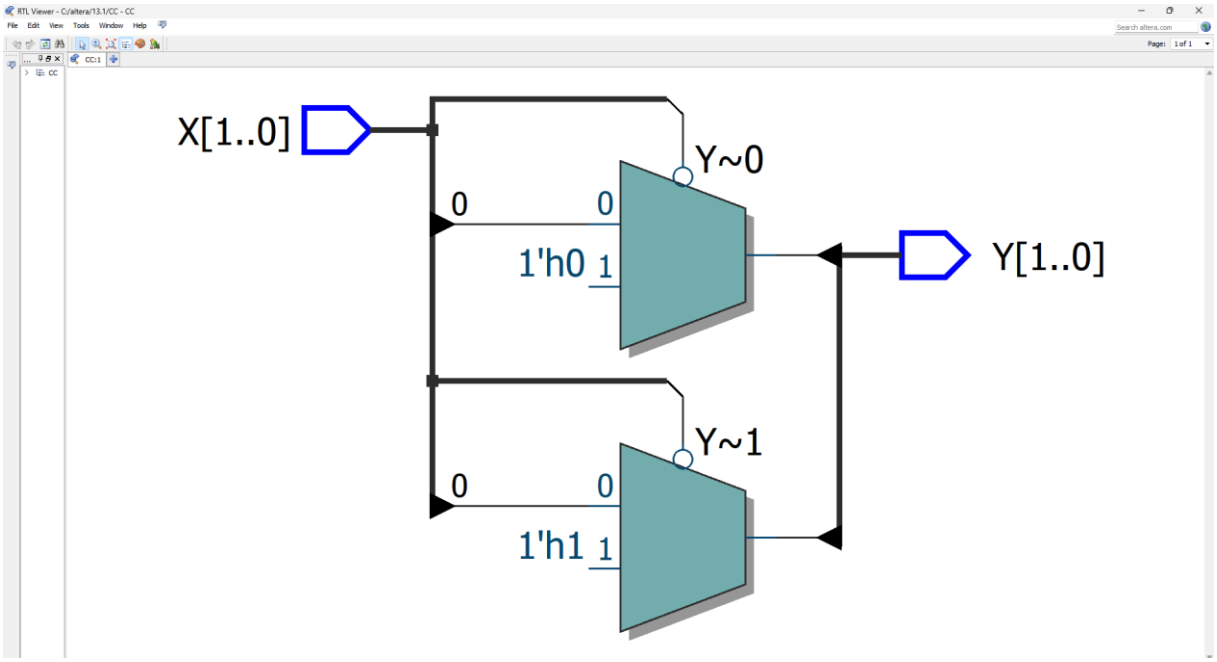
Cette image représente une vue temporelle de la simulation d'un additionneur 4 bits (add4). On y observe les signaux d'entrée (a, b, cin) ainsi que les sorties (s pour la somme et r pour la retenue). Elle permet de vérifier visuellement le bon fonctionnement du circuit pour différentes combinaisons d'entrée, en s'assurant que les sorties correspondent aux résultats attendus.

circuit combinatoire CC

```vhdl
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity CC is
    port (
        X : in bit_vector(1 downto 0);
        Y : out bit_vector(1 downto 0)
    );
end entity CC;

architecture A_CC of CC is
begin
    process (X)
    begin
        if X(1) = '0' then
            Y <= "01";
        elsif X(0) = '0' then
            Y <= "00";
        else
            Y <= "11";
        end if;
    end process;
end architecture A_CC;

```

Cette image montre l'architecture interne du circuit combinatoire CC, qui prend une entrée X (2 bits) et produit une sortie Y (2 bits), en fonction de conditions logiques sur les bits de X.

Schéma RTL généré (Register Transfer Level)



Vue RTL du circuit générée automatiquement par l'outil de synthèse. Elle montre les connexions internes et la logique implémentée selon l'architecture décrite.

Test bench

C:/altera/13.1/CC.vhd (/tb_cc) - Default

```
Ln#
 1  library IEEE;
 2  use IEEE.STD_LOGIC_1164.ALL;
 3
 4  entity TB_CC is
 5  end entity;
 6
 7  architecture TB of TB_CC is
 8
 9      component CC is
10          port (
11              X : in bit_vector(1 downto 0);
12              Y : out bit_vector(1 downto 0)
13          );
14      end component;
15
16      signal X_tb : bit_vector(1 downto 0);
17      signal Y_tb : bit_vector(1 downto 0);
18
```

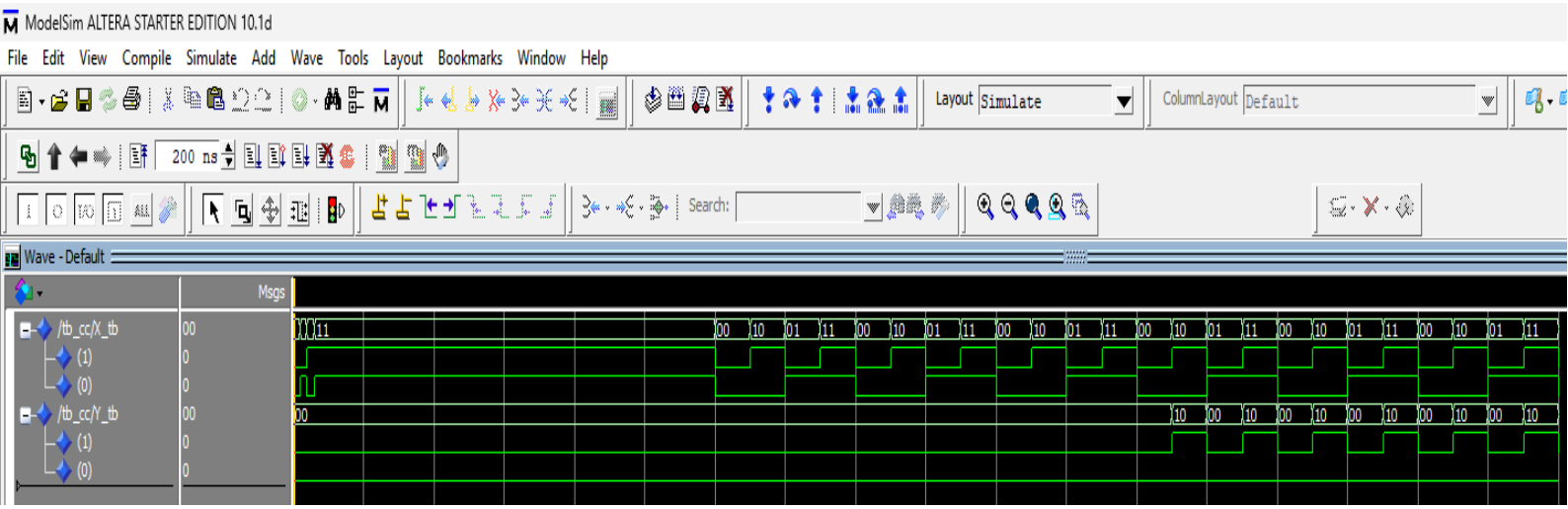
```
19  begin
20
21      DUT: CC
22          port map (
23              X => X_tb,
24              Y => Y_tb
25          );
26
27      process
28          variable i : integer := 0;
29          variable temp : bit_vector(1 downto 0);
30      begin
31          for i in 0 to 3 loop
32              case i is
33                  when 0 => temp := "00";
34                  when 1 => temp := "01";
35                  when 2 => temp := "10";
36                  when 3 => temp := "11";
37                  when others => temp := "00";
38              end case;
39
40              X_tb <= temp;
41              wait for 10 ns;
42          end loop;
43
44          wait;
45      end process;
46
47  end architecture;
48
```

résultats ModelSim

Les résultats sont conformes à la table de vérité d'un Circuit_CC:

X(1)	X(0)	Y(1)	Y(0)
0	0	0	1
0	1	0	1
1	0	0	0
1	1	1	1

Cette table représente toutes les combinaisons possibles de l'entrée X (de "00" à "11") et les valeurs correspondantes de la sortie Y, selon la logique définie dans le code VHDL


ModelSim ALTERA STARTER EDITION 10.1d
File Edit View Compile Simulate Add Wave Tools Layout Bookmarks Window Help
Layout Simulate
ColumnLayout Default
200 ns
Search:
Wave - Default
Msgs
/tb_cc/X_tb
(1)
(0)
/tb_cc/Y_tb
(1)
(0)