Mini ALU

Relazione di progetto

Studenti:
Frega Umberto 239527, frgmrt04a051353d@studenti.unical.it;
Napoli Leonardo 234364, npllrd02s30d086@studenti.unical.it;

Codice Sorgente

Il progetto assegnato consiste nel progettare ed implementare una mini alu, capace di fare addizioni e sottratzioni, tramite linguaggio VHDL. Per la progettazione del sistema si è deciso di utilizzare un pattern comportamentale, andando quindi a definire il comportamento del sistema in base a determinate condizioni, oltretutto si è optato per l'utilizzo del tipo *STD_LOGIC* e quindi *STD_LOGIC_VECTOR* per una maggiore flessibilità e maggiori funzionalità.

Il primo passo della progettazione è stato definire la politica tramite la quale la mini ALU potesse cambiare tra addizione e sottrazione. A questo proposito si è deciso di mantenere un singolo adder, ma cambiare il segno del secondo operando.

1 Adder

1.1 Implementazione

Come componente di base del sistema si è optato per un semplice full-adder. A causa della decisione di approcciare il problema in maniera comportamentale piuttosto che strutturale già nel caso base possiamo vedere l'utilizzo di un assegnamento della variabile *carry_out* tramite condizioni.

1: Codice Full-Adder

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity fulladder_1bit is
5    Port (
6      A : in STD_LOGIC;
7      B : in STD_LOGIC;
8      carry_in : in STD_LOGIC;
9      carry_out : out STD_LOGIC;
10     sum : out STD_LOGIC);
11 end fulladder_1bit;
12
13 architecture Behavioral of fulladder_1bit is
14   signal p: STD_LOGIC;
15 begin
16   p <= A xor B;
17   carry_out <= A when p='0' else
18     carry_in when p='1' else 'X';
19   sum <= p xor carry_in;
20
21 end Behavioral;
```

1.2 Schematica

Il codice precedente ha generato in Vivado la schematica riportata in *Figure 1.* Da notare la creazione di RTL_MUX causata dal blocco condizionale *when else*;

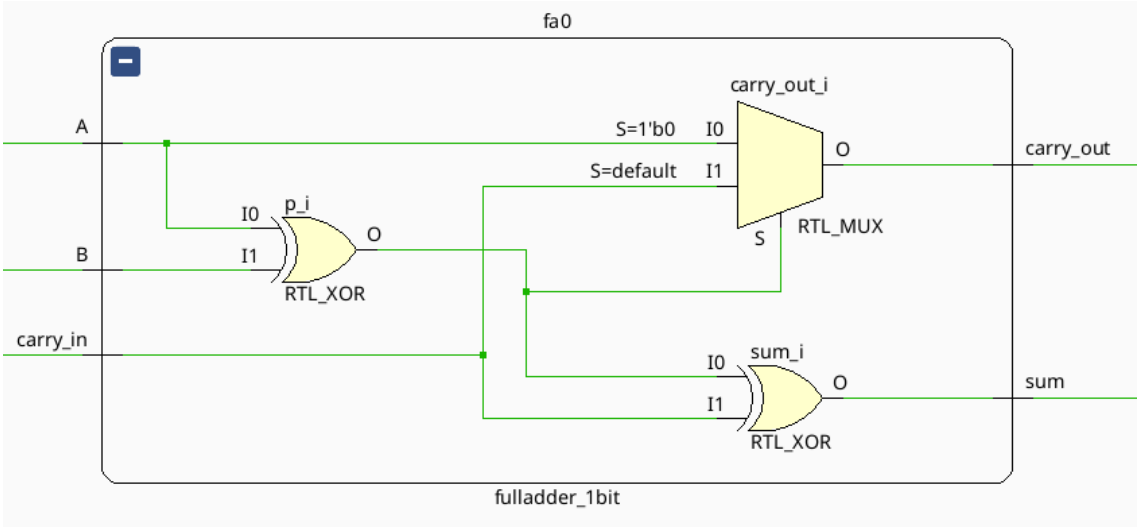

Figure 1: Circuito Logico del Full Adder

1.3 Testbench

Essendo presenti 3 entrate e quindi sole $2^3 = 8$ possibli combinazioni, si è deciso di testare ogni caso possibile. Sono stati implementati anche gli statement *assert* per verificare la correttezza degli output e quindi eseguire dei test.

2: Test Full-Adder

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity testbench_fulladder is
5  end testbench_fulladder;
6
7  architecture Behavioral of testbench_fulladder is
8    component fulladder_1bit is
9      Port (
10       A : in STD_LOGIC;
11       B : in STD_LOGIC;
12       carry_in : in STD_LOGIC;
13       carry_out : out STD_LOGIC;
14       sum : out STD_LOGIC);
15     end component;
16
17   signal Ia,Ib,Icin,Ocout,Osum: STD_LOGIC;
18 begin
19   CUT: fulladder_1bit port map (Ia,Ib,Icin,Ocout,Osum);
20   process begin
```

```
21      --Test 1: A = 0, B = 0, carry_in = 0
22      Ia <= '0'; Ib <= '0'; Icin <= '0';
23      wait for 10ns;
24      assert (Osum  = '0' and Ocout = '0') report "Test 1 Fallito" severity
            error;
25
26      --Test 2: A = 0, B = 0, carry_in = 1
27      Ia <= '0'; Ib <= '0'; Icin <= '1';
28      wait for 10ns;
29      assert (Osum  = '1' and Ocout = '0') report "Test 2 Fallito" severity
            error;
30
31      --Test 3: A = 1, B = 0, carry_in = 0
32      Ia <= '1'; Ib <= '0'; Icin <= '0';
33      wait for 10ns;
34      assert (Osum  = '1' and Ocout = '0') report "Test 3 Fallito" severity
            error;
35
36      --Test 4: A = 1, B = 0, carry_in = 1
37      Ia <= '1'; Ib <= '0'; Icin <= '1';
38      wait for 10ns;
39      assert (Osum  = '0' and Ocout = '1') report "Test 4 Fallito" severity
            error;
40
41      --Test 5: A = 0, B = 1, carry_in = 0
42      Ia <= '0'; Ib <= '1'; Icin <= '0';
43      wait for 10ns;
44      assert (Osum  = '1' and Ocout = '0') report "Test 5 Fallito" severity
            error;
45
46      --Test 6: A = 0, B = 1, carry_in = 1
47      Ia <= '0'; Ib <= '1'; Icin <= '1';
48      wait for 10ns;
49      assert (Osum  = '0' and Ocout = '1') report "Test 6 Fallito" severity
            error;
50
51      --Test 7: A = 1, B = 1, carry_in = 0
52      Ia <= '1'; Ib <= '1'; Icin <= '0';
53      wait for 10ns;
54      assert (Osum  = '0' and Ocout = '1') report "Test 7 Fallito" severity
            error;
55
56      --Test 8: A = 1, B = 1, carry_in = 1
57      Ia <= '1'; Ib <= '1'; Icin <= '1';
58      wait for 10ns;
59      assert (Osum  = '1' and Ocout = '1') report "Test 8 Fallito" severity
            error;
60
61    end process;
62  end Behavioral;
```

1.4 Simulazione

Il risultato della testbench sopra indicata è illustrato nella seguente simulazione:

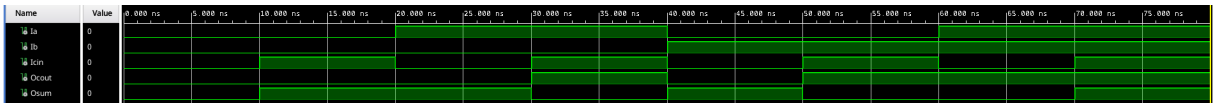

Figure 2: Simulazione Full Adder

2 Ripple-Carry Adder a 4 bit

2.1 Implementazione

L'addizionatore a propagazione del riporto conta al suo interno 4 full adder a 1 bit che lavorano insieme, il *carry_out* finale viene salvato.

3: Codice Ripple Carry

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

entity ripplecarry_4bit is
  Port ( A_4 : in STD_LOGIC_VECTOR (3 downto 0);
    B_4 : in STD_LOGIC_VECTOR (3 downto 0);
    carry_in : in STD_LOGIC;
    carry_out : out STD_LOGIC;
    sum_4 : out STD_LOGIC_VECTOR (3 downto 0));
end ripplecarry_4bit;

architecture Behavioral of ripplecarry_4bit is
  component fulladder_1bit is
    Port (
      A : in STD_LOGIC;
      B : in STD_LOGIC;
      carry_in : in STD_LOGIC;
      carry_out : out STD_LOGIC;
      sum : out STD_LOGIC);
  end component;

  signal sum, carry: STD_LOGIC_VECTOR(3 downto 0);
begin
  fa0: fulladder_1bit PORT MAP( A_4(0), B_4(0), carry_in, carry(0), sum_4
      (0) );
  fa1: fulladder_1bit PORT MAP( A_4(1), B_4(1), carry(0), carry(1), sum_4
      (1) );
  fa2: fulladder_1bit PORT MAP( A_4(2), B_4(2), carry(1), carry(2), sum_4
      (2) );
  fa3: fulladder_1bit PORT MAP( A_4(3), B_4(3), carry(2), carry(3), sum_4
      (3) );
  carry_out <= carry(3);
end Behavioral;
```

2.2 Schematica

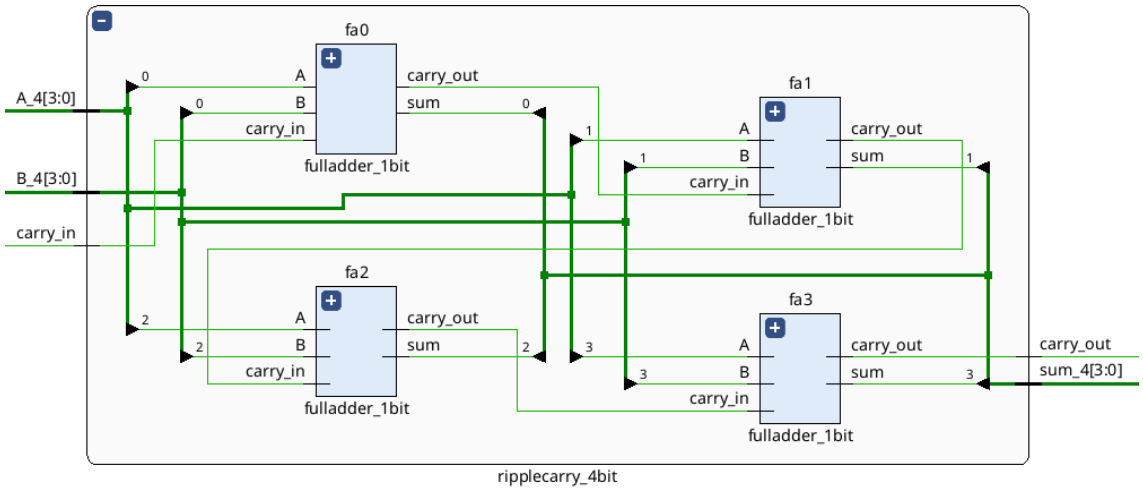

Figure 3: Circuito Logico del Ripple Carry Adder

2.3 TestBench

Nella fase di testing essendo impossibile testare ogni caso sono stati implementati i test soltanto di alcune situazioi notevoli.

4: Codice Ripple Carry

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity testbench_ripplecarry is
5  end testbench_ripplecarry;
6
7  architecture Behavioral of testbench_ripplecarry is
8    component ripplecarry_4bit is
9      Port (
10     A_4 : in STD_LOGIC_VECTOR (3 downto 0);
11     B_4 : in STD_LOGIC_VECTOR (3 downto 0);
12     carry_in : in STD_LOGIC;
13     carry_out : out STD_LOGIC;
14     sum_4 : out STD_LOGIC_VECTOR (3 downto 0));
15   end component;
16
17   signal Ia,Ib,Osum: std_logic_vector (3 downto 0);
18   signal Icin,Ocout:std_logic;
19
20 begin
21   CUT: ripplecarry_4bit port map (Ia,Ib,Icin,Ocout ,Osum);
22   process begin
23     --Test 1: caso base, A = 0, B = 0, carry = 0
24     Ia <= "0000"; Ib <= "0000"; Icin <= '0';
25     wait for 10ns;
```

```
26      assert ( Osum = "0000" and Ocout = '0') report "Test case 1 Failed"
            severity error;
27
28      --Test 2: A = 1, B = 1, carry = 0
29      Ia <= "0001"; Ib <= "0001"; Icin <= '0';
30      wait for 10ns;
31      assert ( Osum = "0010" and Ocout = '0') report "Test case 2 Failed"
            severity error;
32
33      --Test case 3: A = 0101, B = 0011, carry_in = 0
34      Ia <= "0101"; Ib <= "0011"; Icin <= '0';
35      wait for 10 ns;
36      assert (Osum = "1000" and Ocout= '0') report "Test case 3 failed";
37
38      -- Test case 4: A = 1111, B = 0001, carry_in = 0
39      Ia <= "1111"; Ib <= "0001"; Icin <= '0';
40      wait for 10 ns;
41      assert (Osum = "0000" and Ocout = '1') report "Test case 4 failed"
            severity error;
42
43      -- Test case 5: Ia = 1111, Ib = 1111, cin = 0
44      Ia <= "1111"; Ib <= "1111"; Icin <= '0';
45      wait for 10 ns;
46      assert (Osum = "1110" and Ocout = '1') report "Test case 5 failed"
            severity error;
47
48      -- Test case 6: Ia = 1010, Ib = 0101, cin = 1
49      Ia <= "1010"; Ib <= "0101"; Icin <= '1';
50      wait for 10 ns;
51      assert (Osum = "0000" and Ocout = '1') report "Test case 6 failed"
            severity error;
52
53      -- Test case 7: Ia = 1001, Ib = 1001, cin = 0
54      Ia <= "1001"; Ib <= "1001"; Icin <= '0';
55      wait for 10 ns;
56      assert (Osum = "0010" and Ocout = '1') report "Test case 7 failed"
            severity error;
57
58      -- Test case 8: Ia = 0110, Ib = 1001, cin = 1
59      Ia <= "0110"; Ib <= "1001"; Icin <= '1';
60      wait for 10 ns;
61      assert (Osum = "0000" and Ocout = '1') report "Test case 8 failed"
            severity error;
62
63      end process;
64  end Behavioral;
```

2.4 Simulazione

Il risultato della testbench sopra indicata è illustrato nella seguente simulazione:

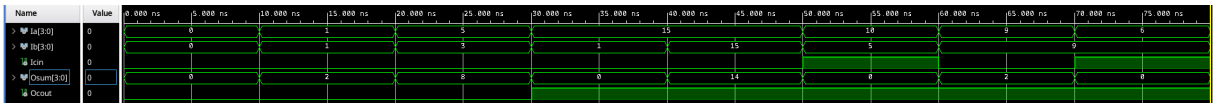

Figure 4: Simulazione Ripple Carry

3 Carry-Select Adder

3.1 Implementazione

Nella fase iniziale vengono istanziati i vari addizionatori Ripple Carry con le loro varianti con *carry_in* '0' e '1', con il costrutto *when else* viene selezionato quale risultato delle somme è quello da utilizzare, la variabile che ha il ruolo di selezionatore è *carry_selector* che è assegnata al valore del riporto dell'addizionatore precedente, è una variabile shared perché deve essere acceduta in più di un processo. Si è optato per una assegnazione di tipo dichiarativa piuttosto che posizionale, producendo un codice più lungo ma anche più robusto. Per il risultato si è deciso di utilizzare un vettore di *STD_LOGIC* a 17 bit piuttosto che a 16 allo scopo di prevenire l'overflow.

5: Codice Carry Select

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

entity carry_select_16bit is
  Port (
    A : in STD_LOGIC_VECTOR (15 downto 0);
    B : in STD_LOGIC_VECTOR (15 downto 0);
    carry_in_start: in STD_LOGIC;
    sum : out STD_LOGIC_VECTOR (16 downto 0);
end carry_select_16bit;

architecture Behavioral of carry_select_16bit is

  component ripplecarry_4bit is
    Port (
      A_4 : in STD_LOGIC_VECTOR (3 downto 0);
      B_4 : in STD_LOGIC_VECTOR (3 downto 0);
      carry_in : in STD_LOGIC;
      carry_out : out STD_LOGIC;
      sum_4 : out STD_LOGIC_VECTOR (3 downto 0));
  end component;

  shared variable carry_selector: STD_LOGIC;
  signal carry_start: STD_LOGIC;
  signal carry0, carry1: STD_LOGIC_VECTOR(2 downto 0);
  signal sum_0, sum_1: STD_LOGIC_VECTOR(12 downto 0);

begin
  ripplecarry0_0: ripplecarry_4bit PORT MAP (
    A_4 => A(3 downto 0),
    B_4 => B(3 downto 0),
    carry_in => carry_in_start,
    carry_out => carry_start,
```

```
    sum_4 => sum(3 downto 0));

  ripplecarry1_0: ripplecarry_4bit PORT MAP (
    A_4 => A(7 downto 4),
    B_4 => B(7 downto 4),
    carry_in => '0',
    carry_out => carry0(0),
    sum_4 => sum_0(3 downto 0));
  ripplecarry1_1: ripplecarry_4bit PORT MAP (
    A_4 => A(7 downto 4),
    B_4 => B(7 downto 4),
    carry_in => '1',
    carry_out => carry1(0),
    sum_4 => sum_1(3 downto 0));

  ripplecarry2_0: ripplecarry_4bit PORT MAP (
    A_4 => A(11 downto 8),
    B_4 => B(11 downto 8),
    carry_in => '0',
    carry_out => carry0(1),
    sum_4 => sum_0(7 downto 4));
  ripplecarry2_1: ripplecarry_4bit PORT MAP (
    A_4 => A(7 downto 4),
    B_4 => B(7 downto 4),
    carry_in => '1',
    carry_out => carry1(1),
    sum_4 => sum_1(7 downto 4));

  ripplecarry3_0: ripplecarry_4bit PORT MAP (
    A_4 => A(15 downto 12),
    B_4 => B(15 downto 12),
    carry_in => '0',
    carry_out => carry0(2),
    sum_4 => sum_0(11 downto 8));
  ripplecarry3_1: ripplecarry_4bit PORT MAP (
    A_4 => A(15 downto 12),
    B_4 => B(15 downto 12),
    carry_in => '1',
    carry_out => carry1(2),
    sum_4 => sum_1(11 downto 8));

  process(carry_start, sum_0, sum_1, carry_start) begin
    case carry_start is
      when '0' =>
        sum( 7 downto 4 ) <= sum_0(3 downto 0);
        carry_selector := carry0(0);
      when  others =>
        sum( 7 downto 4 ) <= sum_1(3 downto 0);
        carry_selector := carry1(0);
    end case;

    case carry_selector is
      when '0' =>
        sum( 11 downto 8 ) <= sum_0(7 downto 4);
        carry_selector := carry0(1);
      when  others =>
        sum( 11 downto 8 ) <= sum_1(7 downto 4);
```

```
91          carry_selector := carry1(1);
92        end case;
93
94        case carry_selector is
95          when '0' =>
96            sum( 15 downto 12 ) <= sum_0(11 downto 8);
97            carry_selector := carry0(2);
98          when  others =>
99            sum( 15 downto 12 ) <= sum_1(11 downto 8);
100           carry_selector := carry1(2);
101       end case;
102         sum(16) <= carry_selector;
103     end process;
104 end Behavioral;
```

3.2 Schematica

La schematica del carry select adder comprende 7 ripplecarry adder a 4 bit e 6 multiplexer. Degno di nota è che nel codice non è presente alcun multiplexer, questi vengono infatti generati automaticamente a partire dalle istruzioni condizionali.

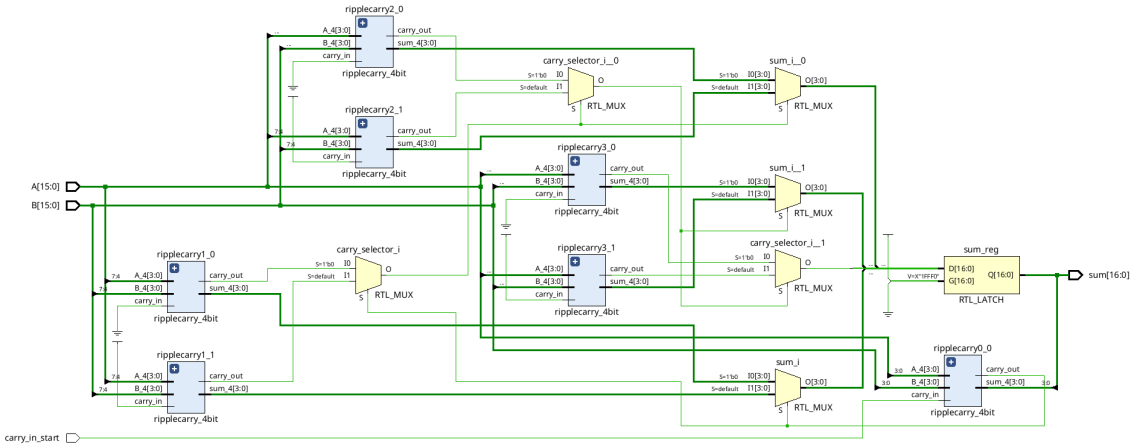

Figure 5: Circuito Logico del Carry Select Adder

3.3 Testbench

Discorso analogo a quello del Ripple Carry Adder.

6: Codice Carry Select

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3
4  entity testbench_carryselect is
5  end testbench_carryselect;
6
7  architecture Behavioral of testbench_carryselect is
```

```
8     component carry_select_16bit is
9       Port (
10        A : in STD_LOGIC_VECTOR (15 downto 0);
11        B : in STD_LOGIC_VECTOR (15 downto 0);
12        carry_in_start: in STD_LOGIC;
13        sum : out STD_LOGIC_VECTOR (16 downto 0));
14    end component;
15    signal Ia,Ib: STD_LOGIC_VECTOR (15 downto 0);
16    signal Os: STD_LOGIC_VECTOR (16 downto 0);
17    signal Icin: STD_LOGIC;
18  begin
19    CUT: carry_select_16bit port map(Ia,Ib,Icin,Os);
20
21    process begin
22      -- Test 1: A = 0000000000000000, B = 0000000000000000, carry_in = 0
23      Ia <= (others => '0');
24      Ib <= (others => '0');
25      Icin <= '0';
26      wait for 10 ns;
27      assert (Os = "00000000000000000")
28      report "Test 1 Fallito: La somma di zero deve essere zero." severity
            error;
29
30      -- Test 2: A = 0000000000000000, B = 0000000000000000, carry_in = 1
31      Ia <= (others => '0');
32      Ib <= (others => '0');
33      Icin <= '1';
34      wait for 10 ns;
35      assert (Os = "00000000000000001")
36      report "Test 2 Fallito: La somma di zero e riporto deve essere uno."
            severity error;
37
38      -- Test 3: A = 0111111111111111, B = 0000000000000001, carry_in = 0
39      Ia <= "0111111111111111";  -- 32767
40      Ib <= "0000000000000001";  -- 1
41      Icin <= '0';
42      wait for 10 ns;
43      assert (Os = "10000000000000000")
44      report "Test 3 Fallito: Somma deve essere 32768 (overflow positivo)."
            severity error;
45
46      -- Test 4: A = 0111111111111111, B = 0111111111111111, carry_in = 0
47      Ia <= "0111111111111111";  -- 32767
48      Ib <= "0111111111111111";  -- 32767
49      Icin <= '0';
50      wait for 10 ns;
51      assert (Os = "11111111111111110")
52      report "Test 4 Fallito: Somma deve essere 65534 (overflow positivo)."
            severity error;
53
54      -- Test 5: A = 0000000000000001, B = 0000000000000001, carry_in = 1
55      Ia <= "0000000000000001";  -- 1
56      Ib <= "0000000000000001";  -- 1
57      Icin <= '1';
58      wait for 10 ns;
59      assert (Os = "00000000000000011") report "Test 5 Fallito: Somma deve
            essere 3 con riporto." severity error;
```

```
60
61      -- Test 6: A = 0000000000000000, B = 1111111111111111, carry_in = 0
62      Ia <= "0000000000000000";
63      Ib <= "1111111111111111";  -- 65535
64      Icin <= '0';
65      wait for 10 ns;
66      assert (Os = "01111111111111111") report "Test 6 Fallito: Somma deve
            essere 65535." severity error;
67
68      -- Test 7: A = 1111111111111111, B = 1111111111111111, carry_in = 1
69      Ia <= "1111111111111111";  -- 65535
70      Ib <= "1111111111111111";  -- 65535
71      Icin <= '1';
72      wait for 10 ns;
73      assert (Os = "11111111111111110")
74      report "Test 7 Fallito: Somma deve essere 65535 con overflow e riporto
            ." severity error;
75
76      -- Test 8: A = 1111111111111111, B = 0000000000000001, carry_in = 0
77      Ia <= "1111111111111111";  -- 65535
78      Ib <= "0000000000000001";  -- 1
79      Icin <= '0';
80      wait for 10 ns;
81      assert (Os = "10000000000000000") report "Test 8 Fallito: Somma deve
            essere 0 (overflow)." severity error;
82
83      --Test Finiti
84      report "Tutti i test sono stati completati con successo!" severity
            note;
85    end process;
86  end Behavioral;
```

3.4 Simulazione

Il risultato della testbench sopra indicata è illustrato nella seguente simulazione:

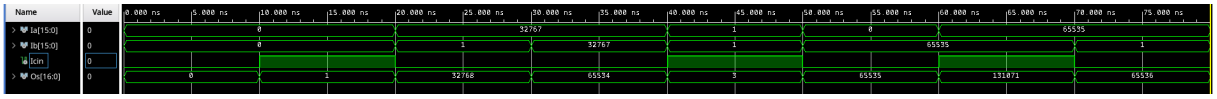

Figure 6: Simulazione Carry Select