Elettronica Digitale

2° Progetto: Moltiplicatore di numeri positivi a otto bit

Studente: Tassone Roberto
Matricola: 149351

Descrizione del circuito e dei componenti

Per la realizzazione di un moltiplicatore di numeri positivi a otto bit (in complemento a due) si è scelto di utilizzare l'algoritmo di Booth. Questo permette di ottimizzare le prestazioni del circuito in quanto si riduce il numero dei prodotti parziali generati. Nel caso di numeri a otto bit si avranno esattamente quattro prodotti parziali.

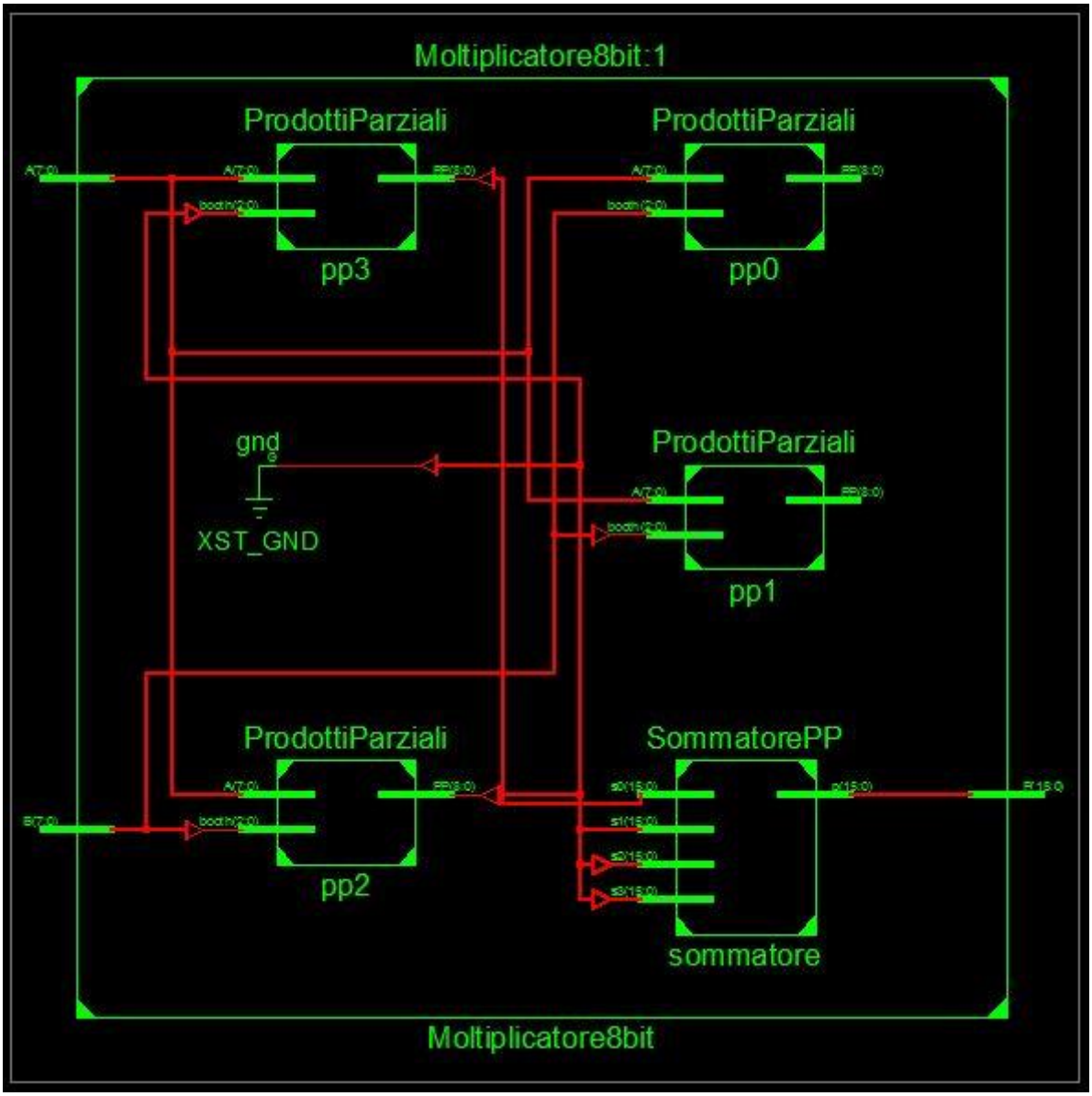
Il modulo **ProdottiParziali** riceve i tre bit delle terne del moltiplicatore B e li passa ad un MUX che li usa come segnali di controllo per generare i prodotti parziali a nove bit. Prima dell'uso del MUX è necessario, partendo dal moltiplicando A, creare alcuni segnali utili, tra cui abbiamo: 2A (realizzato con uno shift sinistro) e -A (realizzato con una NOT e sommando uno al numero ottenuto). Per l'aritmetica dei segnali è necessario importare due librerie: *IEEE.STD_LOGIC_UNSIGNED.ALL* e *IEEE.STD_LOGIC_ARITH.ALL*. Prima di eseguire la somma, questi prodotti parziali devono essere estesi a sedici bit con l'aggiunta di bit di segno e di shift sinistri, seguendo opportunamente l'algoritmo di Booth:

$P(A,C) = \sum_{i=0}^{\frac{n}{2}-1} C_i * A * 2^{2i}$ (C corrisponde alla terna codificata secondo la tabella di Booth)

Una volta che tutti i prodotti parziali sono stati estesi a sedici bit, si può effettuare la loro somma. Si è scelto di creare un modulo **SommatorePP** che effettui la somma dei quattro prodotti parziali. Al suo interno conterrà due **CarrySave a sedici bit** ed un **CarrySelect a sedici bit**. Il primo CarrySave sommerà i primi tre prodotti parziali restituendo un vettore delle somme ed uno dei riporti. Il secondo invece riceverà i due vettori generati dal primo CarrySave (il vettore dei riporti va sempre shiftato a sinistra di una posizione) ed il quarto prodotto parziale, restituendo un secondo vettore delle somme ed uno dei riporti. Questi due vettori vengono infine sommati utilizzando un CarrySelect a sedici bit che è composto da tre CarrySelect a otto bit. Tutti i sommatori utilizzati hanno ingressi ed uscite a sedici bit in quanto, per costruzione, il risultato finale non potrà mai superare i sedici bit.

Struttura e codice VHDL dei componenti

Schema di Moltiplicatore8bit

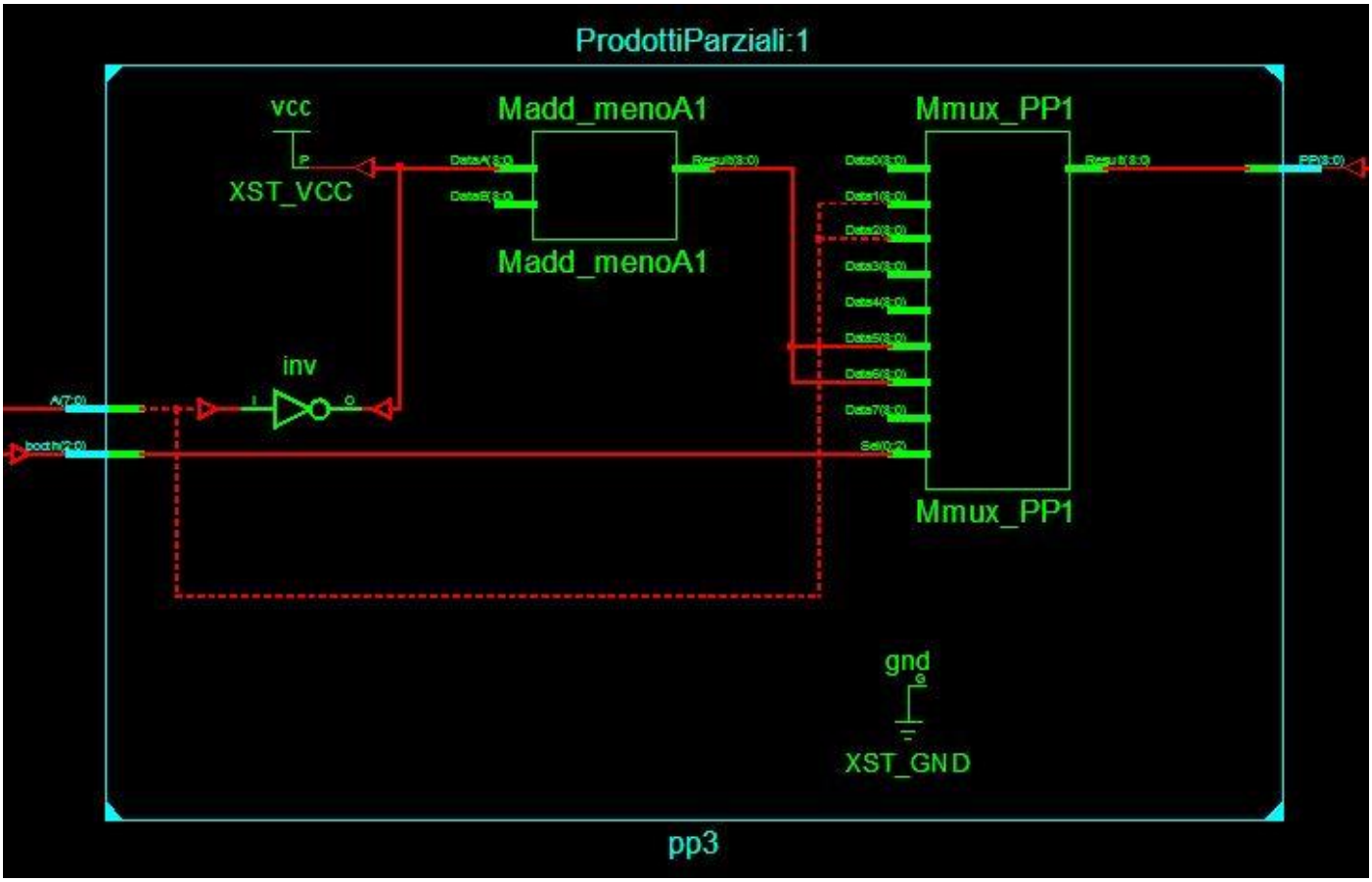


VHDL di Moltiplicatore8bit

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

entity Moltiplicatore8bit is
   Port ( A : in  STD_LOGIC_VECTOR (7 downto 0);
          B : in  STD_LOGIC_VECTOR (7 downto 0);
          P : out  STD_LOGIC_VECTOR (15 downto 0));
end Moltiplicatore8bit;

architecture Structural of Moltiplicatore8bit is

component ProdottiParziali is
```

```
   Port ( booth : in  STD_LOGIC_VECTOR (2 downto 0);
          A : in  STD_LOGIC_VECTOR (7 downto 0);
          PP : out  STD_LOGIC_VECTOR (8 downto 0));
end component;
component SommatorePP is
   Port ( s0 : in  STD_LOGIC_VECTOR (15 downto 0);
          s1 : in  STD_LOGIC_VECTOR (15 downto 0);
          s2 : in  STD_LOGIC_VECTOR (15 downto 0);
          s3 : in  STD_LOGIC_VECTOR (15 downto 0);
          p : out  STD_LOGIC_VECTOR (15 downto 0));
end component;

signal booth4: std_logic_vector(2 downto 0);
signal s0, s1, s2, s3 : std_logic_vector(8 downto 0);
signal s0fixed, s1fixed, s2fixed, s3fixed : std_logic_vector(15 downto 0) := (others => '0');

begin
-- Creo l'ultima terna di booth.
booth4(2 downto 1)<=b(1 downto 0);
booth4(0)<='0';
-- Calcolo i prodotti parziali a 9 bit.
pp0: ProdottiParziali port map(B(7 downto 5),  A,  s3);
pp1: ProdottiParziali port map(B(5 downto 3),  A,  s2);
pp2: ProdottiParziali port map(B(3 downto 1),  A,  s1);
pp3: ProdottiParziali port map(booth4, A, s0);
-- Aggiusto i prodotti parziali portandoli a 16 bit.
s0fixed(8 downto 0)<=s0;
with s0(8) select
        s0fixed(15 downto 9)<="1111111" when '1',
                              "0000000" when '0',
                              "XXXXXXX" when others;
s1fixed(10 downto 2)<=s1;
with s1(8) select
        s1fixed(15 downto 11)<="11111" when '1',
                               "00000" when '0',
                               "XXXXX" when others;
s2fixed(12 downto 4)<=s2;
with s2(8) select
        s2fixed(15 downto 13)<="111" when '1',
                               "000" when '0',
                               "XXX" when others;
s3fixed(14 downto 6)<=s3;
s3fixed(15)<=s3(8);
-- Sommo i prodotti parziali.
sommatore: SommatorePP port map(s0fixed, s1fixed, s2fixed, s3fixed, P);

end Structural;
```

Schema di ProdottiParziali

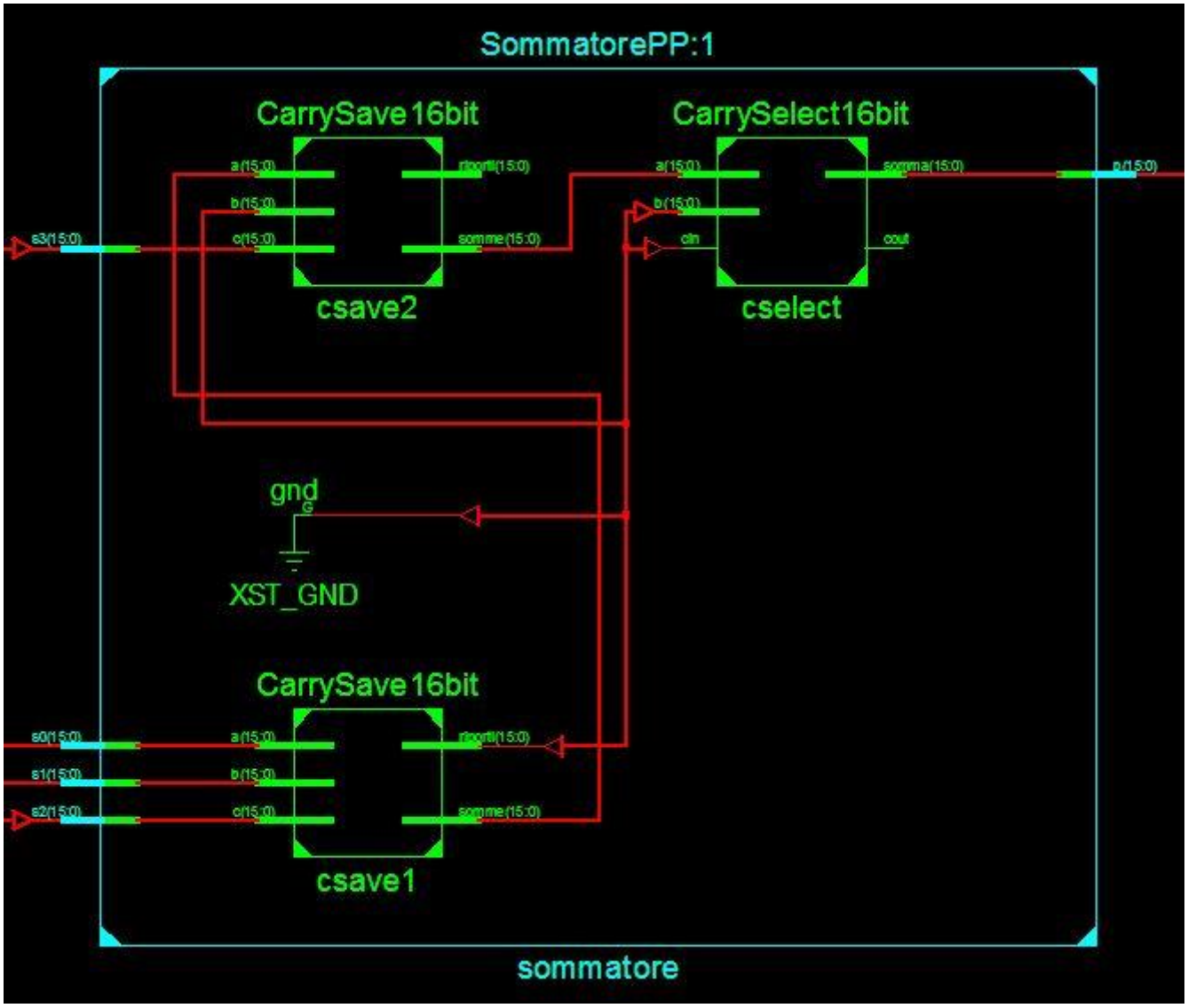


VHDL di ProdottiParziali

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

entity ProdottiParziali is
   Port ( booth : in  STD_LOGIC_VECTOR (2 downto 0);
          A : in  STD_LOGIC_VECTOR (7 downto 0);
          PP : out  STD_LOGIC_VECTOR (8 downto 0));
end ProdottiParziali;

architecture Behavioral of ProdottiParziali is

signal dueA, menodueA, extA, menoA, notA: std_logic_vector(8 downto 0) := (others => '0');

begin
extA(7 downto 0)<=A;
notA<=not extA;
dueA(8 downto 1)<=extA(7 downto 0);
menoA<=std_logic_vector(unsigned(notA+1));
menodueA(8 downto 1)<=menoA(7 downto 0);

with booth(2 downto 0) select
            PP<="000000000" when "000",
                  extA when "001",
                  extA when "010",
```

```
                dueA when "011",
                menodueA when "100",
                menoA when "101",
                menoA when "110",
                "000000000" when "111",
                "XXXXXXXXX" when others;

end Behavioral;
```

Schema di SommatorePP

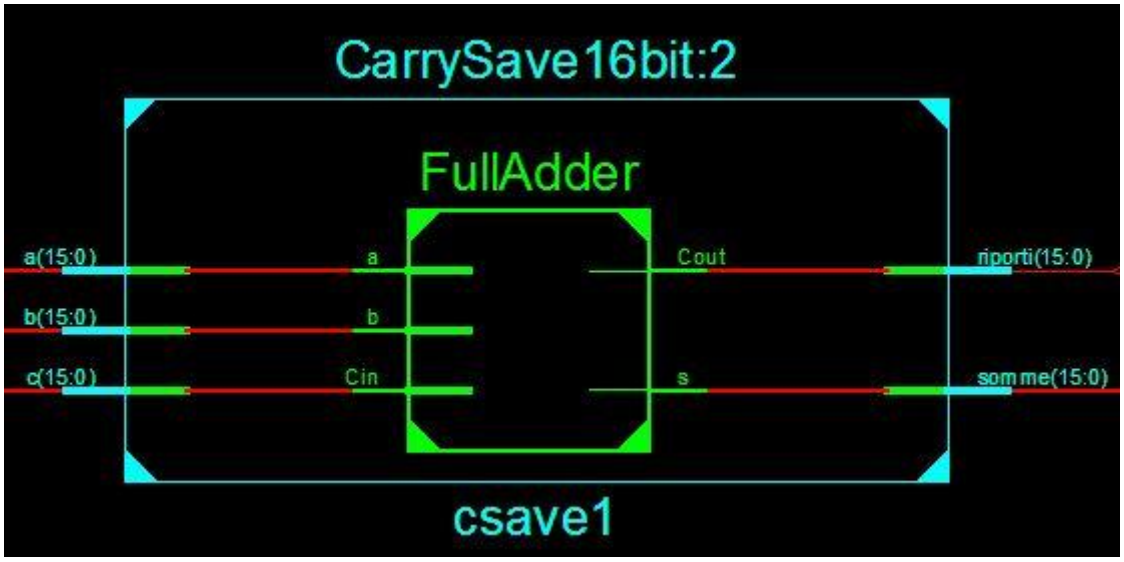


VHDL di SommatorePP

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

entity SommatorePP is
   Port ( s0 : in  STD_LOGIC_VECTOR (15 downto 0);
          s1 : in  STD_LOGIC_VECTOR (15 downto 0);
          s2 : in  STD_LOGIC_VECTOR (15 downto 0);
          s3 : in  STD_LOGIC_VECTOR (15 downto 0);
          p : out  STD_LOGIC_VECTOR (15 downto 0));
end SommatorePP;
```

```
architecture Structural of SommatorePP is

component CarrySave16bit is
   Port ( a : in  STD_LOGIC_VECTOR (15 downto 0);
          b : in  STD_LOGIC_VECTOR (15 downto 0);
          c : in  STD_LOGIC_VECTOR (15 downto 0);
          somme : out  STD_LOGIC_VECTOR (15 downto 0);
          riporti : out  STD_LOGIC_VECTOR (15 downto 0));
end component;

component CarrySelect16bit is
   Port ( a : in  STD_LOGIC_VECTOR (15 downto 0);
          b : in  STD_LOGIC_VECTOR (15 downto 0);
          cin : in  STD_LOGIC;
          somma : out  STD_LOGIC_VECTOR (15 downto 0);
          cout : out  STD_LOGIC);
end component;

signal somme1, riporti1, somme2, riporti2, riporti1fixed, riporti2fixed: std_logic_vector(15 downto 0) :=
(others => '0');
-- Segnale corrispondente al riporto della moltiplicazione.
-- Inutilizzato perché non si potrà mai avere riporto oltre i 16 bit moltiplicando due numeri a 8 bit.
signal unused: std_logic;

begin
-- Sommo i primi tre prodotti parziali usando un CarrySave a 16 bit.
csave1: CarrySave16bit port map(s0, s1, s2, somme1, riporti1);
riporti1fixed(15 downto 1)<=riporti1(14 downto 0);
-- Sommo gli output del CarrySave ed il quarto prodotto parziale con un CarrySave a 16 bit.
csave2: CarrySave16bit port map(somme1, riporti1fixed, s3, somme2, riporti2);
riporti2fixed(15 downto 1)<=riporti2(14 downto 0);
-- Sommo il vettore delle somme e quello dei riporti con un CarrySelect a 16 bit.
cselect: CarrySelect16bit port map(somme2, riporti2fixed, '0', p, unused);

end Structural;
```

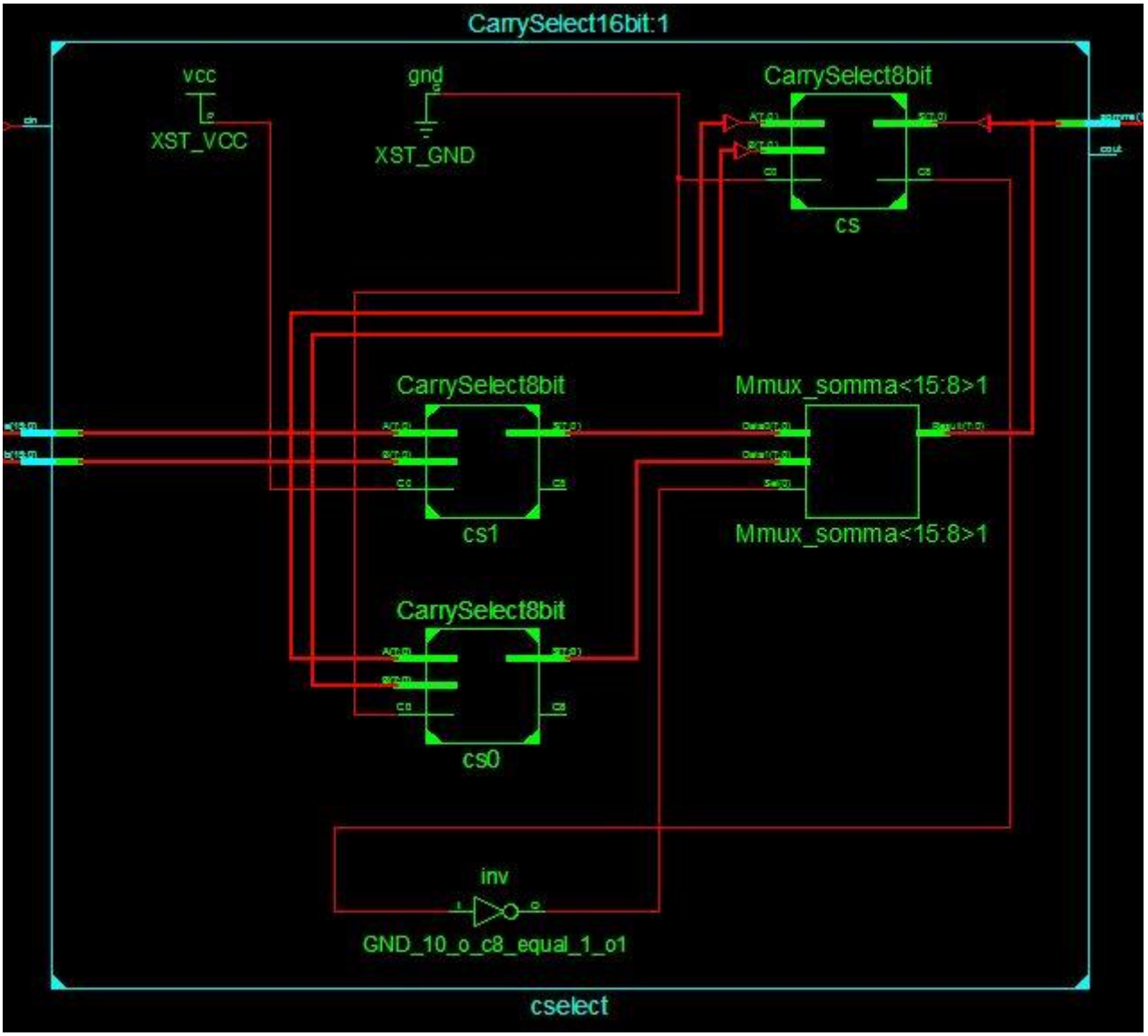
Schema di CarrySave16bit



VHDL di CarrySave16bit

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

entity CarrySave16bit is
   Port ( a : in  STD_LOGIC_VECTOR (15 downto 0);
          b : in  STD_LOGIC_VECTOR (15 downto 0);
          c : in  STD_LOGIC_VECTOR (15 downto 0);
          somme : out  STD_LOGIC_VECTOR (15 downto 0);
          riporti : out  STD_LOGIC_VECTOR (15 downto 0));
end CarrySave16bit;
architecture Structural of CarrySave16bit is

component FullAdder is
 Port ( a, b, Cin : in STD_LOGIC;
        s, Cout :out STD_LOGIC);
end component;

begin
FA0: FullAdder port map(a(0), b(0), c(0), somme(0), riporti(0));
FA1: FullAdder port map(a(1), b(1), c(1), somme(1), riporti(1));
FA2: FullAdder port map(a(2), b(2), c(2), somme(2), riporti(2));
FA3: FullAdder port map(a(3), b(3), c(3), somme(3), riporti(3));
FA4: FullAdder port map(a(4), b(4), c(4), somme(4), riporti(4));
FA5: FullAdder port map(a(5), b(5), c(5), somme(5), riporti(5));
FA6: FullAdder port map(a(6), b(6), c(6), somme(6), riporti(6));
FA7: FullAdder port map(a(7), b(7), c(7), somme(7), riporti(7));
FA8: FullAdder port map(a(8), b(8), c(8), somme(8), riporti(8));
FA9: FullAdder port map(a(9), b(9), c(9), somme(9), riporti(9));
FA10: FullAdder port map(a(10), b(10), c(10), somme(10), riporti(10));
FA11: FullAdder port map(a(11), b(11), c(11), somme(11), riporti(11));
FA12: FullAdder port map(a(12), b(12), c(12), somme(12), riporti(12));
FA13: FullAdder port map(a(13), b(13), c(13), somme(13), riporti(13));
FA14: FullAdder port map(a(14), b(14), c(14), somme(14), riporti(14));
FA15: FullAdder port map(a(15), b(15), c(15), somme(15), riporti(15));
end Structural;
```

Schema di CarrySelect16bit



VHDL di CarrySelect16bit

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity CarrySelect16bit is
   Port ( a : in  STD_LOGIC_VECTOR (15 downto 0);
          b : in  STD_LOGIC_VECTOR (15 downto 0);
          cin : in  STD_LOGIC;
          somma : out  STD_LOGIC_VECTOR (15 downto 0);
          cout : out  STD_LOGIC);
end CarrySelect16bit;

architecture Structural of CarrySelect16bit is

-- Utilizzo il CarrySelect a 8 bit realizzato nel primo progetto.
component CarrySelect8bit is
 Port ( A, B: in STD_LOGIC_VECTOR (7 downto 0);
        C0 : in STD_LOGIC;
        S : out STD_LOGIC_VECTOR (7 downto 0);
        C8 : out STD_LOGIC);
end component;
```

```
signal s, s0, s1: std_logic_vector(7 downto 0);
signal c8, c16_0, c16_1: std_logic;

begin

cs: CarrySelect8bit port map(a(7 downto 0), b(7 downto 0), cin, somma(7 downto 0), c8);
cs0: CarrySelect8bit port map(a(15 downto 8), b(15 downto 8), '0', s0, c16_0);
cs1: CarrySelect8bit port map(a(15 downto 8), b(15 downto 8), '1', s1, c16_1);

with c8 select
        somma(15 downto 8)<=s0 when '0',
                                s1 when '1',
                                "XXXXXXXX" when others;
with c8 select
        cout<=c16_0 when '0',
                c16_1 when '1',
                'X' when others;

end Structural;
```

Test bench

Viene testato il corretto funzionamento del moltiplicatore fornendo degli input e verificandone l'uscita. Dal test si evince che il circuito svolge il suo compito senza errori.

VHDL del test bench

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

ENTITY TestMoltiplicatore8bit IS
END TestMoltiplicatore8bit;

ARCHITECTURE behavior OF TestMoltiplicatore8bit IS
  COMPONENT Moltiplicatore8bit
  PORT(A : IN  std_logic_vector(7 downto 0);
       B : IN  std_logic_vector(7 downto 0);
       P : OUT  std_logic_vector(15 downto 0) );
  END COMPONENT;

 signal A : std_logic_vector(7 downto 0) := (others => '0');
 signal B : std_logic_vector(7 downto 0) := (others => '0');
 signal P : std_logic_vector(15 downto 0);

BEGIN
 uut: Moltiplicatore8bit PORT MAP (
      A => A,
      B => B,
      P => P
    );
```

```
  process
  begin
                A <= "00000000";
                B <= "00000000";
                wait for 10ns;
                A <= "00100101";
                B <= "00000001";
                wait for 10ns;
                A <= "00001111";
                B <= "00001111";
                wait for 10ns;
                A <= "01111111";
                B <= "00000011";
                wait for 10ns;
                A <= "01111111";
                B <= "01111111";
                wait for 10ns;
  end process;
END;
```

Schema dei segnali di input e output