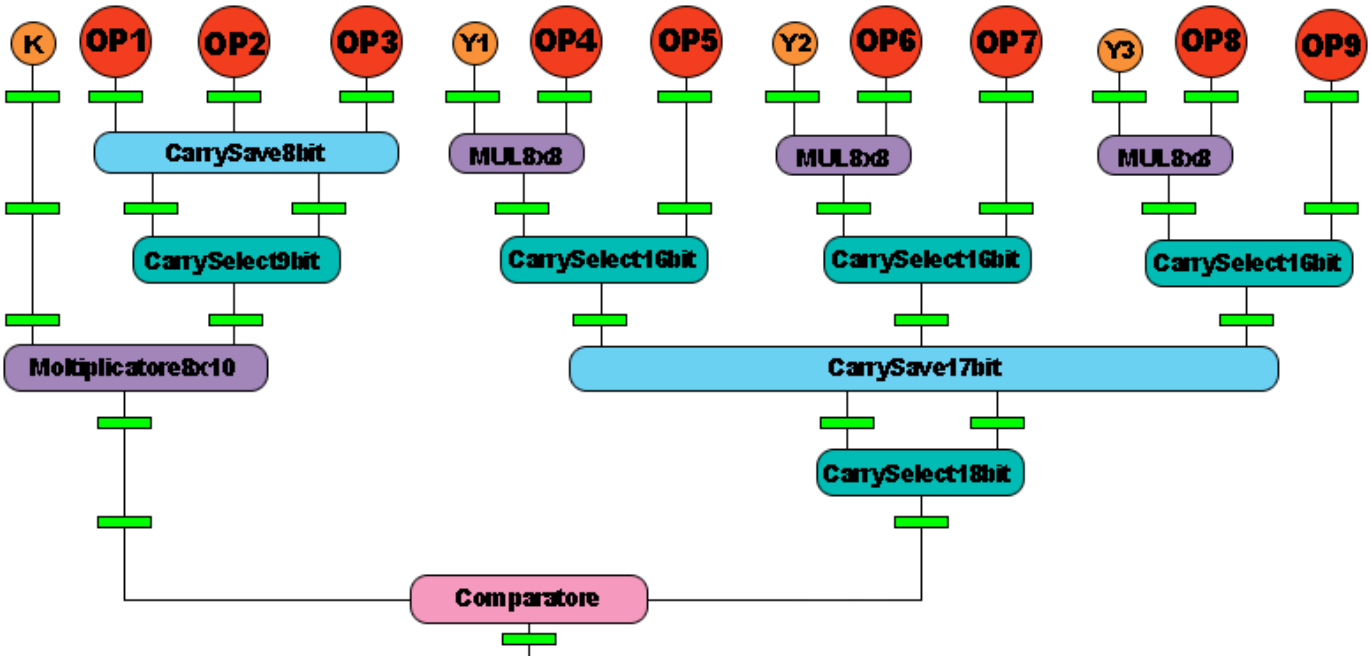
Elettronica Digitale

3° Progetto: Circuito sequenziale

Studente: Tassone Roberto
Matricola: 149351

Descrizione del circuito e dei componenti

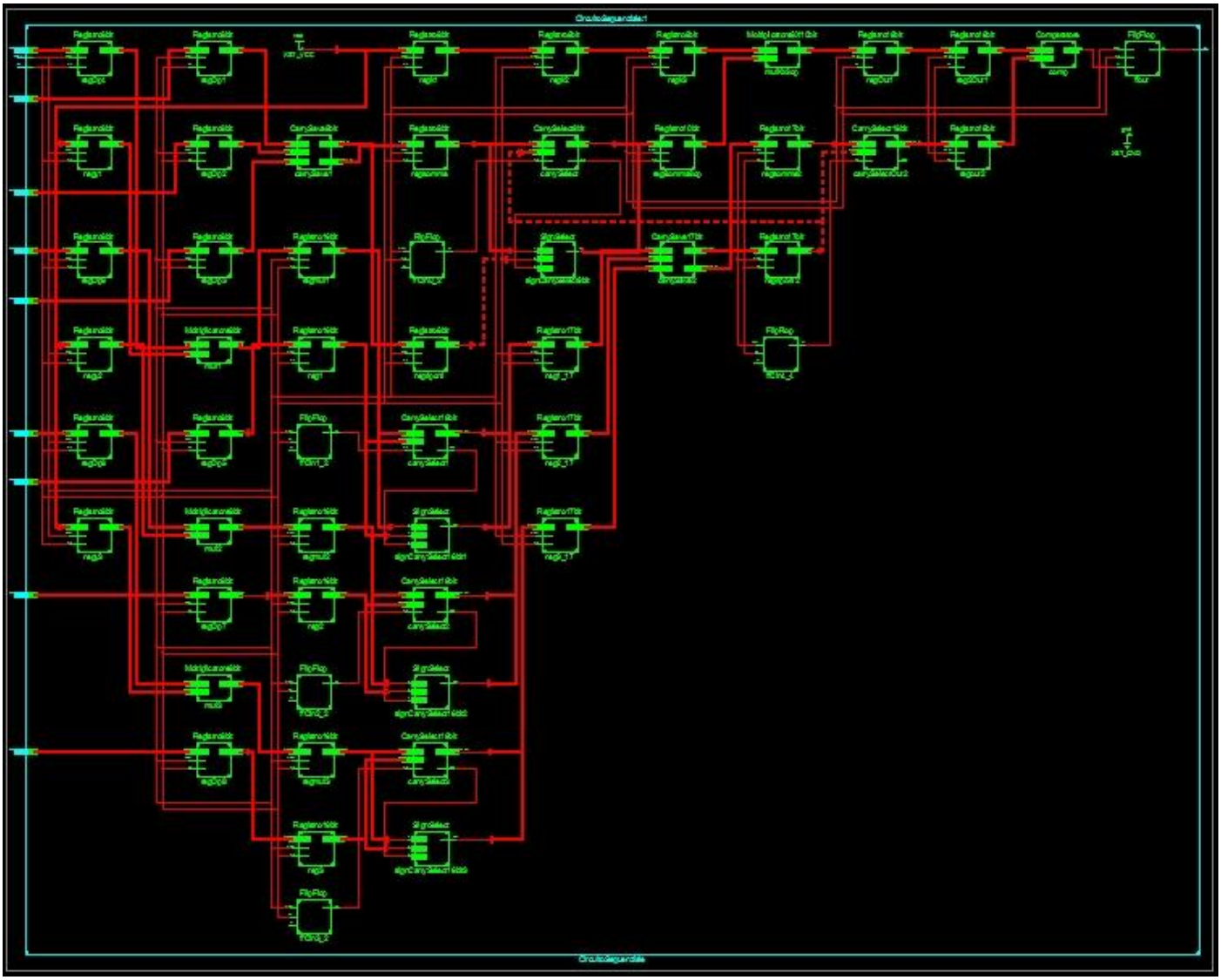
Il circuito sequenziale realizzato riceve in input nove operandi a 8 bit, esegue delle operazioni e restituisce come output un bit. Le operazioni che deve eseguire sugli operandi possono essere divise in due rami: il ramo sinistro somma i primi tre operandi e moltiplica il risultato per un numero costante **k=85**; il ramo destro elabora a due a due gli altri sei operandi, in particolare prende un operando, lo moltiplica per un numero costante **y** e ne somma il prodotto con l'operando successivo. Le tre costanti utilizzare nel ramo destro sono: **y1=103**, **y2=71**, **y3=17**. Sempre nel ramo destro, i tre risultati ottenuti precedentemente vengono a loro volta sommati. Infine viene fatto un confronto tra i risultati dei due rami e viene restituito, come output del circuito, un bit uguale a zero se il risultato del ramo sinistro è maggiore del risultato di destra, uno altrimenti. Di seguito è illustrato lo schema delle operazioni che il circuito dovrà eseguire.



Come si evince dallo schema è necessario che nel **CircuitoSequenziale** tutti i segnali intermedi siamo adeguatamente sincronizzati con dei **Registri** cosicché il ramo più veloce "aspetti" quello più lento e vengano dati gli input corretti a tutti i componenti. Molti di questi componenti sono stati riutilizzati dai progetti precedenti o sono stati modificati per essere adattati al circuito. Ad esempio per il **Moltiplicatore8X10bit** è bastato codificare secondo Booth il numero a 10 bit (somma dei tre operandi del ramo sinistro) e trattare il numero costante **k** come fosse un numero a 8 bit. Avendo però una terna di Booth in più rispetto al precedente moltiplicatore si ha anche un prodotto parziale in più. Per questa ragione è stato progettato un **Sommatore5PP18bit** che esegue le somme dei prodotti parziali. Sono stati quindi introdotti nuovi moduli come il **CarrySelect9bit** (costituito da un **RippleCarry4bit** e da due **RippleCarry5bit**), il **CarrySave17bit** e il **CarrySelect18bit** (costituito da tre **CarrySelect9bit**). Per quanto riguarda il **Comparatore**, il quale deve eseguire un confronto tra il risultato scaturito dal ramo sinistro e da quello destro, si è utilizzato inizialmente un modulo **Complemento18bit** per la complementazione a due del risultato del ramo destro ed in seguito si è eseguita la somma dei due numeri (nonché la sottrazione tra il primo ed il secondo). È bastato infine restituire, come output del circuito, il bit di segno di questa somma.

Struttura e codice VHDL dei componenti

Schema di CircuitoSequenziale



VHDL di CircuitoSequenziale

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

entity CircuitoSequenziale is
    Port ( clk, clear : in std_logic;
           op1 : in  STD_LOGIC_VECTOR (7 downto 0);
           op2 : in  STD_LOGIC_VECTOR (7 downto 0);
           op3 : in  STD_LOGIC_VECTOR (7 downto 0);
           op4 : in  STD_LOGIC_VECTOR (7 downto 0);
           op5 : in  STD_LOGIC_VECTOR (7 downto 0);
           op6 : in  STD_LOGIC_VECTOR (7 downto 0);
           op7 : in  STD_LOGIC_VECTOR (7 downto 0);
           op8 : in  STD_LOGIC_VECTOR (7 downto 0);
           op9 : in  STD_LOGIC_VECTOR (7 downto 0);
           O : out  STD_LOGIC);
```

```
end CircuitoSequenziale;

architecture Structural of CircuitoSequenziale is

component FlipFlop is
	PORT (clk, clear: in std_logic;
		D: in std_logic;
		Q: out std_logic);
end component;

component Registro8bit is
	PORT (clk, clear: in std_logic;
		D: in std_logic_vector(7 downto 0);
		Q: out std_logic_vector(7 downto 0));
end component;

component Registro10bit is
	PORT (clk, clear: in std_logic;
		D: in std_logic_vector(9 downto 0);
		Q: out std_logic_vector(9 downto 0));
end component;

component Registro16bit is
	PORT (clk, clear: in std_logic;
		D: in std_logic_vector(15 downto 0);
		Q: out std_logic_vector(15 downto 0));
end component;

component Registro17bit is
	PORT (clk, clear: in std_logic;
		D: in std_logic_vector(16 downto 0);
		Q: out std_logic_vector(16 downto 0));
end component;

component Registro18bit is
	PORT (clk, clear: in std_logic;
		D: in std_logic_vector(17 downto 0);
		Q: out std_logic_vector(17 downto 0));
end component;

component CarrySave8bit is
   Port ( a : in  STD_LOGIC_VECTOR (7 downto 0);
	b : in  STD_LOGIC_VECTOR (7 downto 0);
	c : in  STD_LOGIC_VECTOR (7 downto 0);
	somme : out  STD_LOGIC_VECTOR (7 downto 0);
	riporti : out  STD_LOGIC_VECTOR (7 downto 0));
end component;

component CarrySelect9bit is
 Port ( A, B: in STD_LOGIC_VECTOR (8 downto 0);
      C0 : in STD_LOGIC;
      S : out STD_LOGIC_VECTOR (8 downto 0);
      C9 : out STD_LOGIC);
end component;

component CarrySelect16bit is
   Port ( a : in  STD_LOGIC_VECTOR (15 downto 0);
```

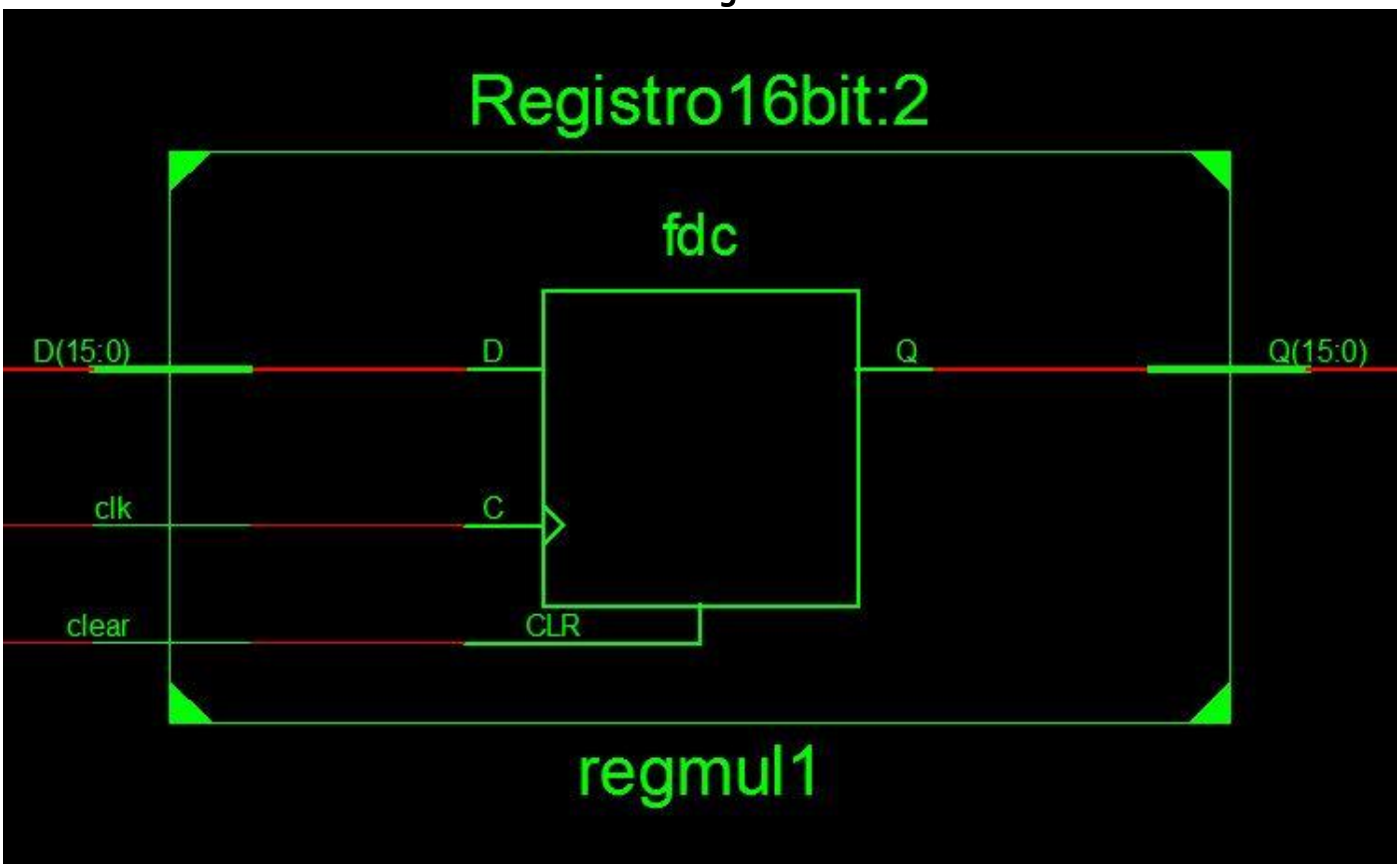
```
        b : in  STD_LOGIC_VECTOR (15 downto 0);
        cin : in  STD_LOGIC;
        somma : out  STD_LOGIC_VECTOR (15 downto 0);
        cout : out  STD_LOGIC);
end component;

component Moltiplicatore8X10bit is
    Port ( A : in  STD_LOGIC_VECTOR (7 downto 0);
        B : in  STD_LOGIC_VECTOR (9 downto 0);
        P : out  STD_LOGIC_VECTOR (17 downto 0));
end component;

component 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 component;

component CarrySave17bit is
    Port ( a : in  STD_LOGIC_VECTOR (16 downto 0);
        b : in  STD_LOGIC_VECTOR (16 downto 0);
        c : in  STD_LOGIC_VECTOR (16 downto 0);
        somme : out  STD_LOGIC_VECTOR (16 downto 0);
        riporti : out  STD_LOGIC_VECTOR (16 downto 0));
end component;
component CarrySelect18bit is
    Port ( a   : in  STD_LOGIC_VECTOR (17 downto 0);
        b   : in  STD_LOGIC_VECTOR (17 downto 0);
        cin : in  STD_LOGIC;
        somma : out  STD_LOGIC_VECTOR (17 downto 0);
        cout : out  STD_LOGIC);
end component;

component Comparatore is
    Port ( A : in  STD_LOGIC_VECTOR (17 downto 0);
        B : in  STD_LOGIC_VECTOR (17 downto 0);
        O : out  STD_LOGIC);
end component;

component SignSelect is
    Port ( a : in  STD_LOGIC;
        b : in  STD_LOGIC;
        c : in  STD_LOGIC;
        sign : out  STD_LOGIC);
end component;

signal op1out, op2out, op3out, op4out, op5out, op6out, op7out, op8out, op9out: std_logic_vector (7 downto
0);
signal vsomma3op, vriporti3op, vsomma3opout, vriporti3opout: std_logic_vector (7 downto 0);
signal vsomma3opfixed, vriporti3opfixed: std_logic_vector (8 downto 0);
signal somma3op9: std_logic_vector (8 downto 0);
signal cSomma3op: std_logic;
signal somma3op10: std_logic_vector (9 downto 0);
signal somma3op10out: std_logic_vector (9 downto 0);
signal k, y1, y2, y3, kout1, kout2, kout3, y1out, y2out, y3out: std_logic_vector (7 downto 0);
signal out1, out2, out1out, out1out2, out2out: std_logic_vector (17 downto 0);
signal p1, p2, p3, p1out, p2out, p3out: std_logic_vector (15 downto 0);
```

```
signal op5ext, op7ext, op9ext, op5extout, op7extout, op9extout: std_logic_vector (15 downto 0):= (others =>
'0');
signal somma1, somma2, somma3: std_logic_vector (15 downto 0);
signal cSomma1, cSomma2, cSomma3: std_logic;
signal somma1_17, somma2_17, somma3_17, somma1_17out, somma2_17out, somma3_17out:
std_logic_vector (16 downto 0);
signal vsomme, vriporti, vsommeout, vriportiout: std_logic_vector (16 downto 0);
signal vsommefixed, vriportifixed: std_logic_vector (17 downto 0);
signal Cin0, Cin0out1, Cin0out2, Cin1, Cin1out1, Cin1out2, Cin2, Cin2out1, Cin2out2, Cin3, Cin3out1,
Cin3out2, Cin4, Cin4out1, Cin4out2, Cin4out3, Cin4out4, unused, output, outputout: std_logic:=('0');

begin

-- Registri per gli operandi in ingresso
regOp1: Registro8bit port map (clk,clear,op1,op1out);
regOp2: Registro8bit port map (clk,clear,op2,op2out);
regOp3: Registro8bit port map (clk,clear,op3,op3out);
regOp4: Registro8bit port map (clk,clear,op4,op4out);
regOp5: Registro8bit port map (clk,clear,op5,op5out);
regOp6: Registro8bit port map (clk,clear,op6,op6out);
regOp7: Registro8bit port map (clk,clear,op7,op7out);
regOp8: Registro8bit port map (clk,clear,op8,op8out);
regOp9: Registro8bit port map (clk,clear,op9,op9out);

-- RAMO SINISTRO
-- Sommo i primi tre operandi
carrySave1: CarrySave8bit port map (op1out, op2out, op3out, vsomma3op, vriporti3op);
regsomme: Registro8bit port map (clk, clear, vsomma3op, vsomma3opout);
regriporti: Registro8bit port map (clk, clear, vriporti3op, vriporti3opout);
vriporti3opfixed<=vriporti3opout & '0';
vsomma3opfixed(7 downto 0)<=vsomma3opout;
vsomma3opfixed(8)<=vsomma3opout(7);
Cin0<='0';
ffCin0_1: FlipFlop port map (clk, clear, Cin0, Cin0out1);
ffCin0_2: FlipFlop port map (clk, clear, Cin0out1, Cin0out2);
carrySelect: CarrySelect9bit port map (vsomma3opfixed, vriporti3opfixed, Cin0out2, somma3op9,
cSomma3op);
somma3op10(8 downto 0)<=somma3op9;
signCarrySelect9bit: SignSelect port map(vsomma3opout(7), vriporti3opfixed(8), cSomma3op,
somma3op10(9));
regSomma3op: Registro10bit port map (clk, clear, somma3op10, somma3op10out);
-- Moltiplico k="85" per la somma dei primi tre operandi
k<="01010101";
regk1: Registro8bit port map(clk, clear, k, kout1);
regk2: Registro8bit port map(clk, clear, kout1, kout2);
regk3: Registro8bit port map(clk, clear, kout2, kout3);
mulKx3op: Moltiplicatore8X10bit port map (kout3, somma3op10out, out1);
regOut1: Registro18bit port map (clk, clear, out1, out1out);
reg2Out1: Registro18bit port map (clk, clear, out1out, out1out2);

-- RAMO DESTRO
-- OP4 OP5
y1<="01100111";
regy1: Registro8bit port map (clk, clear, y1, y1out);
mul1: Moltiplicatore8bit port map (op4out, y1out, p1);
regmul1: Registro16bit port map (clk, clear, p1, p1out);
op5ext(7 downto 0)<=op5out;
```

```
with op5out(7) select
	op5ext(15 downto 8)<= "00000000" when '0',
				"11111111" when '1',
				"XXXXXXXX" when others;
reg1: Registro16bit port map (clk, clear, op5ext, op5extout);
Cin1<='0';
ffCin1_1: FlipFlop port map (clk, clear, Cin1, Cin1out1);
ffCin1_2: FlipFlop port map (clk, clear, Cin1out1, Cin1out2);
carrySelect1: CarrySelect16bit port map (p1out, op5extout, Cin1out2, somma1, cSomma1);
somma1_17(15 downto 0)<=somma1;
signCarrySelect16bit1: SignSelect port map(p1out(15), op5extout(15), cSomma1, somma1_17(16));
reg1_17: Registro17bit port map (clk, clear, somma1_17, somma1_17out);
-- OP6 OP7
y2<="01000111";
regy2: Registro8bit port map (clk, clear, y2, y2out);
mul2: Moltiplicatore8bit port map (op6out, y2out, p2);
regmul2: Registro16bit port map (clk, clear, p2, p2out);
op7ext(7 downto 0)<=op7out;
with op7out(7) select
	op7ext(15 downto 8)<= "00000000" when '0',
				"11111111" when '1',
				"XXXXXXXX" when others;
reg2: Registro16bit port map (clk, clear, op7ext, op7extout);
Cin2<='0';
ffCin2_1: FlipFlop port map (clk, clear, Cin2, Cin2out1);
ffCin2_2: FlipFlop port map (clk, clear, Cin2out1, Cin2out2);
carrySelect2: CarrySelect16bit port map (p2out, op7extout, Cin2out2, somma2, cSomma2);
somma2_17(15 downto 0)<=somma2;
signCarrySelect16bit2: SignSelect port map(p2out(15), op7extout(15), cSomma2, somma2_17(16));
reg2_17: Registro17bit port map (clk, clear, somma2_17, somma2_17out);
-- OP8 OP9
y3<="00010001";
regy3: Registro8bit port map (clk, clear, y3, y3out);
mul3: Moltiplicatore8bit port map (op8out, y3out, p3);
regmul3: Registro16bit port map (clk, clear, p3, p3out);
op9ext(7 downto 0)<=op9out;
with op9out(7) select
	op9ext(15 downto 8)<= "00000000" when '0',
				"11111111" when '1',
				"XXXXXXXX" when others;
reg3: Registro16bit port map (clk, clear, op9ext, op9extout);
Cin3<='0';
ffCin3_1: FlipFlop port map (clk, clear, Cin3, Cin3out1);
ffCin3_2: FlipFlop port map (clk, clear, Cin3out1, Cin3out2);
carrySelect3: CarrySelect16bit port map (p3out, op9extout, Cin3out2, somma3, cSomma3);
somma3_17(15 downto 0)<=somma3;
signCarrySelect16bit3: SignSelect port map(p3out(15), op9extout(15), cSomma3, somma3_17(16));
reg3_17: Registro17bit port map (clk, clear, somma3_17, somma3_17out);
-- Sommo i tre risultati
carrySave2: CarrySave17bit port map (somma1_17out, somma2_17out, somma3_17out, vsomme, vriporti);
regsomme2: Registro17bit port map (clk, clear, vsomme, vsommeout);
regriporti2: Registro17bit port map (clk, clear, vriporti, vriportiout);
vsommefixed(16 downto 0)<=vsommeout;
vsommefixed(17)<=vsommeout(16);
vriportifixed<= vriportiout & '0';
Cin4<='0';
ffCin4_1: FlipFlop port map (clk, clear, Cin4, Cin4out1);
```

```
ffCin4_2: FlipFlop port map (clk, clear, Cin4out1, Cin4out2);
ffCin4_3: FlipFlop port map (clk, clear, Cin4out2, Cin4out3);
ffCin4_4: FlipFlop port map (clk, clear, Cin4out3, Cin4out4);
carrySelectOut2: CarrySelect18bit port map (vsommefixed, vriportifixed, Cin4out4, out2, unused);
regout2: Registro18bit port map (clk, clear, out2, out2out);

-- COMPARATORE FINALE
comp: Comparatore port map (out1out2, out2out, output);
ffout: FlipFlop port map (clk, clear, output, outputout);
O<=outputout;

end Structural;
```

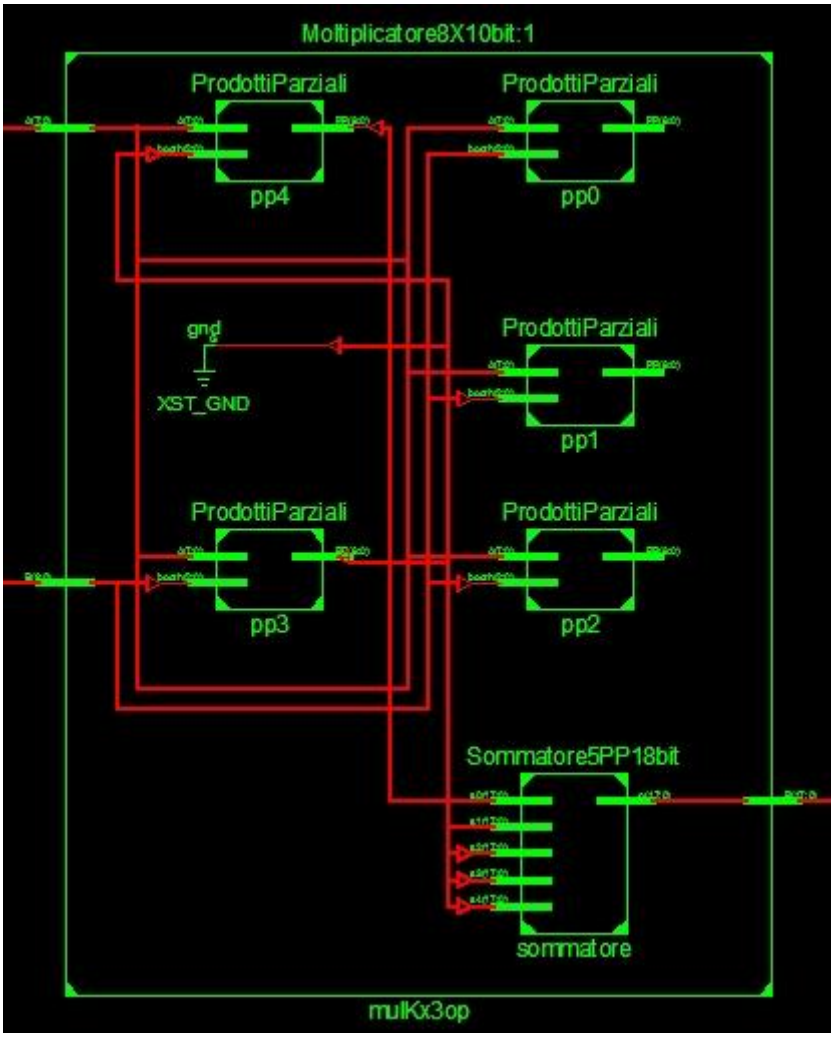
Schema di Registro16bit



VHDL di Registro16bit

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

entity Registro16bit is
	PORT (clk, clear: in std_logic;
		D: in std_logic_vector(15 downto 0);
		Q: out std_logic_vector(15 downto 0));
end Registro16bit;

architecture Behavioral of Registro16bit is

begin
	process(clk, clear)
		begin
		  if clear='1' then
				Q<="0000000000000000";
		  elsif clk'event and clk='1' then
				Q<=D;
		  end if;
	end process;
end Behavioral;
```

Schema di Moltiplicatore8X10bit

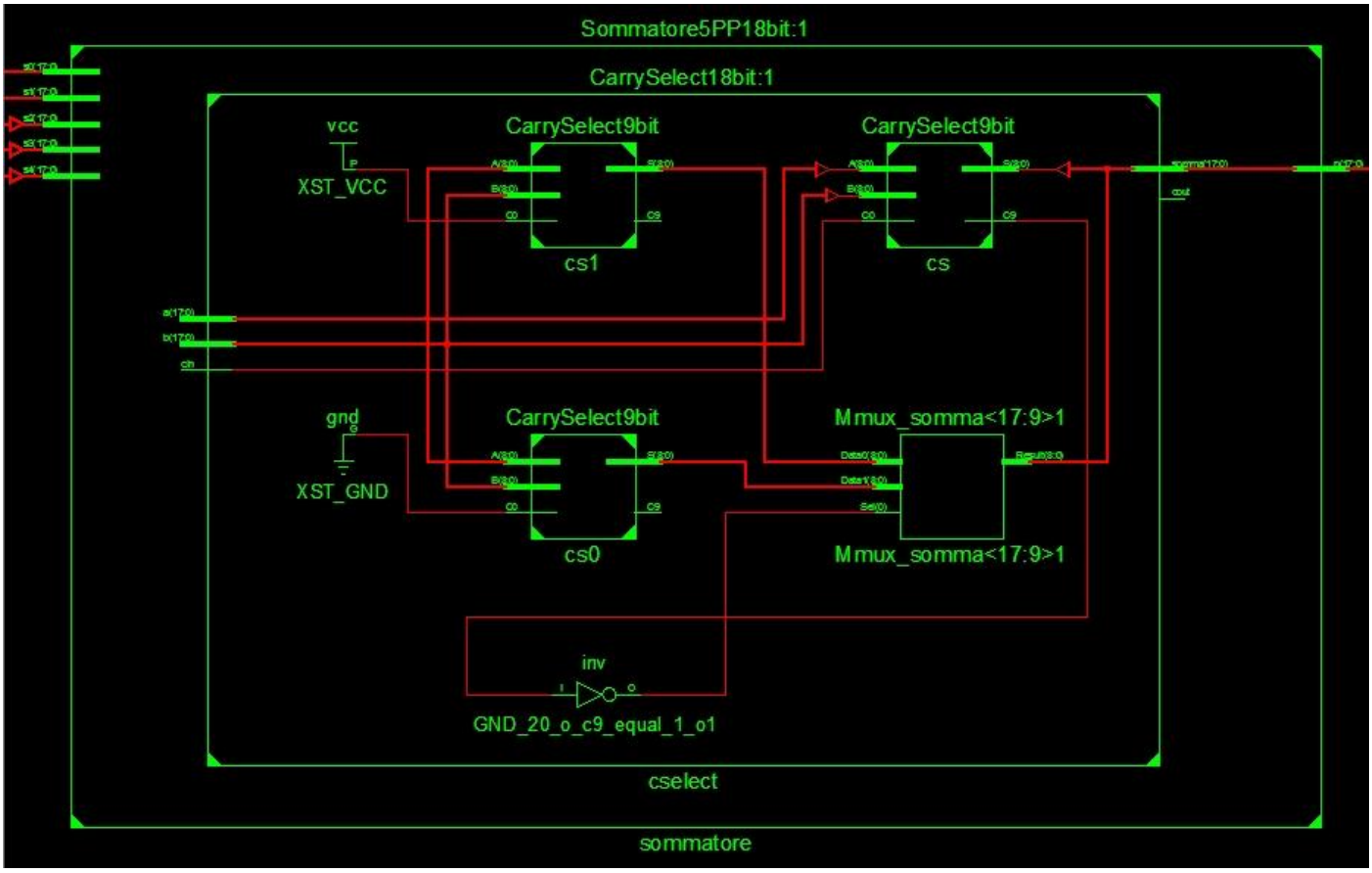


VHDL di Moltiplicatore8X10bit

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

entity Moltiplicatore8X10bit is
   Port ( A : in  STD_LOGIC_VECTOR (7 downto 0);
          B : in  STD_LOGIC_VECTOR (9 downto 0);
          P : out  STD_LOGIC_VECTOR (17 downto 0));
end Moltiplicatore8X10bit;

architecture Structural of Moltiplicatore8X10bit 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 Sommatore5PP18bit is
   Port ( s0 : in  STD_LOGIC_VECTOR (17 downto 0);
          s1 : in  STD_LOGIC_VECTOR (17 downto 0);
          s2 : in  STD_LOGIC_VECTOR (17 downto 0);
          s3 : in  STD_LOGIC_VECTOR (17 downto 0);
          s4 : in  STD_LOGIC_VECTOR (17 downto 0);
          p : out  STD_LOGIC_VECTOR (17 downto 0));
```

```
end component;

signal booth5: std_logic_vector(2 downto 0);
signal s0, s1, s2, s3, s4 : std_logic_vector(8 downto 0);
signal s0fixed, s1fixed, s2fixed, s3fixed, s4fixed : std_logic_vector(17 downto 0):= (others => '0');

begin

booth5(2 downto 1)<=b(1 downto 0);
booth5(0)<='0';

pp0: ProdottiParziali port map(B(9 downto 7), A, s4);
pp1: ProdottiParziali port map(B(7 downto 5), A, s3);
pp2: ProdottiParziali port map(B(5 downto 3), A, s2);
pp3: ProdottiParziali port map(B(3 downto 1), A, s1);
pp4: ProdottiParziali port map(booth5, A, s0);

s0fixed(8 downto 0)<=s0;
with s0(8) select
        s0fixed(17 downto 9)<= "111111111" when '1',
                                                        "000000000" when '0',
                                                        "XXXXXXXXX" when others;
s1fixed(10 downto 2)<=s1;
with s1(8) select
        s1fixed(17 downto 11)<="1111111" when '1',
                                                        "0000000" when '0',
                                                        "XXXXXXX" when others;
s2fixed(12 downto 4)<=s2;
with s2(8) select
        s2fixed(17 downto 13)<="11111" when '1',
                                                        "00000" when '0',
                                                        "XXXXX" when others;
s3fixed(14 downto 6)<=s3;
with s3(8) select
        s3fixed(17 downto 15)<="111" when '1',
                                                        "000" when '0',
                                                        "XXX" when others;
s4fixed(16 downto 8)<=s4;
s4fixed(17)<=s4(8);

sommatore: Sommatore5PP18bit port map(s0fixed, s1fixed, s2fixed, s3fixed, s4fixed,P);

end Structural;
```

Schema di Sommatore5PP18bit

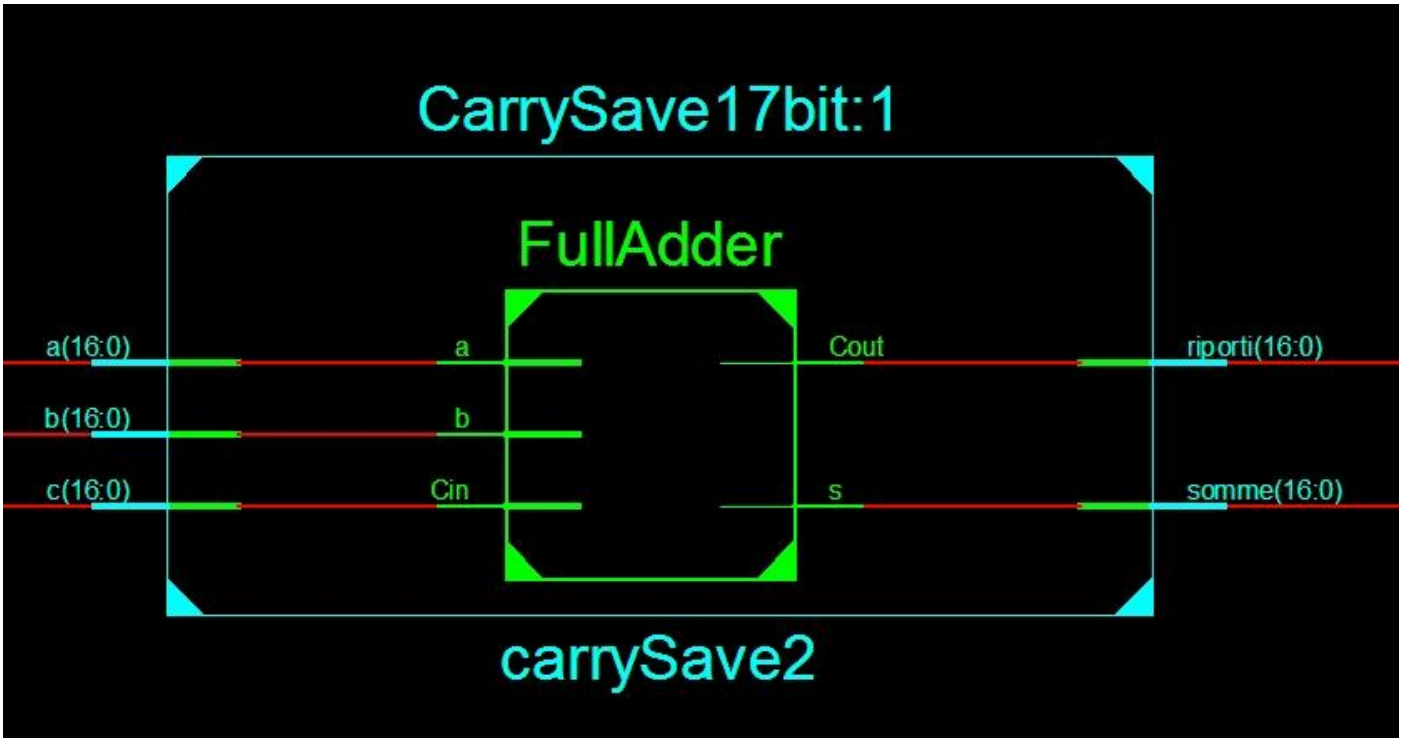


VHDL di Sommatore5PP18bit

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

entity Sommatore5PP18bit is
   Port ( s0 : in  STD_LOGIC_VECTOR (17 downto 0);
          s1 : in  STD_LOGIC_VECTOR (17 downto 0);
          s2 : in  STD_LOGIC_VECTOR (17 downto 0);
          s3 : in  STD_LOGIC_VECTOR (17 downto 0);
          s4 : in  STD_LOGIC_VECTOR (17 downto 0);
          p : out  STD_LOGIC_VECTOR (17 downto 0));
end Sommatore5PP18bit;

architecture Structural of Sommatore5PP18bit is

component CarrySave18bit is
   Port ( a : in  STD_LOGIC_VECTOR (17 downto 0);
          b : in  STD_LOGIC_VECTOR (17 downto 0);
          c : in  STD_LOGIC_VECTOR (17 downto 0);
          somme : out  STD_LOGIC_VECTOR (17 downto 0);
          riporti : out  STD_LOGIC_VECTOR (17 downto 0));
end component;

component CarrySelect18bit is
   Port ( a : in  STD_LOGIC_VECTOR (17 downto 0);
          b : in  STD_LOGIC_VECTOR (17 downto 0);
          cin : in  STD_LOGIC;
          somma : out  STD_LOGIC_VECTOR (17 downto 0);
          cout : out  STD_LOGIC);
```

```
end component;

signal somme1, riporti1, somme2, riporti2, somme3, riporti3, riporti1fixed, riporti2fixed, riporti3fixed:
std_logic_vector(17 downto 0):= (others => '0');
signal unused: std_logic;

begin
-- Sommo i primi tre prodotti parziali usando un CarrySave a 18 bit
csave1: CarrySave18bit port map(s0, s1, s2, somme1, riporti1);
riporti1fixed(17 downto 1)<=riporti1(16 downto 0);
-- Sommo gli output del primo CarrySave ed il quarto prodotto parziale con un CarrySave a 18 bit
csave2: CarrySave18bit port map(somme1, riporti1fixed, s3, somme2, riporti2);
riporti2fixed(17 downto 1)<=riporti2(16 downto 0);
-- Sommo gli output del secondo CarrySave ed il quinto prodotto parziale con un CarrySave a 18 bit
csave3: CarrySave18bit port map(somme2, riporti2fixed, s4, somme3, riporti3);
riporti3fixed(17 downto 1)<=riporti3(16 downto 0);
-- Sommo gli output del terzo CarrySave con un CarrySelect a 18 bit
cselect: CarrySelect18bit port map(somme3, riporti3fixed, '0', p, unused);

end Structural;
```

Schema di CarrySave17bit

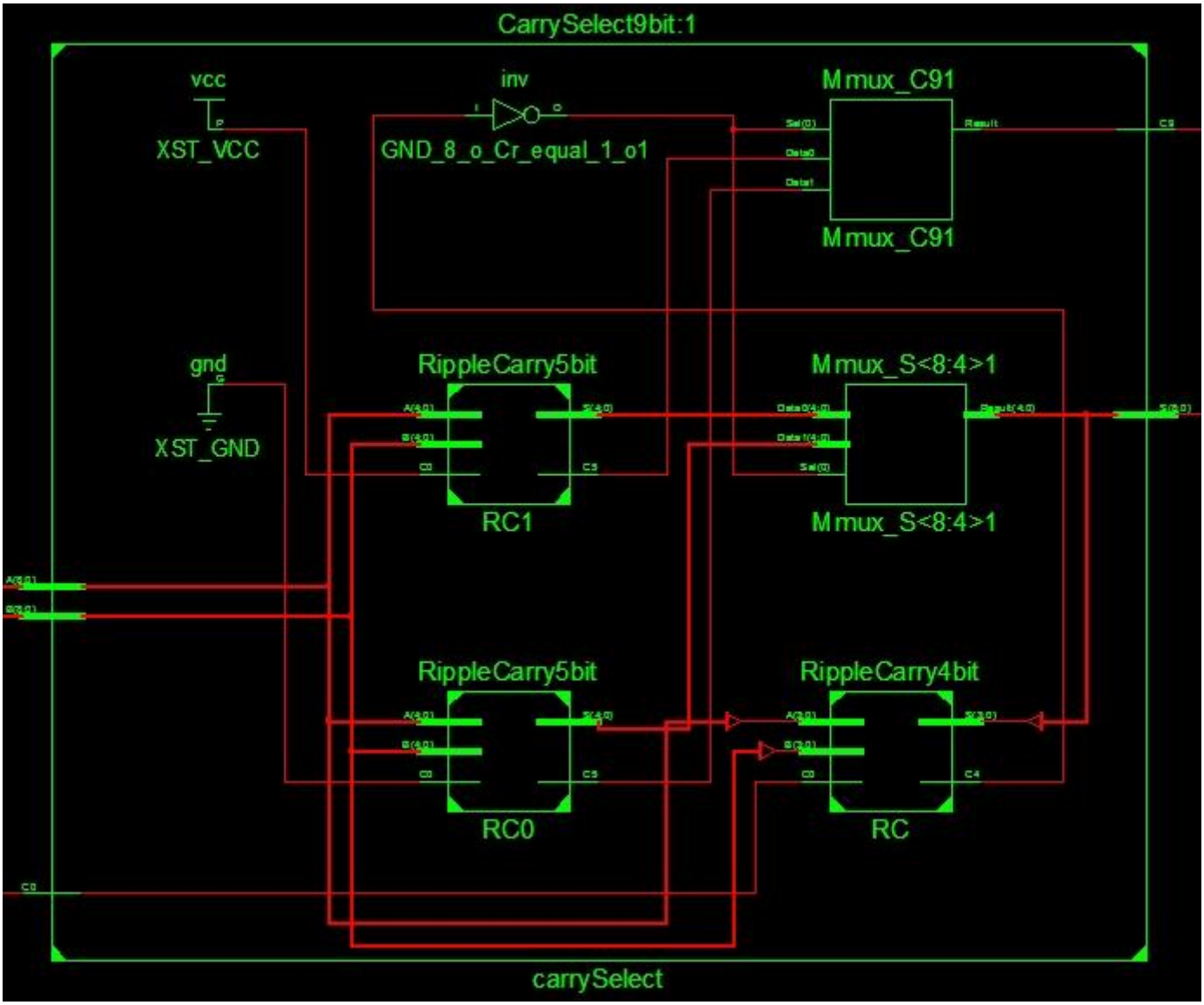


VHDL di CarrySave17bit

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

entity CarrySave17bit is
   Port ( a : in  STD_LOGIC_VECTOR (16 downto 0);
          b : in  STD_LOGIC_VECTOR (16 downto 0);
          c : in  STD_LOGIC_VECTOR (16 downto 0);
          somme : out  STD_LOGIC_VECTOR (16 downto 0);
          riporti : out  STD_LOGIC_VECTOR (16 downto 0));
end CarrySave17bit;

architecture Structural of CarrySave17bit 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));
FA16: FullAdder port map(a(16),b(16),c(16),somme(16),riporti(16));

end Structural;
```

Schema di CarrySelect9bit

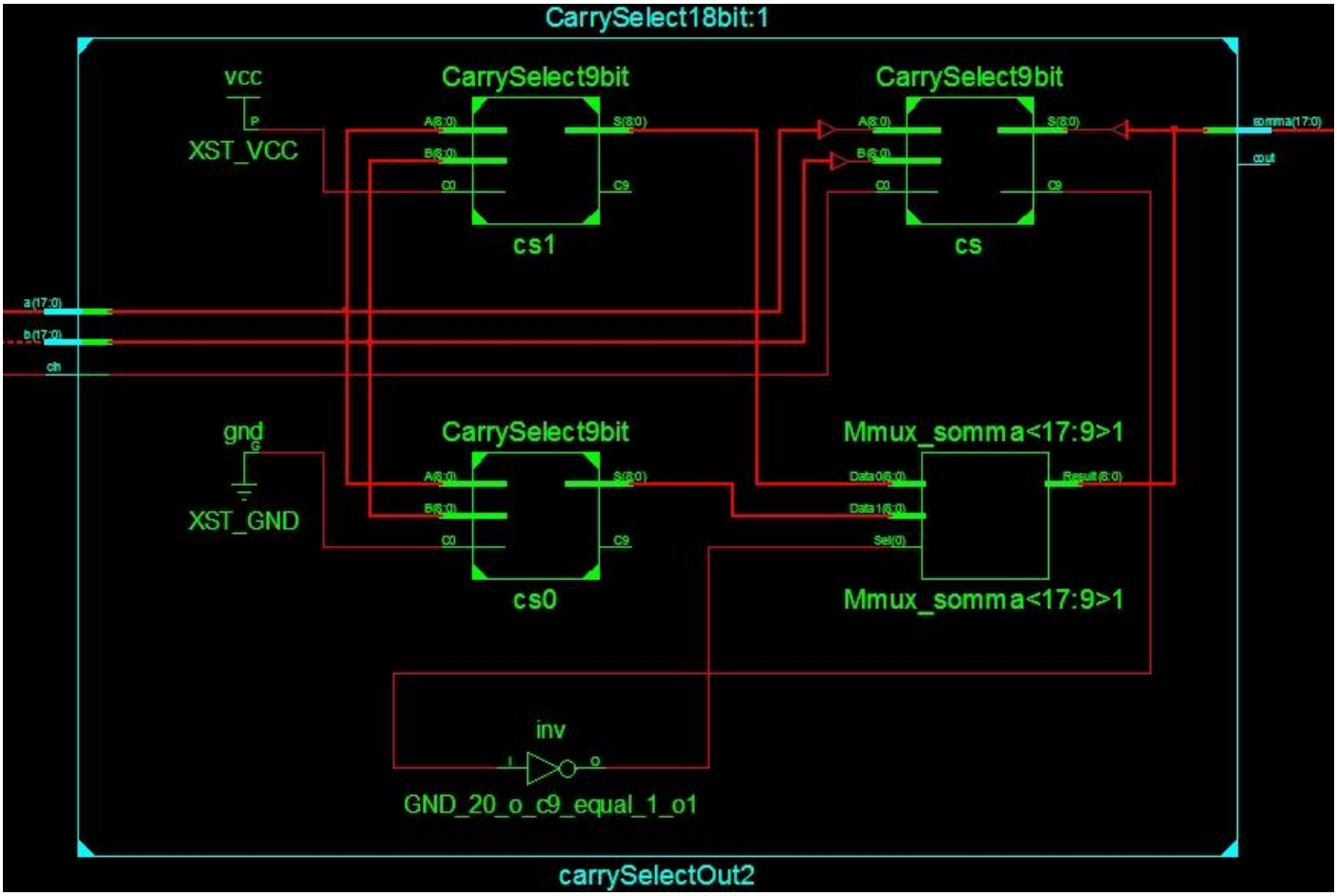


VHDL di CarrySelect9bit

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

entity CarrySelect9bit is
 Port ( A, B: in STD_LOGIC_VECTOR (8 downto 0);
       C0 : in STD_LOGIC;
       S : out STD_LOGIC_VECTOR (8 downto 0);
       C9 : out STD_LOGIC);
end CarrySelect9bit;

architecture Structural of CarrySelect9bit is

component RippleCarry4bit is
 Port ( C0 : in STD_LOGIC;
       A, B : in STD_LOGIC_VECTOR (3 downto 0);
       S : out STD_LOGIC_VECTOR(3 downto 0);
       C4 : out STD_LOGIC);
end component;

component RippleCarry5bit is
 Port ( C0 : in STD_LOGIC;
```

```
    A, B : in STD_LOGIC_VECTOR (4 downto 0);
    S : out STD_LOGIC_VECTOR(4 downto 0);
    C5 : out STD_LOGIC);
end component;

signal Cr, Cr0, Cr1 : STD_LOGIC;
signal S0, S1 : STD_LOGIC_VECTOR (4 downto 0);

begin
RC:RippleCarry4bit port map(C0, A(3 downto 0), B(3 downto 0), S(3 downto 0), Cr);
RC0:RippleCarry5bit port map('0', A(8 downto 4), B(8 downto 4), S0(4 downto 0), Cr0);
RC1:RippleCarry5bit port map('1', A(8 downto 4), B(8 downto 4), S1(4 downto 0), Cr1);
with Cr select
        S(8 downto 4) <= S0 (4 downto 0)when '0',
                         S1 (4 downto 0)when '1',
                         "XXXXX" when others;
with Cr select
        C9 <= Cr0 when '0',
              Cr1 when '1',
              'X' when others;
end Structural;
```

Schema di CarrySelect18bit

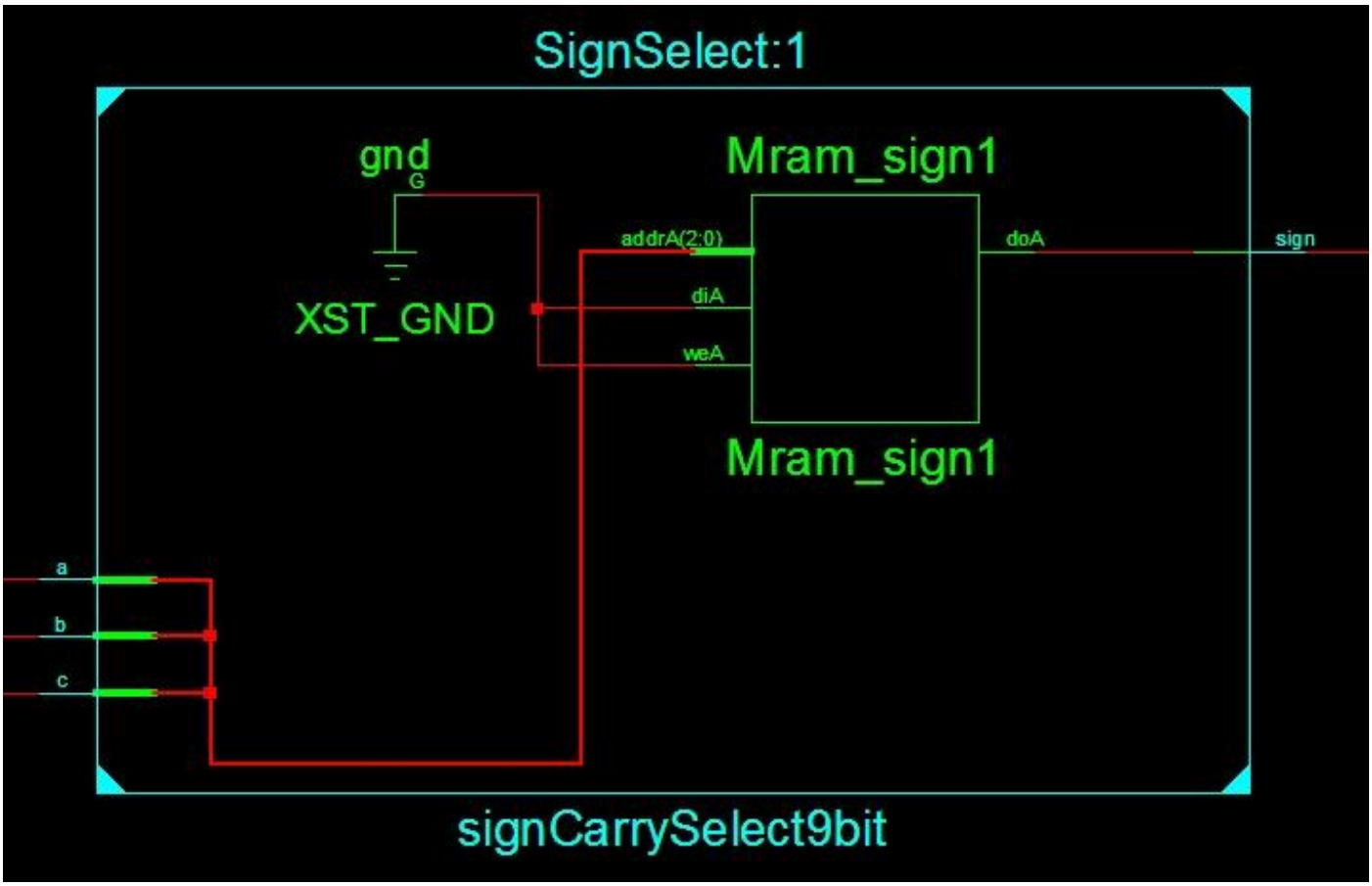


VHDL di CarrySelect18bit

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

entity CarrySelect18bit is
   Port ( a   : in  STD_LOGIC_VECTOR (17 downto 0);
          b   : in  STD_LOGIC_VECTOR (17 downto 0);
          cin : in  STD_LOGIC;
          somma : out  STD_LOGIC_VECTOR (17 downto 0);
          cout : out  STD_LOGIC);
end CarrySelect18bit;

architecture Structural of CarrySelect18bit is

component CarrySelect9bit is
 Port ( A, B: in STD_LOGIC_VECTOR (8 downto 0);
 C0 : in STD_LOGIC;
 S : out STD_LOGIC_VECTOR (8 downto 0);
 C9 : out STD_LOGIC);
end component;

signal s0, s1: std_logic_vector(8 downto 0);
signal c9, c18_0, c18_1: std_logic;

begin

cs: CarrySelect9bit port map(a(8 downto 0), b(8 downto 0), cin, somma(8 downto 0), c9);
cs0: CarrySelect9bit port map(a(17 downto 9), b(17 downto 9), '0', s0, c18_0);
```

```
cs1: CarrySelect9bit port map(a(17 downto 9), b(17 downto 9), '1', s1, c18_1);

with c9 select
        somma(17 downto 9)<=s0 when '0',
                            s1 when '1',
                            "XXXXXXXXX" when others;
with c9 select
        cout<=c18_0 when '0',
                c18_1 when '1',
                'X' when others;

end Structural;
```

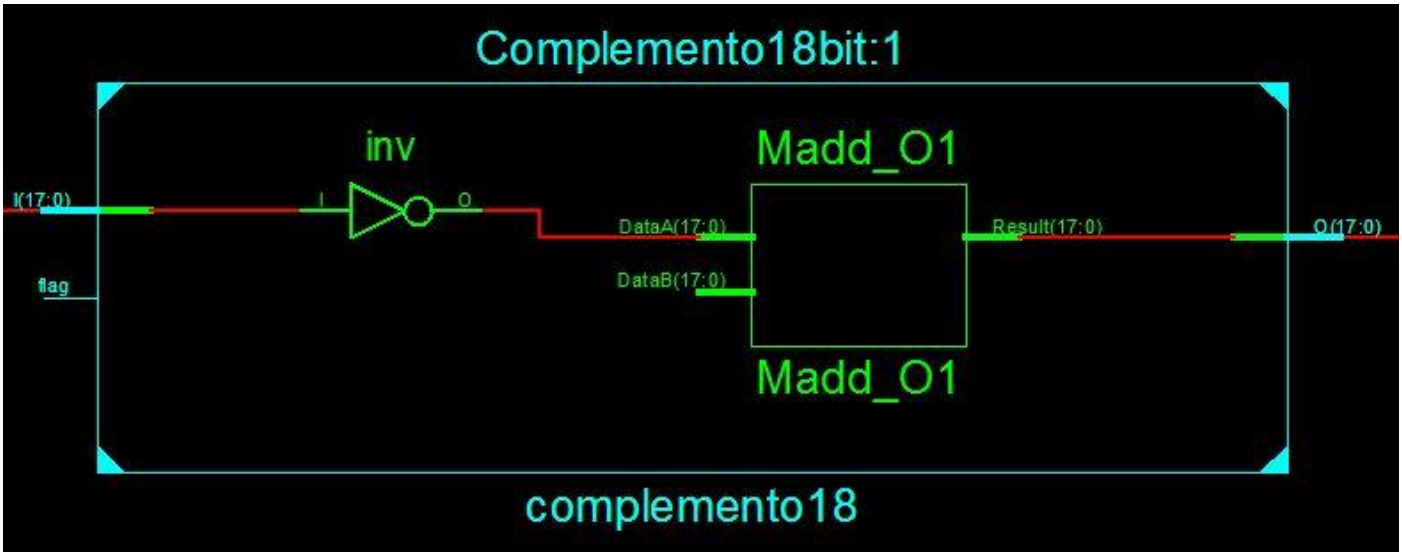
Schema di SignSelect



VHDL di SignSelect

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity SignSelect is
   Port ( a : in  STD_LOGIC;
          b : in  STD_LOGIC;
          c : in  STD_LOGIC;
          sign : out  STD_LOGIC);
end SignSelect;

architecture Behavioral of SignSelect is
signal bits: std_logic_vector(2 downto 0);

begin
bits(0)<=a;
bits(1)<=b;
bits(2)<=c;

with bits select
  sign<='0' when "000",
            '1' when "001",
            '1' when "010",
            '0' when "011",
            '1' when "100",
            '0' when "101",
            '0' when "110",
            '1' when "111",
            'X' when others;
end Behavioral;
```

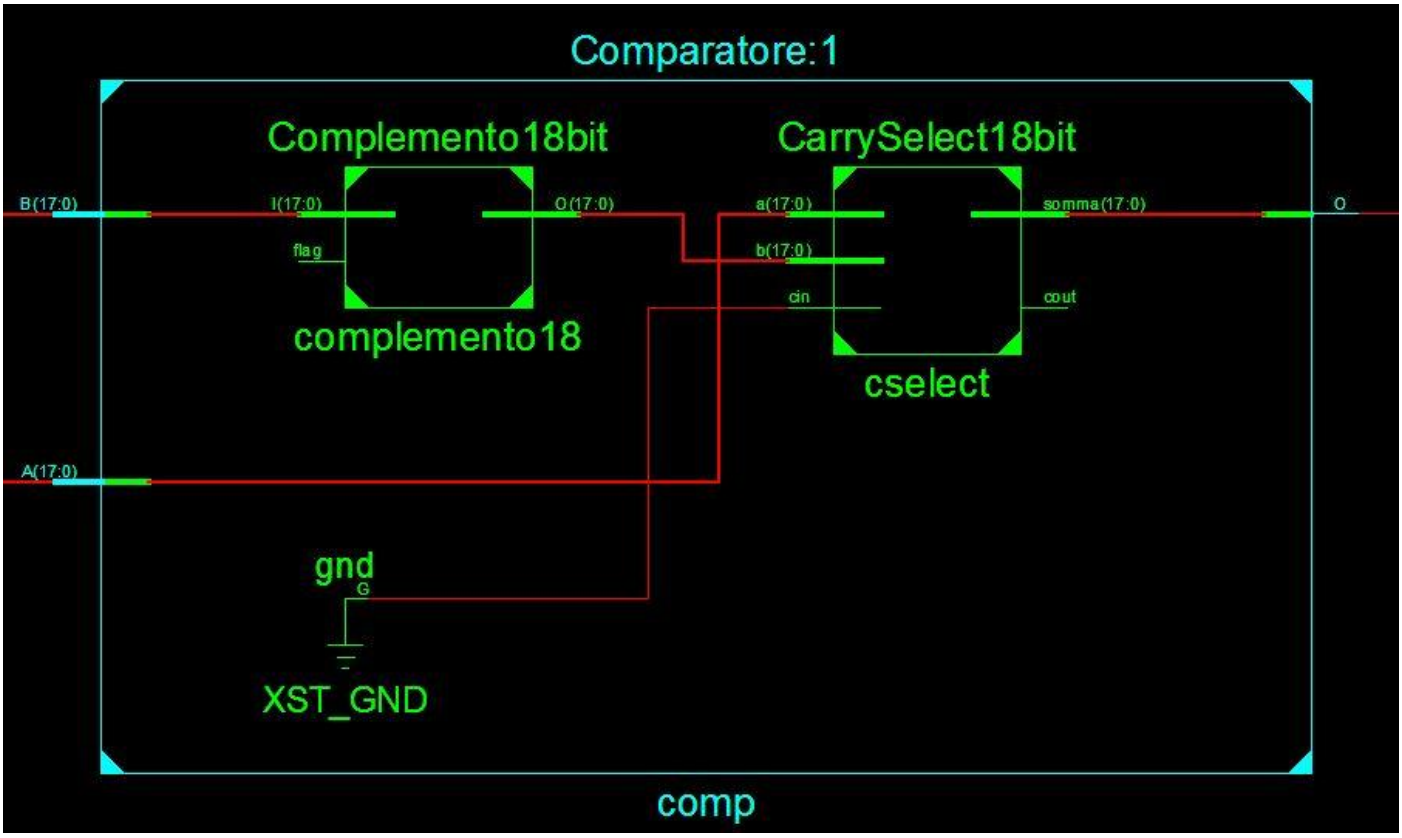
Schema di Complemento18bit



VHDL di Complemento18bit

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

entity Complemento18bit is
   Port ( I : in  STD_LOGIC_VECTOR (17 downto 0);
          O : out  STD_LOGIC_VECTOR (17 downto 0));
end Complemento18bit;

architecture Behavioral of Complemento18bit is

signal ic2, notI, Icomp: std_logic_vector(17 downto 0);

begin

O <= std_logic_vector(unsigned((not I)+1));

end Behavioral;
```

Schema di Comparatore



VHDL di Comparatore

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity Comparatore is
   Port ( A : in  STD_LOGIC_VECTOR (17 downto 0);
          B : in  STD_LOGIC_VECTOR (17 downto 0);
          O : out  STD_LOGIC);
end Comparatore;
architecture Behavioral of Comparatore is
component Complemento18bit is
   Port ( I : in  STD_LOGIC_VECTOR (17 downto 0);
          O : out  STD_LOGIC_VECTOR (17 downto 0));
end component;

component CarrySelect18bit is
   Port ( a : in  STD_LOGIC_VECTOR (17 downto 0);
          b : in  STD_LOGIC_VECTOR (17 downto 0);
          cin : in  STD_LOGIC;
          somma : out  STD_LOGIC_VECTOR (17 downto 0);
          cout : out  STD_LOGIC);
end component;

signal Bcomp, sottrazione: std_logic_vector(17 downto 0);
signal unused: std_logic;

begin
complemento18: Complemento18bit port map(B, Bcomp);
cselect: CarrySelect18bit port map (A, Bcomp, '0', sottrazione, unused);
O <= sottrazione(15);
end Behavioral;
```

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 test IS
END test;

ARCHITECTURE behavior OF test IS

  COMPONENT CircuitoSequenziale
  PORT(
     clk, clear : IN  std_logic;
     op1 : IN  std_logic_vector(7 downto 0);
     op2 : IN  std_logic_vector(7 downto 0);
     op3 : IN  std_logic_vector(7 downto 0);
     op4 : IN  std_logic_vector(7 downto 0);
     op5 : IN  std_logic_vector(7 downto 0);
     op6 : IN  std_logic_vector(7 downto 0);
     op7 : IN  std_logic_vector(7 downto 0);
     op8 : IN  std_logic_vector(7 downto 0);
     op9 : IN  std_logic_vector(7 downto 0);
     O : OUT  std_logic);
  END COMPONENT;


  signal clk, clear : std_logic := '0';
  signal op1 : std_logic_vector(7 downto 0) := (others => '0');
  signal op2 : std_logic_vector(7 downto 0) := (others => '0');
  signal op3 : std_logic_vector(7 downto 0) := (others => '0');
  signal op4 : std_logic_vector(7 downto 0) := (others => '0');
  signal op5 : std_logic_vector(7 downto 0) := (others => '0');
  signal op6 : std_logic_vector(7 downto 0) := (others => '0');
  signal op7 : std_logic_vector(7 downto 0) := (others => '0');
  signal op8 : std_logic_vector(7 downto 0) := (others => '0');
  signal op9 : std_logic_vector(7 downto 0) := (others => '0');

  signal O : std_logic;

  constant clk_period : time := 10 ns;

BEGIN

  uut: CircuitoSequenziale PORT MAP (
       clk => clk,
       clear => clear,
       op1 => op1,
       op2 => op2,
       op3 => op3,
       op4 => op4,
```

```
    op5 => op5,
    op6 => op6,
    op7 => op7,
    op8 => op8,
    op9 => op9,
    O => O);

clk_process :process
begin
                clk <= '0';
                wait for clk_period/2;
                clk <= '1';
                wait for clk_period/2;
end process;

        clear_process :process
begin
                clear <= '1';
                wait for 1ns;
                clear <= '0';
                wait for 99ns;
end process;


stim_proc: process
begin

                op1<="00000000";
                op2<="11110100";
                op3<="11101100";
                op4<="00101111";
                op5<="11110101";
                op6<="00101101";
                op7<="00110011";
                op8<="00111100";
                op9<="01111100";

                wait for 10ns;

                op1<="00101000";
                op2<="10010100";
                op3<="11101000";
                op4<="00100111";
                op5<="11110111";
                op6<="00101001";
                op7<="00100000";
                op8<="00101100";
                op9<="01110100";

                wait for 10ns;

                op1<="10000000";
                op2<="10000000";
                op3<="10000000";
                op4<="11100100";
                op5<="01010100";
                op6<="01001001";
```

```
                op7<="00111110";
                op8<="00000010";
                op9<="01110000";

                wait for 10ns;

                op1<="01111111";
                op2<="01111111";
                op3<="00000011";
                op4<="00000111";
                op5<="11111001";
                op6<="11111110";
                op7<="00000110";
                op8<="00000011";
                op9<="11110011";

                wait for 10ns;

                op1<="00101100";
                op2<="10010100";
                op3<="11101000";
                op4<="00100111";
                op5<="11110111";
                op6<="00101001";
                op7<="00100000";
                op8<="00101100";
                op9<="01110000";

                wait for 10ns;

    wait;
  end process;

END;
```

Schema dei segnali di input e output