UNIVERSITÀ DELLA CALABRIA

DIMES - Dipartimento di INGEGNERIA INFORMATICA
MODELLISTICA, ELETTRONICA E SISTEMISTICA

Anno Accademico 2015-2016

Corso di Laurea Triennale in Ingegneria Informatica

Quarto progetto di Elettronica Digitale

"Circuito Pipeline di Tre Unità Mutliply Accumulate ad otto bit e un Adder che ne somma i risultati a sedici bit"

Professoressa S.Perri

Ivonne Rizzuto
matricola 167058

Descrizione

Il circuito che si deve realizzare per la stesura di questo quarto progetto è un circuito costituito da tre unità MAC (“Multiply Accumulate Unit”), ovvero tre unità di **moltiplicatore** che, ricevendo ognuna due flussi in ingresso, restituiscano, rispettivamente, in uscita, tre risultati da sommare tra di loro, con l'accumulatore che segue loro in cascata, ovvero un Ripple Carry.

Ciascuno dei tre moltiplicatori riceverà in ingresso due flussi di segnali che consistono in operandi ad otto bit, denominati nel modo seguente:

-per il primo moltiplicatore MACa, i flussi in ingresso saranno i segnali A e FA;

-per il secondo moltiplicatore MACb, i flussi in ingresso saranno i segnali contrassegnati da B e FB;

-per il terzo moltiplicatore MACc, i flussi in ingresso saranno denominati C e FC;

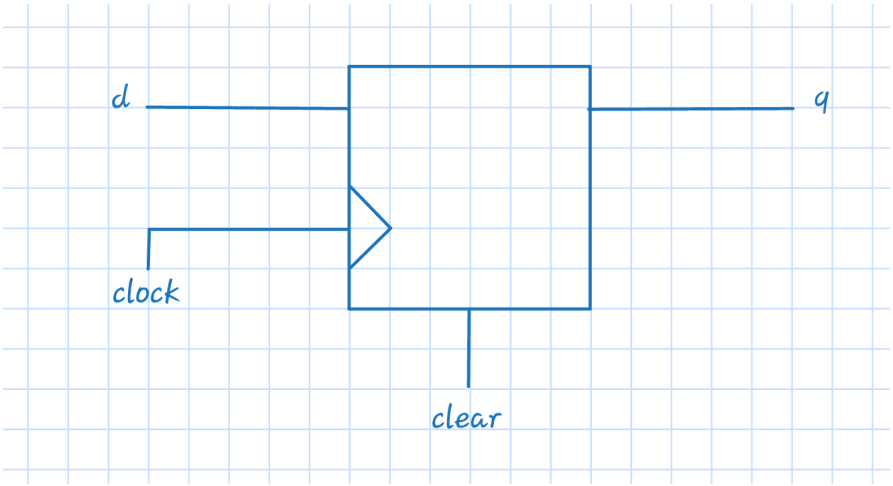
In totale, allora, il circuito riceverà tre coppie di segnali, per un comprensivo di sei flussi in ingresso, costituite tutte da segnali ad otto bit, avendo l'accortezza di forzare a zero l'ultimo bit di ciascuno di questi, ovvero quello più significativo (con indice posizionale “7”, poichè rappresenta proprio il bit di segno).
Gli operandi considerati, infatti, dovranno tutti avere segno algebrico positivo.
I bit coinvolti in questa operazione saranno, dunque: A(7), FA(7), B(7), FB(7), C(7), FC(7).

Il circuito da realizzare è di tipo Pipeline.
Ciò significa che verranno utilizzati un certo numero di registri intermedi, atti a sincronizzare gli ingressi e le uscite del circuito generale, ma anche di alcuni sottomoduli ad esso interni.

Si analizzano, qui di seguito, tutto i componenti che costituiscono l'unità di calcolo sopra descritta, partendo dalle unità più elementari fino a quelle più complesse, tutte di **tipo std_logic**.

Flip Flop

E' l'unità di memoria più elementare del circuito.
I suoi ingressi sono d, il segnale di clock ed il segnale di clear, mentre la sua uscita verrà contrassegnata come q.
Quando il segnale di clear assumerà valore logico alto, allora il valore dell'uscita q verrà azzerato, indipendentemente dal valore in ingresso d e da quello assunto dal segnale di clock;
invece, quando il clear assumerà valore logico basso e il segnale di clock invece varrà “1” , allora nell'uscita q verrà trascritto il valore dell'ingresso d.



Segue il codice Vhdl che descrive il componente in esame:

```
--FlipFlop std_logic
--si include la libreria
library IEEE;
--si include il package di interesse della libreria
use IEEE.std_logic_1164.all;
--Questo componente lavora ad 1 bit.
--ingressi:
--input d,segnale di clock,segnale di clear;
--uscite:
--uscita q;
Entity FlipFlop is

     port(d,clock,clear: in std_logic;
                q: out std_logic
     );
```

```
end FlipFlop;

--definizione dell'architettura
Architecture MyFF of FlipFlop is
--esplicita il comportamento dell'unita' flip flop
     begin
     --sensitivity list: clock e clear
          process(clock,clear)
               begin
               --se si verifica che clear e' pari ad 1,
               --lo stato di uscita del flip flop si azzera
                    if clear = '1' then
                           q <= '0';
               --se invece si verifica che e' lo stato del clock
               --ad essere pari ad 1, allora lo stato di uscita
               --assume lo stato che e' stato ricevuto in ingresso
                    elsif clock'event and clock = '1' then
                                                 q <= d;
                    end if;
          end process;
end MyFF;
```

Registro degli Ingressi

Questo componente è stato utilizzato per la memorizzazione degli ingressi che riceve ciascuno dei tre moltiplicatori ad otto bit, cioè delle tre coppie di segnali ad otto bit che vanno in ingresso al circuito pipeline complessivo.

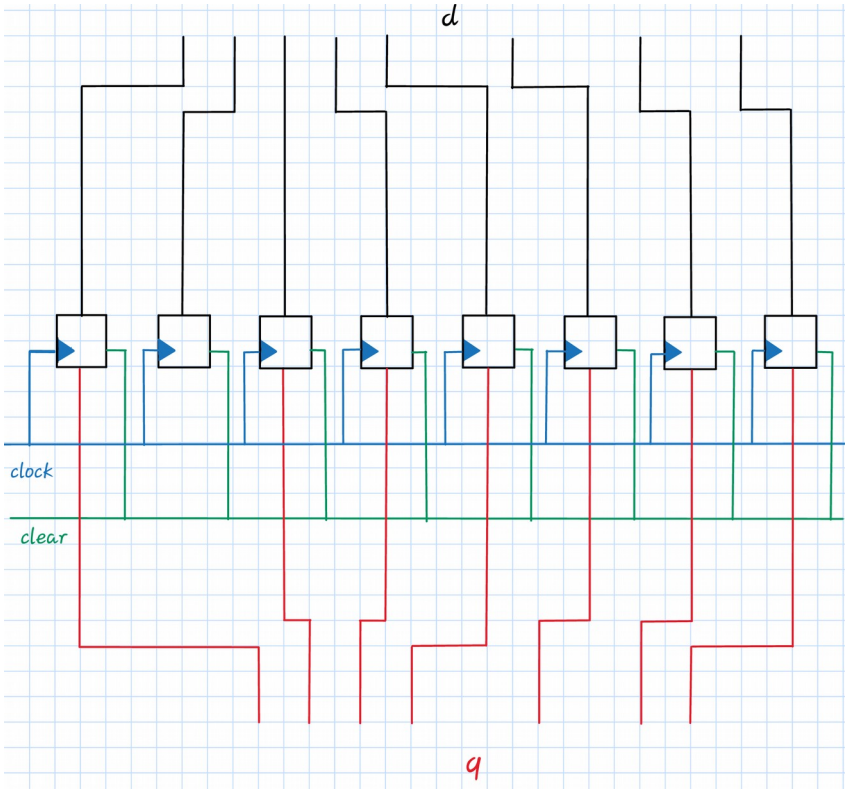
Il registro degli ingressi, infatti, viene utilizzato per sincronizzare gli ingressi A e FA del primo moltiplicatore, B ed FB del secondo, C ed FC del terzo.

Inlotre, è il componente che si occupa di effettuare il cambio del bi t più significativo (quello in posizione “7”) di ciascuno degli operandi in ingresso, a prescindere dal valore che ognuno di questi assuma, nel valore logico basso, cioè “0”.

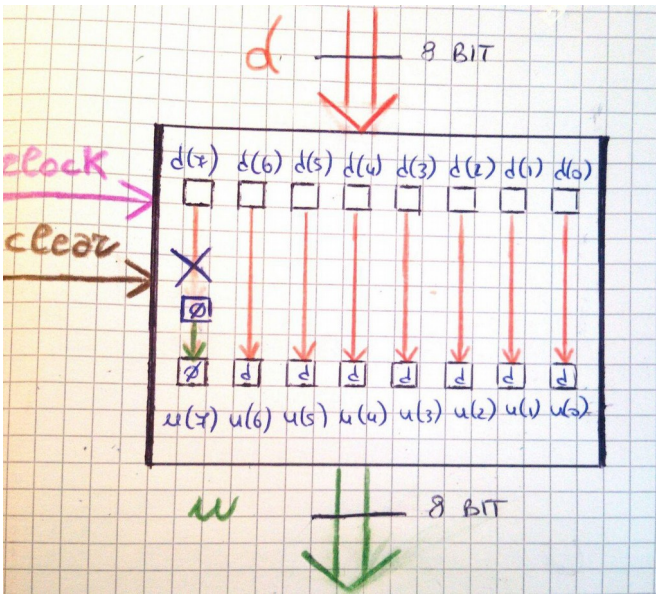
Questo per garantire che tutti gli ingressi non presentino valore negativo in segno.

La sua struttura, in parte analoga a quella di tutti i registri utilizzati fino a questo momento, a meno dell'operazione del cambio di bit, può essere rappresentata come segue.

È costituito da otto unità di Flip Flop che si occupano di memorizzare, quando agisce il colpo di clock, il valore del segnale d ricevuto in ingresso, nell'uscita q, come avviene nel registro ad otto bit sottostante (il tutto se il valore logico del segnale di clear è basso, in quanto, altrimenti, svolgerebbe la funzione di reset):



Nel disegno sottostante, invece, verrà evidenziata, da punto di vista strutturale, come avviene proprio l'operazione di cambio del bit più significativo:



Il codice VHDL che descrive questo componente è il seguente:

```
--Registro che effettua il cambio
--dei bit piu' significativi degli ingressi ricevuti
--dal circuito da realizzare nel valore '0'
--si include la libreria
library IEEE;
--si include il package di interesse della libreria
use IEEE.std_logic_1164.all;
--entita'
Entity RegistroIngressi is
--REGISTRO DI CAMBIO BIT SIGNIFICATIVO
--riceve in ingresso un vettore di otto bit
--ne cambia il bit piu' significativo,
--quello in posizione 7 e salva tutto in
--un vettore ad otto bit
     port(
          i: in std_logic_vector(7 downto 0);
          clock,clear: in std_logic;
          u: out std_logic_vector(7 downto 0)
     );
end RegistroIngressi;
--architettura
Architecture MyRI of RegistroIngressi is
--si assegna l'ingresso modificato
--nel vettore di uscita
begin
--inizio delle operazioni

u(7)<='0';  --si effettua il cambio del bit in posizione 7
u(6)<=i(6);
u(5)<=i(5);
u(4)<=i(4);
u(3)<=i(3);
u(2)<=i(2);
u(1)<=i(1);
u(0)<=i(0);


end MyRI;
```

Registro a 16 bit

E' il componente di memorizzazione utilizzato nel modulo Sommatore3Prod che si occupa di sommare i prodotti a sedici bit ottenuti dai tre moltiplicatori.

Essendo questi suoi ingressi, è questo stesso componente ad occuparsi della loro sincronizzazione e quindi conseguente trasferimento in tre registri a sedici bit ciascuno.

La struttura di questo componente è analoga a quella che è stata utilizzata nella composizione del registro ad otto bit illustrato, infatti, l'unica differenza consiste nell'aggiunta di flip flop in base alle necessità.
Allora, in totale, questo registro conterrà 16 di queste unità elementari.

Si riporta il codice del componente:

```
--Registro a 16 Flip Flop di tipo std_logic
--e' realizzato in forma compatta, senza istanziare i flip flop ma
--scrivendo direttamente il process
--si include la libreria
library IEEE;
--si include il modulo di interesse della libreria
use IEEE.std_logic_1164.all;
--componente usato per immagazzinare le otto cifre con cui lavorera'
il sommatore
Entity Register16 is
--ingressi:
--d vettore da 16 celle,clock,clear;
--uscite:
--q vettore di uscita da 16 celle;
     port(
          d : in std_logic_vector(15 downto 0);
          clock,clear: in std_logic;
          q : out std_logic_vector(15 downto 0)
     );
end Register16;

Architecture MyRegistro of Register16 is

     begin

     process(clock,clear)
```

```
                begin
                     if clear = '1' then
                            q <= "0000000000000000";

                     elsif clock'event and clock = '1'
                           then q <= d;

                     end if;
     end process;

end MyRegistro;
```

Registro a 17 bit

Questo componente di memorizzazione è utilizzato sia nel modulo Sommatore3Prod, che già impiega il registro a sedici bit descritto precedentemente, sia nel modulo CircuitoMul che lo impiega per sincronizzare l'uscita del circuito complessivo, ovvero la somma che si è generata addizionando tra loro i tre progotti ottenuti dai moltiplicatori.

Sia la struttura che il codice scritto per descriverlo non presentano alcuna differenza sostanziale rispetto al registro a sedici bit, anche se in questo caso le unità fondamentali Flip Flop da istanziare per il suo funzionameto saranno diciassette.

Segue il suo codice VHDL:

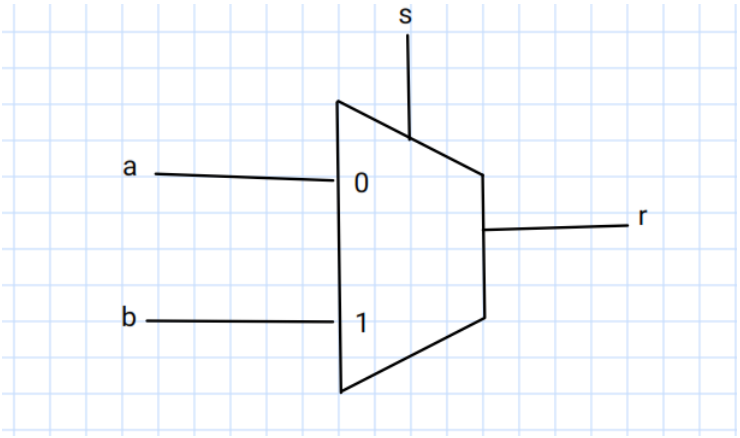
```
--Registro a 17 Flip Flop di tipo std_logic
--e' realizzato in forma compatta, senza istanziare i flip flop ma
--scrivendo direttamente il process
--si include la libreria
library IEEE;
--si include il modulo di interesse della libreria
use IEEE.std_logic_1164.all;
--componente usato per immagazzinare le otto cifre con cui lavorera'
--il sommatore
```

```
Entity Register17 is

--ingressi:
--id vettore da 17 celle,clock,clear;
--uscite:
--uq vettore di uscita da 17 celle;
     port(
          id : in std_logic_vector(16 downto 0);
          clock,clear: in std_logic;
          uq : out std_logic_vector(16 downto 0)
     );

end Register17;

Architecture MyRegistro of Register17 is
--inizio delle operazioni
begin

process(clock,clear)
     begin
          if clear = '1' then
                 uq <= "00000000000000000";

          elsif clock'event and clock = '1'
                     then uq <= id;

          end if;
     end process;

end MyRegistro;
```

Multiplexer

Il Mux riceve tre bit: due ricevuti come ingressi ed un bit di controllo che servirà per instradare uno tra i due ingressi verso l'uscita.
Questo componente è utilizzato nell'implementazione del Full Adder.

Può essere rappresentato, graficamente, in questo modo:



Il codice che lo descrive è questo:

```
--Multiplexer std_logic
--si importa la libreria
library IEEE;
--si include il modulo di interesse della libreria
use IEEE.std_logic_1164.all;
--questo componente, tra i piu' ingressi ricevuti, in base al valore del selettore,
--ne instradera' uno soltanto verso l'uscita
entity Mux is
--a e b saranno gli ingressi, s il selettore e r il risultato
     port(
          a,b,s: in std_logic;
          r: out std_logic);
end Mux;

architecture MyMux of Mux is
--parte dichiarativa di segnali ausiliari
--parte descrittiva
begin
--si definisce la funzione s
```

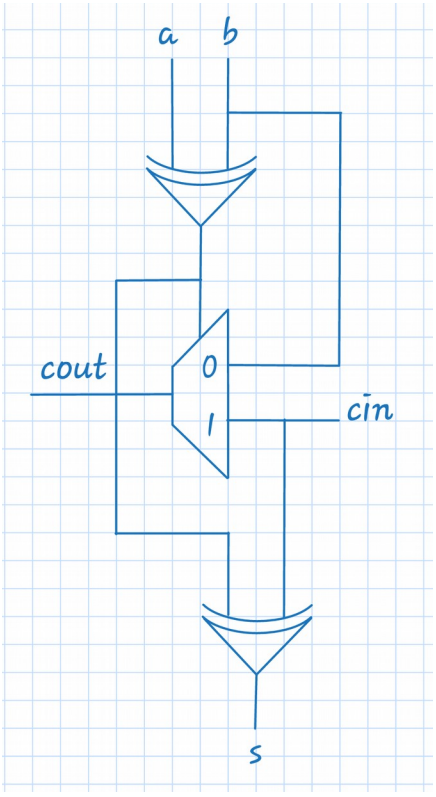
```
r <= ( a and (not s) ) or (b and s);
end MyMux;
```

Full Adder

E' il componente basilare sia per il funzionamento dei Ripple carry ad otto, quindici, sedici e diciassette ingressi impiegati per svolgere le varie somme calcolate nel progetto, sia per il calcolo del riporto totale che si genera all'interno del modulo Adder15.

Il Full Adder è realizzato con due xor e due mux, riceve tre bit in ingresso e restituisce la somma di questi ultimi, con l'eventuale riporto che si genera.

Lo si può rappresentare in questo modo:



Segue il codice Vhdl:

```
--Full Adder std_logic
--usa un componente Mux
--inclusione della libreria
library IEEE;
--inclusione del package di interesse
use IEEE.std_logic_1164.all;

--definizione dell'entita'
Entity FullAdder is
--elenco delle porte e specificazione del loro utilizzo
     port(
           a,b,cin: in std_logic;
           cout,s:out std_logic);

end FullAdder;
--definizione dell'architettura
Architecture FA of FullAdder is
--parte dichiarativa
--segnali ausiliari per il FA:propagate p
signal p:std_logic;
--si include il componente mux
--questo instradera' uno solo degli ingressi verso l'uscita,
--scegliendo quello il cui valore posizionale, in nbn, corrisponde
--del selettore
component Mux is
--si specificano le porte che utlizza nell'ordine in cui
--sono state definite
     port(
          a,b,s: in std_logic;
          r: out std_logic);
end component;
--parte descrittiva
begin
--si istanzia il componente che si e' incluso dandogli un nome
--ingressi:b e cin, uscita: cout, selettore: p
MX: Mux port map(b,cin,p,cout);
--p
p <= a xor b;
--somma s
s <= p xor cin;
--riporto in uscita cout
end FA;
```

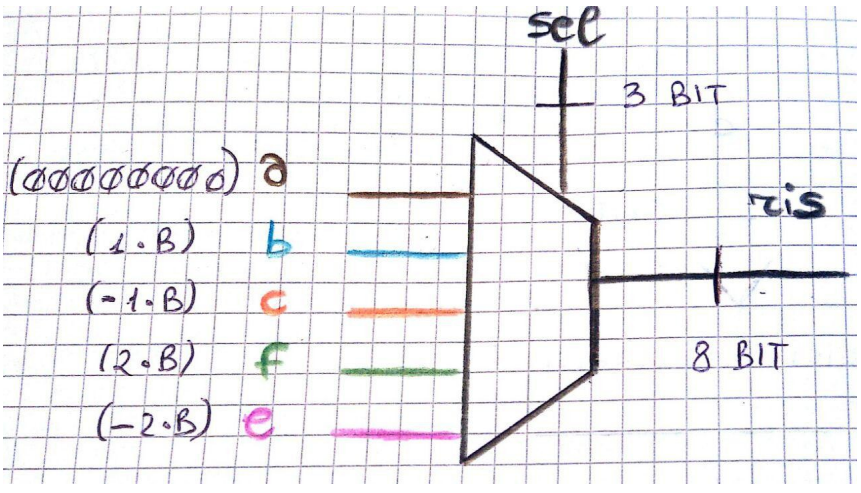
Muliplexer a cinque ingressi

Il componente in esame è ul Mux a cinque ingressi, oltre il segnale di controllo “sel”.
Il suo scopo è analogo a quello del basilare Multiplexer che è stato utilizzato fino a questo momento, infatti, dei cinque segnali a nove bit ciascuno che riceve in ingresso, si occupa, in base al valore del selettore a tre bit, di instradarne uno soltanto verso l'uscita, che sarà quindi, anch'essa, a nove bit.

Per la sua realizzazione è stato ultilizzato utilizzato il costrutto “with..select..then”.
In base al valore del selettore sel a tre bit, l'uscita “ris” assumerà uno tra i cinque valori ricevuti in ingresso, ovvero a,b,c,d,e.
Essendo questo componente utilizzato all'interno del moltiplicatore, la sua funzione sarà proprio quella di generale il “prototipo” del prodotto parziale.
Il selettore altro non è che quello che si è generato, a partire da una delle quattro terne in cui è stato scomposto il **moltiplicando**, secondo la **codifica di Booth**.

Assegnati A e B come ingressi del moltiplicatore, allora i valori possibili che potranno essere assegnati da questo tipo di Mux all'uscita sono:
-a, che rappresenta il valore “00000000”;
-b, che rappresenta il moltiplicatore così com'è, ovvero “1*B”;
-c, che rappresenta il moltiplicatore negativo in segno, cioè “-1*B”;
-d, che rappresenta il doppio del moltiplicatore, “2*B”;
-e, che è il doppio del moltiplicatore negativo in segno, “-2*B”;
Ulteriori precisazioni verranno fatte nella descrizione del funzionamento della struttura complessiva del moltiplicatore, dopo aver descritto separatamente i suoi componenti interni.

Dal punto di vista strutturale il Mux sarà sì fatto:



Segue il codice:

```
--Multiplexer std_logic a cinque ingressi a nove bit
--si importa la libreria

library IEEE;
--si include il modulo di interesse della libreria
use IEEE.std_logic_1164.all;
--questo componente, tra i 5 ingressi ricevuti, in base al valore del
--selettore,
--ne instradera' uno soltanto verso l'uscita
--lavora con ingressi a 9 bit, restituendo, in uscita un risultato a
--nove bit
entity Mux5i9b is
     port(
           a,b,c,d,e:in std_logic_vector(8 downto 0);
           sel: in std_logic_vector(2 downto 0);
           ris: out std_logic_vector(8 downto 0)
         );
end Mux5i9b;

architecture MyMux of Mux5i9b is
     --operazioni del mux a cinque ingressi
     --controlli sul valore del selettore a tre ingressi
     begin
          with sel select
               ris <=
                    --sel=0
                    a when "000",
                    --sel=1
                    b when "001",
                    --sel=2
                    c when "010",
                    --sel=3
                    d when "011",
                    --sel=4
                    e when "100",
               "XXXXXXXXX" when others;
--altrimenti, per qualsiasi altro valore rimanente di sel
--Nota:si sceglie la seguente notazione:
--a come "00000000";
--b come "moltiplicatore";
--c come "- moltiplicatore";
--d come "2 per moltiplicatore";
--e come "-2 per moltiplicatore";
end MyMux;
```

Shift Left

Lo Shift verso a sinistra, di un certo numero di posizioni, viene utilizzato per simulare e ottenere lo stesso risultato che si avrebbe effettuando un'operazione di moltiplicazione.
In questo progetto, i moduli di Shift utilizzati sono quattro, e tutti quanti verrano impiegati per “shiftare” tre delle quattro uscite generate, ognuna, da ciascuno di due dei quattro Mux a cinque ingressi utilizzati nel moltiplicatore, il modulo denominato MulBooth.

L'unico modulo che non viene utilizzato in questo contesto è lo shift di una sola posizione, che, al contrario degli altri tipi di shift, che avranno come ingresso un numero a nove bit, il suo segnale di input sarà un numero ad otto bit.

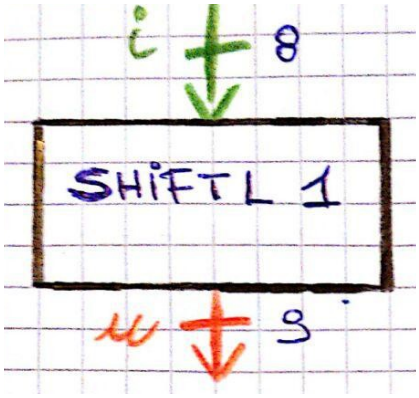
L'utilizzo di uno qualsiasi di questi componenti fa sì che, il risultato ottenuto, incrementi il suo numero di bit e che gli “spazi vuoti” nelle posizioni meno significative che si generano dallo spostamento del numero verso sinistra, cioè verso le posizioni più significative, vengano riempiti con degli zeri.

Shift di una posizione

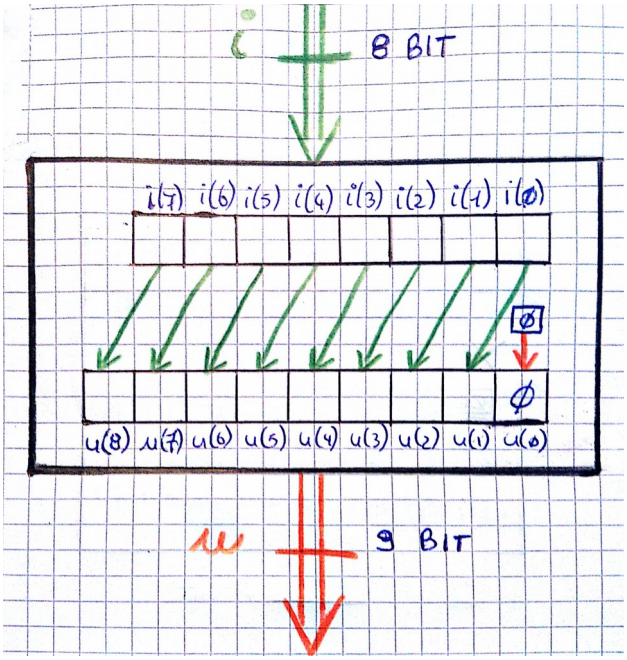
Equivale a moltiplicare per un fattore “2” il numero che riceve in ingresso, ad otto bit.
L'uscita corrispondente sarà, in questo caso, un numero a nove bit, che sarà costituito dall'aver spostato di una sola posizione tutti i bit del numero iniziale e dall' aver riempito la posizione meno significativa, rimasta di conseguenza vuota, con uno “0”.

Questo modulo è utilizzato sia nel componente Complemento2, sia nel moltiplicatore MulBooth, per generare uno degli ingressi dei Mux5i, ovvero “e”, che è il doppio del moltiplicatore negativo in segno, “-2*B”, ottenuto proprio shiftando di una sola posizione l'altro ingresso del mux, “c”,cioè il moltiplicatore negativo in segno “-1*B”.

Verrà indicato da questa simbologia:



Graficamente, il suo funzionamento può essere rappresentato in questo modo:



Segue il codice VHDL del componente in esame:

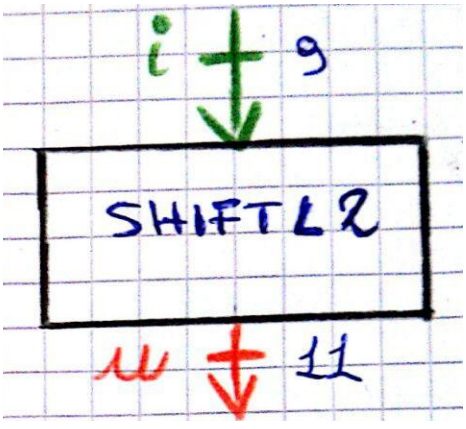
```
--modulo che si occupa di fare lo shift
--verso sinistra, di una sola posizione
--si include la libreria
library IEEE;
--si include il modulo di interesse della libreria
use IEEE.std_logic_1164.all;
--MOLTIPLICAZIONE PER 2
--Da 8 bit a 9 bit
Entity Shiftl1 is
--riceve in ingresso un numero ad otto bit
--restituendo in uscita un numero a nove bit
--che presenta, nella sua cella meno significativa
--(quella piu' a destra) uno zero in piu'
--per occupare la casella rimasta vuota dopo
--l'operazione di shifting
     port(
          i : in std_logic_vector(7 downto 0);
          u : out std_logic_vector(8 downto 0)
          );

end Shiftl1;
```

```
--definizione del funzionamento del componente
Architecture MyS1 of Shiftl1 is
     --operazioni: da otto bit a nove bit
     begin

     u(8)<=i(7);
     u(7)<=i(6);
     u(6)<=i(5);
     u(5)<=i(4);
     u(4)<=i(3);
     u(3)<=i(2);
     u(2)<=i(1);
     u(1)<=i(0);
     u(0)<='0';

end MyS1;
```

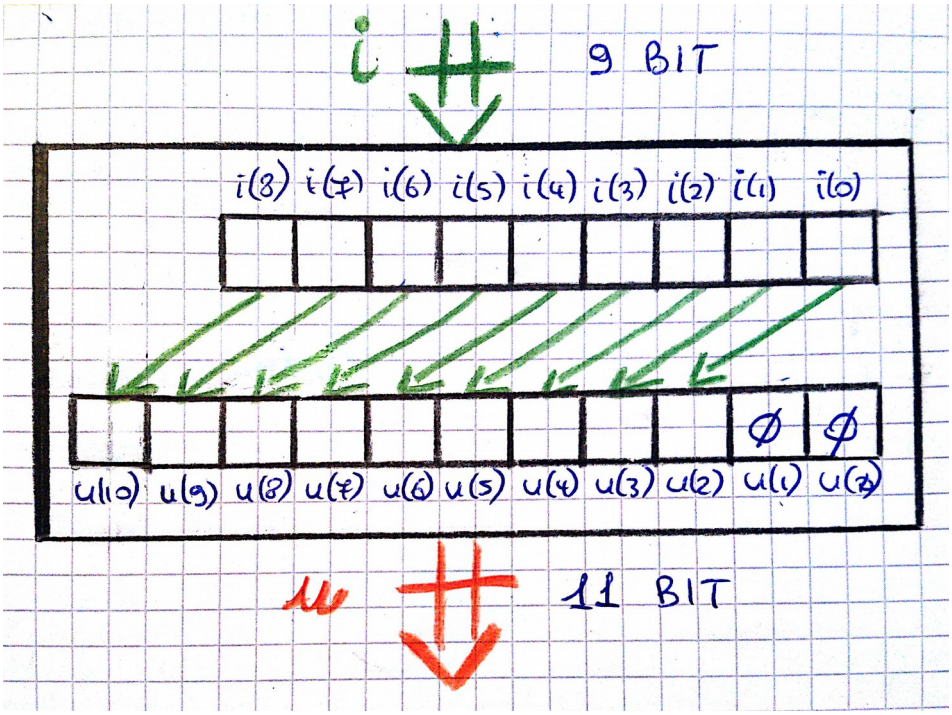
Shift di due posizioni

Equivale a moltiplicare per un fattore “4” il numero che riceve in ingresso, a nove bit.
L'uscita corrispondente sarà, in questo caso, un numero ad undici bit, che sarà costituito dall'aver spostato di due posizioni tutti i bit del numero iniziale e dall'aver riempito le due posizioni meno significative, rimaste, di conseguenza, vuote, con degli “0”.
Questo modulo è utilizzato all'interno di MulBooth e viene utilizzato per shiftare di due posizioni l'uscita del secondo mux, Mux2.

Verrà indicato in questo modo:



Ecco il suo funzionamento:



Il codice VHDL che lo implementa è il seguente:

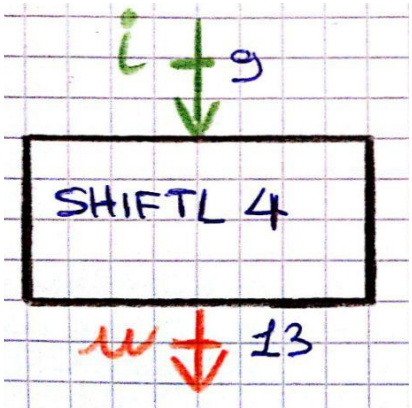
```
--modulo che si occupa di fare lo shift
--verso sinistra, di due posizioni
--si include la libreria
library IEEE;
--si include il modulo di interesse della libreria
use IEEE.std_logic_1164.all;
--MOLTIPLICAZIONE PER 4
--Da 9 bit a 11 bit
Entity Shiftl2 is
--riceve in ingresso un numero a 9 bit
--restituendo in uscita un numero a 11 bit
--che presenta, nelle sua celle meno significative
--(quelle piu' a destra) degli zeri in piu'
--per occupare le caselle rimasta vuota dopo
--l'operazione di shifting
     port(
          i : in std_logic_vector(8 downto 0);--9 bit in ingresso
          u : out std_logic_vector(10 downto 0)--11 bit in uscita
          );

end Shiftl2;
```

```
--definizione del funzionamento del componente
Architecture MyS2 of Shiftl2 is
     --operazioni: da nove bit ad undici bit
     begin

     u(10)<=i(8);
     u(9)<=i(7);
     u(8)<=i(6);
     u(7)<=i(5);
     u(6)<=i(4);
     u(5)<=i(3);
     u(4)<=i(2);
     u(3)<=i(1);
     u(2)<=i(0);
     u(1)<='0';
     u(0)<='0';

end MyS2;
```

Shift di quattro posizioni

Equivale ad effettuare una motliplicazione del numero ricevuto in ingresso, a nove bit.
L'uscita corrispondente sarà, in questo caso, un numero a tredici bit, che sarà costituito dall'aver spostato di quattro posizioni tutti i bit del numero iniziale e dall'aver riempito le quattro posizioni meno significative, rimaste, di conseguenza, vuote, con degli “0”.
Questo modulo è utilizzato all'interno di MulBooth e viene utilizzato per shiftare di quattro posizioni l'uscita del terzo mux, Mux3.

Essendo la logica che descrive il suo funzionamento la stessa di quella impiegata negli altri moduli Shiftl, si riporta soltanto la simbologia con cui questo componente verrà indicato:



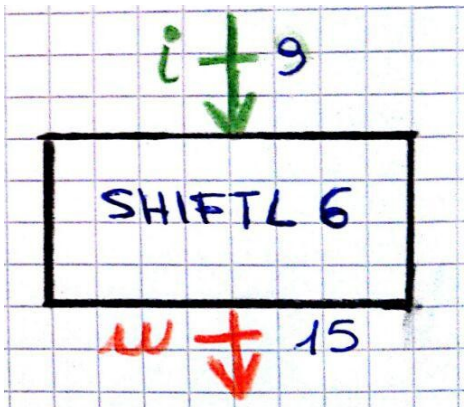
Segue il codice VHDL:

```
--modulo che si occupa di fare lo shift
--verso sinistra, di quattro posizioni
--si include la libreria
library IEEE;
--si include il modulo di interesse della libreria
use IEEE.std_logic_1164.all;
--MOLTIPLICAZIONE
--Da 9 bit a 13 bit
Entity Shiftl4 is
--riceve in ingresso un numero a 9 bit
--restituendo in uscita un numero a 13 bit
--che presenta, nelle sua celle meno significative
--(quelle piu' a destra) degli zeri in piu'
--per occupare le caselle rimasta vuota dopo
--l'operazione di shifting
     port(
          i : in std_logic_vector(8 downto 0);--9 bit in ingresso
          u : out std_logic_vector(12 downto 0)--13 bit in uscita
          );

end Shiftl4;

--definizione del funzionamento del componente
Architecture MyS4 of Shiftl4 is
     --operazioni: da nove bit a tredici bit
     begin

     u(12)<=i(8);
     u(11)<=i(7);
     u(10)<=i(6);
     u(9)<=i(5);
     u(8)<=i(4);
     u(7)<=i(3);
     u(6)<=i(2);
     u(5)<=i(1);
     u(4)<=i(0);
     u(3)<='0';
     u(2)<='0';
     u(1)<='0';
     u(0)<='0';

end MyS4;
```

Shift di sei posizioni

Equivale ad effettuare una motliplicazione del numero ricevuto in ingresso, a nove bit.
L'uscita corrispondente sarà, in questo caso, un numero a quindici bit, che sarà costituito dall'aver spostato di sei posizioni tutti i bit del numero iniziale e dall'aver riempito queste sei posizioni meno significative, rimaste, di conseguenza, vuote, con degli “0”.
Questo modulo è utilizzato all'interno di MulBooth e viene utilizzato per shiftare di quattro posizioni l'uscita del quarto mux, Mux4.

Essendo la logica che descrive il suo funzionamento la stessa di quella impiegata negli altri moduli Shiftl, si riporta soltanto la simbologia usata per identificarlo:



Segue il condice VHDL che descrive le funzioni del componente:

```
--modulo che si occupa di fare lo shift
--verso sinistra, di sei posizioni
--si include la libreria
library IEEE;
--si include il modulo di interesse della libreria
use IEEE.std_logic_1164.all;
--MOLTIPLICAZIONE
--Da 9 bit a 15 bit
Entity Shiftl6 is
--riceve in ingresso un numero a 9 bit
--restituendo in uscita un numero a 15 bit
--che presenta, nelle sua celle meno significative
--(quelle piu' a destra) degli zeri in piu'
--per occupare le caselle rimasta vuota dopo
```

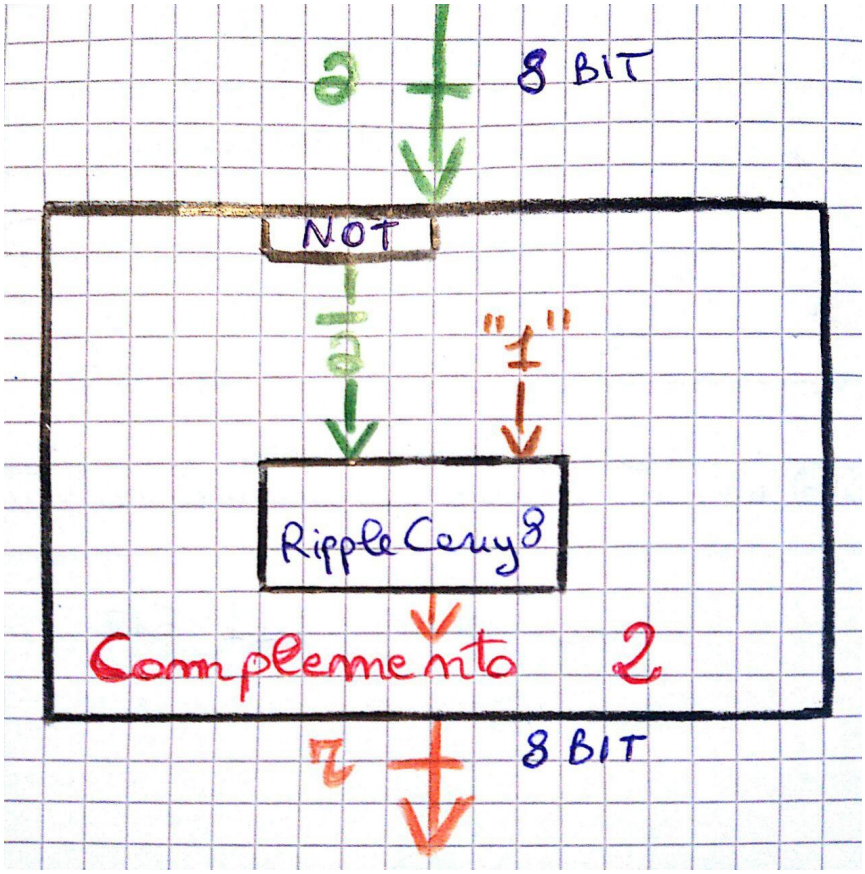
```
--l'operazione di shifting

     port(
          i : in std_logic_vector(8 downto 0);--9 bit in ingresso
          u : out std_logic_vector(14 downto 0)--15 bit in uscita
          );

end Shiftl6;

--definizione del funzionamento del componente
Architecture MyS6 of Shiftl6 is
     --operazioni: da nove bit a quindici bit
     begin

     u(14)<=i(8);
     u(13)<=i(7);
     u(12)<=i(6);
     u(11)<=i(5);
     u(10)<=i(4);
     u(9)<=i(3);
     u(8)<=i(2);
     u(7)<=i(1);
     u(6)<=i(0);
     u(5)<='0';
     u(4)<='0';
     u(3)<='0';
     u(2)<='0';
     u(1)<='0';
     u(0)<='0';


end MyS6;
```

Complemento a Due

Questo modulo, dato in ingresso un numero ad otto bit, restituisce come uscita il suo complemento a due, sempre ad otto bit.

Questo viene realizzato effettuato il negato, bit a bit, del segnale ricevuto in ingresso e sommando il valore “00000001” al vettore risultante, rneg.

L'operazione di somma viene effettuata tramite un Ripple Carry ad otto bit, non sequenziale, di cui verrà riportato il codice ma non il suo disegno strutturale, in quanto già descritto e rappresentato nei progetti a questo precedenti.
Si riporta la sua rappresentazione grafica ai fini di esplicitarne il funzionamento:



Codice VHDL che lo implementa:

```
--modulo che esegue il Complemento a due di un numero ad otto bit
--Il complemento a due viene effettuato negando ogni bit e
--sommando uno al numero così ottenuto
--si include la libreria
library IEEE;
--si include il package di interesse della libreria
use IEEE.std_logic_1164.all;
--entita'
```

```
Entity Complemento2 is
--riceve in ingresso un numero ad otto bit
--e restituisce il suo complemento a due
     port(
          a:in std_logic_vector(7 downto 0);
          r: out std_logic_vector  (7 downto 0)--8 bit
          );
end Complemento2;

--architettura

Architecture C2 of Complemento2 is
--inclusione del componente

--inclusione del ripple carry ad otto bit
component RippleCarry8 is
     port(
          a,b: in std_logic_Vector(7 downto 0);
          cin: in std_logic;
          s: out std_logic_Vector(7 downto 0);
          cout: out std_logic);
end component;

--segnali ausiliari
signal rneg: std_logic_vector(7 downto 0);
--vettore che contiene il negato dell'ingresso

signal nshift: std_logic_vector(7 downto 0):="00000001";

--inizio delle operazioni
begin

--negato bit a bit

rneg(0)<=not a(0);
rneg(1)<=not a(1);
rneg(2)<=not a(2);
rneg(3)<=not a(3);
rneg(4)<=not a(4);
rneg(5)<=not a(5);
rneg(6)<=not a(6);
rneg(7)<=not a(7);

--si istanzia il componente
PiuUno: RippleCarry8 port map(rneg,nshift,'0',r);
```

```
--il riporto va scartato, quindi non predispongo l'uscita
corrispondente

end C2;
```

Per completezza, ecco il codice del Ripple Carry ad otto bit:

```
library IEEE;
use  IEEE.std_logic_1164.all;
Entity RippleCarry8 is
--ingressi:a vettore di 8 bit, b vettore di 8 bit;
--uscite:s vettore di 9 bit, e' il risultato della somma tra i due
--        numeri a 8 bit a e b;cout,riporto in uscita

     port(
          a,b: in std_logic_Vector(7 downto 0);
          cin: in std_logic;
               s: out std_logic_Vector(7 downto 0);
               cout: out std_logic);
end RippleCarry8;

Architecture MyRC8 of RippleCarry8 is
--segnali ausiliari
signal c: std_logic_vector(6 downto 0);--mantiene memoria dei riporti
--si include il FullAdder che eseguira' le somme
component FullAdder is
     port(
          a,b,cin: in std_logic;
             cout,s:out std_logic);
end component;
begin


--Operazioni del RC

FA0: FullAdder port map(a(0),b(0),cin,c(0),s(0));

FA1: FullAdder port map(a(1),b(1),c(0),c(1),s(1));

FA2: FullAdder port map(a(2),b(2),c(1),c(2),s(2));

FA3: FullAdder port map(a(3),b(3),c(2),c(3),s(3));

FA4: FullAdder port map(a(4),b(4),c(3),c(4),s(4));
```

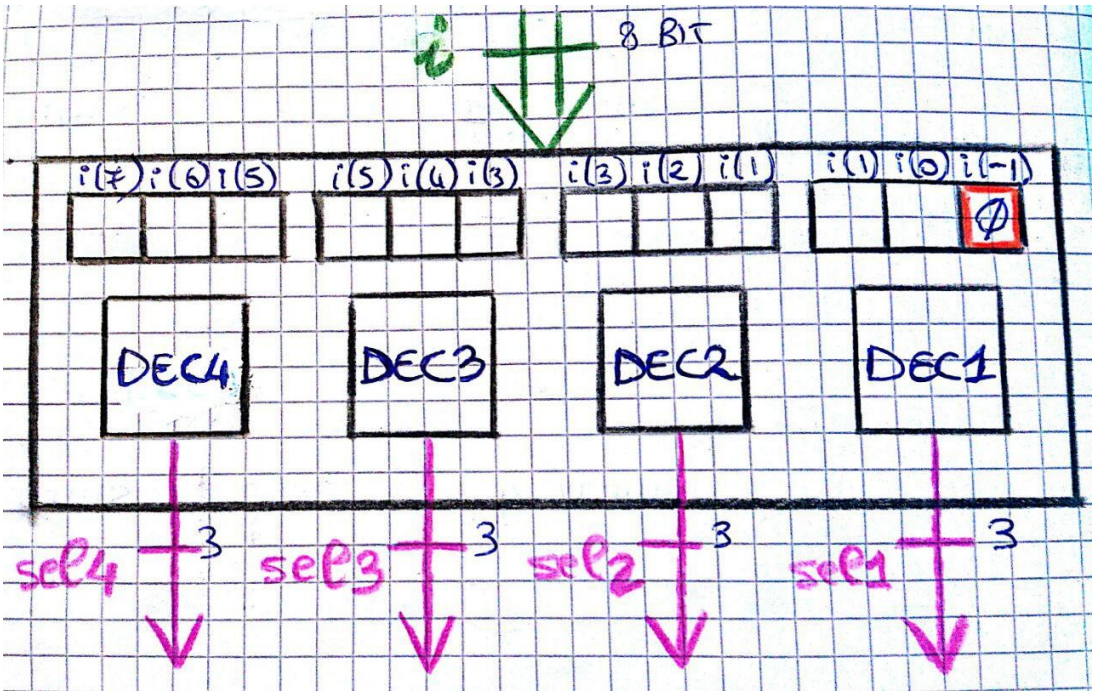
```
FA5: FullAdder port map(a(5),b(5),c(4),c(5),s(5));

FA6: FullAdder port map(a(6),b(6),c(5),c(6),s(6));

FA7: FullAdder port map(a(7),b(7),c(6),cout,s(7));

end MyRC8;
```

Divisore del moltiplicando in quattro terne

Questo modulo, denominato DivisoreTerne, si occupa di dividere in quattro terne da tre bit ciascuna, il moltiplicando, ovvero il flusso A in ingresso al circuito complessivo.
È necessario effettuare questa operazione perché, in base alle terne ottenute, verranno stabiliti i valori dei quattro selettori (sel1,sel2,sel3,sel3) che costituiranno i segnali di controllo a tre bit dei quattro Mux5i9b utilizzati per la generazione dei “prototipi” dei prodotti parziali, ovvero i prodotti parziali a nove bit, non ancora resi a quindici bit e shiftati, a meno di quello generato dal Mux1.

Oltre che separare l'ingresso in quattro terne, questo modulo si occuperà di istanziare i componenti Dec che dovranno attuare la codifica secondo il codice di Booth, il fondamento dell'unità moltiplicatore che si vuole realizzare.

Si riporta, per chiarezza, un disegno che descriva la suddivisione delle terne ed anche il funzionamento di questo componente:



Si può notare come, nella generazione delle terne, ogni volta venga ripreso, come primo bit della terna successiva, l'ultimo bit della terna precedente.
Inoltre, per quanto riguarda la prima terna, il suo primo bit, quello nella posizione meno significativa, indicata dall'indice “-1”, assumerà il valore logico “0”, in quanto non viene preso dai bit che compongono il flusso A in ingresso al DivisoreTerne.

Si riporta il suo codice in VHDL:

```
--Divisore in terne
--Riceve un numero ad otto bit
--e lo scompone in terne da passare
--al modulo Dec che attua la decodifica
--in codice di Booth
--si include la libreria
library IEEE;
--si include il package di interesse della libreria
use IEEE.std_logic_1164.all;
```

```
--entita'
Entity DivisoreTerne is
--riceve in ingresso un numero ad otto bit
--fornisce in uscita quattro flussi a tre bit ciascuno
--che saranno i selettori dei quattro mux usati nel circuito
     port(
          i: in std_logic_vector(7 downto 0);
          sel1: out std_logic_vector(2 downto 0);
          sel2: out std_logic_vector(2 downto 0);
          sel3: out std_logic_vector(2 downto 0);
          sel4: out std_logic_vector(2 downto 0)
     );

end DivisoreTerne;
--definizione dell'architettura
Architecture MyDT of DivisoreTerne is
--si include il componente Decodificatore
component Dec is
     port(
          i: in std_logic_vector(2 downto 0);
          u: out std_logic_vector(2 downto 0)
     );
end component;
--si istanziano 4 dec, ciascuno dei quali ricevera'
--la propria terna
--i segnali ausiliari saranno costituiti dai vettori
--da tre celle che quindi gli verranno dati in ingresso
signal terna1, terna2, terna3, terna4 : std_logic_vector(2 downto 0);

--inizio delle operazioni
begin
--assegnamento dei vettori
--PRIMA TERNA
terna1(0)<='0';--aggiunta i(-1)
terna1(1)<=i(0);
terna1(2)<=i(1);

--SECONDA TERNA

--Nota: il primo bit della terna successiva e'
--l'ultimo bit della terna precedente

terna2(0)<=i(1);
terna2(1)<=i(2);
terna2(2)<=i(3);
```

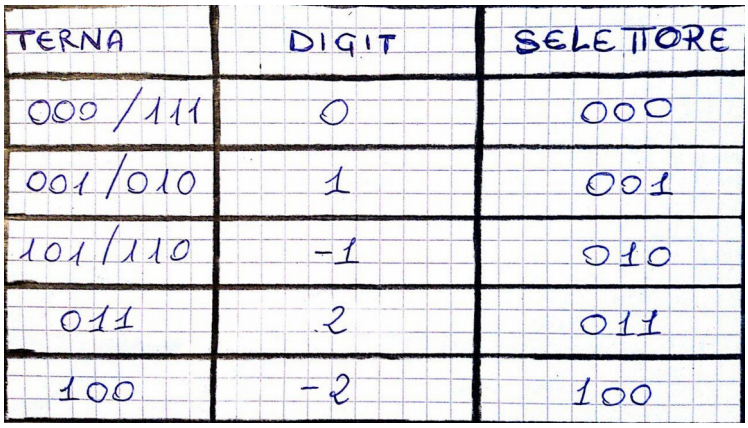
```
--TERZA TERNA
terna3(0)<=i(3);
terna3(1)<=i(4);
terna3(2)<=i(5);

--QUARTA TERNA
terna4(0)<=i(5);
terna4(1)<=i(6);
terna4(2)<=i(7);

--si istanziano i codificatori secondo la tabella di Booth


--DEC1 per la prima terna
Dec1: Dec port map(terna1,sel1);

--DEC2 per la seconda terna
Dec2: Dec port map(terna2,sel2);

--DEC3 per la terza terna
Dec3: Dec port map(terna3,sel3);

--DEC4 per la quarta terna
Dec4: Dec port map(terna4,sel4);

end MyDT;
```

Decodificatore delle terne secondo il codice di Booth

Questo modulo, denominato Dec, si occupa di decodificare le terne ricevute, secondo il codice di Booth.
Infatti, ricevendo in ingresso un segnale a tre bit, lo interpreta, secondo la tabella di Booth riportata qui di seguito, per restituire in uscita, un vettore di tre bit che andrà a costituire il segnale di controllo dei mux che generano i prodotti parziali.

Dalla prima tabella si può vedere come, in base al valore del Digit, venga poi stabilito il risultante prodotto parziale, senza fare nessun altro calcolo intermedio.
Nella seconda tabella, invece, viene messo in evidenza, più che il risultato che si otterrà alla fine di questa operazione, la logica su cui si basa il blocchetto Dec.
Infatti, dovendo questo componente produrre in uscita il valore del selettore che dovranno ricevere in ingresso i quattro mux, si mostra come, a partire dal valore della terna che riceve in ingresso, il valore da assegnare al selettore, in base all'ordine con cui i digit, moltiplicati ad A, che nella prima tabella rappresenta il moltiplicatore, vengano moltiplicati a quest'ultimo ed i risultati passati come i cinque ingressi di ciascun Mux, ovvero “a”, “b”, “c”, “d”, “e”.

Terna b_{i+1} b_i b_{i-1}	Digit codificato	Prodotto parziale
0 0 0	0	0
0 0 1	1	A
0 1 0	1	A
0 1 1	2	2A
1 0 0	-2	-2A
1 0 1	-1	-A
1 1 0	-1	-A
1 1 1	0	0



Si riporta il codice VHDL che realizza questo componente, implementato utilizzando il costrutto with “terna in ingresso” select… selettore <= “valore da assegnare al selettore” when “valore della terna” :

```
--modulo che si occupa di decodificare le terne ricevute
--secondo la tabella di CODIFICA DI BOOTH
library IEEE;
use IEEE.std_logic_1164.all;
--definizione dell'entita'
Entity Dec is
--riceve in ingresso tre bit e
--restituisce un numero in base
--ai valori stabiliti dalla tabella di Booth
     port(
          i: in std_logic_vector(2 downto 0);
          u: out std_logic_vector(2 downto 0)
     );
--l'uscita in pratica sara' il selettore del mux a cinque ingressi
end Dec;
--definizione del comportamento di questo componente
Architecture MyD of Dec is
--operazioni
--in base al valore del digit verra' mandato in output
--quello che sara' il valore del selettore dei mux a cinque ingressi
--l'uscita sara':
--"000", che comporta l'uscita a come "00000000";
--"001" che comporta l'uscita b come "moltiplicatore";
--"010" che comporta l'uscita c come "- moltiplicatore";
--"011" che comporta l'uscita d come "2 per moltiplicatore";
--"100" che comporta l'uscita e come "-2 per moltiplicatore";
     begin

          with i select       --”i” e' il segnale di controllo
               --assegnamento
               u <=
               --scelta
               "000" when "000" | "111",
               --prodotto parziale = 0, digit pari a 0
               "001" when "001" | "010",
               --prodotto parziale = moltiplicatore, digit pari a 1
               "010" when "101" | "110",
               --prodotto parziale = - moltiplicatore,digit pari a -1
               "011" when "011",
          --prodotto parziale=due volte moltiplicatore,digit pari a 2
               "100" when "100",
       --prodotto parziale= - due volte moltiplicatore,digit pari a -2
               "XXX" when others;
               --altri casi, non previsti
     end MyD;
```

Moltiplicatore di Booth ad otto bit

Questo modulo è il cuore del progetto, perché tre di queste unità si occuperanno di generare i prodotti che , sommati, costituiranno l'uscita del circuito da realizzare.
Come già è stato detto, i suoi ingressi, a e b, saranno due segnali ad otto bit che, dopo essere stati elaborati secondo l'algoritmo che utilizza la codifica di Booth, andranno a comporre un'uscita p, di sedici bit.

Allora, si ripercorrono, passo passo, i moduli che attraversano gli ingressi A e B e le operazioni a cui questi verranno sottoposti.

Per prima cosa, verranno composti i segnali dati in ingresso ai Mux a 5 ingressi di nove bit, mentre l'ingresso a verrà passato a DivisoreTerne, che restituirà i selettori degli stessi.

È solo a questo punto che si possono iniziare a costruire i prodotti parziali, in questo modo:

-il primo prodotto pp1, prendendo l'uscita del Mux1, denominata uop1 e rendendola a quindici bit, anteponendo, al suo valore, nelle celle più significative, il valore del bit uop1(8), per mantenerne il segno, sia questo “0” oppure “1”.

-il secondo prodotto pp2, prendendo l'uscita del Mux2, denominata uop2, shiftarla tramite il modulo Shift2l, che la restituisce ad unidici bit ed infine rendendola a quindici bit, anteponendo, al suo valore, nelle celle più significative, il valore del bit suop2(10), per mantenerne il segno.

-il terzo prodotto pp3, prendendo l'uscita del Mux3, denominata uop3, shiftarla tramite il modulo Shift4l, che la restituisce a tredici bit ed infine rendendola a quindici bit, anteponendo, al suo valore, nelle celle più significative, il valore del bit suop3(12), per mantenerne il segno.

-il quarto prodotto pp4, prendendo l'uscita del Mux4, denominata uop4, shiftarla tramite il modulo Shift6l, che la restituisce direttamente a quindici bit, senza che ci sia la necessità di riempire altre celle perché rimaste vuote, mantenendo il segno.

L'ultima cosa di cui si occupa ancora questo componente è di sommare i prodotti parziali calcolati, mandandoli in ingresso al componente Adder15, che verrà descritto in seguito.

Segue il codice dell'intero componente:

```
--Moltiplicatore di Booth
--e' una composizione di vari moduli
--che si occupa di realizzare l'operazione
--di moltiplicazione tra due operandi ad otto bit
--utilizzando gli shiftleft per poi sommare i prodotti parziali
--Riepilogo:
--riceve due ingressi ad otto bit e restituisce il loro prodotto a 16
bit
--si include la libreria
library IEEE;
--si include il package di interesse della libreria
use IEEE.std_logic_1164.all;
--definizione del componente
Entity MulBooth is
--porte coinvolte
     port(
           a,b: in std_logic_vector(7 downto 0);--otto bit
           p: out std_logic_vector(15 downto 0)--sedici bit
          );
end MulBooth;
--definizione dell'architettura
Architecture MyMB of MulBooth is
--inclusione dei componenti:
--all'inizio verranno usati, su A:
component DivisoreTerne is
--riceve in ingresso un numero ad otto bit
--fornisce in uscita quattro flussi a tre bit ciascuno
--che saranno i selettori dei quattro mux usati nel circuito
     port(
          i: in std_logic_vector(7 downto 0);
          sel1: out std_logic_vector(2 downto 0);
          sel2: out std_logic_vector(2 downto 0);
          sel3: out std_logic_vector(2 downto 0);
          sel4: out std_logic_vector(2 downto 0)
     );
--scompone l'ingresso in terne da passare
--al modulo Dec
end component;
```

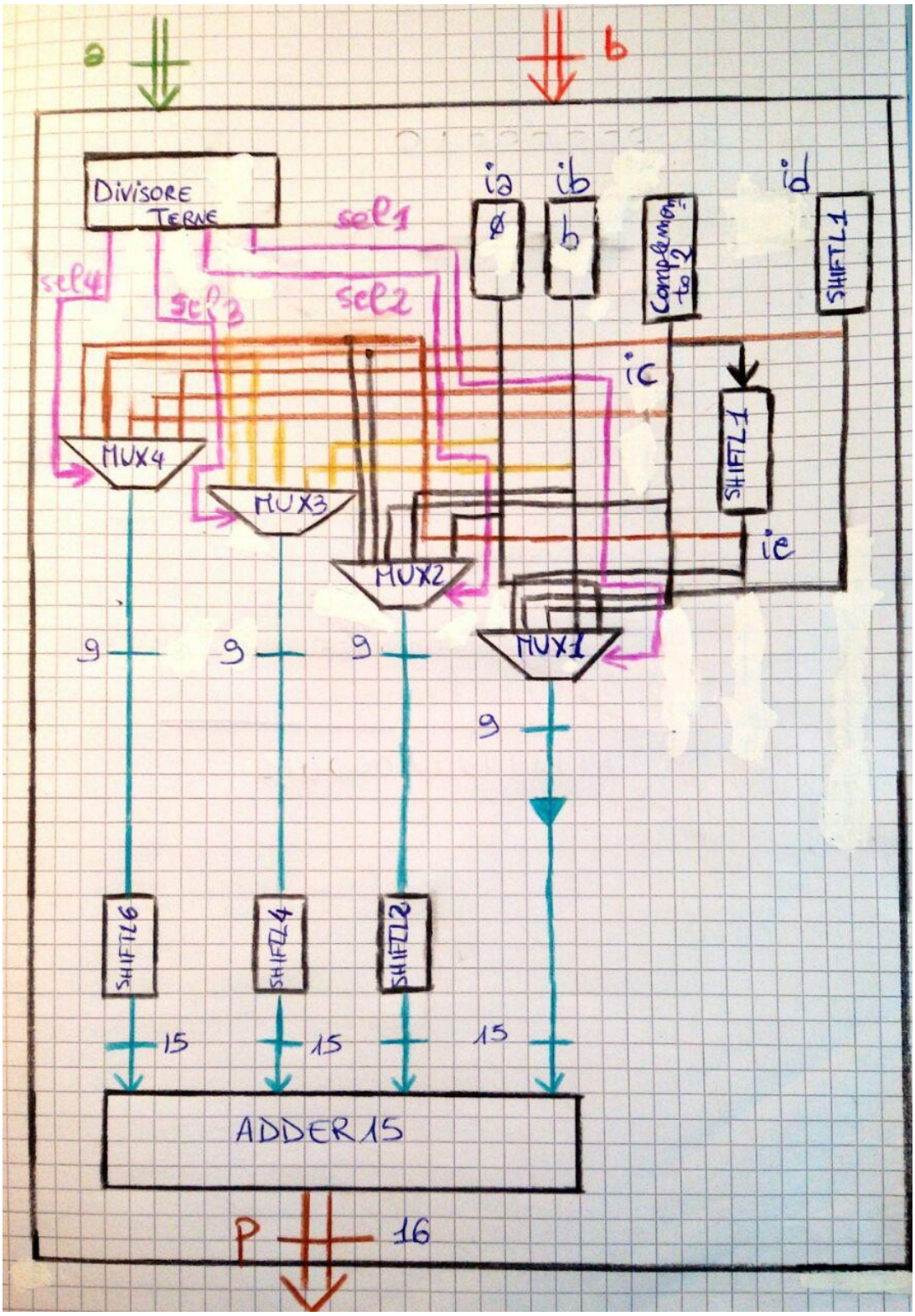
```
--all'inizio verranno usati su B:
component Complemento2 is
     port(
          a:in std_logic_vector(7 downto 0);
          r: out std_logic_vector(7 downto 0)--otto bit
          );
end component;
--Mux a 5 ingressi
component Mux5i9b is
     port(
          a,b,c,d,e:in std_logic_vector(8 downto 0);
          sel: in std_logic_vector(2 downto 0);
          ris: out std_logic_vector(8 downto 0)
         );
end component;
--Si includono tutti i moduli di Shifting
component Shiftl1 is
     port(
          i : in std_logic_vector(7 downto 0);
          u : out std_logic_vector(8 downto 0)
          );--da otto bit a nove
end component;

component Shiftl2 is
     port(
          i : in std_logic_vector(8 downto 0);--9 bit in ingresso
          u : out std_logic_vector(10 downto 0)--11 bit in uscita
          );
end component;

component Shiftl4 is
     port(
          i : in std_logic_vector(8 downto 0);--9 bit in ingresso
          u : out std_logic_vector(12 downto 0)--13 bit in uscita
          );
end component;

component Shiftl6 is
     port(
          i : in std_logic_vector(8 downto 0);--9 bit in ingresso
          u : out std_logic_vector(14 downto 0)--15 bit in uscita
          );
end component;
```

```
--sommatore dei prodotti parziali
component Adder15 is
--riceve i prodotti parziali gia' resi a 15 bit
--e li somma tra di loro, dando in output
--il risultato effettivo del prodotto tra i due operandi
--iniziali ad otto bit
  port(
     prodp1,prodp2,prodp3,prodp4:in std_logic_vector(14 downto 0);
           prodtot: out std_logic_vector(15 downto 0)
     );
end component;


--INIZIO DELLE OPERAZIONI

--segnali ausiliari usati per i quattro MUX
--ingressi
signal ia,ib,ic,id,ie: std_logic_vector(8 downto 0);
signal ictemp: std_logic_vector(7 downto 0);
--selettori
signal usel1,usel2,usel3,usel4: std_logic_vector(2 downto 0);
--uscite, numeri a nove bit
signal uop1,uop2,uop3,uop4: std_logic_vector(8 downto 0);
--variabile ausiliarie per trasformare le uscite dei mux, dopo le
--operazioni
--che calcolo i prodotti parziali, a 15 bit: prodotti parziali
signal pp1,pp2,pp3,pp4: std_logic_vector(14 downto 0);
--variabili temporanee per salvare le uscite dei mux che devono
--essere shiftate(tre di quattro)
signal suop2:std_logic_vector(10 downto 0);
signal suop3:std_logic_vector(12 downto 0);

begin --CALCOLI
--si ricavano i vari ingressi a nove bit, a partire da B, che
--ricevera' il mux1
--ia: "000000000"
ia<="000000000";

--ib: B stesso reso a nove bit
ib(8)<=b(7);
ib(7)<=b(7);
ib(6)<=b(6);
ib(5)<=b(5);
ib(4)<=b(4);
ib(3)<=b(3);
ib(2)<=b(2);
```

```
ib(1)<=b(1);
ib(0)<=b(0);

--ic: sara' B opposto in segno, si ottiene con il complemento a due
--rendendolo poi a nove bit
C2: Complemento2 port map(b,ictemp);
--trasformazione in nove bit
ic(8)<=ictemp(7);
ic(7)<=ictemp(7);
ic(6)<=ictemp(6);
ic(5)<=ictemp(5);
ic(4)<=ictemp(4);
ic(3)<=ictemp(3);
ic(2)<=ictemp(2);
ic(1)<=ictemp(1);
ic(0)<=ictemp(0);

--id: sara' B moltiplicato per due e reso a nove bit,
--si utilizza per farlo lo shift1 applicato sul complemento a due
--fatto su b,cioe' ictemp
B2: Shiftl1 port map(b,id);

--ie: sara' l'opposto di B moltiplicato per 2, sempre a nove bit
--lo ottengo shiftando di uno il valore di ic( che e' proprio -B)
Bmeno2: Shiftl1 port map(ictemp,ie);

--ora si devono costruire i selettori che riceveranno i quattro mux
da istanziare
--questi sono le quattro uscite generate dai blocchetti Dec
--Si ottiene come segue:
DT: DivisoreTerne port map(a,usel1,usel2,usel3,usel4);

--Si istanziano i quattro mux
--per la prima terna
Mux1: Mux5i9b port map(ia,ib,ic,id,ie,usel1,uop1);
--per la seconda terna
Mux2: Mux5i9b port map(ia,ib,ic,id,ie,usel2,uop2);
--per la terza terna
Mux3: Mux5i9b port map(ia,ib,ic,id,ie,usel3,uop3);
--per la quarta terna
Mux4: Mux5i9b port map(ia,ib,ic,id,ie,usel4,uop4);

--ora bisogna trattare le uscite dei mux e renderle tutte a 15 bit
--Calcolo dei prodotti parziali
--PRIMO PRODOTTO: uscita di mux1
--il riempimento delle celle vuote fino a rendere upo1 a nove bit un
```

```
--numero a 15 bit verra' effettuato ricopiando il valore del suo bit
--piu'significativo in ciascuna di queste, in modo che:
--se questo sara' uno, vorra' dire che il numero ottenuto era
--negativo in segno e di conseguenza lo sara' anche il prodotto
--parziale che si sta generando se questo sara' zero, allora il
--numero era positivo in segno.
--il numero piu' significativo di uop1 e' quello in posizione 8
pp1(14)<=uop1(8);pp1(13)<=uop1(8);pp1(12)<=uop1(8);pp1(11)<=uop1(8);p
p1(10)<=uop1(8);pp1(9)<=uop1(8);--riempimento delle celle vuote
pp1(8)<=uop1(8);pp1(7)<=uop1(7);pp1(6)<=uop1(6);pp1(5)<=uop1(5);
pp1(4)<=uop1(4);pp1(3)<=uop1(3);pp1(2)<=uop1(2);pp1(1)<=uop1(1);
pp1(0)<=uop1(0);

--SECONDO PRDODOTTO: uscita mux2
SM2: Shiftl2 port map(uop2,suop2);--shifting
--da 11 a 15 bit
--lle celle vuote verranno riempite con suop2(10)
pp2(14)<=suop2(10);pp2(13)<=suop2(10);pp2(12)<=suop2(10);pp2(11)<=suo
p2(10);--riempimento delle celle vuote
pp2(10)<=suop2(10);pp2(9)<=suop2(9);pp2(8)<=suop2(8);pp2(7)<=suop2(7)
;pp2(6)<=suop2(6);pp2(5)<=uop2(5);
pp2(4)<=suop2(4);pp2(3)<=suop2(3);pp2(2)<=suop2(2);pp2(1)<=suop2(1);p
p2(0)<=suop2(0);

--TERZO PRODOTTO: uscita mux3
SM3: Shiftl4 port map(uop3,suop3);--shifting
--da 13 a 15 bit
--le celle vuote verranno riempite con suop3(12)
pp3(14)<=suop3(12);pp3(13)<=suop3(12);--riempimento delle celle vuote
pp3(12)<=suop3(12);pp3(11)<=suop3(11);pp3(10)<=suop3(10);pp3(9)<=suop
3(9);pp3(8)<=suop3(8);pp3(7)<=suop3(7);
pp3(6)<=suop3(6);pp3(5)<=suop3(5);pp3(4)<=suop3(4);pp3(3)<=suop3(3);p
p3(2)<=suop3(2);pp3(1)<=suop3(1);pp3(0)<=suop3(0);

--QUARTO PRODOTTO
SM4: Shiftl6 port map(uop4,pp4);--shifting
--a 15 bit(non c'e' bisogno di spostarlo)
--DEBUGGIN
--p<=pp1; --per debugging
--RISULTATO
--si sommano i prodotti parziali: risultato a 16 bit
SommaProdottiParziali: Adder15 port map(pp1,pp2,pp3,pp4,p);


end MyMB;
```

Graficamente, può essere rappresentato in questo modo:



Sommatore a quindici bit

E' il modulo che si occupa, nel moltiplicatore MulBooth, di sommare i prodotti parziali che sono stati generati.
Quindi riceve in ingresso quattro flussi da quindici bit ciascuno e restituisce, in uscita, il prodotto complessivo calcolato dal moltiplicatore in questione, sempre a quindici bit.

È costituito da tre Ripple Carry a 15 bit, di cui:
-il primo si occupa di effettuare la somma tra i prodotti parziali prodp1 e prodp2;
-il secondo di effettuare la somma tra i prodotti parziali prodp3 e prodp4;
-il terzo, invece, addiziona tra di loro i risultati ottenuti dai due Ripple Carry precendenti, ottenendo così il prodotto vero e proprio.

Inoltre, viene utilizzato un FullAdder che si occupa di calcolare il riporto complessivo, prendendo, come ingressi, sia il “riporto2” generato dal primo ripple carry, sia il “riporto” generato dal secondo.
Non verrà riportata una rappresentazione grafica perché il funzionamento di un blocco sommatore è già stato trattato nei progetti precedenti.

Segue il codice che lo implementa:

```
--Sommatore usato per addizionare i prodotti parziali
--deve essere a 15 bit
--si importa la libreria
library IEEE;
--si include il modulo di intresse della libreria
use  IEEE.std_logic_1164.all;
--definizione del componente
Entity Adder15 is
--riceve i prodotti parziali gia' resi a 15 bit
--e li somma tra di loro, dando in output
--il risultato effettivo del prodotto tra i due operandi
--iniziali ad otto bit
     port(
       prodp1,prodp2,prodp3,prodp4: in std_logic_vector(14 downto 0);
           prodtot: out std_logic_vector(15 downto 0)
     );

end Adder15;
```

```
--definizione dell'architettura
Architecture MyA15 of Adder15 is
--variabili ausiliarie
signal s:std_logic_vector(14 downto 0);--contiene la somma quasi
--completa
signal cout: std_logic;--riporto generato dall'ultimo ripple carry
signal stemp1,stemp2: std_logic_vector(14 downto 0);--somme a 15 bit
signal riporto,riporto2,finalcout: std_logic;--riporto generato dal
--secondo ripple carry(ingresso dell'ultimo)

--inclusione del componente ripple carry
component RippleCarry15 is
     port(
          a,b: in std_logic_Vector(14 downto 0);
          cin: in std_logic;--riporto iniziale in ingresso
          s: out std_logic_Vector(14 downto 0);--15 bit
          cout: out std_logic);
end component;

--si include il FullAdder
component FullAdder is
     port (
          a,b,cin: in std_logic;
          cout,s:out std_logic);
end component;

--non c'e' riporto iniziale nei primi due ripple carry: cin='0'

--inizio delle operazioni da effettuare
begin
--somma tra op1 e op2 operandi a 15 bit
Primo: RippleCarry15 port map(prodp1,prodp2,'0',stemp1,riporto2);
--somma tra op3 e op4 operandi a 15 bit


Secondo: RippleCarry15 port map(prodp3,prodp4,'0',stemp2,riporto);

--somma tra le due somme parziali stemp1 e stemp2
Ultimo: RippleCarry15 port map(stemp1,stemp2,'0',s,cout);

--calcolo del riporto complessivo
RiportoFinale: FullAdder port map(riporto,riporto2,cout,finalcout);

--dove s e' un numero a 15 bit e cout un bit
--si scrive l'uscita del blocco Adder15 che deve essere un numero a
--16 bit
```

```
--cosi' fatto: riporto e s =1+15= 16 bit
prodtot(15)<=finalcout;
prodtot(14)<=s(14);
prodtot(13)<=s(13);
prodtot(12)<=s(12);
prodtot(11)<=s(11);
prodtot(10)<=s(10);
prodtot(9)<=s(9);
prodtot(8)<=s(8);
prodtot(7)<=s(7);
prodtot(6)<=s(6);
prodtot(5)<=s(5);
prodtot(4)<=s(4);
prodtot(3)<=s(3);
prodtot(2)<=s(2);
prodtot(1)<=s(1);
prodtot(0)<=s(0);
--il risultato e' stato assegnato
end MyA15;
```

Per completezza, si riporta anche il codice del RippleCarry15, ovvero a quindici bit:

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

Entity RippleCarry15 is
     port(
           a,b: in std_logic_Vector(14 downto 0);
           cin: in std_logic;
           s: out std_logic_Vector(14 downto 0);--15 bit
           cout: out std_logic);
end RippleCarry15;
--definizione dell'architettura
Architecture MyRC15 of RippleCarry15 is
--segnali ausiliari
signal c: std_logic_vector(14 downto 0);

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

begin
```

```
--Operazioni del RC
FA0: FullAdder port map(a(0),b(0),cin,c(0),s(0));
FA1: FullAdder port map(a(1),b(1),c(0),c(1),s(1));
FA2: FullAdder port map(a(2),b(2),c(1),c(2),s(2));
FA3: FullAdder port map(a(3),b(3),c(2),c(3),s(3));
FA4: FullAdder port map(a(4),b(4),c(3),c(4),s(4));
FA5: FullAdder port map(a(5),b(5),c(4),c(5),s(5));
FA6: FullAdder port map(a(6),b(6),c(5),c(6),s(6));
FA7: FullAdder port map(a(7),b(7),c(6),c(7),s(7));
FA8: FullAdder port map(a(8),b(8),c(7),c(8),s(8));
FA9: FullAdder port map(a(9),b(9),c(8),c(9),s(9));
FA10: FullAdder port map(a(10),b(10),c(9),c(10),s(10));
FA11: FullAdder port map(a(11),b(11),c(10),c(11),s(11));
FA12: FullAdder port map(a(12),b(12),c(11),c(12),s(12));
FA13: FullAdder port map(a(13),b(13),c(12),c(13),s(13));
FA14: FullAdder port map(a(14),b(14),c(13),cout,s(14));
end MyRC15;
```

Sommatore dei prodotti calcolati dai tre moltiplicatori

Questo componente, ricevendo in ingresso i tre prodotti calcolati dalle tre unità di moltiplicatore, li somma tra di loro, restituendo il risultato che si richiedeva al circuito complessivo.
Anche in questo caso, trattandosi di un sommatore, non verranno forniti ulteriori dettagli.

Segue il codice VHDL del componente:

```
--Sommatore totale (SEQUENZIALE)
--si occupa di effettuare la somma tra i tre prodotti
--che si sono calcolati a partire dalle tre coppie
--di operandi ad otto bit dati in ingresso al circuito
--si importa la libreria
library IEEE;
--si include il modulo di intresse della libreria
use  IEEE.std_logic_1164.all;
--definizione del componente
Entity Sommatore3Prod is
--riceve in ingresso tre operandi a 16 bit
--cin, riporto in ingresso, che e' nullo;
--uscite:
--s vettore di 16 bit, e' il risultato della somma tra i tre numeri a
--16 bit;
--cout, riporto in uscita
port(
     a,b,c: in std_logic_Vector(15 downto 0);
          cin: in std_logic;
     clock,clear:in std_logic;
     s: out std_logic_Vector(16 downto 0);
        cout: out std_logic);

end Sommatore3Prod;
--definizione dell'architettura
Architecture MyS3P of Sommatore3Prod is
--segnali ausiliari per i registri
signal atemp,btemp,ctemp: std_logic_vector(15 downto 0);
signal ctemp1,ctemp2,sabtemp,sab: std_logic_vector(16 downto 0);
signal cintemp,coutemp,cotemp1: std_logic;
--COMPONENTI UTILIZZATI
```

```
component RippleCarry16 is
--ingressi:
--a vettore di 16 bit, b vettore di 16 bit;
--cin, riporto in ingresso al ripple carry, di tipo bit;
--uscite:
--s vettore di 16 bit, e' il risultato della somma tra i due numeri a
--16 bit, a e b;
--cout, riporto in uscita dal ripple carry, di tipo bit;
port(
      a,b: in std_logic_Vector(15 downto 0);
           cin: in std_logic;
      clock, clear: in std_logic;
           s: out std_logic_Vector(16 downto 0);--risultato a 17 bit
      cout: out std_logic);
end component;
--per memorizzare gli ingressi e la somma risultante
component Register16 is
--ingressi:
--d vettore da 16 celle,clock,clear;
--uscite:
--q vettore di uscita da 16 celle;
     port(
          d : in std_logic_vector(15 downto 0);
          clock,clear: in std_logic;
          q : out std_logic_vector(15 downto 0)
     );
end component;
--inclusione registro a 17 bit
component Register17 is
     port(
          id : in std_logic_vector(16 downto 0);
          clock,clear: in std_logic;
          uq : out std_logic_vector(16 downto 0)
     );
end component;
--per memorizzare il riporto iniziale ed il riporto in uscita cout
--ripple carry che somma operandi a 17 bit
component RippleCarry17 is
     port(
          a,b: in std_logic_Vector(16 downto 0);
               cin: in std_logic;
          clock, clear: in std_logic;
               s: out std_logic_Vector(16 downto 0);
             cout: out std_logic);
end component;
```

```
--INIZIO
begin
--operazioni dei registri
--SINCRONIZZAZIONE DEGLI INGRESSI a 16 bit

--ARegister e' il registro che immagazzina i valori del vettore a
--ricevuto in ingresso,inserendoli nel suo valore di uscita
ARegister : Register16 port map(a,clock,clear,atemp);

--BRegister e' il registro che immagazzina i valori del vettore a
--ricevuto in ingresso,inserendoli nel suo valor di uscita
BRegister : Register16 port map(b,clock,clear,btemp);

--CRegister e' il registro che immagazzina i valori del vettore a
--ricevuto in ingresso,inserendoli nel suo valor di uscita
CRegister : Register16 port map(c,clock,clear,ctemp);

--riporto iniziale
FR: FlipFlop port map('0',clock,clear,cintemp);

--PRIMO SOMMATORE
SommaAB:RippleCarry16 port
map(atemp,btemp,cintemp,clock,clear,sabtemp);

--la sua somma viene resa gia' internamente a 17 bit, anteponendo al
--risultato della somma tra a e b , il riporto che si e' generato

--SECONDO SOMMATORE
--sabtemp e' gia' stata resa a 17 bit e deve essere passata in un –
registro per essere sincronizzata
SABRegister: Register17 port map(sabtemp,clock,clear,sab);

--secondo operando del secondo sommatore: passaggio da 15 a 16 bit
ctemp1(0)<=ctemp(0);
ctemp1(1)<=ctemp(1);
ctemp1(2)<=ctemp(2);
ctemp1(3)<=ctemp(3);
ctemp1(4)<=ctemp(4);
ctemp1(5)<=ctemp(5);
ctemp1(6)<=ctemp(6);
ctemp1(7)<=ctemp(7);
ctemp1(8)<=ctemp(8);
ctemp1(9)<=ctemp(9);
ctemp1(10)<=ctemp(10);
ctemp1(11)<=ctemp(11);
ctemp1(12)<=ctemp(12);
```

```
ctemp1(13)<=ctemp(13);
ctemp1(14)<=ctemp(14);
ctemp1(15)<=ctemp(15);--sedicesimo bit
ctemp1(16)<=ctemp(15);--cella vuota
--c viene reso, da 16 a 17 bit

--CRegister: passaggio di ctemp nel registro a 16 bit
C2Register : Register17 port map(ctemp1,clock,clear,ctemp2);
--Operando C a 16 bit

--SECONDO SOMMATORE: somma tra (A+B) + C

SommaSC:RippleCarry17 port
map(sab,ctemp2,cintemp,clock,clear,s,cout);

end MyS3P;
```

Si riportano, per completezza, anche i codici dei moduli Ripple Carry, rispettivamente, a sedici ed a diciassette bit:

```
--Ripple Carry a 16 bit
library IEEE;
use  IEEE.std_logic_1164.all;
--definizione del componente
Entity RippleCarry16 is
     port(
           a,b: in std_logic_Vector(15 downto 0);
           cin: in std_logic;
           clock, clear: in std_logic;
           s: out std_logic_Vector(16 downto 0);--risultato a 17 bit
           cout: out std_logic);
end RippleCarry16;
--definizione dell'architettura
Architecture MyRC16 of RippleCarry16 is
--segnali ausiliari
signal c: std_logic_vector(14 downto 0);
signal coutemp:std_logic;
```

```
--si include il FullAdder che eseguira' le somme
component FullAdder is
     port(
          a,b,cin: in std_logic;
          cout,s:out std_logic);
end component;



--si include il Register16
component Register16 is
     port(
          d : in std_logic_vector(15 downto 0);
          clock,clear: in std_logic;
          q : out std_logic_vector(15 downto 0)
          );

end component;

--si include il Register17
component  Register17 is
     port(
          id : in std_logic_vector(16 downto 0);
          clock,clear: in std_logic;
          uq : out std_logic_vector(16 downto 0)
     );
end component;

--inclusione del Flip Flop
component FlipFlop is
     port(
          d,clock,clear: in std_logic;
          q: out std_logic
     );
end component;
--operazioni dei registri
begin
--Operazioni del RC

FA0: FullAdder port map(a(0),b(0),cin,c(0),s(0));

FA1: FullAdder port map(a(1),b(1),c(0),c(1),s(1));

FA2: FullAdder port map(a(2),b(2),c(1),c(2),s(2));

FA3: FullAdder port map(a(3),b(3),c(2),c(3),s(3));
```

```
FA4: FullAdder port map(a(4),b(4),c(3),c(4),s(4));

FA5: FullAdder port map(a(5),b(5),c(4),c(5),s(5));

FA6: FullAdder port map(a(6),b(6),c(5),c(6),s(6));

FA7: FullAdder port map(a(7),b(7),c(6),c(7),s(7));

FA8: FullAdder port map(a(8),b(8),c(7),c(8),s(8));

FA9: FullAdder port map(a(9),b(9),c(8),c(9),s(9));

FA10: FullAdder port map(a(10),b(10),c(9),c(10),s(10));

FA11: FullAdder port map(a(11),b(11),c(10),c(11),s(11));

FA12: FullAdder port map(a(12),b(12),c(11),c(12),s(12));

FA13: FullAdder port map(a(13),b(13),c(12),c(13),s(13));

FA14: FullAdder port map(a(14),b(14),c(13),c(14),s(14));

FA15: FullAdder port map(a(15),b(15),c(14),coutemp,s(15));
--sedicesimo bit
--si trasforma in 17 bit anteponendo al valore di s il riporto cout
s(16)<=coutemp;
--riporto
cout<=coutemp;

end MyRC16;


--Ripple Carry a 17 bit
library IEEE;
use  IEEE.std_logic_1164.all;

Entity RippleCarry17 is
     port(
          a,b: in std_logic_Vector(16 downto 0);
          cin: in std_logic;
          clock, clear: in std_logic;
          s: out std_logic_Vector(16 downto 0);
          cout: out std_logic);
end RippleCarry17;
```

```
Architecture MyRC17 of RippleCarry17 is

signal atemp,btemp:std_logic_vector(16 downto 0);
signal stemp:std_logic_vector(16 downto 0);
signal cintemp:std_logic;
--segnali ausiliari
signal c: std_logic_vector(15 downto 0);
--si include il FullAdder che eseguira' le somme
component FullAdder is
     port(
          a,b,cin: in std_logic;
          cout,s:out std_logic);
end component;
--si include il Register17
component  Register17 is
     port(
          id : in std_logic_vector(16 downto 0);
          clock,clear: in std_logic;
          uq : out std_logic_vector(16 downto 0)
     );
end component;
--inclusione del Flip Flop
component FlipFlop is
     port(
          d,clock,clear: in std_logic;
          q: out std_logic
     );
end component;
--operazioni dei registri
begin
--SINCRONIZZAZIONE DEGLI INGRESSI

--ARegister e' il registro che immagazzina i valori del vettore a
--ricevuto in ingresso,
--inserendoli nel suo valore di uscita
ARegister : Register17 port map(a,clock,clear,atemp);

--SBRegister e' il registro che immagazzina i valori del vettore a
--ricevuto in ingresso,inserendoli nel suo valor di uscita
BRegister : Register17 port map(b,clock,clear,btemp);

--CinFlipFlop e' il componente che immagazzina il valore del riporto
--iniziale cin, rendendolo disponibile come stato della sua uscita
CinFlipFlop : FlipFlop port map(cin,clock,clear,cintemp);
--Operazioni del RC
```

```
FA0: FullAdder port map(atemp(0),btemp(0),cintemp,c(0),s(0));

FA1: FullAdder port map(atemp(1),btemp(1),c(0),c(1),s(1));

FA2: FullAdder port map(atemp(2),btemp(2),c(1),c(2),s(2));

FA3: FullAdder port map(atemp(3),btemp(3),c(2),c(3),s(3));

FA4: FullAdder port map(atemp(4),btemp(4),c(3),c(4),s(4));

FA5: FullAdder port map(atemp(5),btemp(5),c(4),c(5),s(5));

FA6: FullAdder port map(atemp(6),btemp(6),c(5),c(6),s(6));

FA7: FullAdder port map(atemp(7),btemp(7),c(6),c(7),s(7));

FA8: FullAdder port map(atemp(8),btemp(8),c(7),c(8),s(8));

FA9: FullAdder port map(atemp(9),btemp(9),c(8),c(9),s(9));

FA10: FullAdder port map(atemp(10),btemp(10),c(9),c(10),s(10));

FA11: FullAdder port map(atemp(11),btemp(11),c(10),c(11),s(11));

FA12: FullAdder port map(atemp(12),btemp(12),c(11),c(12),s(12));

FA13: FullAdder port map(atemp(13),btemp(13),c(12),c(13),s(13));

FA14: FullAdder port map(atemp(14),btemp(14),c(13),c(14),s(14));

FA15: FullAdder port map(atemp(15),btemp(15),c(14),c(15),s(15));
--sedicesimo bit
FA16: FullAdder port map(atemp(16),btemp(16),c(15),cout,s(16));--
diciassettesimo bit

end MyRC17;
```

Circuito complessivo

Questo modulo racchiude in sé tutti i componenti trattati finora. Riceve in ingresso le sei coppie di operandi ad otto bit che dovranno essere moltiplicate tra loro e di cui si dovranno, infine, sommare i risultati.
È un circuito PIPELINE, in cui sia gli ingressi che le uscite verranno sincronizzati mediante gli appositi registri e flip flop.

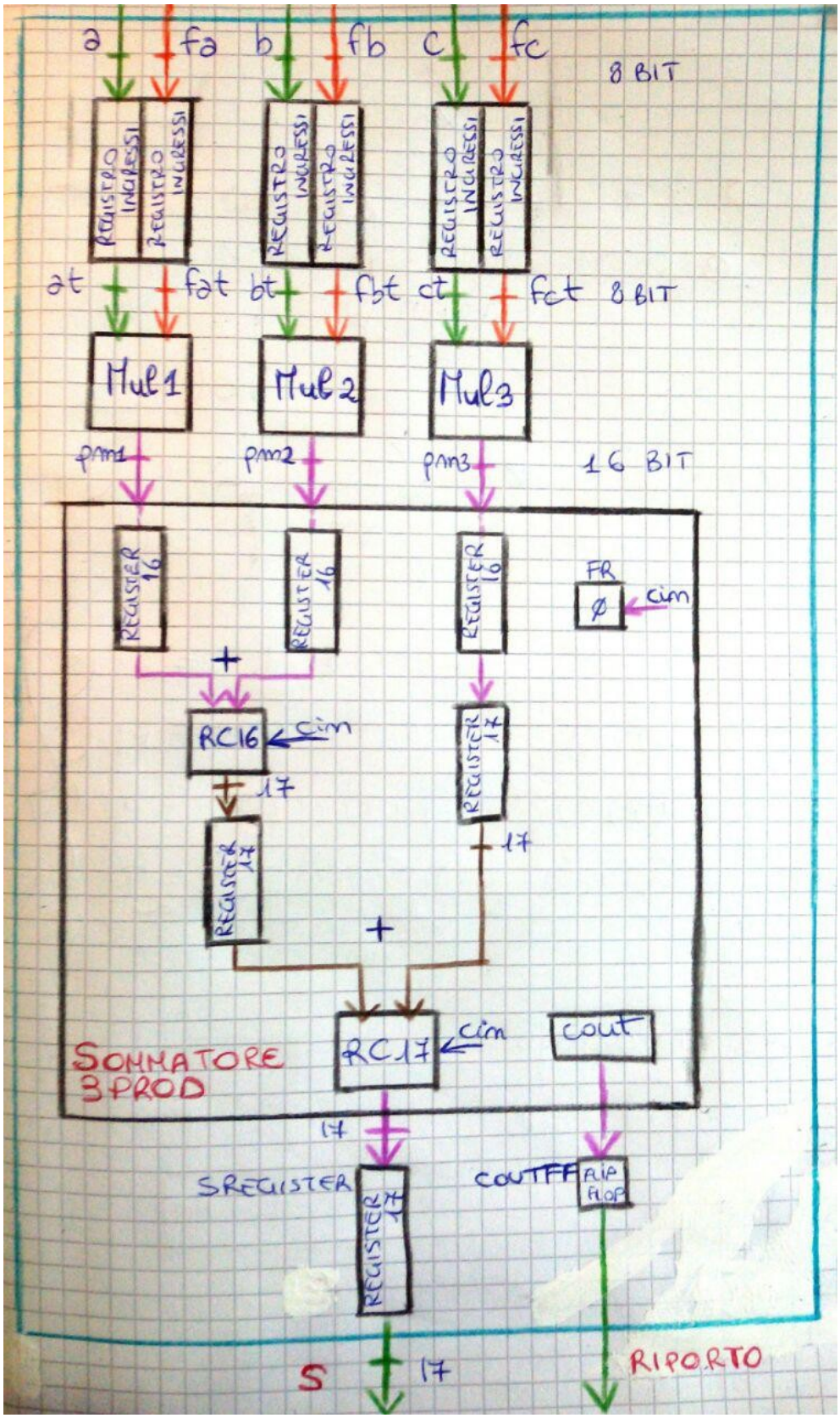
Si riporta il codice VHDL che lo descrive:

```
--circuito complessivo pipeline
--costituito da tre moltiplicatore in parallelo
--i cui risultati vengono mandati ad un sommatore a tre ingressi
--si importa la libreria
library IEEE;
--si include il modulo di intresse della libreria
use  IEEE.std_logic_1164.all;
--definizione del componente
Entity CircuitoMulSum is
     port(
          a,fa,b,fb,c,fc: in std_logic_vector(7 downto 0);
          clock,clear:in std_logic;
          risultato: out std_logic_vector(16 downto 0);
          riporto: out std_logic
     );
end CircuitoMulSum;

--definizione dell'architettura
Architecture MyCMS of CircuitoMulSum is
--variabile ausiliarie per la sincronizzazione
--da ingressi circuito complessivo ad ingressi moltiplicatori
signal at,fat,bt,fbt,ct,fct:std_logic_vector(7 downto 0);
--uscite moltiplicatori 16 bit
signal pm1,pm2,pm3:std_logic_vector(15 downto 0);
--uscita sommatore tre prodotti
signal ris:std_logic_vector(16 downto 0);
--riporto generato dalla somma dei tre prodotti
signal carry:std_logic;
--inclusione dei componenti
--flip flop
```

```
component FlipFlop is
     port(
          d,clock,clear: in std_logic;
          q: out std_logic
     );
end component;
--registro speciale che si occupa di cambiare il bit
--piu' significativo degli ingressi in '0', oltre che di
--sincronizzarli e passarli, a coppie, ai tre moltiplicatori di Booth
component RegistroIngressi is
--REGISTRO DI CAMBIO BIT SIGNIFICATIVO
--riceve in ingresso un vettore di otto bit
--ne cambia il bit piu' significativo,
--quello in posizione 7 e salva tutto in
--un vettore ad otto bit
     port(
          i: in std_logic_vector(7 downto 0);
          clock,clear:in std_logic;
          u: out std_logic_vector(7 downto 0)
     );
end component;


--moltiplicatore di booth: si occupa di calcolare i prodotti tra le
--coppie
--se ne dovranno istanziare tre

component MulBooth is
--porte coinvolte
     port(
           a,b: in std_logic_vector(7 downto 0);--otto bit
           p: out std_logic_vector(15 downto 0)--sedici bit
          );
end component;
--modulo che si occupa di sommare i tre prodotti calcolati
component Sommatore3Prod is
--riceve in ingresso tre operandi a 16 bit
--cin, riporto in ingresso, che e' nullo;
--uscite:
--s vettore di 16 bit, e' il risultato della somma tra i tre numeri a
--16 bit;
--cout, riporto in uscita
     port(
          a,b,c: in std_logic_Vector(15 downto 0);
          cin: in std_logic;
          clock,clear:in std_logic;
```

```
          s: out std_logic_Vector(16 downto 0);
          cout: out std_logic);
end component;
--registro a 17 bit
component Register17 is
     port(
          id : in std_logic_vector(16 downto 0);
          clock,clear: in std_logic;
          uq : out std_logic_vector(16 downto 0)
     );
end component;
--inizio delle operazioni
begin
--SINCRONIZZAZIONE DEGLI INGRESSI
ra: RegistroIngressi port map(a,clock,clear,at);
rfa: RegistroIngressi port map(fa,clock,clear,fat);
rb: RegistroIngressi port map(b,clock,clear,bt);
rfb: RegistroIngressi port map(fb,clock,clear,fbt);
rc: RegistroIngressi port map(c,clock,clear,ct);
rfc: RegistroIngressi port map(fc,clock,clear,fct);
--Calcolo dei prodotti

--Primo:
Mul1: MulBooth port map(at,fat,pm1);
--Secondo:
Mul2: MulBooth port map(bt,fbt,pm2);
--Terzo:
Mul3: MulBooth port map(ct,fct,pm3);

--Somma dei prodotti
SommaProdotti: Sommatore3Prod port
map(pm1,pm2,pm3,'0',clock,clear,ris,carry);
--queso modulo sincronizza gli ingressi che riceve internamente a se
--stesso,di sincronizzare le uscite invece se ne occupa il modulo
--corrente, CircuitoMulSum

--SINCRONIZZAZIONE DELLE USCITE

--riporto finale
COUTFF : FlipFlop  port map(carry,clock,clear,riporto);
--somma finale
SRegister : Register17 port map(ris,clock,clear,risultato);

end MyCMS;
```

Graficamente, si può rappresentare come segue:



Simulazione

Segue il codice VHDL scritto per avviare la simulazione del circuito, verificandone così il corretto funzionamento.

```
--Test del circuito
--si importa la libreria
library IEEE;
--si include il modulo di intresse della libreria
use  IEEE.std_logic_1164.all;
--entita' vuota
Entity TestCircuito is
end TestCircuito;
--definizione dell'architettura
Architecture TC of TestCircuito is
--si inizializzano i valori che si vuole assumano i segnali
signal iclock : std_logic :='1';--segnale di clock
signal iclear : std_logic :='1';--segnale di clear
--segnali usati
signal ia,ifa,ib,ifb,ic,ifc :std_logic_vector(7 downto 0);
signal us :std_logic_vector(16 downto 0);
signal ucout : std_logic;

--si definisce il componente circuitale da testare
component CircuitoMulSum is
     port(
          a,fa,b,fb,c,fc: in std_logic_vector(7 downto 0);
          clock,clear:in std_logic;
          risultato: out std_logic_vector(16 downto 0);
          riporto: out std_logic
     );
end component;

--attivita'
begin
--si scrive il process relativo al segnale di clock
clock:
process
     begin
     loop
          wait for 5 ns;
          iclock<= not iclock;
     end loop;
end process;
--si scrive il process relativo al segnale di clear
```

```
clear:
process
     begin
          iclear<= '0';
          wait for 200 ns;
end process;
--si scrive il process relativo agli altri ingressi
inputs:
--si useranno tre operandi ad otto bit ed il riporto iniziale cin
--i valori attesi saranno cout, che e' il riporto finale
--e la somma s a nove bit
process
     begin
--1)
      ia<="00000000";
     ifa<="00000010";
      ib<="00000001";
     ifb<="10000011";
      ic<="10100000";
     ifc<="11100000";
     wait for 10 ns;
--2)
      ia<="11111110";
     ifa<="10000000";
      ib<="00000001";
     ifb<="11111111";
     ic<="10100000";
     ifc<="01010101";
     wait for 10 ns;
--3)
      ia<="10101010";
     ifa<="01010101";
      ib<="10000000";
     ifb<="00000000";
      ic<="00000001";
     ifc<="00001111";
     wait for 10 ns;
--4)
      ia<="01111111";
     ifa<="11000000";
      ib<="00000011";
     ifb<="00110011";
      ic<="11000000";
     ifc<="11001100";
     wait for 10 ns;
```

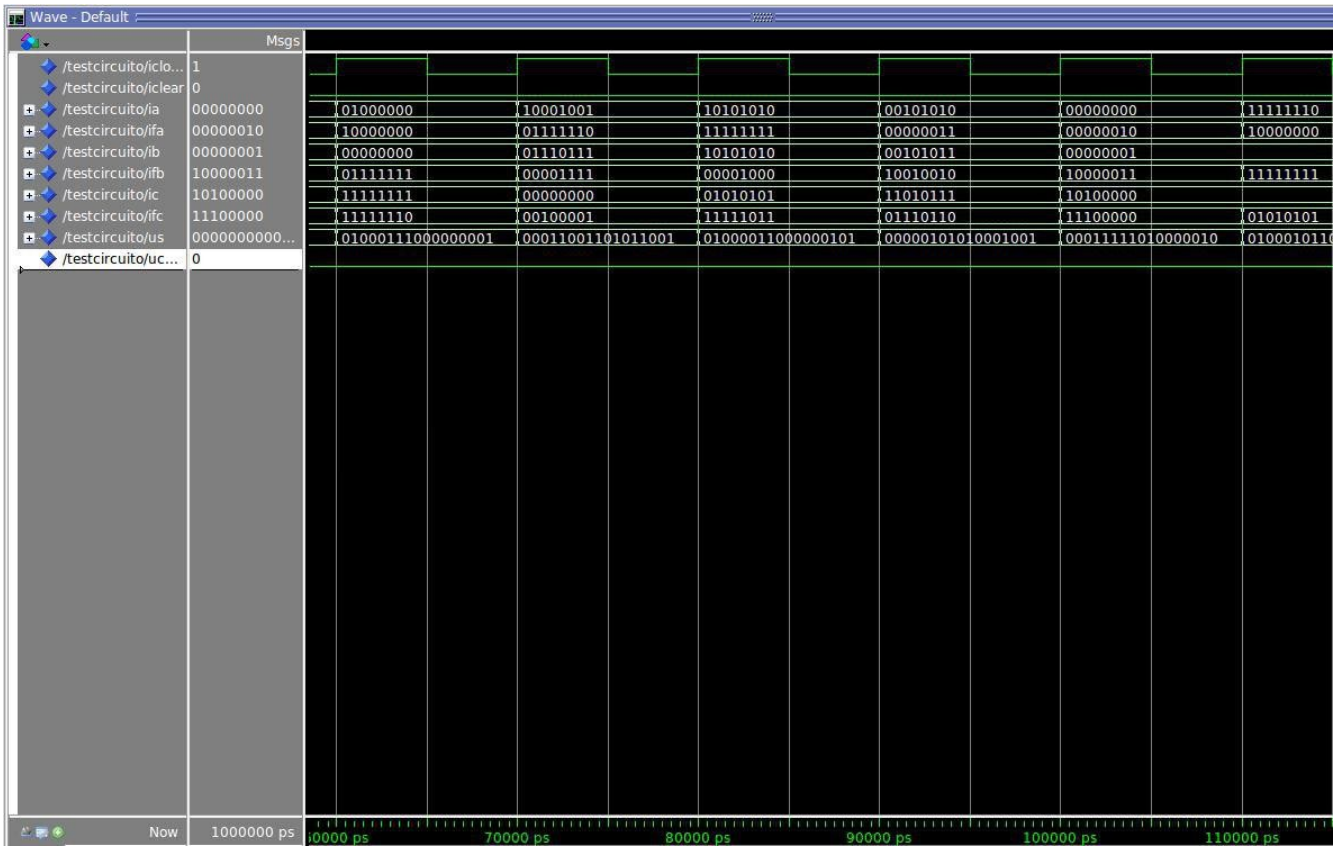
```
--5)
      ia<="10001001";
     ifa<="00000100";
      ib<="01110111";
     ifb<="10000001";
      ic<="00100001";
     ifc<="10101010";
     wait for 10 ns;

--6)
      ia<="00100000";
     ifa<="00010010";
      ib<="00001001";
     ifb<="10000001";
      ic<="10100001";
     ifc<="11000000";
     wait for 10 ns;

--7)
      ia<="01000000";
     ifa<="10000000";
      ib<="00000000";
     ifb<="01111111";
      ic<="11111111";
     ifc<="11111110";
     wait for 10 ns;

--8)
      ia<="10001001";
     ifa<="01111110";
      ib<="01110111";
     ifb<="00001111";
      ic<="00000000";
     ifc<="00100001";
     wait for 10 ns;

--9)
      ia<="10101010";
     ifa<="11111111";
      ib<="10101010";
     ifb<="00001000";
      ic<="01010101";
     ifc<="11111011";
     wait for 10 ns;
```

```
--10)
      ia<="00101010";
     ifa<="00000011";
      ib<="00101011";
     ifb<="10010010";
      ic<="11010111";
     ifc<="01110110";
     wait for 10 ns;

end process;

circuito: CircuitoMulSum
          port map(ia,ifa,ib,ifb,ic,ifc,iclock,iclear,us,ucout);

end TC;
```

Ai fini di mostrare il corretto funzionamento del circuito che si doveva realizzare si inviano delle immagini estratte dalla simulazione:

Simulazione da 0 a 60000 ps.

Simulazione da 60000 ps a 110000 ps.



Si può notare come la struttura pipeline presenti un ciclo di latenza pari a quattro colpi di clock dovuto ai quattro livelli di registri intermendi che sono stati inseriti.

Inoltre, i risultati ottenuti in base agli ingressi forniti al circuito sono esatti ed il componente in esame, dopo i cicli di latenza riscontrati, fornisce il risultato atteso, in base agli ingressi ricevuti, ad ogni colpo di clock.