

UNIVERSITÀ DELLA CALABRIA



DIMES - Dipartimento di INGEGNERIA INFORMATICA
MODELSTICA, ELETTRONICA E SISTEMISTICA

Anno Accademico 2015-2016

Corso di Laurea Triennale in Ingegneria Informatica

Terzo progetto di Elettronica Digitale

“Circuito PipeLine con due sommatori”

Professoressa S.Perri

Ivonne Rizzuto
matricola 167058

Descrizione

Il circuito che si deve realizzare per la stesura di questo terzo progetto è un sommatore che, ricevendo tre flussi in ingresso ad otto bit, restituisca, in uscita, la somma di questi ed il conseguente riporto che si genera.

In particolare, questa unità di calcolo è costituita da due sommatori collegati, tra loro, in cascata.

Il primo sommatore, un **carry select** ad otto bit, riceverà in ingresso due flussi di segnali che consistono negli operandi ad otto bit, denominati A e B, ed il riporto iniziale cin.

Al termine delle sue operazioni, ovvero la somma tra A, B, e Cin, restituirà, in uscita, la somma S, anch'essa ad otto bit ed il riporto finale Cout.

I flussi in ingresso A e B vengono prelevati da due registri ad otto bit, mentre il valore di Cin viene immagazzinato in un Flip Flop. Il risultato delle operazioni svolte verrà immagazzinato in un vettore ad otto bit, che conterrà il valore della somma S, che poi verrà passato, in ingresso, come uno dei due operandi del secondo sommatore.

Per quanto riguarda il secondo sommatore, un **ripple carry** a nove bit, questo riceverà, come flussi di ingresso, da sistemare in due registri a nove bit, l'operando D, un vettore a nove celle, costituito dal valore della somma S calcolato dal primo sommatore, per quanto riguarda quelle da 0 a 7, e dal valore del riporto cout in uscita, sempre del primo sommatore, in posizione 8, ovvero la cella meno significativa (se si sta utilizzando l'ordinamento "8 down to 0") e ed il flusso in ingresso C, il terzo operando ad otto bit, ricevuto come ingresso dell'unità di calcolo complessiva, che dovrà essere anch'esso trasferito in un registro a nove bit.

Il circuito complessivo, allora, avrà come risultati:

- il valore finale della somma effettuata tra A, B, C ed il riporto Cin in entrata al primo sommatore (il riporto in ingresso al secondo sommatore, infatti, verrà considerato pari a '0');
- il riporto Cout, che si genera quando la somma precedentemente calcolata tra A e B, chiamata D, verrà addizionata al valore di C, generando sia il valore di somma finale S Finale, da salvare nel registro a nove bit, sia il riporto Cout Finale, da salvare in un flip flop.

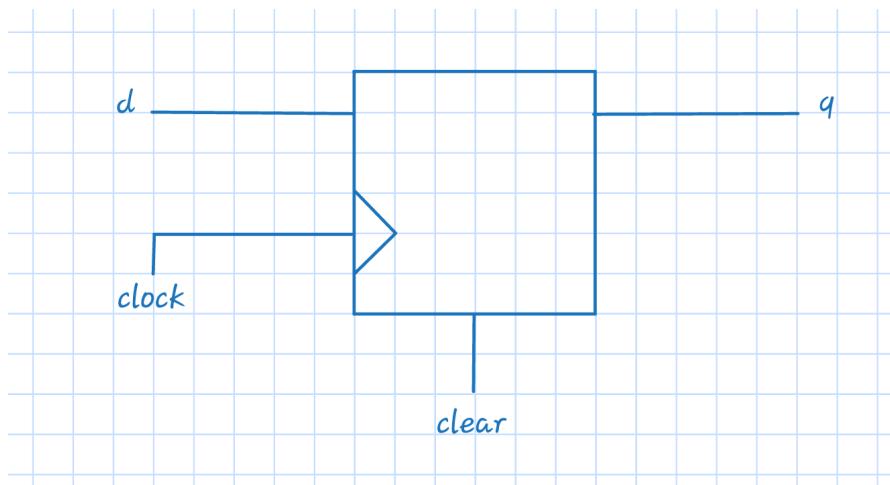
Si analizzano, qui di seguito, tutto i componenti che costituiscono l'unità di calcolo sopra descritta.

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 se 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
--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
--espllicita 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 azzerà
      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;

```

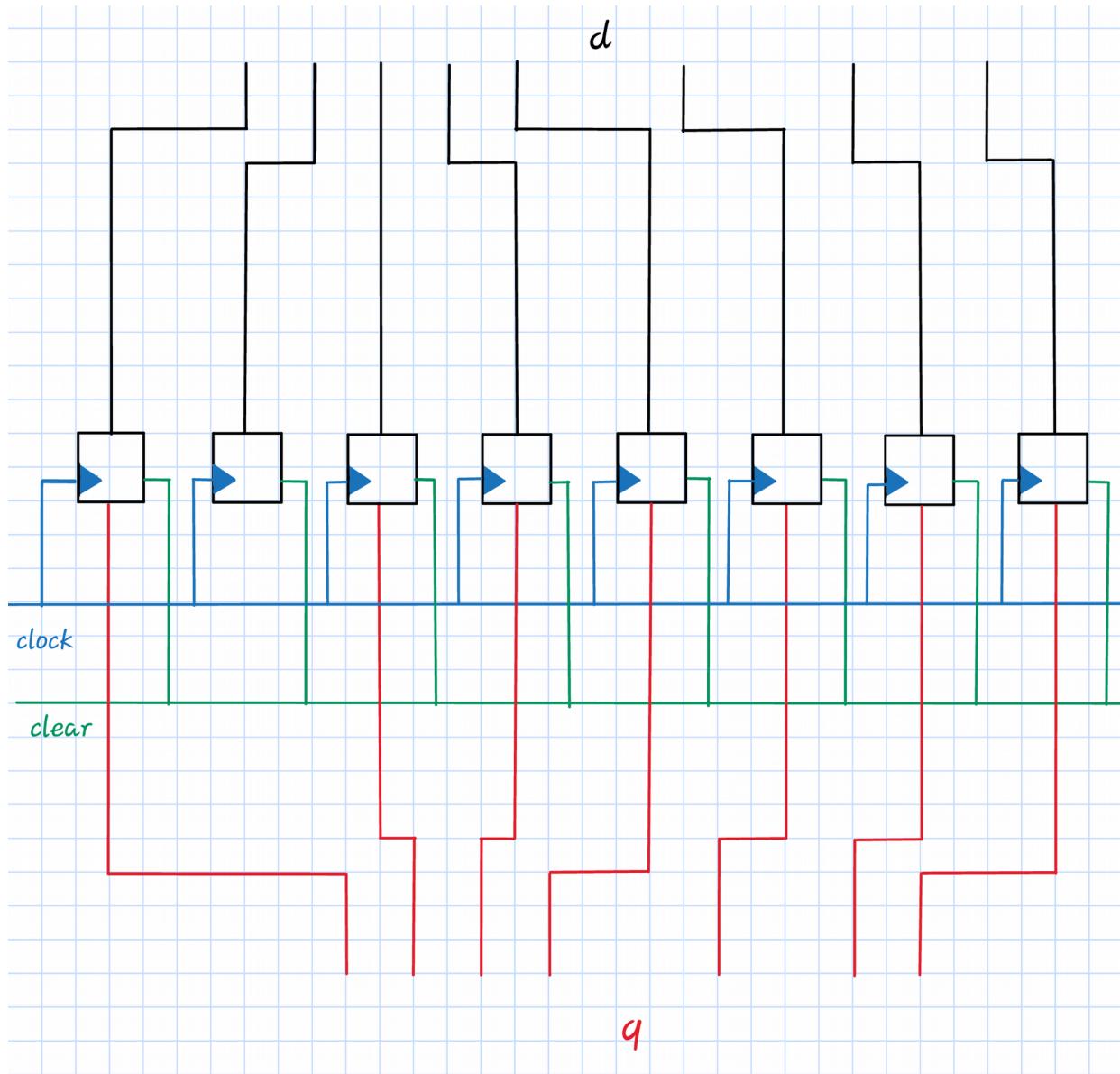
Il Flip Flop è stato usato per gestire la memorizzazione dei segnali quali cin, cout e anche i riporti prestabiliti passati in ingresso a due dei tre ripple carry a 4 bit che compongono il carry select, ovveri inc0 e inc1.

Altri componenti usati per la memorizzazione di valori dei segnali utilizzati sono i registri a quattro e ad otto bit.

Registro ad 8 bit

Questo componente è stato utilizzato per la memorizzazione degli ingressi che riceve il primo sommatore, il Carry Select ad otto bit. Viene, infatti, utilizzato per sincronizzare gli ingressi A e B e l'uscita S.

Si può rappresentare, graficamente, in questo modo:



Il codice vhdl che descrive il registro ad 8 bit è il seguente:

```
--Registro ad 8 Flip Flop
```

```

--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 lavorerà
il sommatore

Entity Register8 is
--ingressi:
--d vettore da 8 celle,clock,clear;
--uscite:
--q vettore di uscita da 8 celle;
  port(
    d : in std_logic_vector(7 downto 0);
    clock,clear: in std_logic;
    q : out std_logic_vector(7 downto 0)
  );
end Register8;

Architecture MyRegistro of Register8 is

  begin
    process(clock,clear)
      begin
        if clear = '1' then
          q <= "00000000";
        elsif clock'event and clock = '1' then
          q <= d;
        end if;
      end process;
end MyRegistro;

```

La struttura di questo componente sarà quella che verrà utilizzata nella composizione dell'altro registro usato nel progetto.
 Infatti, l'unica differenza consiste nell'aggiunta di un flip flop, quindi, in totale, conterrà nove di queste unità elementari

Registro a 9 bit

E' il componente di memorizzazione utilizzato nel secondo sommatore, il Ripple Carry a nove bit, ma anche nell'unità di calcolo complessiva, chiamata Sommatore.

Esso, all'interno di questo modulo, viene utilizzato per sincronizzare gli ingressi D e C, ovvero quelli del componente Ripple Carry a nove bit.

In particolare, l'operando D si genera dall'unione del vettore sabtemp, prodotto in uscita dal primo sommatore, con il riporto couttemp sempre di quest'ultimo, inserito nella posizione meno significativa.

Allora, il vettore a nove celle che si forma, denominato sabtemp1, ai fini della sincronizzazione, viene fatto passare in un registro a nove bit. Alla stessa operazione verrà sottoposto l'ingresso all'intero Sommatore denominato C , un vettore ad otto bit che, dopo essere stato trasferito in un vettore a nove celle denominato ctemp, verrà dato in ingresso ad un altro registro a nove bit, C2Register, la cui uscita verrà chiamata ctemp2.

Si riporta il codice del componente:

```
--Registro a 9 Flip Flop
--e' realizzato in forma compatta
--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 nove cifre con cui lavorera' il sommatore
--in particolare salvera' il risultato che si genera' dal carry
--select ad otto bit.
--Si ha la necessita' di avere un registro a nove bit perche', dalla
--somma tra due operandi
--ad otto cifre, potrebbe generarsi, come risultato, un numero a
--nove cifre.
```

```
Entity Register9 is
  --ingressi:
  --d vettore da 9 celle,clock,clear;
  --uscite:
  --q vettore di uscita da 9 celle;
    port(d : in std_logic_vector(8 downto 0);
          clock,clear: in std_logic;
          q : out std_logic_vector(8 downto 0)
        );
end Register9;
```

```
Architecture MyRegistro of Register9 is
```

```
begin
```

```

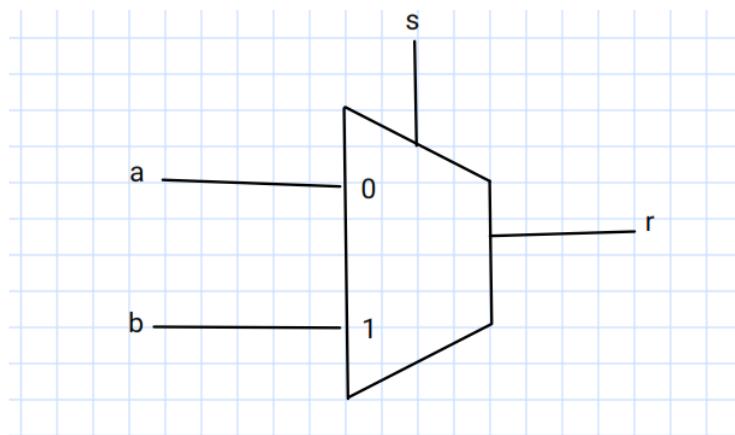
process(clock,clear)
begin
  if clear = '1' then
    q <= "000000000";
  elsif clock'event and clock = '1' then
    q <= d;
  end if;
end process;
end MyRegistro;

```

Per completezza, si riportano i codici dei componenti più elementari utilizzati nella realizzazione del circuito complessivo, quali:

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 svolge un ruolo fondamentale nel calcolo della parte meno significativa della somma ("7 downto 4") che si svolge nel Carry Select ad otto bit, perché permette di stabilire, per ogni bit di somma che deve essere calcolato, se prendere quello generato dal Ripple Carry RP0, che riceve come un riporto iniziale inc0 pari a '0', oppure quello generato dal Ripple CarryRP1, che riceve come riporto iniziale inc1 un segnale pari a '1'.



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;

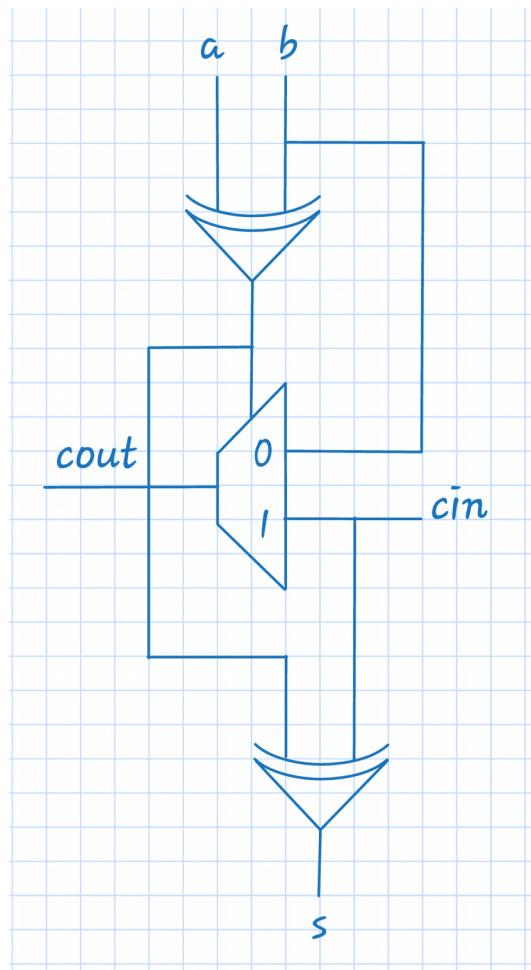
architecture MyMux of Mux is
--parte dichiarativa di segnali ausiliari
--parte descrittiva
begin
    --si definisce la funzione s
    r <= ( a and (not s) ) or (b and s);
end MyMux;
```

Full Adder

E' il componente basilare sia per il funzionamento del Ripple carry a quattro bit, sia per lo stesso componente ad otto bit.

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
--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;

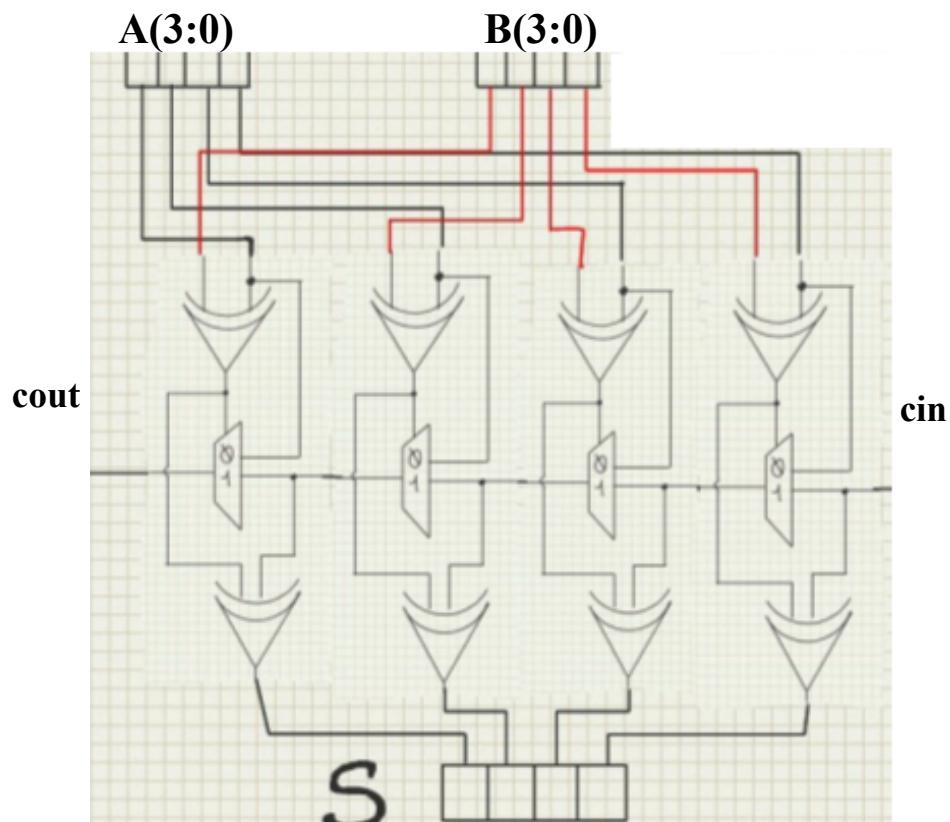
--definizione dell'architettura
Architecture FA of FullAdder is
--parte dichiarativa
    --segnali ausiliari per il FA:propagate p
    signal p:std_logic;
    --si include il componente mux
    component Mux is
        --si specificano le porte che utlizza nell'ordine in cui
        --sono state definite
        port(a,b,s: in std_logic;
              r: out std_logic);
    end component;
--parte descrittiva
--e' introdotta dal begin
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);
    --propagate
    p <= a xor b;
    --somma s
    s <= p xor cin;
    --riporto in uscita cout
end FA;
```

Ripple Carry a 4 bit

Tre di questi componenti sono utilizzati nel carry select ad otto bit.

Il Ripple Carry riceve, in ingresso, due flussi di quattro bit, quali A e B, oltre che il riporto iniziale Cin, e restituisce, in uscita, la loro somma S.

E' stato costituito richiamando, al suo interno, quattro istanze del componente Full Adder, secondo lo schema sottostante:



Segue il codice di implementazione:

```
--Ripple carry a 4 bit
--si importa la libreria
library IEEE;
--si include il modulo di interesse della libreria
use IEEE.std_logic_1164.all;
```

```

--definizione del componente

Entity RippleCarry4b is

--ingressi:
--a vettore di 4 bit, b vettore di 4 bit;
--cin, riporto in ingresso al ripple carry, di tipo bit (e' il
--riporto iniziale cin = g0 = a0 and b0);

--uscite:
--s vettore di 4 bit, e' il risultato della somma tra i due numeri a
--4 bit a e b;
--cout, riporto in uscita dal ripple carry, di tipo bit (e' il
--riporto finale c3,
--infatti in tutto i riporti sono cin -esterno-, c0, c1, c2, c3 da
--calcolare );

    port( a,b: in std_logic_vector(3 downto 0);
                cin: in std_logic;
                s: out std_logic_vector(3 downto 0);
                cout: out std_logic);

end RippleCarry4b;

--definizione dell'architettura

Architecture MyRC of RippleCarry4b is

signal c: std_logic_vector(2 downto 0);

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

--Operazioni del RC

begin

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),cout,s(3));

end MyRC;

```

Si procede, ora, nella descrizione dei componenti più significativi del progetto, ovvero quelli che costituiscono la totalità dell'unità di calcolo che si deve realizzare.

Carry Select ad 8 bit

Così come alcuni dei componenti utilizzati finora erano già stati utilizzati nella realizzazione del primo progetto, anche il Carry Select ad otto bit è uno di questi.

Questo componente utilizza, per i calcoli da eseguire, tre Ripple Carry a 4 bit e 5 mux, ma quello che è importante evidenziare è che, rispetto al componente già visto in precedenza, è stato necessario apportare delle modifiche, ovvero introdurre dei registri.

Il codice vhdl che lo descrive è quello sottostante:

```
--si importa la libreria
library IEEE;
--si include il modulo di interesse della libreria
use IEEE.std_logic_1164.all;

Entity CarrySelect8b is
--ingressi A(0:7), B(0:7), cin, clock, clear
--uscite: cout, S(0:8)
  port(
    a,b : in std_logic_Vector(7 downto 0 );
    cin, clock, clear : in std_logic;
    s : out std_logic_Vector(7 downto 0 );
    cout: out std_logic
  );
end CarrySelect8b;
--stesura dell'architettura

Architecture CS8b of CarrySelect8b is
--inclusione del RippleCarry a 4 bit
component RippleCarry4b is

  port( a,b: in std_logic_Vector(3 downto 0 );
        cin: in std_logic;
        s: out std_logic_Vector(3 downto 0 );
        cout: out std_logic);

end component;

--inclusione del Mux
component Mux is
--s selettore, r risultato
  port(a,b,s: in std_logic;
       r: out std_logic
     );
end component;
```

```

--inclusione del registro ad 8
component Register8 is
    port(d : in std_logic_vector(7 downto 0);
          clock,clear: in std_logic;
          q : out std_logic_vector(7 downto 0)
        );
end component;

--inclusione del Flip Flop
component FlipFlop is
    port(d,clock,clear: in std_logic;
          q: out std_logic
        );
end component;

--inclusione del registro ad 9
component Register9 is
    port(d : in std_logic_vector(8 downto 0);
          clock,clear: in std_logic;
          q : out std_logic_vector(8 downto 0)
        );
end component;

--INGRESSI
--segnali utilizzati per il trasferimento nei registri
signal atemp,btemp : std_logic_vector(7 downto 0);
signal cintemp: std_logic;

--il ripoto in ingresso considerato pari a "0" e il riporto in
--ingresso considerato pari a "1" devono essere salvati in dei flip
--flop intermedi per poi essere mandati, rispettivamente, come
--ingressi del CLA0 e del CLA1.
signal inc0, inc1 : std_logic;

--SEGNALI AUSILIARI
--s0 per conservare la somma calcolata con il riporto (cioe' 1)
--e s1 quella senza riporto (cioe' 0)
--si e' la somma calcolata per i bit piu' significativi dal RCI
signal s0,s1,si : std_logic_vector(3 downto 0);--SOMME PARZIALI

--ci primo riporto del RCI che calcola la somma per i bit della
parte
--meno significativa;
--c0 e' il riporto generato dal RC0 che calcola la somma
considerando -il riporto in ingresso pari a "0";
--c1 e' il riporto generato dal RC1 che calcola la somma
considerando -il riporto in ingresso pari a "1";

```

```

signal c0,c1,ci : std_logic;--RIPORTI IN USCITA
--parte descrittiva
begin
  --operazioni dei registri
  --SINCRONIZZAZIONE DEGLI INGRESSI
  --ARegister e' il registro che immagazzina i valori del vettore a
  --ricevuto in ingresso, inserendoli nel suo valore di uscita
  ARegister : Register8 port map(a,clock,clear,atemp);
  --BRegister e' il registro che immagazzina i valori del vettore B
  --ricevuto in ingresso,inserendoli nel suo valore di uscita
  BRegister : Register8 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);
  --C0FF e' il componente che immagazzina il valore del riporto c0,
  --che sarebbe il riporto
  --dato in ingresso al RC0, considerato pari a "0", rendendolo
  --disponibile come stato della
  --sua uscita denominata inc0
  C0FF : FlipFlop port map('0',clock,clear,inc0);
  --C1FF e' il componente che immagazzina il valore del riporto c1,
  --che sarebbe il riporto
  --dato in ingresso al RC1, considerato pari a "1", rendendolo
  --disponibile come stato della
  --sua uscita denominata inc1
  C1FF : FlipFlop port map('1',clock,clear,inc1);
  --CALCOLI
  --operazione dei Ripple Carry modificate tenendo conto
  --dell'introduzione dei registri
  --RCI "iniziale" somma di std_logic in posizione da 0 a 3, parte
  --meno --significativa:riceve, come ingressi, il riporto iniziale
  --cintemp, --il vettore atemp da cui prende i primi 4 bit e il vettore
  --btemp, con cui fa la stessa cosa; il risultato verrà
  --scritto nelle prime quattro celle del vettore risultante
  RCI : RippleCarry4b port map(atemp(3 downto 0),btemp(3 downto
  0),cintemp,s(3 downto 0),ci);
  --RC0 somma da std_logic in posizione da 4 a 7
  --questo si suppone lavori con riporto in ingresso '0'
  --c0 e' il suo riporto in uscita,salva la somma in s0
  RC0 : RippleCarry4b port map(atemp(7 downto 4),btemp(7 downto
  4),inc0,s0,c0);

```

```
--RC1 somma da std_logic in posizione da 4 a 7
--questo si suppone lavori con riporto in ingresso '1'
--c1 e' il suo riporto in uscita,salva la somma in s1

RC1 : RippleCarry4b port map(atemp(7 downto 4),btemp(7 downto
4),inc1,s1,c1);

--operazioni dei Mux

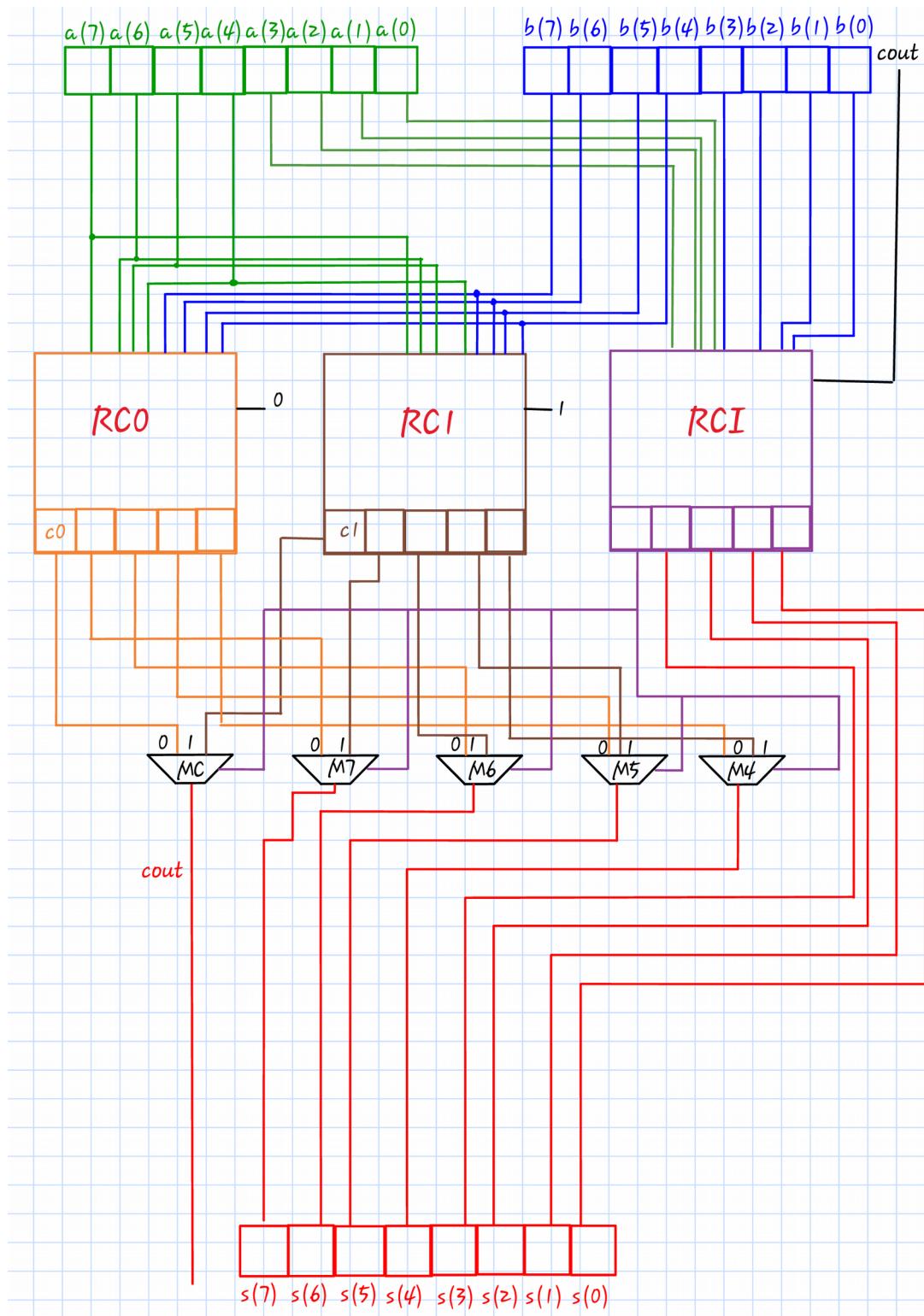
--il selettore di ogni mux sara' il riporto ci ottenuto dal primo
ripple carry rci
--la scelta tra s1 e s0 verra' messa in stemp

M4 : Mux port map(s0(0),s1(0),ci,s(4));
M5 : Mux port map(s0(1),s1(1),ci,s(5));
M6 : Mux port map(s0(2),s1(2),ci,s(6));
M7 : Mux port map(s0(3),s1(3),ci,s(7));

--segue mux che calcola il riporto finale
MC :Mux port map(c0,c1,ci,cout);

end CS8b;
```

Graficamente, può essere schematizzato in questo modo:



Nel disegno non sono stati riportati i Flip Flop che si era previsto di inserire per motivi di maggiore leggibilità.

Ripple Carry ad 9 bit

In cascata al primo sommatore, il Carry Select ad 8 bit, si trova questo componente, il Ripple Carry a 9 bit.

La sua struttura è analoga a quella del Ripple Carry a 4 bit, soltanto che utilizza nove Full Adder invece di quattro ed un Registro ad otto bit, anziché i vettori di 4 bit.

E' per questo motivo che si è deciso di non riportare il grafico, ma soltanto il codice vhdl che lo implementa:

```
--si importa la libreria
library IEEE;
--si include il modulo di interesse della libreria
use IEEE.std_logic_1164.all;

--definizione del componente

Entity RippleCarry9bit is

--ingressi:
--a vettore di 9 bit, b vettore di 9 bit;
--cin, riporto in ingresso al ripple carry, di tipo bit (e' il
--riporto iniziale cin = g0 = a0 and b0);
--uscite:
--s vettore di 9 bit, e' il risultato della somma tra i due numeri a
--9 bit a e b;
--cout, riporto in uscita dal ripple carry, di tipo bit (e' il
--riporto finale c3,
--infatti in tutto i riporti sono cin -esterno-, c0, c1, c2, c3 da
--calcolare );

port(
    a,b: in std_logic_vector(8 downto 0);
    cin: in std_logic;
    clock, clear: in std_logic;
    s: out std_logic_vector(8 downto 0);
    cout: out std_logic);

end RippleCarry9bit;

--definizione dell'architettura

Architecture MyRC9 of RippleCarry9bit is

--ingressi: atemp,btemp,cintemp

signal atemp,btemp:std_logic_vector(8 downto 0);
signal cintemp:std_logic;

--segnali ausiliari
signal c: std_logic_vector(7 downto 0);--mantiene memoria dei
riporti
```

```

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

--si include il Register9
component Register9 is
    port(d : in std_logic_vector(8 downto 0);
          clock,clear: in std_logic;
          q : out std_logic_vector(8 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,presi, inserendoli nel suo valore di uscita
ARegister : Register9 port map(a,clock,clear,atemp);

--BRegister e' il registro che immagazzina i valori del vettore a
--ricevuto in ingresso,presi, inserendoli nel suo valore di uscita
BRegister : Register9 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);

--parte dichiarativa

--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),cout,s(8));

end MyRC9;
```

Questo componente basa il suo funzionamento sulla presenza di nove Full Adder, ognuno dei quali calcola, rispettivamente, il bit di somma i-esimo ed il riporto che si genera.

La somma complessiva verrà salvata in un vettore stemp, che poi sarà trasferito in un registro a nove bit denominato Sregister, la cui uscita sarà s, il valore effettivo della somma.

Per quanto riguarda il riporto finale cout, questo coinciderà con quello calcolato dall'ultimo Full Adder, FA8, che calcola il bit di somma meno significativo. Il segnale risultante, coutemp, verrà poi trasferito in un Flip Flop, COUTFF, che si occuperà di instradarlo verso l'uscita effettiva del sommatore in esame.

E' questo il meccanismo utilizzato per realizzare la sincronizzazione.

Infatti, vengono utilizzati dei registri per memorizzare gli ingressi, in modo che l'unità, che di volta in volta si sta considerando, li riceva tutti nello stesso istante temporale.

Lo stesso ragionamento viene implementato nella gestione delle uscite che, dopo essere state calcolate e memorizzate tramite dei segnali ausiliari, per mezzo dei registri, verranno rese effettivamente disponibili nello stesso momento, garantendo, in questo modo, la correttezza del risultato che si sarebbe dovuto produrre in uscita.

Sommatore

E' l'unità di calcolo complessiva.

Come già si è scritto precedentemente, è costituito da un Sommatore Carry Select ad otto bit, a cui segue, in cascata, un Sommatore Ripple Carry a nove bit.

Questa disparità di numero di bit ricevuti come ingresso è dovuta al fatto che, dal primo sommatore, potrebbe generarsi un risultato con riporto, di cui si deve quindi tenere conto nel momento in cui questo viene instradato come ingresso del secondo sommatore.

Allora, si è deciso di salvare questo risultato di somma ad otto bit, con il relativo riporto che si genera, in un vettore a nove bit ordinato in maniera "8 downto 0", posizionando nella cella meno significativa, cioè la "8", il riporto cout prodotto in uscita dal primo sommatore, e nelle altre, dalla "7" alla "0", il vettore di somma ad otto bit che si è generato.

Il tutto verrà passato come ingresso al secondo sommatore, assieme al terzo operando del circuito complessivo, il vettore C.

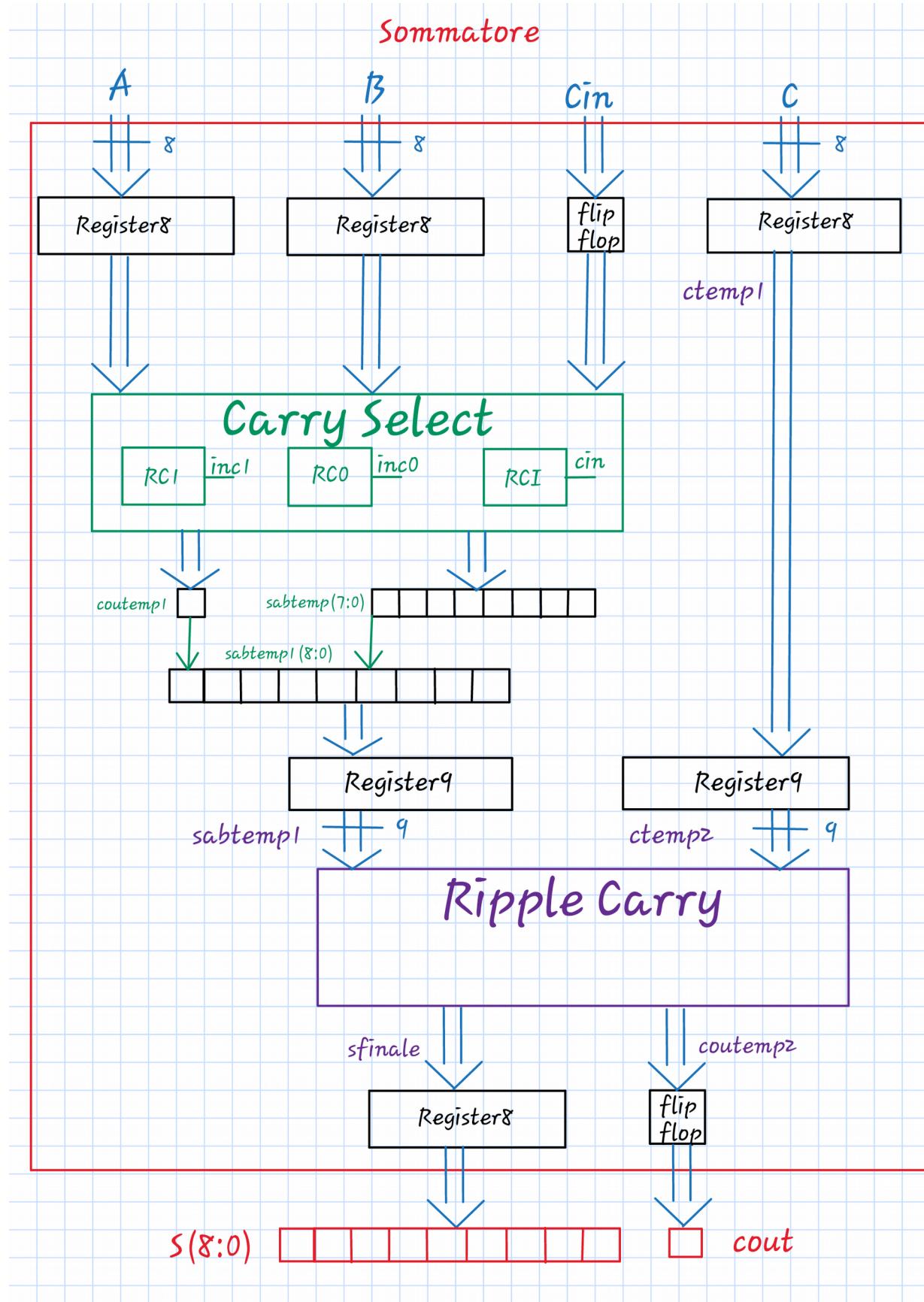
Questo componente, per come è stato strutturato, si occuperà di trasferire gli ingressi, rispettivamente, in due registri di nove bit e, considerando il riporto che riceve in ingresso pari a '0', poiché nulla, inizierà le sue operazioni.

Le uscite prodotte, allora, per essere sincronizzate, verranno sistemate in questo modo:

-il registro Sregister a nove bit, ricevendo in ingresso la somma sfinale che si è generata dal Ripple Carry a nove bit, si occuperà di mandare in uscita quest'ultima, denominata come s.

-il flip flop Coutff che, ricevendo in ingresso il riporto finale coutemp2, si occuperà di mandarlo in uscita come cout.

Si riporta uno schema esemplificativo:



Il codice che ne descrive il funzionamento è riportato qui di seguito:

```
--Sommatore totale
--si importa la libreria
library IEEE;
--si include il modulo di interesse della libreria
use IEEE.std_logic_1164.all;

--definizione del componente

Entity Sommatore is

--1 si riferisce al primo sommatore: Carry Select ad otto bit
--2 si riferisce al secondo sommatore: Ripple Carry a nove bit
--
--ingressi:
--a vettore di 8 bit, b vettore di 8 bit, c vettore di 8 bit;
--cin, riporto in ingresso;
--uscite:
--s vettore di 9 bit, e' il risultato della somma tra i tre numeri a
9 bit;
--cout, riporto in uscita

port
  a,b,c: in std_logic_vector(7 downto 0);
  cin: in std_logic;
  clock,clear:in std_logic;
  s: out std_logic_vector(8 downto 0);
  cout: out std_logic);

end Sommatore;

--definizione dell'architettura

Architecture MyS of Sommatore is

--segnali ausiliari per i registri

signal atemp,btemp,sabtemp: std_logic_vector(7 downto 0);
signal ctemp,sabtemp1,sfinale,ctemp2:std_logic_vector(8 downto 0);
--a nove bit!

signal coutemp1,coutemp2,coutfinale:std_logic;

--variabile ausiliaria usata del passaggio dell'ingresso c ad otto
--bit nel registro a nove bit
signal cintemp2:std_logic:='0';
```

```

--COMPONENTI UTILIZZATI

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

--si include il Register8
component Register8 is
    port(
        d : in std_logic_vector(7 downto 0);
        clock,clear: in std_logic;
        q : out std_logic_vector(7 downto 0)
    );
end component;

--si include il Register9
component Register9 is
    port(d : in std_logic_vector(8 downto 0);
          clock,clear: in std_logic;
          q : out std_logic_vector(8 downto 0)
    );
end component;

--inclusione del Flip Flop
component FlipFlop is
    port(d,clock,clear: in std_logic;
          q: out std_logic
    );
end component;

--inclusione del carry select ad otto bit
component CarrySelect8b is
    port(
        a,b : in std_logic_vector(7 downto 0 );
        cin, clock, clear : in std_logic;
        s : out std_logic_Vector(7 downto 0);
        cout: out std_logic
    );
end component;

--inclusione del ripple carry a nove bit
component RippleCarry9bit is
    port(
        a,b: in std_logic_vector(8 downto 0);
        cin: in std_logic;
        clock, clear: in std_logic;
        s: out std_logic_vector(8 downto 0);
        cout: out std_logic
    );
end component;

```

```

--INIZIO
begin
--OPERAZIONI

--PRIMO SOMMATORE

Primo:CarrySelect8b port map(a,b,cin,clock,clear,sabtemp,coutemp1);

--assegnamento della somma tra a e b complessiva, come cifra in
--posizione 8, del riporto coutemp1: risultato a 9 bit

sabtemp1(0)<=sabtemp(0);
sabtemp1(1)<=sabtemp(1);
sabtemp1(2)<=sabtemp(2);
sabtemp1(3)<=sabtemp(3);
sabtemp1(4)<=sabtemp(4);
sabtemp1(5)<=sabtemp(5);
sabtemp1(6)<=sabtemp(6);
sabtemp1(7)<=sabtemp(7);
sabtemp1(8)<=coutemp1;

--secondo operando del secondo sommatore

ctemp(0)<=c(0);
ctemp(1)<=c(1);
ctemp(2)<=c(2);
ctemp(3)<=c(3);
ctemp(4)<=c(4);
ctemp(5)<=c(5);
ctemp(6)<=c(6);
ctemp(7)<=c(7);
ctemp(8)<=cintemp2;

--CRegister: passaggio di c nel registro a 9 bit

C2Register : Register9 port map(ctemp,clock,clear,ctemp2);--Operando
C a 9 bit

--SECONDO SOMMATORE

Secondo: RippleCarry9bit port
map(sabtemp1,ctemp2,'0',clock,clear,sfinale,coutemp2);

--SINCRONIZZAZIONE DELLE USCITE

--riporto finale
COUTFF : FlipFlop port map(coutemp2,clock,clear,cout);

--somma finale
SFRegister : Register9 port map(sfinale,clock,clear,s);

end MyS;

```

Simulazione

Ai fini di verificare il corretto funzionamento del circuito Sommatore, con l'ausilio del software appropriato, si è scritto un test bench che ne simuli il comportamento, in base a dieci terne di ingressi differenti, oltre che al riporto iniziale.

Il codice è il seguente:

```
--Test SOMMATORE
--si importa la libreria
library IEEE;
--si include il package che si vuole usare
use IEEE.std_logic_1164.all;

--descrizione della classe di test

Entity TestSommatoreComplessivo is
end TestSommatoreComplessivo;

Architecture TS of TestSommatoreComplessivo is

--si inizializzano i valori che si vuole assumano i segnali
signal iclock : std_logic := '0';--segnalet di clock
signal iclear : std_logic := '1';--segnalet di clear
--segnali usati
signal ia,ib,ic :std_logic_vector(7 downto 0);
signal us :std_logic_vector(8 downto 0);
signal icin,ucout : std_logic;

--si definisce il componente CarrySelect8
component Sommatore is

port(
    a,b,c: in std_logic_vector(7 downto 0);
        cin: in std_logic;
    clock,clear:in std_logic;
        s: out std_logic_vector(8 downto 0);
        cout: out std_logic);
end component;

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 60 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 calori attesi saranno cout, che e' il riporto finale
--e la somma s a nove bit
process
begin
--1)cout=0,s=0  1010 0001
    icin<='1';
    ia<="00000000";
    ib<="00000001";
    ic<="10100000";
    wait for 10 ns;
--2)cout=1,s=0  1010 0000
    icin<='1';
    ia<="11111110";
    ib<="00000010";
    ic<="10100001";
    wait for 10 ns;
--3)cout=1,s=0
    ia<="11111111";
    ib<="10000000";
    ic<="00000001";
    wait for 10 ns;
--4)cout=1,s=0  0100 0010
    icin<='0';
    ia<="01111111";
    ib<="00000011";
    ic<="11000000";
    wait for 10 ns;
--5)cout=1,s=0  0010 0001
    ia<="10001001";
    ib<="01110111";
    ic<="00100001";
    wait for 10 ns;

```

```

--6)cout=1,s=0  0010 0010
    icin<='1';
    ia<="10000000";
    ib<="00000001";
    ic<="10000001";
    wait for 10 ns;
--7)cout=1,s=0  1000 1001
    icin<'0';
    ia<="00000001";
    ib<="00000000";
    ic<="11111111";
    wait for 10 ns;
--8)cout=1,s=0  1000 1001
    icin<='1';
    ia<="10001001";
    ib<="01110111";
    ic<="00100001";
    wait for 10 ns;
--9)cout=1,s=0  1010 1001
    icin<'0';
    ia<="10101010";
    ib<="10101010";
    ic<="01010101";
    wait for 10 ns;
--10)
    icin<='1';
    ia<="00101010";
    ib<="00101011";
    ic<="11000000";
    wait for 10 ns;
end process;

```

**circuito: Sommatore port
map(ia,ib,ic,icin,iclock,iclear,us,ucout);**

end TS;

Si è scelto di usare tre process:

-il primo è dedicato alla variazione del segnale di **clock** ;

-il secondo è dedicato al segnale di **clear** ;

-il terzo, ovvero l'ultimo, è quello che descrive la variazione dei tre operandi ad otto bit forniti come ingresso al circuito, cioè A, B e C, insieme al riporto iniziale Cin.

Si riportano, quindi, alcune immagini estratte dalla simulazione:

	Msgs	00000000	0	0	010100010	110100010	110000001	101000010
+ ◆ /testsommatorecomplessivo/us	00000000	000000000						
◆ /testsommatorecomplessivo/uout	0							
◆ /testsommatorecomplessivo/idock	0							
◆ /testsommatorecomplessivo/idear	0							
◆ /testsommatorecomplessivo/icin	1							
+ ◆ /testsommatorecomplessivo/ic	10100000	10100000	10100001	00000001	11000000	00100001	10000001	
+ ◆ /testsommatorecomplessivo/ib	00000001	00000001	00000010	10000000	00000011	01110111	00000001	
+ ◆ /testsommatorecomplessivo/ia	00000000	00000000	11111110	11111111	01111111	10001001	10000000	

I primi 4 Risultati

	100100001	100000011	100000000	100100010
+ ◆ /testsommatorecomplessivo/us				
◆ /testsommatorecomplessivo/uout				
◆ /testsommatorecomplessivo/idock				
◆ /testsommatorecomplessivo/idear				
◆ /testsommatorecomplessivo/icin				
+ ◆ /testsommatorecomplessivo/ic	11111111	00100001	01010101	11000000
+ ◆ /testsommatorecomplessivo/ib	00000000	01110111	10101010	00101011
+ ◆ /testsommatorecomplessivo/ia	00000001	10001001	10101010	00101010

La simulazione procede fino a 100 ns

	100100010	110101001	100010110	010100010	110100010	110000001	101000010
+ ◆ /testsommatorecomplessivo/us	00000000						
◆ /testsommatorecomplessivo/uout	0						
◆ /testsommatorecomplessivo/idock	0						
◆ /testsommatorecomplessivo/idear	0						
◆ /testsommatorecomplessivo/icin	1						
+ ◆ /testsommatorecomplessivo/ic	10100000	10100000	10100001	00000001	11000000	00100001	10000001
+ ◆ /testsommatorecomplessivo/ib	00000001	00000001	00000010	10000000	00000011	01110111	00000001
+ ◆ /testsommatorecomplessivo/ia	00000000	00000000	11111110	11111111	01111111	10001001	10000000

Da 100 fino a 150 ns

	101000010	100100001	100000011	100000000	100100010
+ ◆ /testsommatorecomplessivo/us					
◆ /testsommatorecomplessivo/uout					
◆ /testsommatorecomplessivo/idock					
◆ /testsommatorecomplessivo/idear					
◆ /testsommatorecomplessivo/icin					
+ ◆ /testsommatorecomplessivo/ic	10000001	11111111	00100001	01010101	11000000
+ ◆ /testsommatorecomplessivo/ib	00000001	00000000	01110111	10101010	00101011
+ ◆ /testsommatorecomplessivo/ia	10000000	00000001	10001001	10101010	00101010

Fino a 200 ns, fine simulazione.

La durata complessiva che si è scelta è di 200 ns.

Si può notare che il primo risultato utile, ovvero quello della somma appena considerata, **sopraggiunge dopo tre colpi di clock**.

Il tempo di latenza è quindi di tre colpi di clock.

Questo è dovuto al numero di registri presenti, infatti, i segnali di interesse devono attraversare, prima i registri di ingresso del primo sommatore, il Carry Select, poi i registri in ingresso del secondo sommatore, il Ripple Carry, ed infine i registri che instradano i risultati verso l'uscita del circuito complessivo, il sommatore.

Dagli screenshots finora riportati è possibile notare come, una volta che i registri sono stati riempiti, il risultato, relativo agli ingressi che si stanno considerando, verrà reso disponibile ogni colpo di clock evidenziando così la vera funzionalità del pipeline.

In conclusione, il circuito che si voleva realizzare, funziona correttamente.