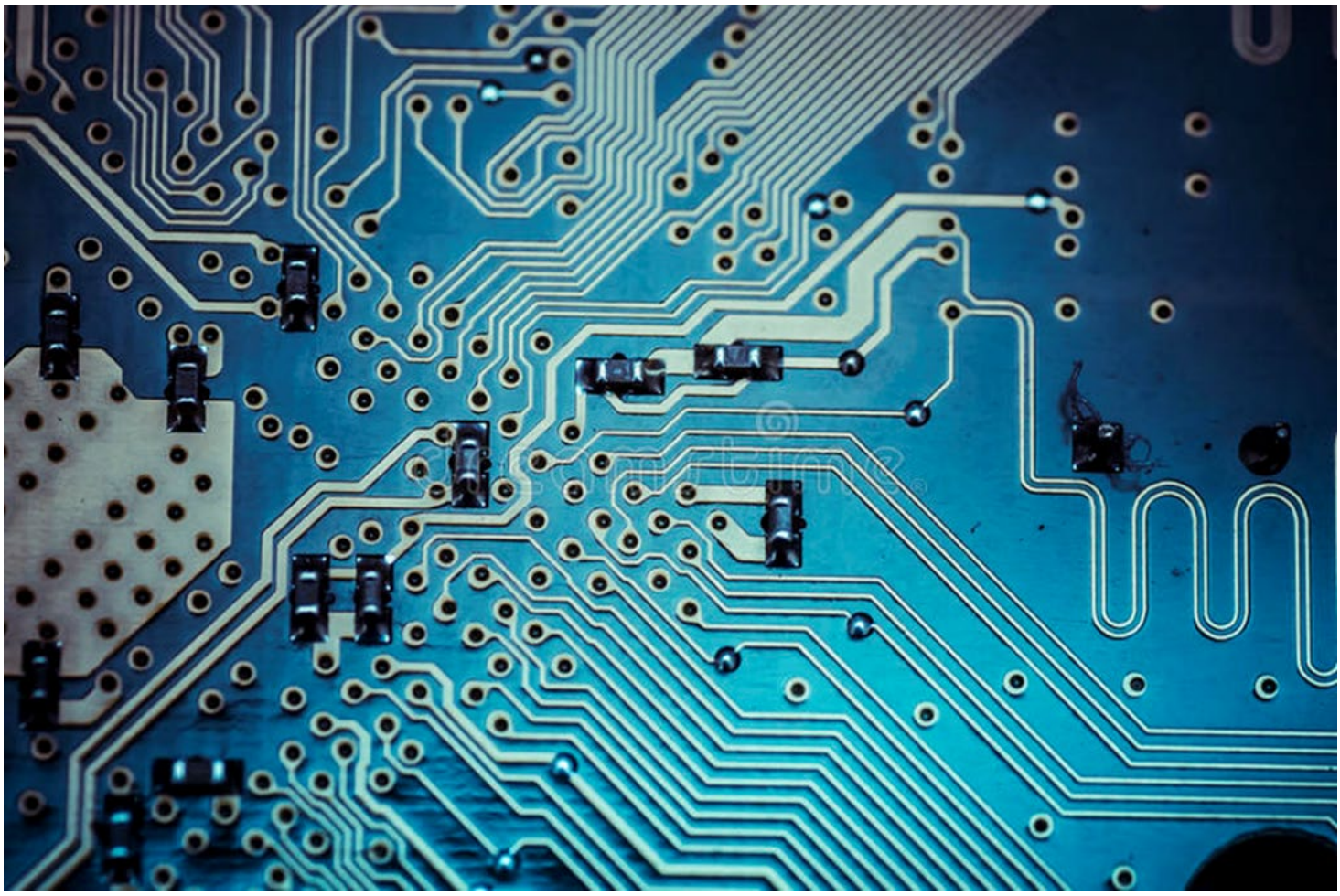
Progetto I
Elettronica Digitale

Anno accademico 2021/2022



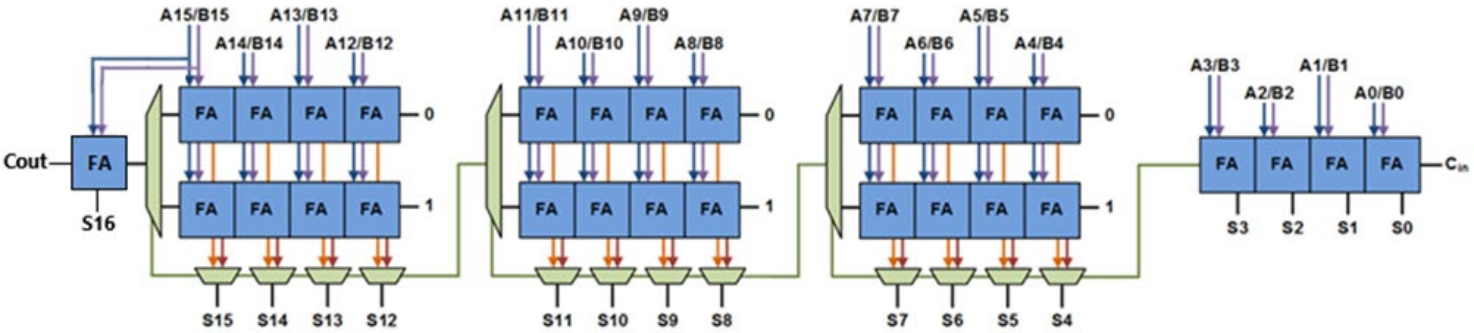
Francesco Zumpano, MAT: 209693

Luana Pulignano, MAT: 209471

Presentazione del circuito

La traccia del progetto chiede di implementare un carry select parametrico che lavora su numeri espressi in complemento a due. In particolare, gli addendi usati devono essere a 16 bit.

La struttura da noi scelta prevede la divisione dei bit in gruppi da 4 bit ciascuno. Abbiamo quindi utilizzato 3 carry select, un ripple carry e un full adder aggiuntivo per gestire l'ultimo bit di riporto.



Le componenti elementari utilizzate sono il full adder e il multiplexer a due ingressi.

Con questa struttura i primi 4 bit dei due numeri da sommare vengono processati dal Ripple Carry, il quale genera, oltre alla somma, un bit di riporto che viene usato come selettore dai multiplexer del primo Carry Select.

Nel frattempo, i blocchi di ripple carry hanno generato i due possibili valori della somma dei successivi 4 bit, dunque bisogna aspettare solo il tempo di selezione del multiplexer per terminare la generazione della somma del primo carry select. Lo stesso procedimento è ripetuto per tutti gli altri carry select a 4 bit che, in un circuito ad n bit, sono in numero pari a n/4 -1. Il riporto in uscita dall'ultimo carry select viene dato in ingresso ad un full adder aggiuntivo insieme ai bit più significativi dei due numeri che stiamo sommando. Il risultato di somma generato avrà quindi un bit in più rispetto agli addendi.

La struttura scelta (gruppi da 4 bit) fa uso di più componenti fisiche rispetto alla configurazione di base ma ci permette di ridurre la latenza del tempo di calcolo. In particolare, esso è pari a:

Tempo di calcolo = $\tau_{RC} + \left(\frac{n}{4} - 1\right) \cdot \tau_{MUX} + \tau_{FA} = 4 \cdot \tau_{FA} + \left(\frac{n}{4} - 1\right) \cdot \tau_{MUX} + \tau_{FA} = 5 \cdot \tau_{FA} + \left(\frac{n}{4} - 1\right) \cdot \tau_{MUX}$

Che nel nostro caso diventa: $5 \cdot \tau_{FA} + 3 \cdot \tau_{MUX}$

Di seguito elenchiamo i vari file di supporto allo sviluppo del circuito.

Libreria

Per soddisfare la richiesta di circuito parametrico, abbiamo inserito il parametro *n* all'interno di una libreria per non appesantire il codice degli altri componenti.

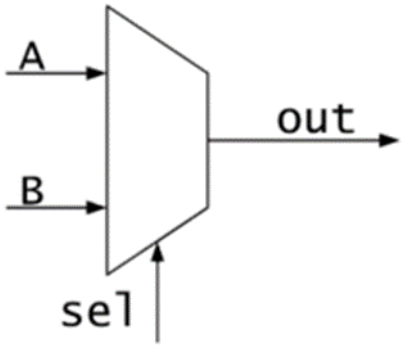
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

package MyDef is
    constant n:integer:=16;
end package MyDef;
```

Multiplexer

Il multiplexer è una rete combinatoria che permette di selezionare, tra i segnali in ingresso, quello che deve passare in uscita.

Tale scelta è effettuata dai bit detti di selezione. In particolare, con n bit di selezione si può scegliere tra 2^n ingressi.

Nel nostro caso è necessario utilizzare dei multiplexer con due ingressi, poiché dobbiamo scegliere tra due possibili risultati in base al valore del carry in ingresso. Di conseguenza è sufficiente un solo bit di selezione.

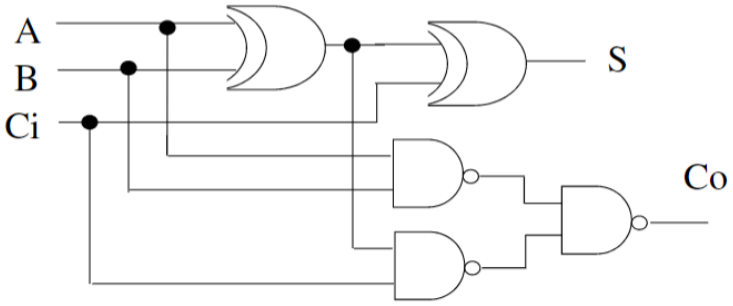


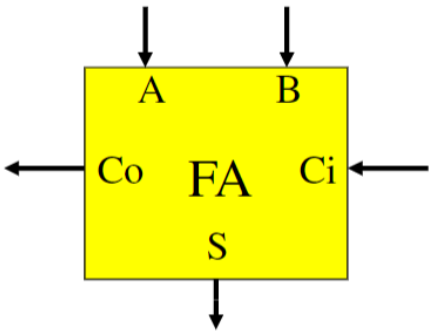
Di seguito è riportata l'implementazione in VHDL:

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

entity Multiplexer is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           sel : in STD_LOGIC;
           u : out STD_LOGIC);
end Multiplexer;

architecture Behavioral of Multiplexer is
begin
u<=a when sel='0' else
    b when sel='1' else 'X';
end Behavioral;
```

Full Adder

Il full adder è una rete logica che esegue la somma tra due bit con un eventuale riporto. Esso, dunque, riceve 3 ingressi, i due bit da sommare e il riporto in entrata e genera due uscite, il bit di somma e il bit di riporto in output.





$$S = \overline{A} \cdot Y + A \cdot \overline{Y} = A \oplus B \oplus Ci$$

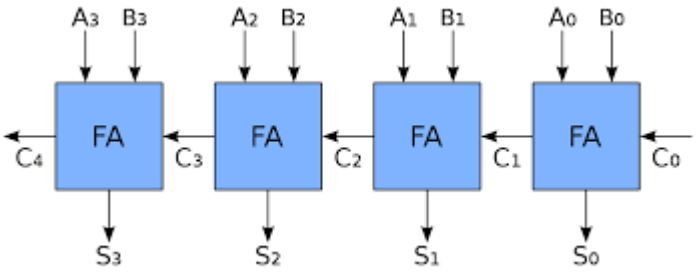
$$Co = \overline{\overline{(A \oplus B) \cdot Ci} \cdot \overline{A \cdot B}}$$

Per la scrittura del codice ci siamo serviti di due segnali ausiliari, il propagate e il generate che hanno il compito rispettivamente di propagare e generare il riporto.

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

entity FullAdder is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           cin : in STD_LOGIC;
           s : out STD_LOGIC;
           cout : out STD_LOGIC);
end FullAdder;

architecture Behavioral of FullAdder is
signal p,g: STD_LOGIC;
begin
p<=a xor b;
g<=a and b;
s<=p xor cin;
cout<=g or (p and cin);
end Behavioral;
```

Ripple Carry

Il ripple carry adder è il circuito più semplice per implementare la somma tra due numeri binari. Il suo vantaggio è che richiede poche porte logiche, dunque è poco costoso. D'altra parte, è poco efficiente in quanto ogni full adder che lo compone deve aspettare il precedente per effettuare la somma (problema della propagazione del riporto).



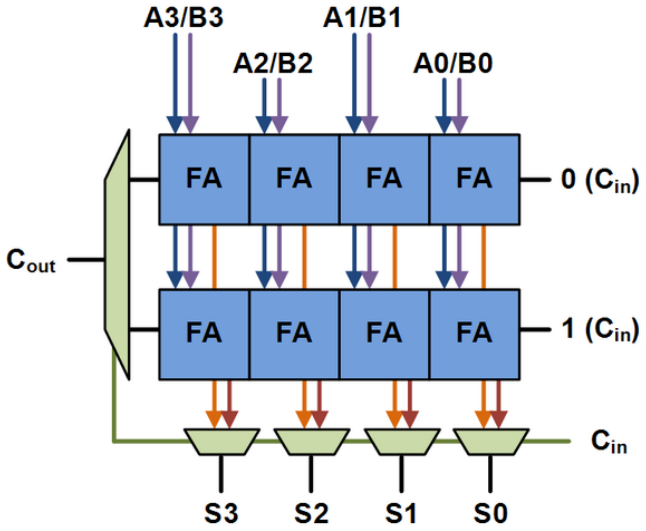
Il ripple carry da noi utilizzato lavora a gruppi di 4 bit. Abbiamo scelto un approccio strutturale che impiega come componente il full adder precedentemente definito.

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

entity RippleCarry4 is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           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 RippleCarry4;
architecture Struct of RippleCarry4 is
component FullAdder is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           cin : in STD_LOGIC;
           s : out STD_LOGIC;
           cout : out STD_LOGIC);
end component;
signal carry: STD_LOGIC_VECTOR (4 downto 0);
begin
MyFor: for i in 0 to 3 generate
    MyFA: FullAdder port map( A(i), B(i), carry(i), S(i), carry(i+1));
end generate MyFor;
carry(0)<=cin;
cout<=carry(4);
end Struct;
```

Carry Select a 4 bit

Il carry select adder è un circuito sommatore che utilizza, oltre ai full adder, dei multiplexer che consentono di ottimizzare i tempi.

La sua struttura si compone di due ripple carry che ricevono gli stessi bit da sommare ma riporti opposti, per generare i due possibili valori di somme. In base al reale valore del riporto in input, verrà selezionato il valore corretto in output dai multiplexer. La sua configurazione più complessa consente di ridurre i tempi di calcolo, rendendo il carry select più efficiente del ripple carry.



Nella nostra implementazione a 4 bit abbiamo utilizzato dei segnali aggiuntivi oltre ai componenti Mux e Ripple. In particolare:

- i segnali *c0* e *c1* rappresentano i due riporti in ingresso che assumono rispettivamente i valori 0 e 1;
- i segnali *r0* e *r1* rappresentano i riporti in uscita dai due ripple carry (gli indici si riferiscono al valore del carry in ingresso);
- i segnali *s0* e *s1* rappresentano i due valori di somma uscenti dai due ripple carry (gli indici si riferiscono al valore del carry in ingresso);

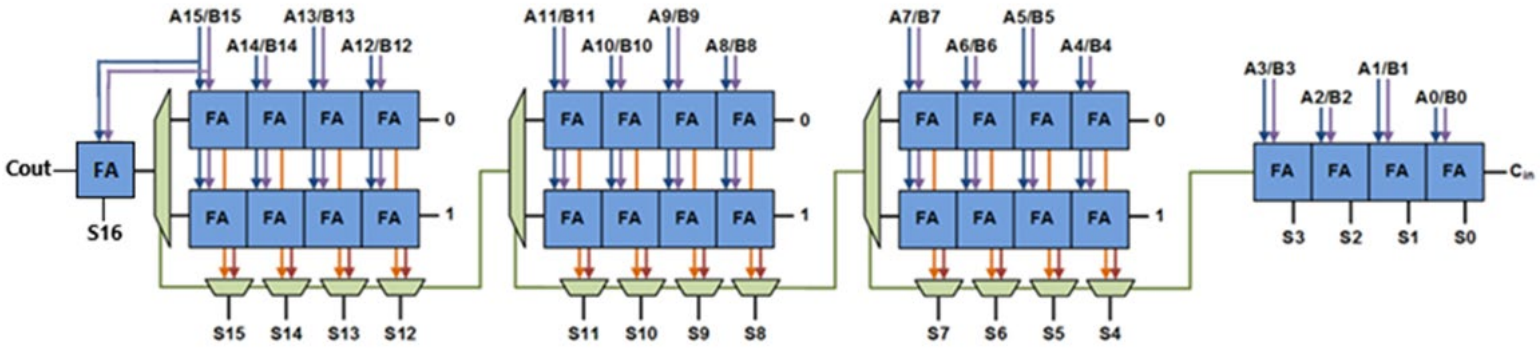
Inoltre, abbiamo istanziato 5 multiplexer, 4 per ogni bit di somma e uno per il riporto in output.

Di seguito il codice del Carry Select a 4 bit:

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

entity CarrySelect4 is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           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 CarrySelect4;

architecture Struct of CarrySelect4 is
component RippleCarry4 is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           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;

component Multiplexer is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           sel : in STD_LOGIC;
           u : out STD_LOGIC);
end component;
signal r0, r1: STD_LOGIC;   --riporti in uscita dai due ripple carry (r0 è riferito al bit in ingresso 0)
signal c0,c1: STD_LOGIC;
signal s0, s1:  STD_LOGIC_VECTOR (3 downto 0); --uscite dei ripple carry (s0 è riferito al bit in ingresso 0)
begin
c0<='0'; c1<='1';
RC0: RippleCarry4 port map(A,B,c0,s0,r0);
RC1: RippleCarry4 port map(A,B,c1,s1,r1);

MyFor: for i in 0 to 3 generate
    MyMux: Multiplexer port map(s0(i),s1(i),Cin,S(i));
end generate MyFor;
MuxFin: Multiplexer port map(r0,r1,Cin,Cout);

end Struct;
```

Carry Select a 16 bit

Il carry select a 16 bit utilizza in cascata 3 carry select a 4 bit, un ripple carry per il primo gruppo di bit e un full adder in chiusura per la gestione dell'ultimo bit di riporto.



Per propagare il riporto abbiamo impiegato un segnale aggiuntivo *carry* che conduce il riporto calcolato nel blocco corrente al successivo blocco di carry select. Esso viene usato come selettore nei mux.

Essendo in forma parametrica, il carry select ha sempre n/4-1 stadi di carry select a 4 bit (3 nel nostro caso).

Nel full adder finale abbiamo copiato gli ultimi bit dei vettori A e B per calcolare il riporto conclusivo, estendendo i numeri per segno poiché espressi in notazione complemento a due.

Questa l'implementazione del carry select a 16 bit:

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
library work;
use work.myDef.all;

entity CarrySelect16 is

    Port ( A : in STD_LOGIC_VECTOR (n-1 downto 0);
           B : in STD_LOGIC_VECTOR (n-1 downto 0);
           Cin : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (n downto 0));
end CarrySelect16;

architecture Struct of CarrySelect16 is
component CarrySelect4 is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           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;


component RippleCarry4 is
    Port ( A : in STD_LOGIC_VECTOR (3 downto 0);
           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;
component FullAdder is
    Port ( a : in STD_LOGIC;
           b : in STD_LOGIC;
           cin : in STD_LOGIC;
           s : out STD_LOGIC;
           cout : out STD_LOGIC);
end component;
signal carry: STD_LOGIC_VECTOR (n/4 downto 0);
begin
RC4: RippleCarry4 port map(A(3 downto 0),B(3 downto 0),Cin,S(3 downto 0),carry(0));

MyFor: for i in 1 to (n/4)-1 generate
    CSi:CarrySelect4 port map(A(i*4+3 downto i*4),B(i*4+3 downto i*4),carry(i-1),S(i*4+3 downto i*4),carry(i));
end generate MYFor;

FA: FullAdder port map(A(n-1),B(n-1),carry(n/4-1),S(n),carry(n/4));
end Struct;
```

Test Bench

Nel test bench abbiamo istanziato il carry select a 16 bit e i segnali di input (*IA, IB, ICarry*) e output (*OS)*, i quali trasportano i valori di prova in ingresso al circuito.

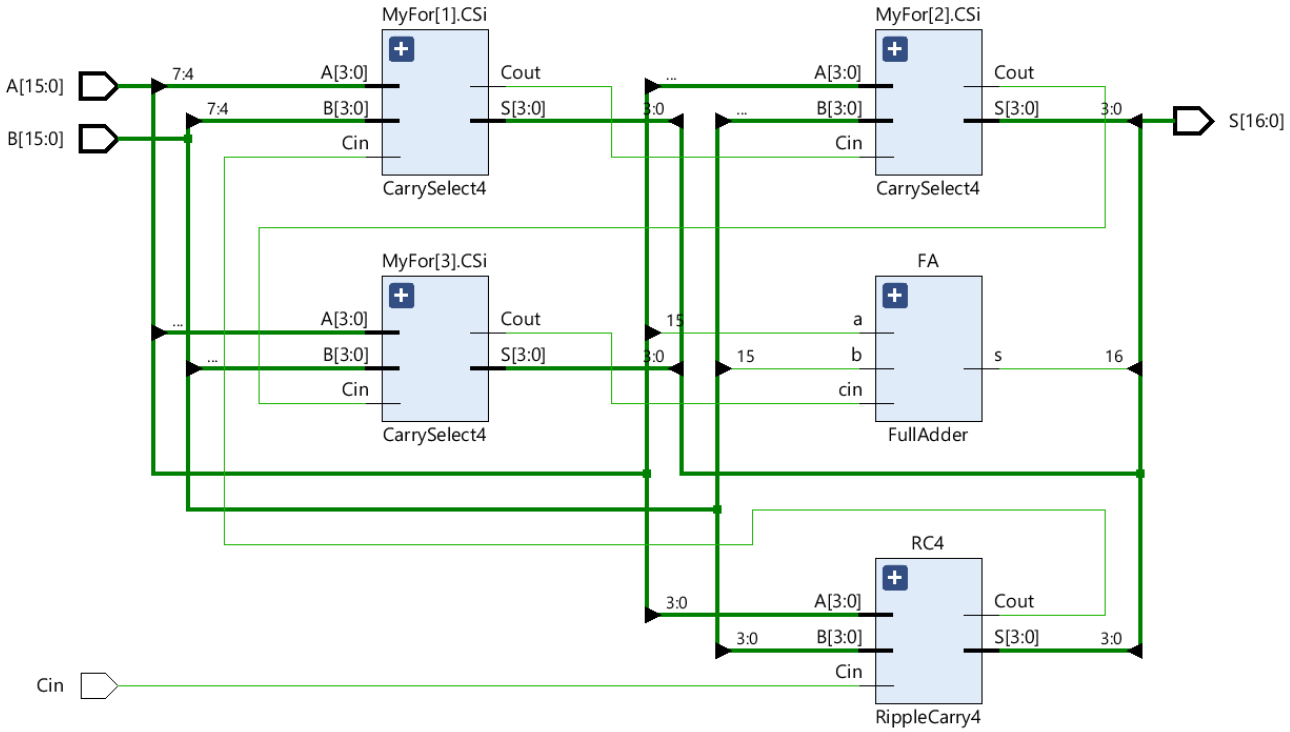
Per testare tutti i possibili valori di input, il range varia tra -2^{n-1} e $2^{n-1}-1$, ovvero tutti i numeri esprimibili con *n* bit.

Per ogni somma calcolata il circuito attende 10ns in modo da visualizzare il corrispondente valore.

Il bit di riporto in input è settato a 0 perché nella somma tra due numeri il primo riporto è sempre 0.

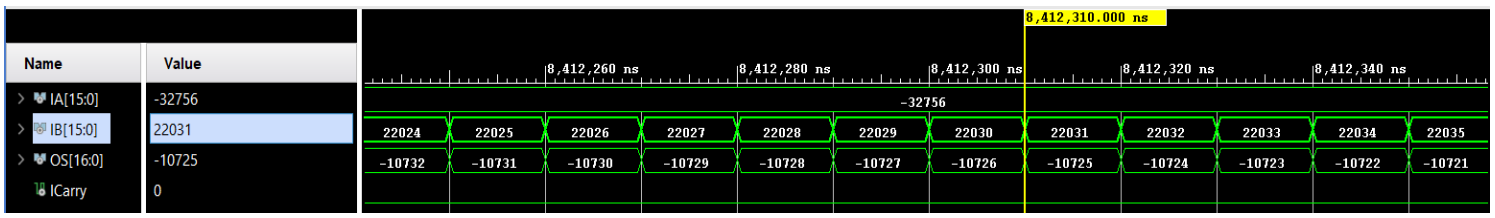
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_arith.ALL;
library work;
use work.myDef.all;

entity TestBench is
--  Port ( );
end TestBench;

architecture Behavioral of TestBench is
component CarrySelect16 is
    Port ( A : in STD_LOGIC_VECTOR (n-1 downto 0);
           B : in STD_LOGIC_VECTOR (n-1 downto 0);
           Cin : in STD_LOGIC;
           S : out STD_LOGIC_VECTOR (n downto 0));
end component;
signal IA,IB:STD_LOGIC_VECTOR (n-1 downto 0);
signal OS:STD_LOGIC_VECTOR (n downto 0);
signal ICarry:STD_LOGIC;
begin
    CS16: CarrySelect16 port map(IA,IB,ICarry,OS);
    ICarry<='0';
    process
    begin

    for va in -(2**(n-1)) to (2**(n-1)-1) loop
        IA<=conv_std_logic_vector(va,n);
        for vb in -(2**(n-1)) to (2**(n-1)-1) loop
            IB<=conv_std_logic_vector(vb,n);
            wait for 10ns;
        end loop;
    end loop;
    end process;
end Behavioral;
```

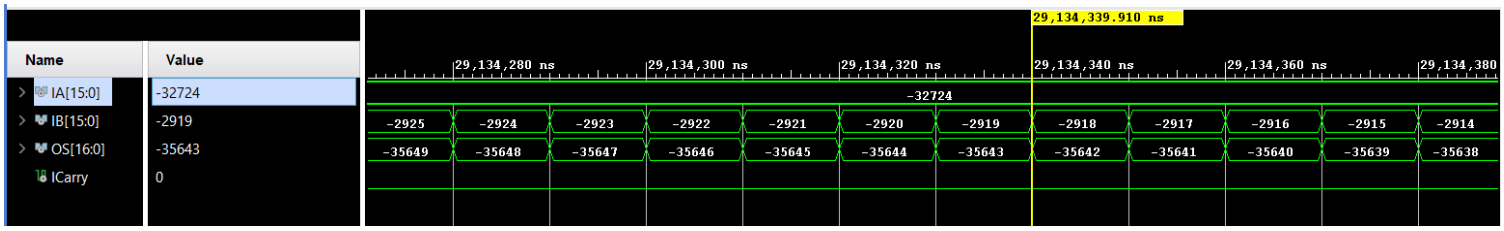
Lo schema iniziale del circuito è il seguente:



Simulazione Behavioral

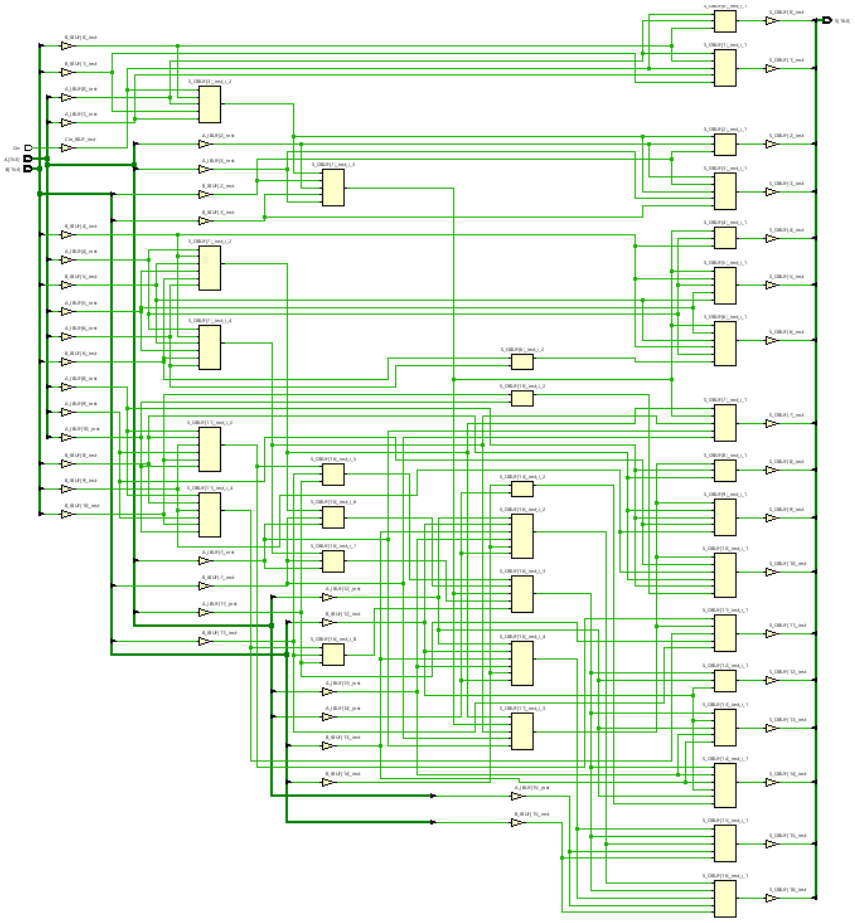
Questi sono alcuni esempi di somme nella simulazione behavioral. Come si può notare, i calcoli sono corretti e non è presente ritardo nei calcoli poiché si tratta di una simulazione ideale.



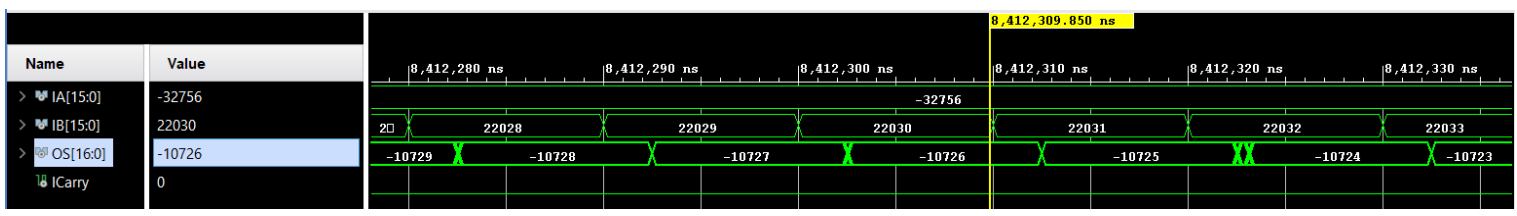


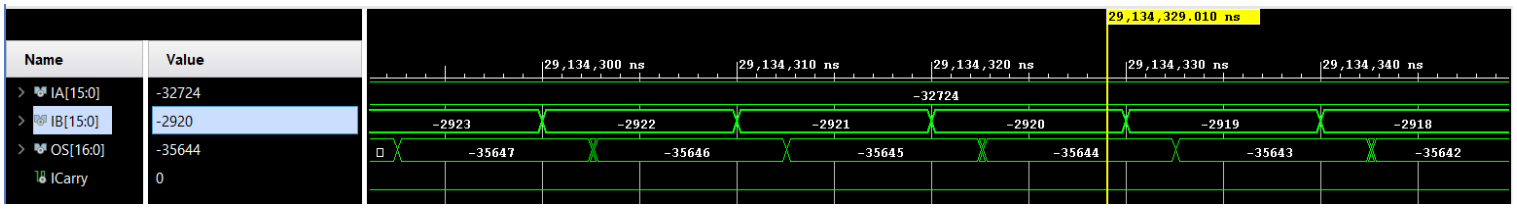
Simulazione Post-Sintesi

Lo schema post-sintesi è il seguente:



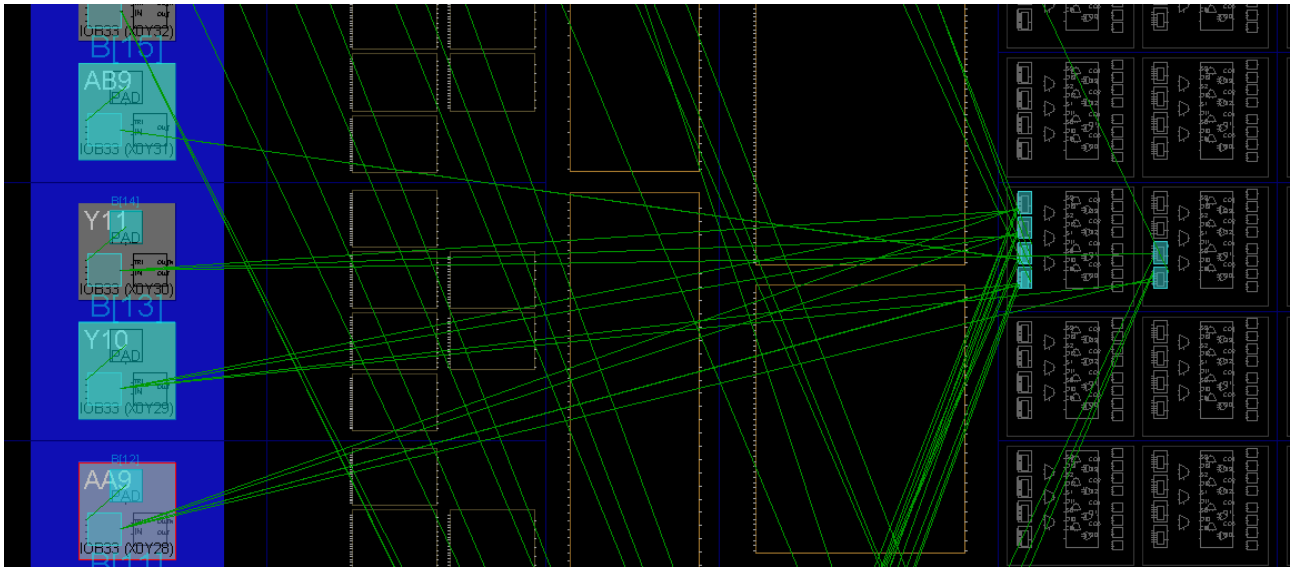
Le somme al tempo corrispondente coincidono con quelle calcolate nella simulazione behavioral ma è stato introdotto un lieve ritardo di 2.520 ns tra il cambio dell'ingresso e la produzione del risultato.



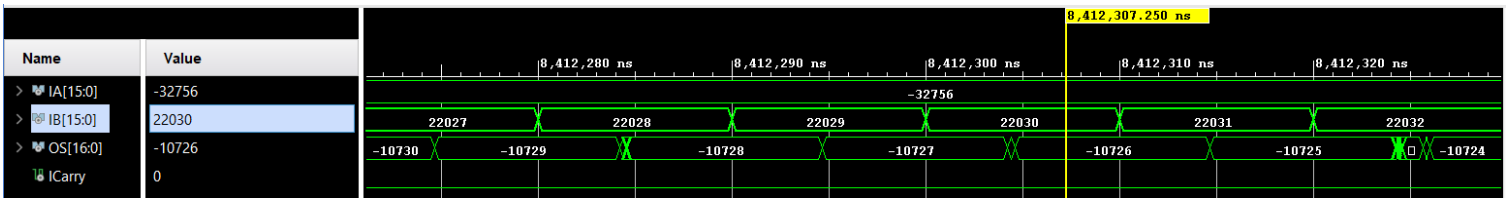


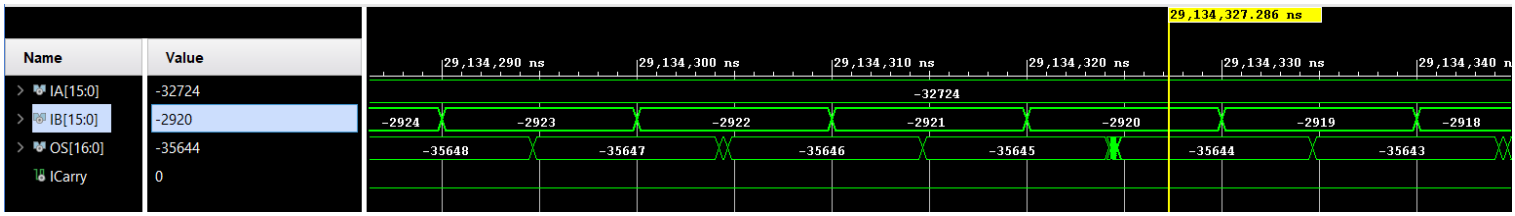
Simulazione Post-Implementazione

Dopo l'implementazione, sono visibili tutti i collegamenti tra le componenti utilizzate nel dispositivo scelto, in questo caso la ZedBoard. Di seguito una porzione del dispositivo:



Anche in questo caso le somme coincidono con le precedenti simulazioni ma il ritardo è aumentato, passando a 4.637 ns, quasi il doppio del ritardo introdotto dalla sintesi, a causa dei collegamenti fisici creati dall'implementazione.





Risorse occupate

Attraverso il file report utilization è possibile controllare le risorse usate per il circuito. In particolare, è di nostro intresse valutare il numero di Look Up Table e di IOBuffer utilizzati. Le LUT impiegate in questo caso sono 25, pari allo 0.05% del totale, mentre gli IOBuffer usati sono 50, pari al 25% del totale.

Site Type	Used	Fixed	Available	Util%
Slice LUTs	25	0	53200	0.05
LUT as Logic	25	0	53200	0.05
LUT as Memory	0	0	17400	0.00

Site Type	Used	Fixed	Available	Util%
Bonded IOB	50	0	200	25.00
IOB Master Pads	24			
IOB Slave Pads	24			