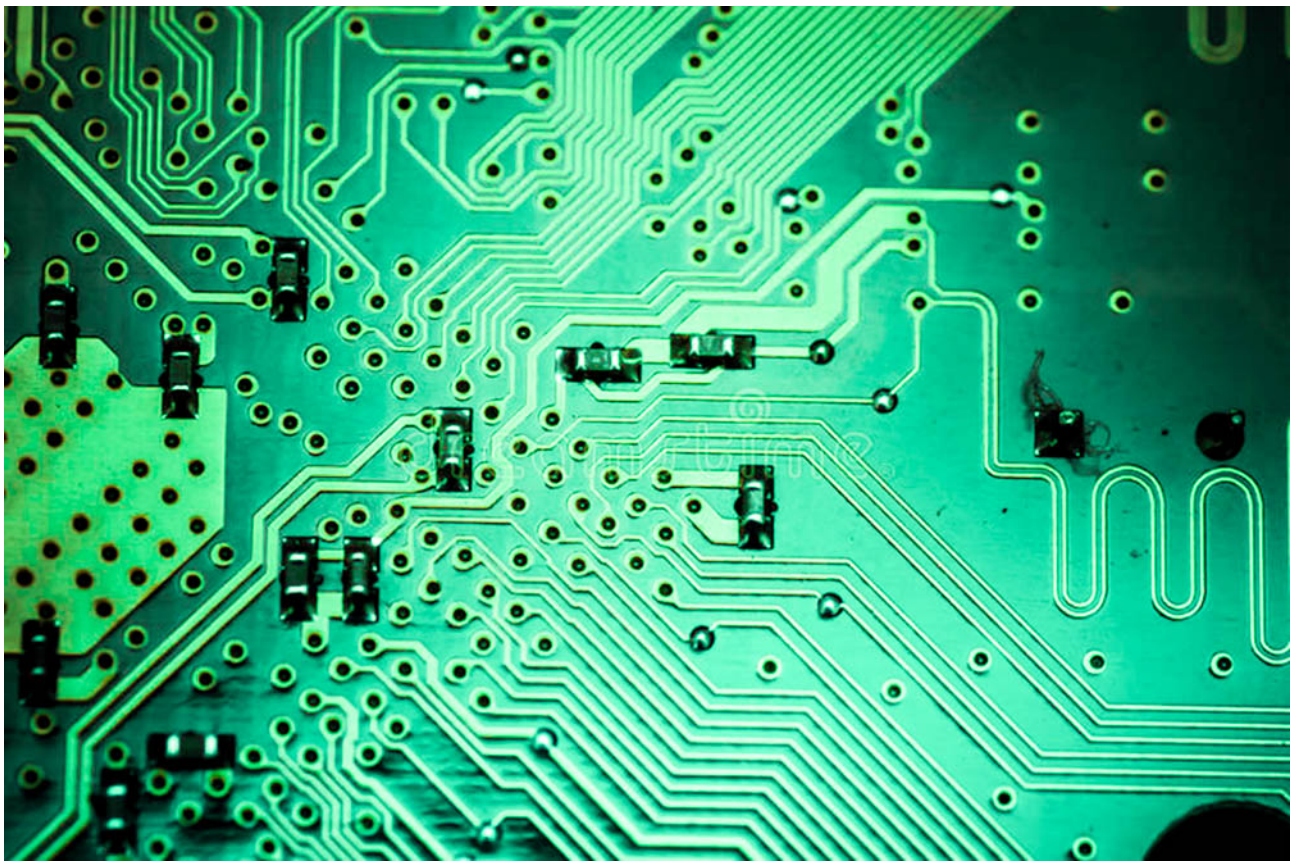
Progetto II
Elettronica Digitale

Anno accademico 2021/2022



Francesco Zumpano, MAT: 209693

Luana Pulignano, MAT: 209471

Presentazione del circuito

La traccia del progetto chiede di creare un circuito parametrico che riceve quattro operandi (*A, B, C, D*) a n bit in complemento a due e un segnale di controllo *Contr* a 2 bit e, in base a quest'ultimo, di calcolare le seguenti operazioni:

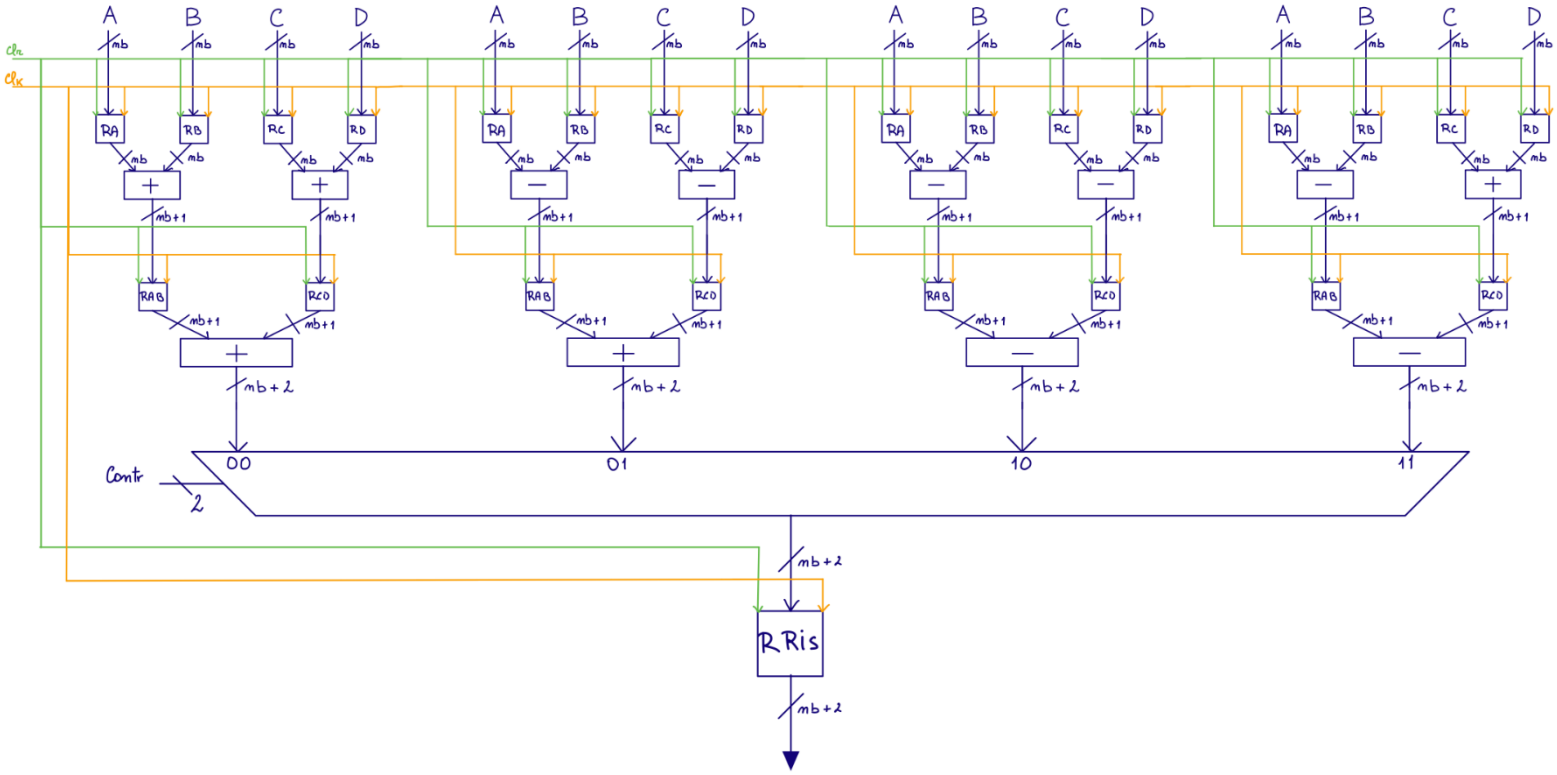
Operazione
A+B+C+D, se Contr="00"
A−B+C−D, se Contr="01"
A+B−C+D, se Contr="10"
A−B−C−D, se Contr="11"

Il circuito da noi implementato utilizza una struttura pipeline al fine di velocizzare le operazioni. Essa, infatti, utilizza dei registri in cui salvare i valori dei segnali che possono essere utilizzati per le successive operazioni in cascata, mentre i segnali di ingresso possono variare senza alterare i calcoli in corso.

I registri del nostro circuito si comportano come dei flip-flop in quanto sensibili ai fronti del clock. In questo caso la traccia richiede che i segnali varino ai fronti di discesa. Inoltre, i registri contengono il segnale di clear: esso è un segnale asincrono che, quando è alto, pone a zero tutti i valori salvati nei registri.

Le componenti principali che svolgono i calcoli sono un adder e un subtracter combinati fra loro in base a una delle possibili operazioni da svolgere.

Di seguito lo schema del circuito:



Dal disegno si evince che il circuito calcola in parallelo tutte e 4 le possibili operazioni e, in base al valore di *Contr*, il multiplexer seleziona il risultato corretto. Tutti i registri condividono i segnali di clock e clear (*clk* e *clr)*.

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

Libreria

All'interno della libreria è stata definita la costante *nbit* che indica il numero di bit dei segnali in ingresso al circuito. Essa ci ha permesso di rendere il circuito parametrico e di semplificare la richiesta di testarlo con ingressi sia ad 8 che a 16 bit. Infatti, basta cambiare il valore della costante nella libreria per riadattare il codice.

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

package costanti is
constant nbit: integer:=8; --16;
end package costanti;
```

Adder

L'adder è stato implementato come un ripple carry adder, usando i segnali di propagate e generate insieme al carry. L'adder è provvisto di due registri, *RA* e *RB*, i quali contengono i valori in ingresso al circuito. I registri comprendono i segnali di *clk* e *clr*, entrambi presenti nella sensitivity list del process in quanto tra loro indipendenti. In particolare il clear è più prioritario e, quando è alto, assegna ai segnali salvati nel registro il valore 0.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  library work;
4  use work.costanti.all;
5  entity Adder is
6  generic (nb:integer:=nbit);
7  Port ( A,B : in STD_LOGIC_VECTOR (nb-1 downto 0);
8         clk,clr : in STD_LOGIC;
9         Sum : out STD_LOGIC_VECTOR (nb downto 0));
10 end Adder;
11
12 architecture Behavioral of Adder is
13 signal RA, RB: STD_LOGIC_VECTOR (nb-1 downto 0);
14 signal p,g: STD_LOGIC_VECTOR (nb downto 0);
15 signal c: STD_LOGIC_VECTOR (nb+1 downto 0);
16 begin
17     process(clk,clr)
18     begin
19     if(clr='1')then
20         RA<=(others=>'0'); RB<=(others=>'0');
21     elsif(falling_edge(clk)) then
22         RA<=A; RB<=B;
23     end if;
24     end process;
25 c(0)<='0';
26 p<=(RA(nb-1)xor RB(nb-1))& (RA xor RB);
27 g<=(RA(nb-1)and RB(nb-1))& (RA and RB);
28 c(nb+1 downto 1)<= g or (p and c(nb downto 0));
29 Sum<=p xor c(nb downto 0);
30 end Behavioral;
```

Subtracter

Il Subtracter presenta una struttura analoga all'Adder. È stato anch'esso implementato come un ripple carry adder e include due registri per salvare i segnali in ingresso comprendenti clock e clear. Poiché i numeri sono in complemento a due, per effettuare la sottrazione bisogna ricondursi a una somma: è necessario dunque invertire tutti i bit del secondo operando e sommare '1'. In particolare l'uno viene dato in ingresso al circuito come riporto iniziale.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  library work;
4  use work.costanti.all;
5  entity Subtracter is
6  generic (nb:integer:=nbit);
7  Port ( A,B : in STD_LOGIC_VECTOR (nb-1 downto 0);
8         clk,clr : in STD_LOGIC;
9         Sub : out STD_LOGIC_VECTOR (nb downto 0));
10 end Subtracter;
11 architecture Behavioral of Subtracter is
12 signal RA, RB: STD_LOGIC_VECTOR (nb-1 downto 0);
13 signal p,g: STD_LOGIC_VECTOR (nb downto 0);
14 signal c: STD_LOGIC_VECTOR (nb+1 downto 0);
15 begin
16     process(clk,clr)
17     begin
18     if(clr='1')then
19     RA<=(others=>'0'); RB<=(others=>'0');
20     elsif(falling_edge(clk)) then
21     RA<=A; RB<=B;
22     end if;
23     end process;
24 c(0)<='1';
25 p<=(RA(nb-1)xor not RB(nb-1))& (RA xor not RB);
26 g<=(RA(nb-1)and not RB(nb-1))& (RA and not RB);
27 c(nb+1 downto 1)<= g or (p and c(nb downto 0));
28 Sub<=p xor c(nb downto 0);
29 end Behavioral;
```

Circuit

Il circuito generale utilizza come component l'Adder e il Subtracter precedentemente descritti e dei segnali ausiliari che servono per calcolare i possibili risultati delle operazioni. È importante notare che il risultato del primo livello di operazioni aritmetiche restituisce un segnale con un bit in più. Di conseguenza, poiché il circuito complessivo è provvisto di due livelli, restituirà un risultato a *nb+2* bit. Per soddisfare questo requisito, è stato impiegato il costrutto generic map in fase di istanziazione che ci consente di generalizzare il numero di bit dei segnali.

La struttura è organizzata in modo tale da eseguire in contemporanea l'operazione tra *A* e *B* e quella tra *C* e *D*, generando due risultati parziali. Al secondo livello, questi ultimi vengono combinati per generare l'uscita del circuito.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  library work;
4  use work.costanti.all;
5
6  entity Circuit is
7  generic (nb:integer:=nbit);
8  Port ( A : in STD_LOGIC_VECTOR (nb-1 downto 0);
9         B : in STD_LOGIC_VECTOR (nb-1 downto 0);
10        C : in STD_LOGIC_VECTOR (nb-1 downto 0);
11        D : in STD_LOGIC_VECTOR (nb-1 downto 0);
12        Contr : in STD_LOGIC_VECTOR (1 downto 0);
13        clk : in STD_LOGIC;
14        clr : in STD_LOGIC;
15        Ris : out STD_LOGIC_VECTOR (nb+1 downto 0));
16 end Circuit;
17
18 architecture Behavioral of Circuit is
19 component Adder is
20     generic (nb:integer);
21     Port ( A,B : in STD_LOGIC_VECTOR (nb-1 downto 0);
22            clk,clr : in STD_LOGIC;
23            Sum : out STD_LOGIC_VECTOR (nb downto 0));
24 end component;
25
26 component Subtracter is
27     generic (nb:integer);
28     Port ( A,B : in STD_LOGIC_VECTOR (nb-1 downto 0);
29            clk,clr : in STD_LOGIC;
30            Sub : out STD_LOGIC_VECTOR (nb downto 0));
31 end component;
```

Nell' architecture, oltre ai segnali di supporto, sono stati istanziati tutti i possibili sotto-circuiti, ognuno dei quali calcola una delle quattro operazioni aritmetiche.

L'operazione corretta viene poi selezionata in base al valore del segnale *Contr* nello statement when-else e salvata nel segnale *RegRis*. A livello circuitale, questa operazione corrisponde a selezionare uno dei quattro ingressi all'interno del multiplexer.

A valle del circuito è presente un ulteriore registro, anch'esso dotato dei segnali *clk* e *clr,* in cui salvare l'uscita complessiva.

```
33  signal AB1,CD1,AB2,CD2,AB3,CD3,AB4,CD4: STD_LOGIC_VECTOR (nb downto 0);
34  signal ORis1,ORis2,ORis3,ORis4: STD_LOGIC_VECTOR (nb+1 downto 0);
35  signal RegRis: STD_LOGIC_VECTOR(nb+1 downto 0);
36  begin
37
38  Add1: Adder generic map(nb) port map(A,B,clk,clr,AB1);
39  Add2: Adder generic map(nb) port map(C,D,clk,clr,CD1);
40  Add3: Adder generic map(nb+1) port map(AB1,CD1,clk,clr,ORis1);
41
42  Sub1: Subtracter generic map(nb) port map(A,B,clk,clr,AB2);
43  Sub2: Subtracter generic map(nb) port map(C,D,clk,clr,CD2);
44  Add4: Adder generic map(nb+1) port map(AB2,CD2,clk,clr,ORis2);
45
46  Add5: Adder generic map(nb) port map(A,B,clk,clr,AB3);
47  Sub3: Subtracter generic map(nb) port map(C,D,clk,clr,CD3);
48  Sub4: Subtracter generic map(nb+1) port map(AB3,CD3,clk,clr,ORis3);
49
50  Sub5: Subtracter generic map(nb) port map(A,B,clk,clr,AB4);
51  Add6: Adder generic map(nb) port map(C,D,clk,clr,CD4);
52  Sub6: Subtracter generic map(nb+1) port map(AB4,CD4,clk,clr,ORis4);
53
54       RegRis<=ORis1 when Contr="00" else
55               ORis2 when Contr="01" else
56               ORis3 when Contr="10" else
57               ORis4 when Contr="11" else
58               (others=>'X');
59      process(clk,clr)
60      begin
61       if(clr='1') then
62           Ris<=(others=>'0');
63       elsif(falling_edge(clk))then
64           Ris<=RegRis;
65       end if;
66       end process;
67
68  end Behavioral;
```

Test Bench

Nel test bench è stato istanziato il circuito a cui sono stati passati i segnali di input e output. È stata inoltre definita la costante *Tclk,* una costante di tipo *time* che definisce la durata del periodo di clock.

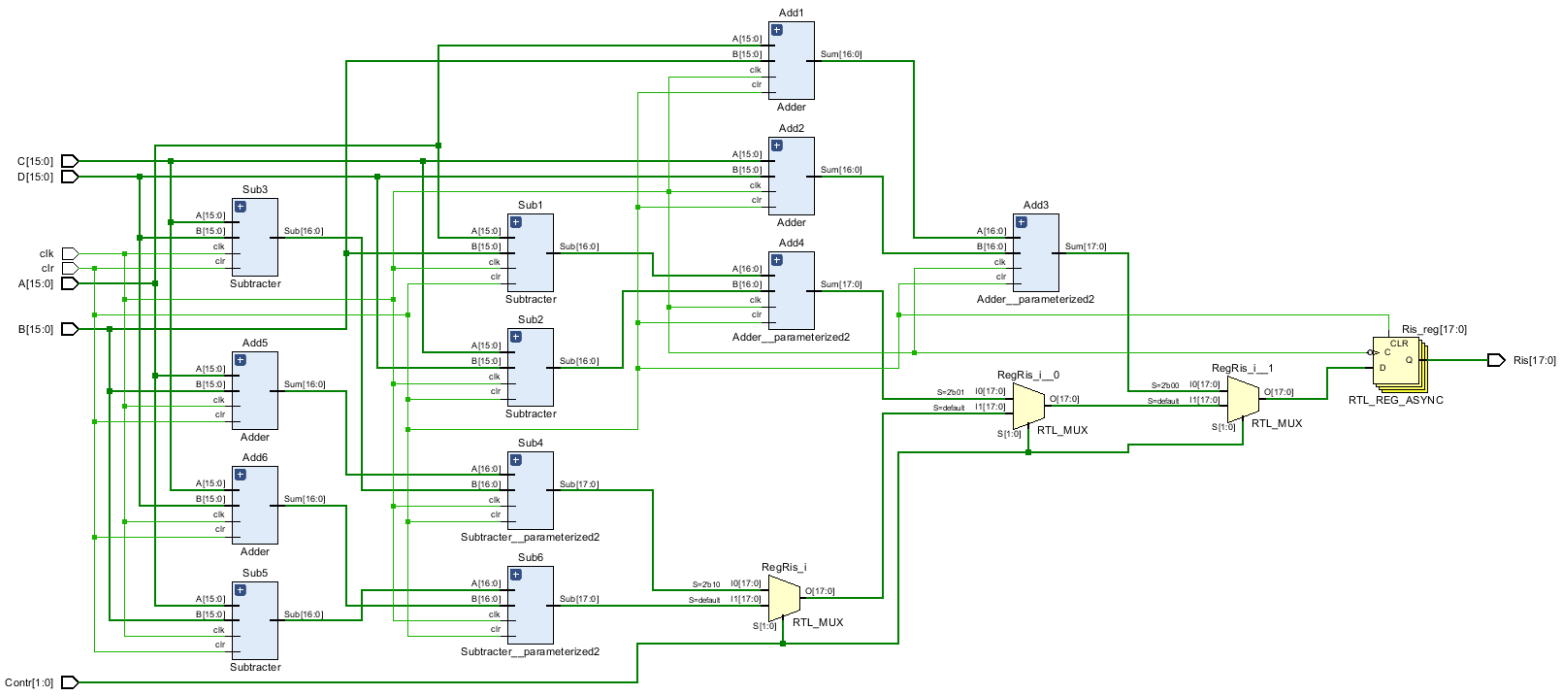
Nel primo process viene scandito l'andamento del clock mentre nel secondo process è presente un'attesa iniziale di 100ns per il global reset, periodo nel quale tutti i registri vengono inizializzati.

Nel codice è presente un doppio ciclo di for tramite il quale vengono generati dei valori di prova dei segnali *IA IB IC ID.* Le combinazioni di *IContr* testate in questo esempio sono "00" e "10" e, per ogni ciclo del for esterno, il segnale *Iclr* cambia il suo stato per testare la consistenza dei registri.

```
1  library IEEE;
2  use IEEE.STD_LOGIC_1164.ALL;
3  use IEEE.STD_LOGIC_arith.ALL;
4  library work;
5  use work.costanti.all;
6
7  entity SimCircuit is
8  generic (nb:integer:=nbit);
9  end SimCircuit;
10 architecture Behavioral of SimCircuit is
11 component Circuit is
12 Port ( A : in STD_LOGIC_VECTOR (nb-1 downto 0);
13         B : in STD_LOGIC_VECTOR (nb-1 downto 0);
14         C : in STD_LOGIC_VECTOR (nb-1 downto 0);
15         D : in STD_LOGIC_VECTOR (nb-1 downto 0);
16         Contr : in STD_LOGIC_VECTOR (1 downto 0);
17         clk : in STD_LOGIC;
18         clr : in STD_LOGIC;
19         Ris : out STD_LOGIC_VECTOR (nb+1 downto 0));
20
21 end component;
22 signal IA, IB,IC,ID: STD_LOGIC_VECTOR (nb-1 downto 0);
23 signal Iclk, Iclr: STD_LOGIC:='1';
24 signal IContr: STD_LOGIC_VECTOR (1 downto 0);
25 signal ORis: STD_LOGIC_VECTOR (nb+1 downto 0);
26 constant Tclk: time:=10 ns;
```

```
27  begin
28      circ: Circuit port map (IA,IB,IC,ID,IContr,Iclk,Iclr,ORis);
29      process
30      begin
31          wait for Tclk/2;
32          Iclk<=not Iclk;
33      end process;
34      process
35      begin
36          wait for 100ns; -- attesa per il global reset
37          wait until falling_edge(Iclk);
38          wait for Tclk;
39          IContr<="00";
40          for va in -(2**(4)) to (2**(4)-1) loop
41              IA<=conv_std_logic_vector(va,nb);
42              IC<=conv_std_logic_vector(va,nb);
43              Iclr<=not Iclr;
44              for vb in -(2**(4)) to (2**(4)-1) loop
45                  IB<=conv_std_logic_vector(vb,nb);
46                  ID<=conv_std_logic_vector(vb,nb);
47                  wait for Tclk;
48              end loop;
49              IContr<="10";
50          end loop;
51      end process;
52  end Behavioral;
```

Lo schema iniziale del circuito è il seguente ed è uguale per entrambe le versioni a *nb=8* e *nb=16,* ad eccezione del numero di bit dei segnali di input e output:



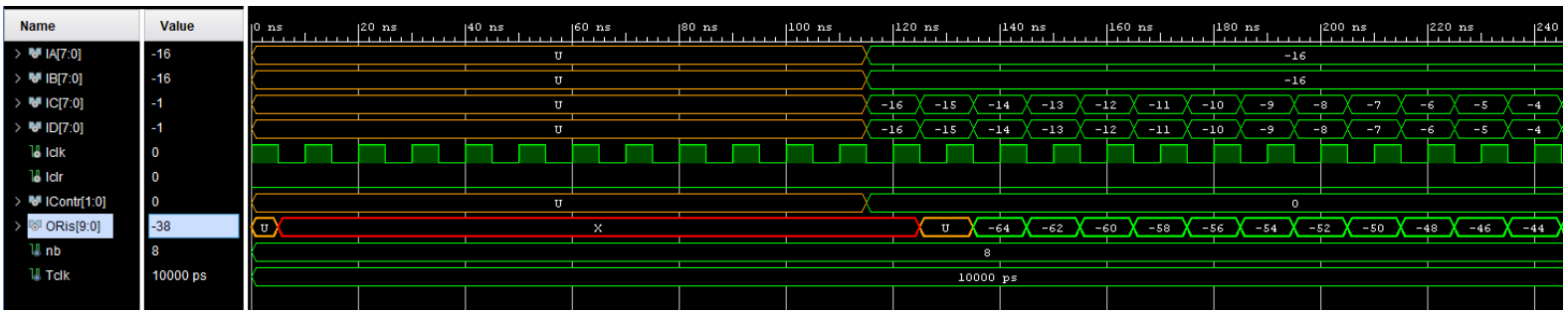
Simulazione Behavioral

In questa prima parte si può notare che per i primi 100ns più un ulteriore ciclo di clock il circuito non lavora perché è in fase di global reset. Successivamente a questa fase il circuito calcola la prima operazione in quanto *IContr* è in configurazione “00”.

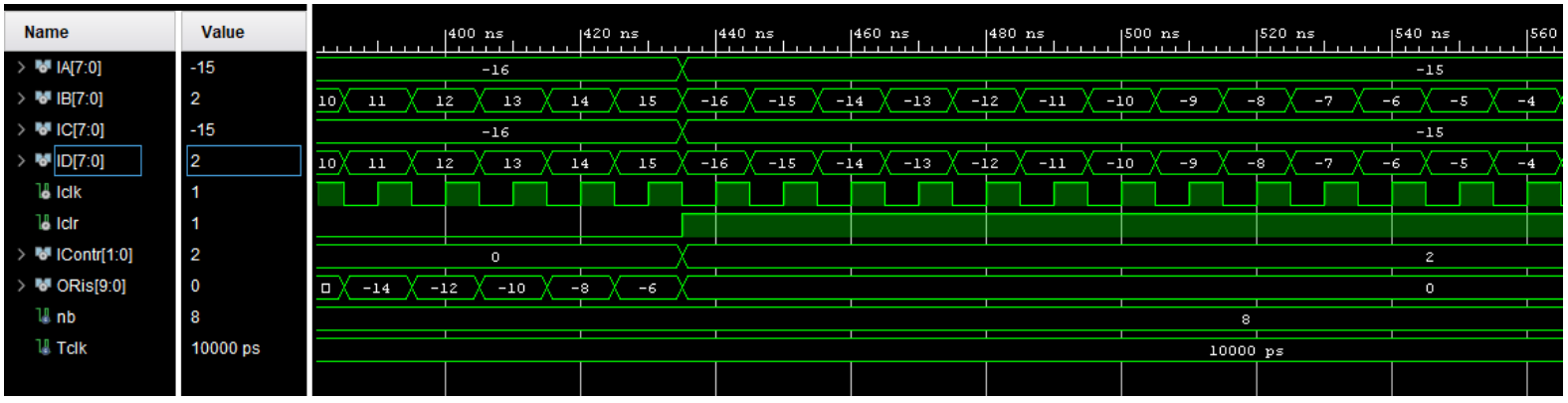
Il primo risultato corretto è visibile dopo due cicli di clock per rispettare il requisito di una latenza di due periodi. La latenza indica dopo quanto tempo viene prodotto il primo risultato utile a partire dalla ricezione degli ingressi. In termini pratici, la latenza corrisponde al numero di livelli di registri meno uno. Infatti l’uscita risulta non inizializzata fino al primo fronte di discesa, poi diventa non specificata a causa della configurazione ignota di *IContr.*

Dal primo risultato in poi il circuito produce un output ad ogni ciclo di clock senza introdurre ritardi, definendo quindi un throughput pari ad 1.

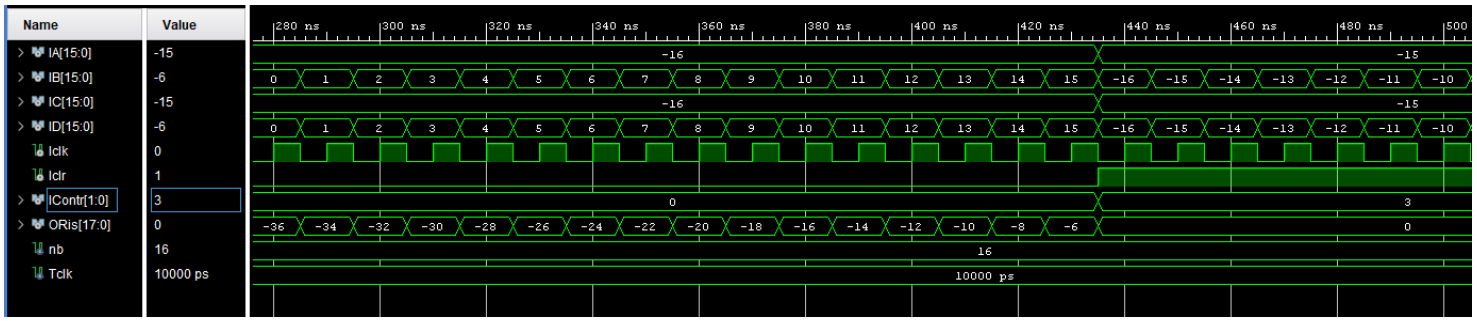
Esempi a 8 bit:



Di seguito un esempio del clear che diventa attivo e azzera *ORis:*



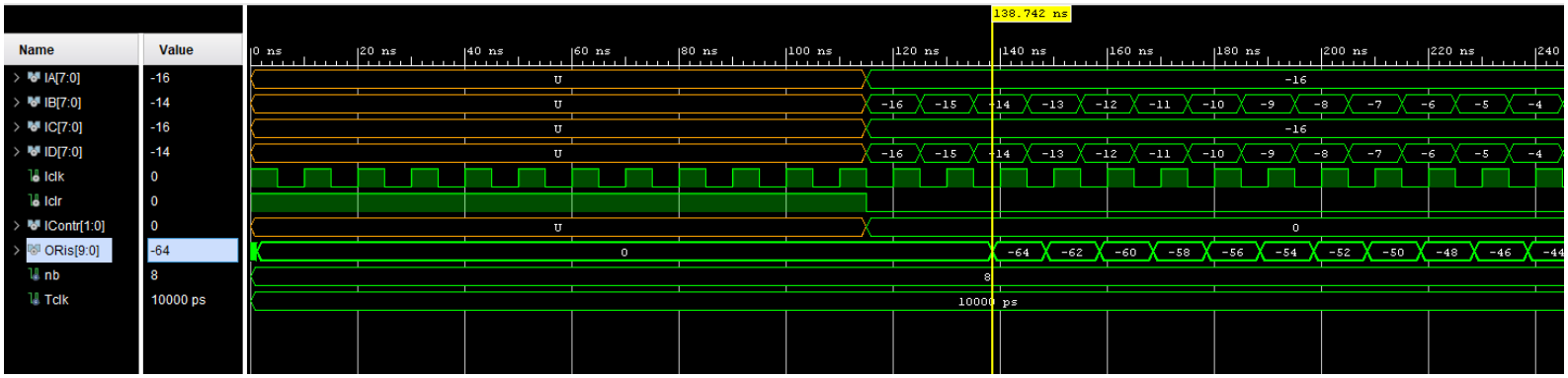
Esempio a 16 bit:



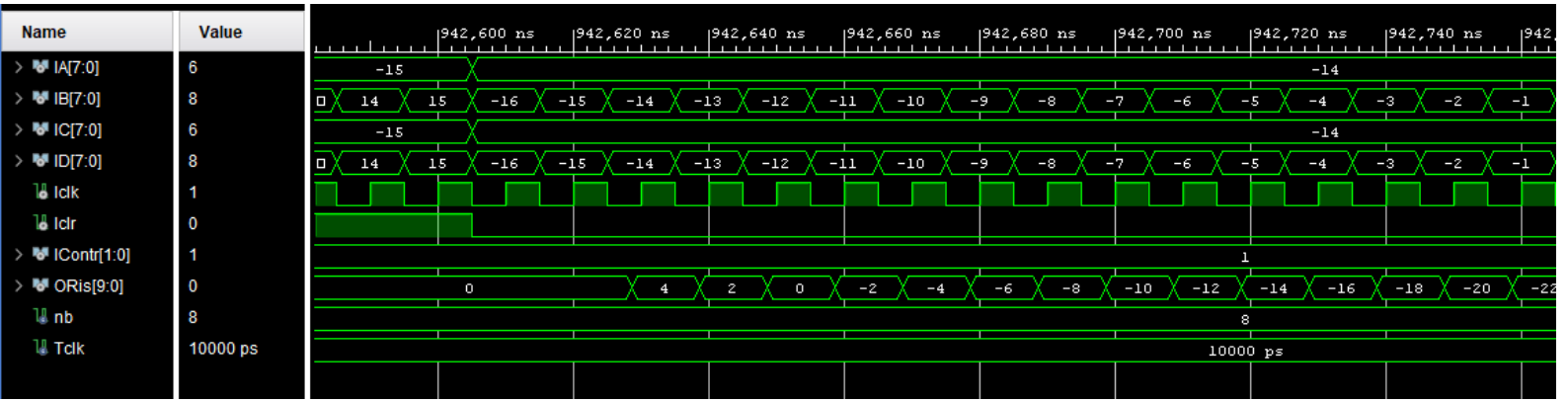
Simulazione Post-Sintesi

Nella foto seguente si può notare che dopo la sintesi è stato introdotto un ritardo pari a 3.742 ns dalla produzione del risultato rispetto alla simulazione behavioral.

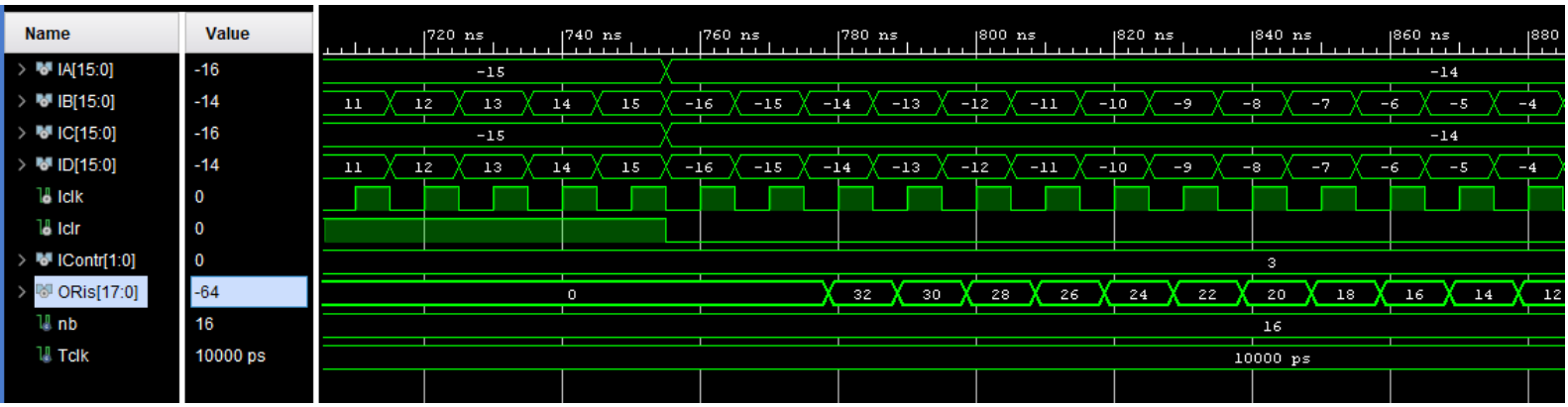
Esempi a 8 bit:



Di seguito un esempio con *IContr* pari a “01”:



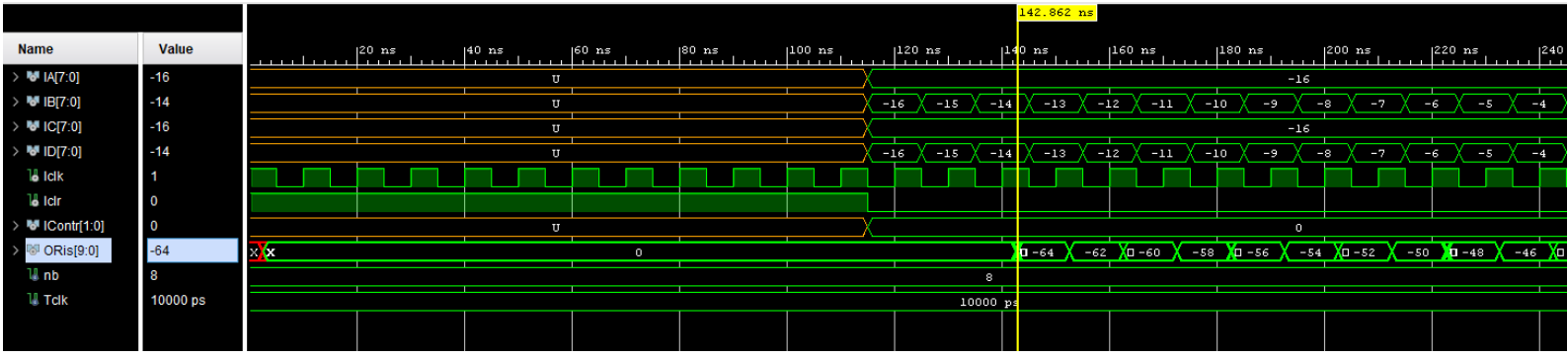
Esempio a 16 bit:



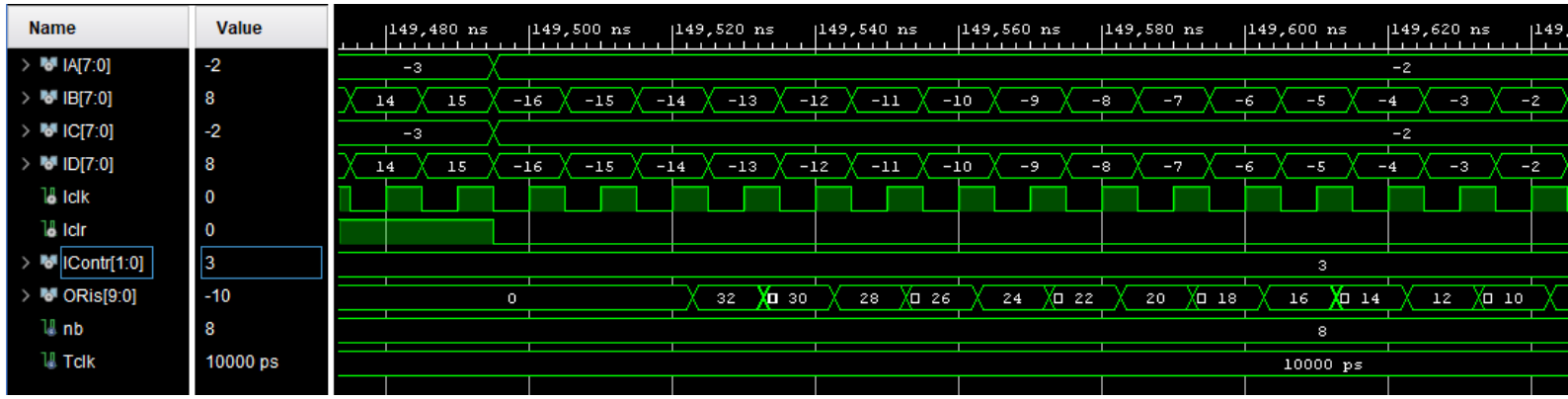
Simulazione Post-Implementazione

Dopo l'implementazione è possibile notare che il ritardo riferito alla generazione dei risultati è aumentato da 3.742 a 7.862 a causa dei collegamenti fisici creati durante l'implementazione.

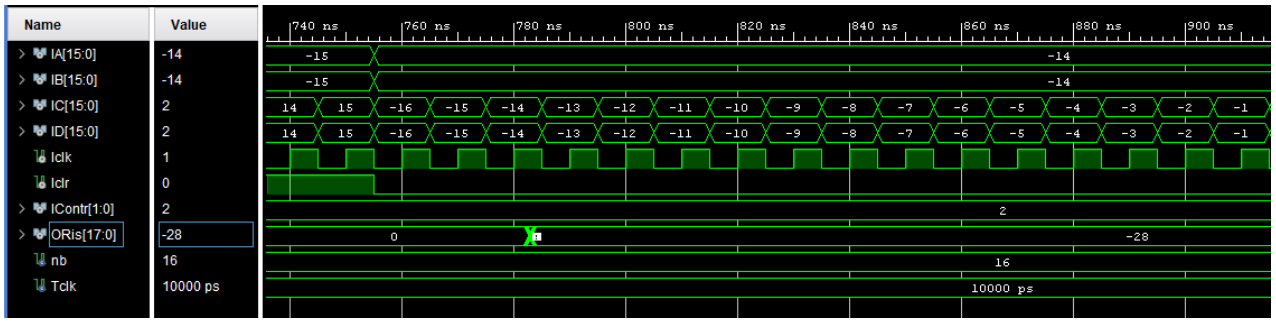
Esempi a 8 bit:



Nella foto seguente un esempio con *IC*ontr pari a "11":

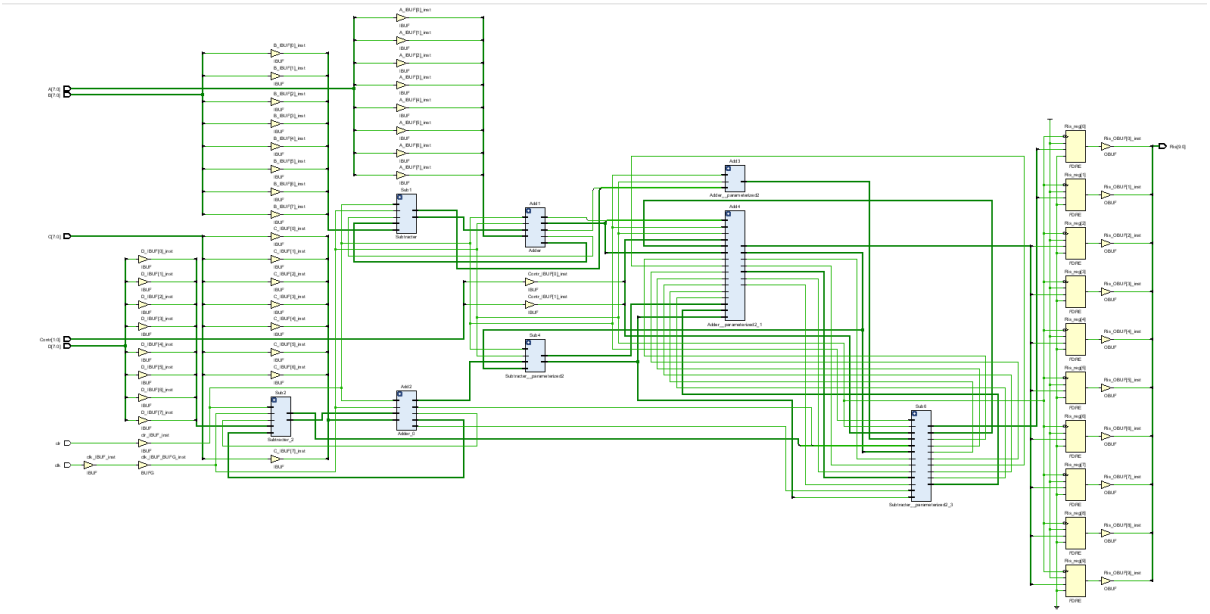


Esempio a 16 bit con IContr = “10”:

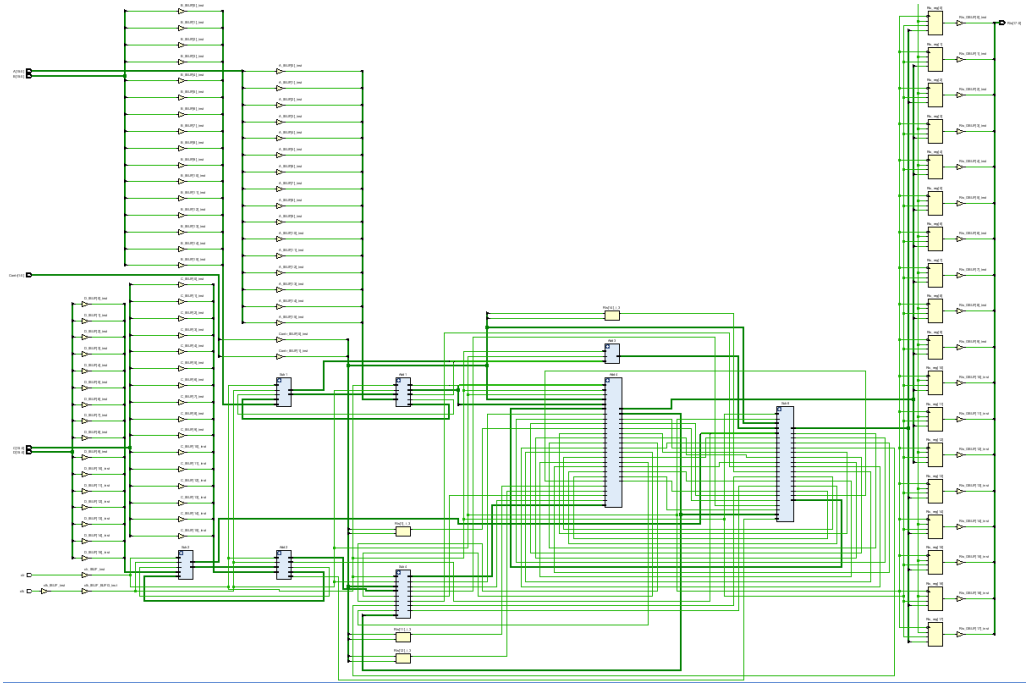


Sono qui riportate le schematic post-sintesi e post-implementazione ad 8 e 16 bit a confronto:

Schematic 8 bit:

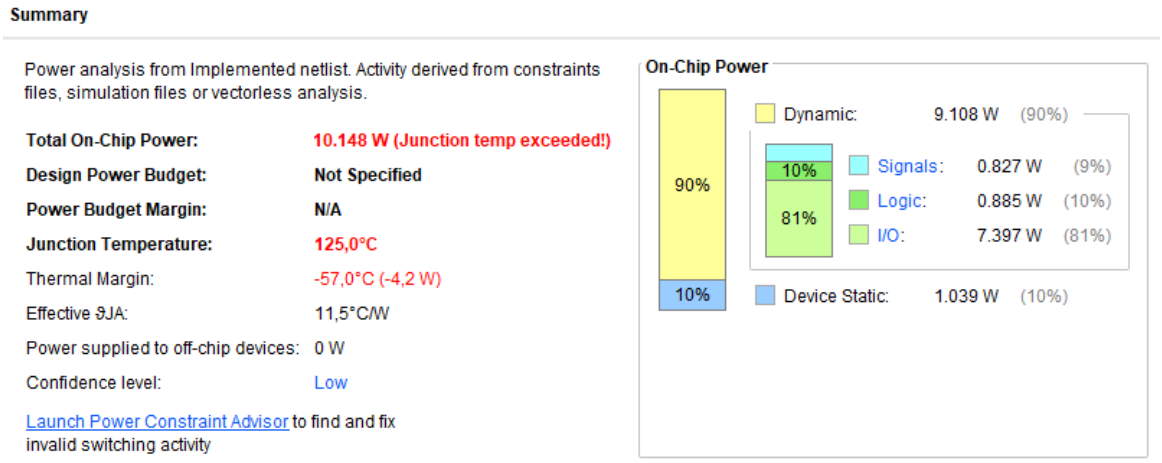


Schematic 16 bit:



Report Power

Successivamente all'implementazione ci siamo preoccupati di valutare il circuito con *nb=8* da un punto di vista energetico. Infatti dopo l'implementazione il circuito presenta una temperatura di giunzione troppo elevata (125°C) e una potenza dissipata pari a 10.148 W, indice di una frequenza di funzionamento eccessiva che causerebbe dei malfunzionamenti.



Per risolvere questa problematica è stata introdotta una *Timing Constraint* che permette di regolare il periodo di clock, in questo caso di abbasarlo, per far diminuire la frequenza di lavoro e quindi garantire il corretto funzionamento del circuito.

Il nuovo vincolo sul periodo scelto da noi è di 5 ns:

```
1 create_clock -period 5.000 -name myClock -waveform {0.000 2.500} [get_nets clk]
```

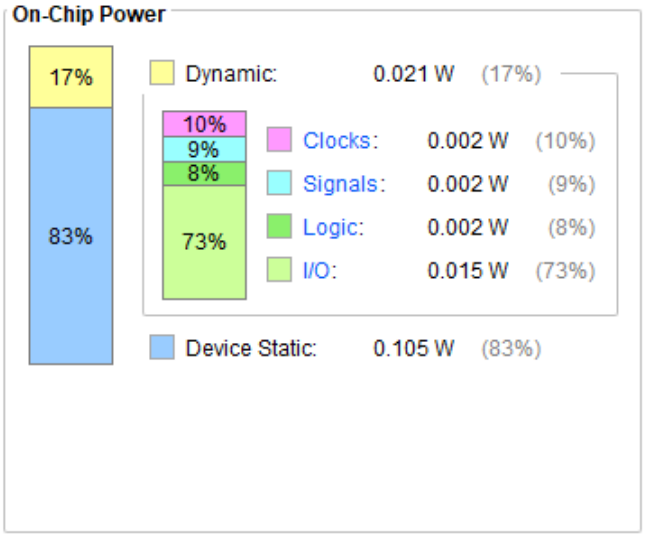
Rieseguendo l'implementazione dopo l'introduzione della constraint la potenza dissipata assume un valore coerente al vincolo di tempo (0.126 W), così come la temperatura di giunzione (26.4°C).

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

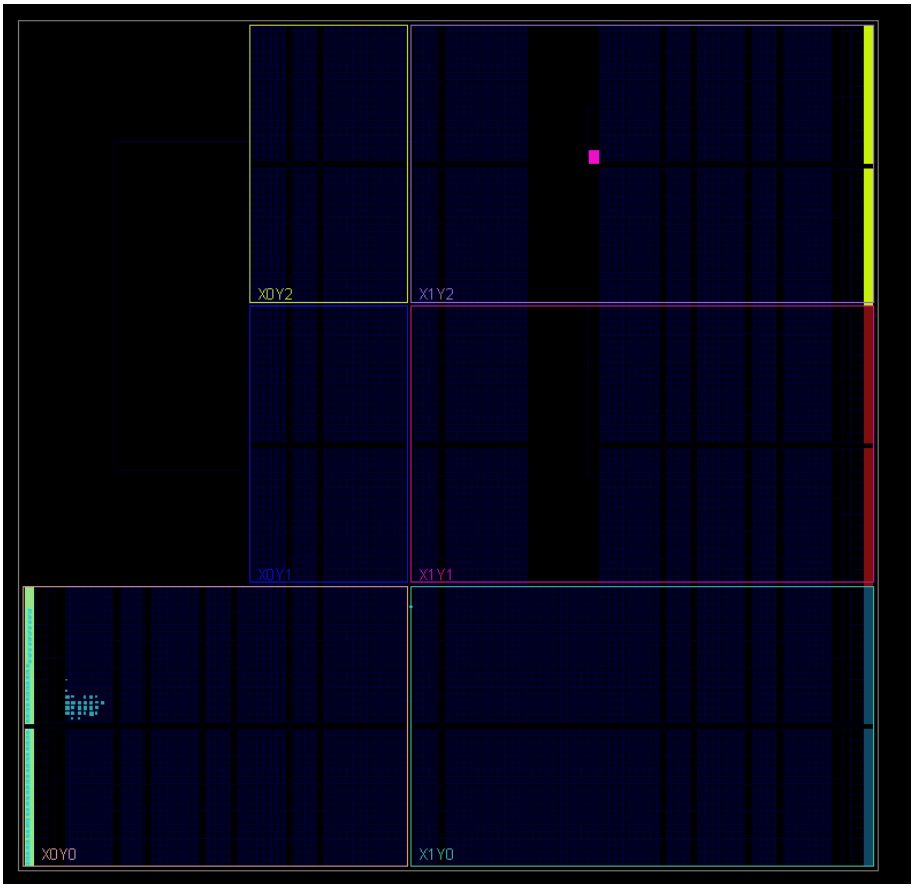
Total On-Chip Power:	**0.126 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26,4°C**
Thermal Margin:	58,6°C (4,9 W)
Effective ϑJA:	11,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity



La nuova potenza dissipata comprende una bassa percentuale di potenza dinamica (17%), come è possibile notare anche dallo schema del device: la parte evidenziata nella foto è l'unica a compiere lavoro e quindi a dissipare potenza dinamica. Il resto del device, non compiendo nessuna operazione, dissipa potenza statica (83%).

Inoltre tra le componenti della potenza dinamica è comparso un 10% dissipato dal clock che prima del vincolo non era presente. Il clock infatti ha bisogno di risorse dedicate alla sua distribuzione poiché coinvolge tutti i registri.



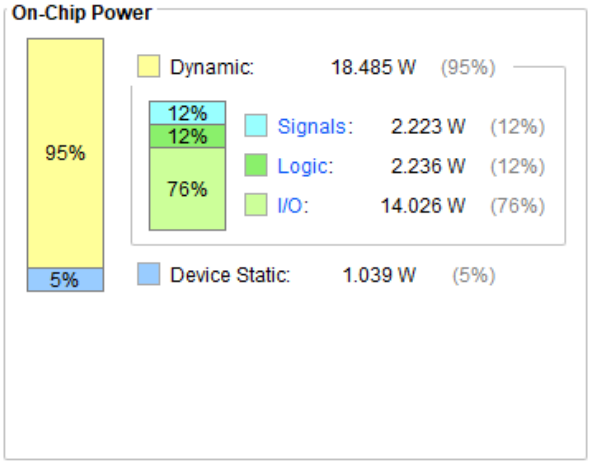
Una valutazione analoga è stata fatta per *nb=16*.

Notiamo che in questo caso la temperatura di giunzione è simile al caso *nb=8* (125°C) mentre la potenza dissipata è quasi raddoppiata (19.525W). Inoltre è aumentata la percentuale della potenza dinamica, da 90% a 95%.

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**19.525 W (Junction temp exceeded!)**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**125,0°C**
Thermal Margin:	-165,2°C (-13,6 W)
Effective ϑJA:	11,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity



In questo caso una constraint sul periodo pari a 5 ns non era sufficiente per far rispettare i vincoli, dunque il periodo di clock è stato incrementato a 7 ns rientrando cosi nei limiti.

```
1 create_clock -period 7.000 -name myClock -waveform {0.000 3.500} [get_nets clk]
```

È stata rieseguita l'implementazione successivamente all'introduzione della constraint e la potenza dissipata risulta coerente con il vincolo di tempo introdotto (0.138W), così come la temperatura di giunzione (26.6°C).

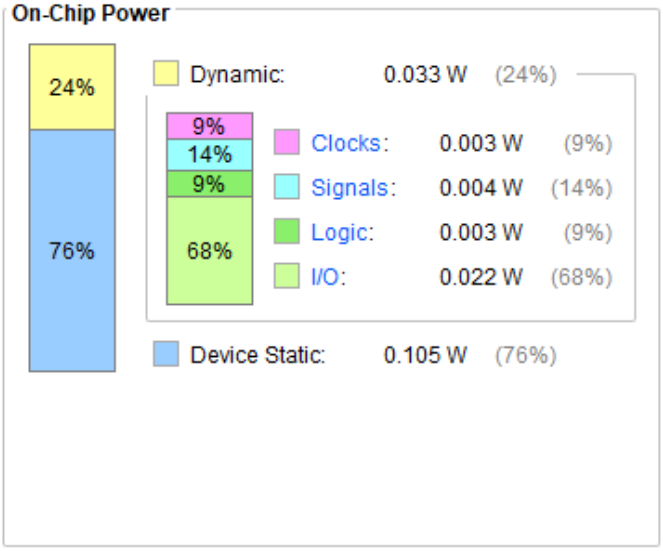
La potenza dissipata è composta dal 24% di potenza dinamica, che è aumentata, e dal 76% di potenza statica, che invece è diminuita. Anche in questo caso è presente una parte di potenza dissipata dal clock (9%).

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.138 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26,6°C**
Thermal Margin:	58,4°C (4,9 W)
Effective ϑJA:	11,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Low

Launch Power Constraint Advisor to find and fix invalid switching activity



Risorse occupate

Analizzando i report è possibile visualizzare le caratteristiche temporali e le risorse fisiche del circuito.

I seguenti risultati sono riferiti a un circuito con *nb=8:*

Tutti i vincoli di tempo del circuito sono soddisfatti e inoltre è presente un WNS (worst negative slack) di 0.189 ns, che ci dice di quanto possiamo ancora ridurre il periodo affinché i vincoli rimangano soddisfatti.

```
------------------------------------------------------------------------------
| Design Timing Summary
| ---------------------
------------------------------------------------------------------------------

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints
    -------      -------  ---------------------  -------------------
      0.189        0.000                      0                   44


All user specified timing constraints are met.


------------------------------------------------------------------------------
| Clock Summary
| -------------
------------------------------------------------------------------------------

Clock    Waveform(ns)         Period(ns)      Frequency(MHz)
-----    ------------         ----------      --------------
myClock  {0.000 2.500}        5.000           200.000
```

Per quanto riguarda le risorse fisiche impiegate notiamo un utilizzo di:

- 81 Look Up Table, pari allo 0.15%;
- 76 registri Flip-Flop, pari allo 0.07%;
- 46 IO buffer, pari al 23%.

Site Type	Used	Fixed	Available	Util%
Slice LUTs	81	0	53200	0.15
LUT as Logic	81	0	53200	0.15
LUT as Memory	0	0	17400	0.00
Slice Registers	76	0	106400	0.07
Register as Flip Flop	76	0	106400	0.07
Register as Latch	0	0	106400	0.00

Site Type	Used	Fixed	Available	Util%
Bonded IOB	46	0	200	23.00
IOB Master Pads	22			
IOB Slave Pads	23			

Questi invece i dati del circuito con *nb=16*:

Anche in questo caso tutti i vincoli di tempo sono soddisfatti e in particolare il WNS è pari a 0.306 ns.

```
------------------------------------------------------------------------------
| Design Timing Summary
| ---------------------
------------------------------------------------------------------------------

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints
    -------      -------  ---------------------  -------------------
      0.306        0.000                      0                   84


All user specified timing constraints are met.


------------------------------------------------------------------------------
| Clock Summary
| -------------
------------------------------------------------------------------------------

Clock    Waveform(ns)         Period(ns)      Frequency(MHz)
-----    ------------         ----------      --------------
myClock  {0.000 3.500}        7.000           142.857
```

Analizzando le risorse fisiche notiamo un forte incremento di queste:

- le LUT sono quasi triplicate (215 contro le 81), pari allo 0.40%;
- i registri (148 contro 76) sono quasi raddoppiati, pari allo 0.14%;
- gli IO buffer sono circa il doppio (86 contro i 46), pari al 43%.

Site Type	Used	Fixed	Available	Util%
Slice LUTs	215	0	53200	0.40
LUT as Logic	215	0	53200	0.40
LUT as Memory	0	0	17400	0.00
Slice Registers	148	0	106400	0.14
Register as Flip Flop	148	0	106400	0.14
Register as Latch	0	0	106400	0.00

Site Type	Used	Fixed	Available	Util%
Bonded IOB	86	0	200	43.00
IOB Master Pads	41			
IOB Slave Pads	43			