Secondo Progetto Elettronica Digitale

Carmelo Gugliotta Mat:213477

Anno Accademico: 2021/2022

Contents

1 Introduzione

Il pipelining è una tecnica di progettazione utilizzata per accelerare il funzionamento dei percorsi in data-processing digitali. In particolare permette di migliorare l'utilizzo delle risorse e aumentare il throughput funzionale, o meglio, massimizzare il ritmo con cui vengono prodotti i risultati elaborando istruzioni in maniera parallela. Il parallelismo è raggiunto, senza costi eccessivi, tramite l'utilizzo di Registri, che permettono d'introdurre step di elaborazione intermedia all'interno del circuito. Da ciò consegue che i circuiti Pipeline non producono risultati in maniera immediata, ma necessitano di un intervallo di tempo, detto LATENZA, per elaborare un output.

Un registro rappresenta un sistema di memoria non permanente, che acquisisce informazioni in forma binaria. Si tratta infatti, di una memoria volatile i cui dati si perderebbero qualora mancasse la corrente elettrica. Possono essere costruiti attraverso varie configurazioni circuitali con un prefissato numero n di flip-flop opportunamente collegati con il terminale di clock e di reset comuni. In questo modo i registri possono memorizzare più bit d'informazione.

2 Circuito a quattro operandi

2.1 Progettazione Struttura

Si vuole realizzare un circuito che, ricevuti in ingresso i quattro operandi A, B, C e D ad n *bit* in complemento a due, ed il segnale di controllo Contr a 2-bit, svolga le seguenti operazioni:

- $A + B + C + D$ se $Contr$= "00";
- $A - B + C - D$ se $Contr$="01";
- $A + B - C + D$ se $Contr$="10";
- $A - B - C - D$ se $Contr$="11";

Si descrive quindi, la struttura pipeline utilizzata per implementare il circuito:

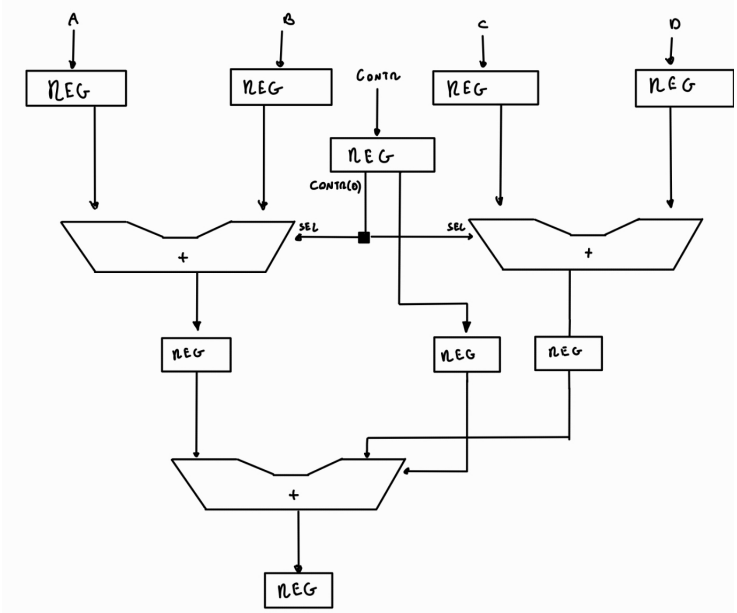


Figure 1: Circuito

Si osservi che, supponendo segnali ad **n bit** in ingresso, si elaborano in uscita segnali ad n+2 bit; inoltre una struttura così composta, garantisce una latenza di due colpi di clock e un throughput unitario, a partire dall'istante di tempo in cui viene elaborato il primo risultato.

2.2 Implementazione VHDL

2.2.1 Package

Generalmente è preferibile inserire tutte le costanti che definiranno il numero di bit in ingresso, e il numero di bit assegnato a ogni blocco, in unico file package, invece di copiare e incollare quest'ultimi ove necessario. In questo modo, se si vuole analizzare il comportamento del circuito al variare di un determinato parametro, è sufficiente aggiornare solo il file package.

```
1     package MyDef is
2         constant n:integer:=8;
3     end package;
```

Listing 1: MyDef

2.2.2 Modulo Ripple Carry Adder e Multiplexer

Andiamo a descrivere il primo componente:**Il Ripple Carry Adder** è un circuito puramente combinatorio. Quest'ultimo deve essere in grado di sommare 3 operandi, A e B, di n bit, e il riporto in ingresso c_{in} per generare in uscita il risultato S .
È possibile notare in Fig.1 che vi sono 3 Moduli di Ripple Carry Adder; i primi due sono adibiti, rispettivamente, alla somma di $A + / - B$ e $\overline{C}/C + / - D$, con Ingresso pari ad n bit e Uscita ad n+1 bit; l'operando C viene negato, in caso di necessità, nel circuito generale. Il terzo modulo di RCA è adibita alla somma $Ris_{AB} + Ris_{CD} + 0/1$, in base al bit di controllo più significativo, con Ingresso ad n+1 bit e uscita ad n+2 bit. Ecco perché definiamo due moduli diversi RCA_{AddSub} e $RCA_{AddSub2}$.

Un altro importantissimo componente, sebbene non dichiarato come entità è Il **Multiplexer**. Quest'ultimo è un dispositivo che può ricevere in ingresso 2^n segnali, e tramite n segnali di controllo, sintetizza un unico segnale in output tra quelli ricevuti in ingresso. Introdotto all'interno dell'RCA, tramite specifiche istruzioni, per permettere la selezione del segnale, diretto o negato, al variare del segnale di controllo $Contr$.

```
1  library IEEE;
2  library WORK;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use WORK.MyDef.ALL;
5
6  entity RCA_AddSub is
7      Port ( A : in STD_LOGIC_VECTOR (n-1 downto 0);
8             B : in STD_LOGIC_VECTOR (n-1 downto 0);
9             Sel : in STD_LOGIC ;
10            S : out STD_LOGIC_VECTOR (n downto 0));
11 end RCA_AddSub;
12 Architecture behav of RCA_AddSub is
13     signal IA,IB,nB,Curr_B:std_logic_Vector(n downto 0);
14     signal carry: STD_LOGIC_VECTOR(n+1 downto 0);
15     signal p,g: STD_LOGIC_VECTOR(n downto 0);
16     begin
17         IA<=A(n-1)&A;
```

```
18          IB<=B(n-1)&B;
19          nB<=not IB;
20
21          Curr_B<=IB when Sel='0'
22              else nB when Sel='1'
23              else (others=>'X');
24
25
26          p<=(IA xor Curr_B);
27          g<=(IA and Curr_B);
28          carry(0)<=Sel;
29          carry(n+1 downto 1)<=g or(p and carry(n downto 0));
30          S<=p xor carry(n downto 0);
31
32  end behav;
33
34  library IEEE;
35  library WORK;
36  use IEEE.STD_LOGIC_1164.ALL;
37  use WORK.MyDef.ALL;
38
39  entity RCA_AddSub_2 is
40      Port ( A : in STD_LOGIC_VECTOR (n downto 0);
41             B : in STD_LOGIC_VECTOR (n downto 0);
42             Sel : in STD_LOGIC;
43             S : out STD_LOGIC_VECTOR (n+1 downto 0));
44  end RCA_AddSub_2;
45  Architecture behav of RCA_AddSub_2 is
46      signal IA,IB:std_logic_Vector(n+1 downto 0);
47      signal carry: STD_LOGIC_VECTOR(n+2 downto 0);
48      signal p,g: STD_LOGIC_VECTOR(n+1 downto 0);
49      begin
50          IA<=A(n)&A;
51          IB<=B(n)&B;
52
53
54          p<=(IA xor IB);
55          g<=(IA and IB);
56          carry(0)<=Sel;
57          carry(n+2 downto 1)<=g or(p and carry(n+1 downto 0));
58          S<=p xor carry(n+1 downto 0);
59
60  end behav;
```

Listing 2: Modulo Ripple Carry Adder

2.2.3 Modulo Circuito

2.2.4 Implementazione Circuito e Schema Logico

Il circuito visibile in Fig.1 non solo necessita di 3 Ripple Carry Adder, ma anche di Registri. Il **Registro** è un particolare componente, composto da più Flip Flop, ognuno dei quali presenta: Un bit d'ingresso, Un bit di uscita, Clock e Clear.
Il segnale di Clear è un segnale asincrono, in quanto non appena esso è alto, l'uscita cambia immediatamente.
Il segnale di Clock consente di scandire con precisione gli intervalli di tempo in cui ogni elemento deve tenere conto del segnale in input, permettendo così di gestire la sincronizzazione e la latenza del circuito. Difatti il segnale di clock, simula, il concetto del "prima" e "dopo" ed è un segnale periodico nel tempo, generalmente un'onda quadra.

Sono stati utilizzati 9 registri:

- 4 per gli ingressi A,B,C,D;
- 2 per memorizzare e sincronizzare il segnale di Contr:

  -liv0 Il primo per sincronizzare Contr(0) con i primi due moduli di RCA;

  -liv1 Il secondo per sincronizzare Contr(1) con il terzo modulo di RCA;
- 2 per i Risultati intermedi;
- 1 per il Risultato finale;

```
1  library IEEE;
2  library WORK;
3  use IEEE.STD_LOGIC_1164.ALL;
4  use WORK.MyDef.all;
5
6
7  entity MyCircuit is
8      Port ( A : in STD_LOGIC_VECTOR (n-1 downto 0);
9             B : in STD_LOGIC_VECTOR (n-1 downto 0);
10            C : in STD_LOGIC_VECTOR (n-1 downto 0);
11            D : in STD_LOGIC_VECTOR (n-1 downto 0);
12            clk: in STD_LOGIC;
13            clr: in STD_LOGIC;
14            Contr : in STD_LOGIC_VECTOR (1 downto 0);
15            S : out STD_LOGIC_VECTOR (n+1 downto 0)); ---uscita in complemento a
      due
16 end MyCircuit;
17
18 architecture Struct_v1 of MyCircuit is
19
20     component RCA_AddSub is
21         port( A : in STD_LOGIC_VECTOR (n-1 downto 0);
22               B : in STD_LOGIC_VECTOR (n-1 downto 0);
23               Sel : in STD_LOGIC;
24               S : out STD_LOGIC_VECTOR (n downto 0));
25     end component;
26
27     component RCA_AddSub_2 is
28         port( A : in STD_LOGIC_VECTOR (n downto 0);
29               B : in STD_LOGIC_VECTOR (n downto 0);
```

```
30              Sel : in STD_LOGIC;
31              S : out STD_LOGIC_VECTOR (n+1 downto 0));
32    end component;
33
34    signal IA,IB,IC,TrueC,ID:std_logic_vector(n-1 downto 0);
35    signal Ris_AB,Ris_CD:STD_LOGIC_VECTOR (n downto 0);
36
37    signal IContr_0,IContr_1: STD_LOGIC_VECTOR( 1 downto 0);
38
39    signal IRis_AB,IRis_CD:STD_LOGIC_VECTOR (n downto 0);
40    signal ORis,ORis_F:STD_LOGIC_VECTOR (n+1 downto 0);
41 begin
42
43    process(clk, clr)
44        begin
45        if clr='1' then
46            IA<=(others=>'0');
47            IB<=(others=>'0');
48            IC<=(others=>'0');
49            ID<=(others=>'0');
50            IRis_AB<=(others=>'0');
51            IRis_CD<=(others=>'0');
52            ORis_F<=(others=>'0');
53            IContr_0<=(others=>'0');
54            IContr_1<=(others=>'0');
55        elsif falling_edge(clk) then
56            IA<= A;
57            IB<= B;
58            IC<= C;
59            ID<= D;
60            IRis_AB<=Ris_AB;
61            IRis_CD<=Ris_CD;
62            ORis_F<=ORis;
63            IContr_1<=IContr_0;
64            IContr_0<=Contr;
65        end if;
66    end process;
67
68    Adder_liv0_1: RCA_AddSub port map(IA,IB,IContr_0(0),Ris_AB);
69
70    TrueC<=IC when IContr_0(1)= '0' else
71            not IC when IContr_0(1)='1' else (others=>'X');
72
73    Adder_liv0_2: RCA_AddSub port map(TrueC,ID,IContr_0(0),Ris_CD);
74    Adder_liv1: RCA_AddSub_2 port map(IRis_AB,IRis_CD,IContr_1(1),ORis);
75    S<=ORis_F;
76 end Struct_v1;
```

Listing 3: Modulo Carry Select Adder

Lo schema logico corrispondente, definendo n pari a 8 è il seguente:

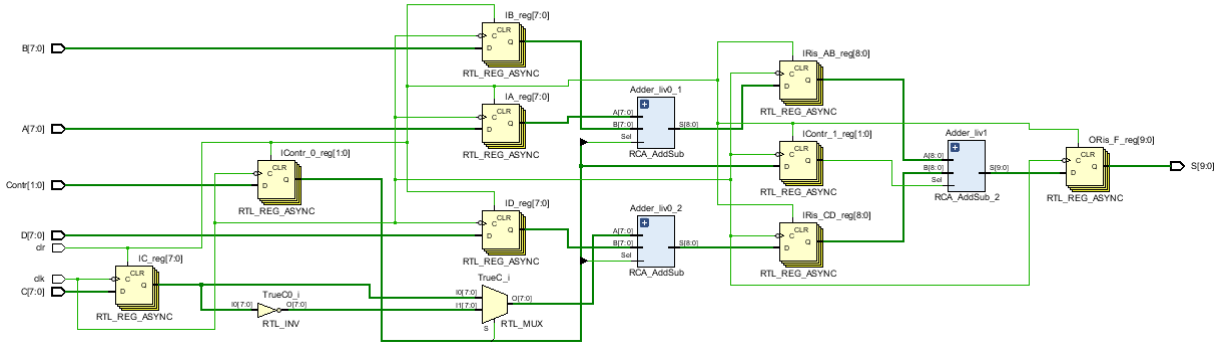


Figure 2: Schema Logico *n=8*

Se definiamo n pari a 16 il design risulta essere il seguente:

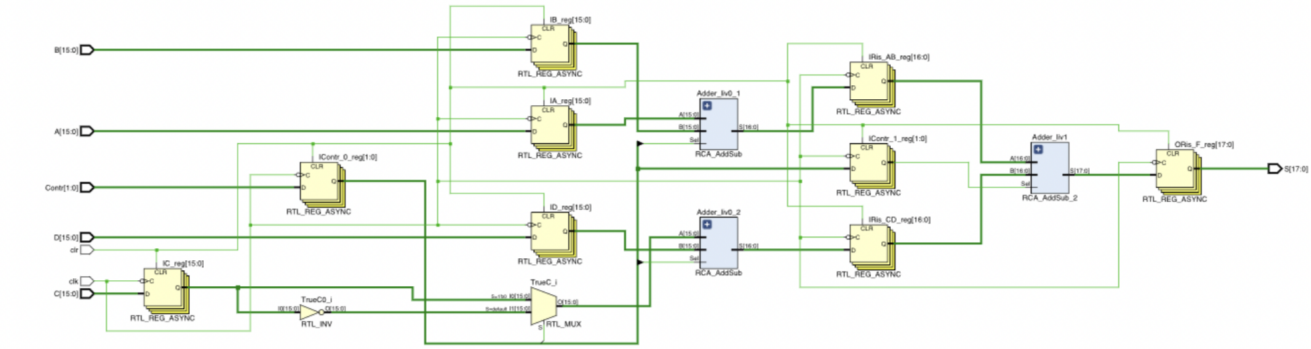


Figure 3: Schema Logico *n=16*

2.3 Simulazione Comportamentale

Modelliamo ora gli aspetti salienti dell'ambiente in cui il circuito si troverà a operare, definendo un TestBench esaustivo, che permette di specificare gli stimoli in ingresso e di collezionare risultati, per la simulazione del circuito e per la verifica della correttezza delle operazioni.

Si noti che i segnali A, B, C e D che immettiamo in ingresso hanno valori appartenenti all'intervallo $[-2^{n-1}, 2^{n-1} - 1]$.

```
1 library IEEE;
2 library WORK;
3 use IEEE.STD_LOGIC_1164.ALL;
4 use IEEE.STD_LOGIC_ARITH.ALL;
5 use WORK.MyDef.all;
6
7 entity SimBench_v1 is
8 end SimBench_v1;
9
```

```
10 architecture Behavioral of SimBench_v1 is
11 component MyCircuit is
12     Port ( A : in STD_LOGIC_VECTOR (n-1 downto 0);
13            B : in STD_LOGIC_VECTOR (n-1 downto 0);
14            C : in STD_LOGIC_VECTOR (n-1 downto 0);
15            D : in STD_LOGIC_VECTOR (n-1 downto 0);
16            clk: in STD_LOGIC;
17            clr: in STD_LOGIC;
18            Contr : in STD_LOGIC_VECTOR (1 downto 0);
19            S : out STD_LOGIC_VECTOR (n+1 downto 0));
20 end component;
21
22 signal IA,IB,IC,ID: std_logic_vector (n-1 downto 0);
23 signal Icontr: STD_LOGIC_VECTOR (1 downto 0);
24 signal ORis: std_logic_vector (n+1 downto 0);
25 signal Iclk: std_logic:='0';
26 signal Iclear: STD_LOGIC;
27 constant Tclk: Time:=10ns;
28
29 begin
30     CIRC: MyCircuit port map(IA,IB,IC,ID,Iclk,IClear,Icontr,ORis);
31
32     process
33         begin
34         wait for Tclk/2;
35         Iclk<= not Iclk;
36     end process;
37
38     process
39         begin
40
41         Iclear<='0';
42         wait for Tclk/2;
43         for va in -(2**( n -1) ) to (2**( n -1) -1) loop
44             IA <= conv_std_logic_vector ( va , n ) ;
45             for vb in -(2**( n -1) ) to (2**( n -1) -1) loop
46                 IB <= conv_std_logic_vector ( vb , n ) ;
47                 for vc in -(2**(n -1) ) to (2**( n -1) -1) loop
48                     IC <= conv_std_logic_vector ( vc , n ) ;
49                     for vd in -(2**( n -1) ) to (2**( n -1) -1) loop
50                         ID <= conv_std_logic_vector ( vd , n ) ;
51                         for sel1 in std_logic range '0' to '1' loop
52                             Icontr(1)<=sel1;
53                             for sel0 in std_logic range '0' to '1' loop
54                             Icontr(0)<=sel0;
55                             wait for Tclk;
56                             end loop;
57                         end loop;
58                     end loop;
59                 end loop;
60             end loop ;
61         end loop;
62     end process;
63 end Behavioral;
```

Listing 4: Test Bench

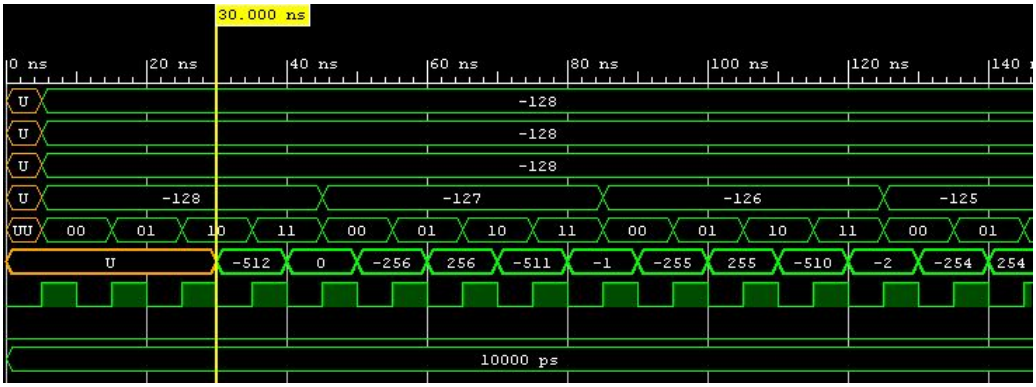


Figure 4: Grafico Simulazione Comportamentale n=8bit

Dai grafici è possibile osservare, che ogni ingresso ricevuto viene elaborato con **ritardo nullo**. Questo non è possibile nella realtà, e avviene in quanto si sta eseguendo una simulazione comportamentale.

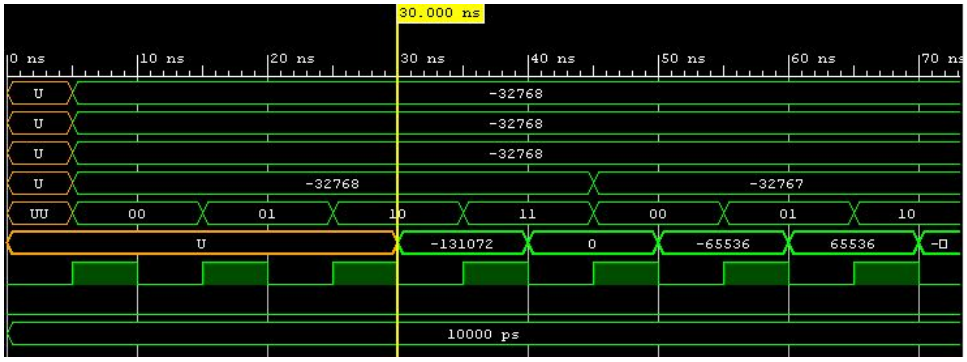


Figure 5: Grafico Simulazione Comportamentale n=16bit

Per effettuare un' analisi circuitale più approfondita, è necessario effettuare la **sintesi del circuito** e successivamente la **simulazione con timing**.

2.4 Sintesi Circuitale e Implementazione

2.4.1 Sintesi con n posto a 8bit

Le board sulle quali vengono sviluppati circuiti di questo tipo, sono dispositivi elettronici formati da un circuito integrato le cui funzionalità logiche di elaborazione sono appositamente programmabili e modificabili tramite opportuni linguaggi di descrizione hardware, e vengono detti **Field Programmable Gate Aways** (solitamente abbreviato in **FPGA**). La board in utilizzo è la ZedBoard e il risultato della sintesi è il seguente:

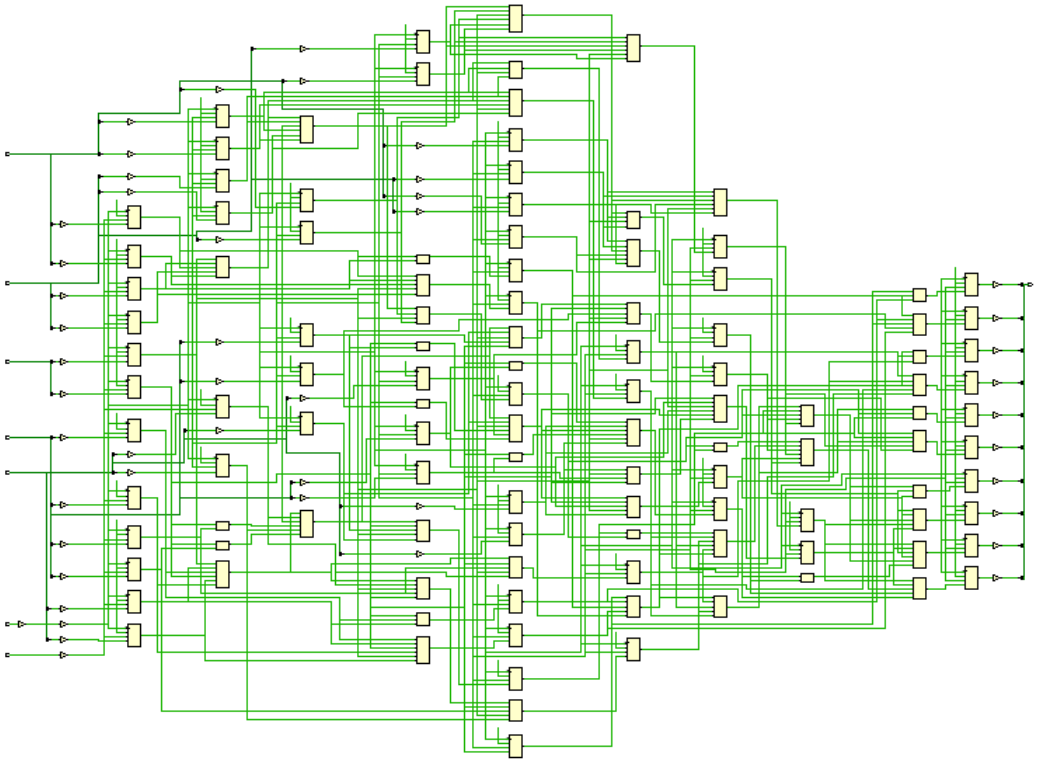


Figure 6: Sintesi

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	37	0	53200	0.07
LUT as Logic	37	0	53200	0.07
LUT as Memory	0	0	17400	0.00
Slice Registers	63	0	106400	0.06
Register as Flip Flop	63	0	106400	0.06
Register as Latch	0	0	106400	0.00
F7 Muxes	0	0	26600	0.00
F8 Muxes	0	0	13300	0.00

Figure 7: Risorse Occupate

2.4.2 Implementazione con n posto a 8bit

Tramite l'implementazione, si ottiene un circuito realistico, caratterizzato da un ritardo strettamente legato alle LUT che lo compongono, e dal modo in cui esse sono interconnesse tra di loro.

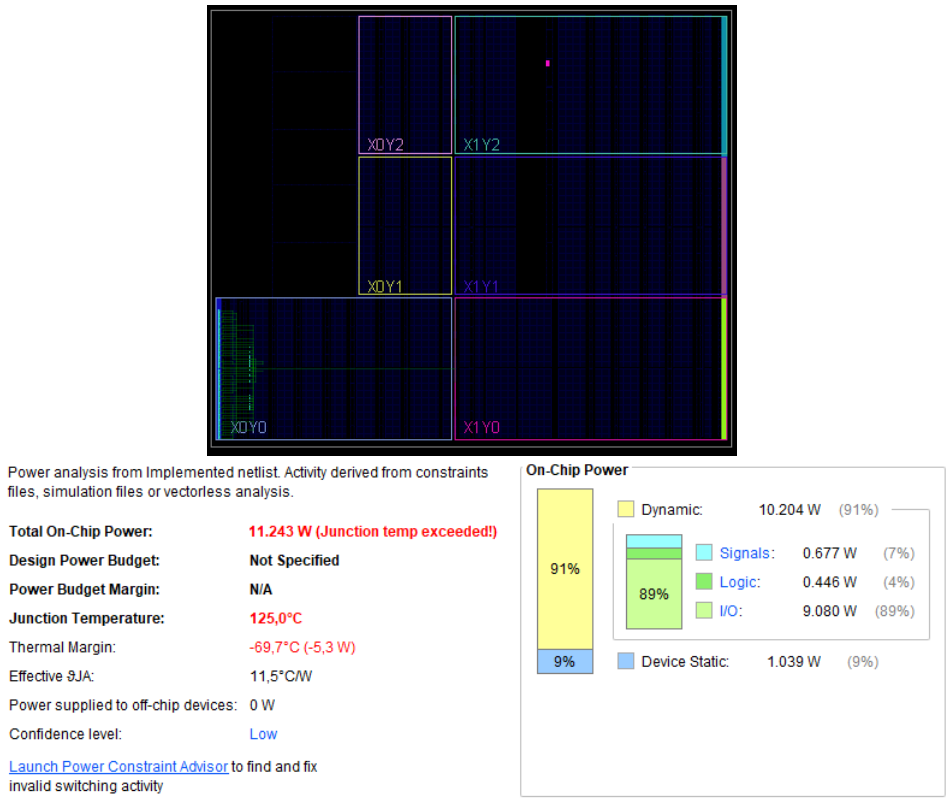


Figure 8: Implementazione n=8

Si osservi in Fig.8 che il Circuito ottenuto, potrebbe incorrere in gravi malfunzionamenti, dato il valore della Temperatura di Giunzione, che supera il valore massimo consentito. Affinchè vi sia una corretta dissipazione di potenza, vengono inseriti dei vincoli (*Constraint*) di frequenza sul segnale di Clock. Introduciamo il seguente vincolo in frequenza:

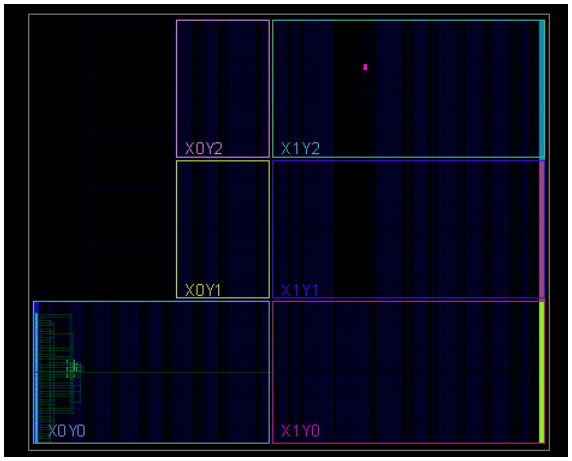
```
1 create_clock -period 5.000 -name myclock -waveform {0.000 2.500} [get_nets clk]
```

Listing 5: Test Bench

Implementando nuovamente il circuito è possibile osservare immediatamente gli effetti del vincolo di frequenza in termini di potenza; in particolare la dissipazione di potenza è stata stimata alla frequenza di funzionamento che corrisponde ai 5ns definiti nel Constraint e la Temperatura di Giunzione risulta profondamente migliorata. Si osserva inoltre che:

$$\text{Dissipazione Statica} >> \text{Dissipazione Dinamica}$$

Questo accade poiché il quantitativo di risorse responsabili della Dissipazione Statica sono nettamente maggiori del quantitativo di risorse responsabili della Dissipazione Dinamica. Si noti inoltre che, essendo il Clock un segnale ricevuto da più componenti circuitali, per una migliore gestione di carico, necessita di risorse dedicate alla sua distribuzione e per tale ragione ha un contributo di una quota pari al 9% sulla Dissipazione di potenza dinamica.



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

Total On-Chip Power:	**0.127 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**26,5°C**
Thermal Margin:	58,5°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

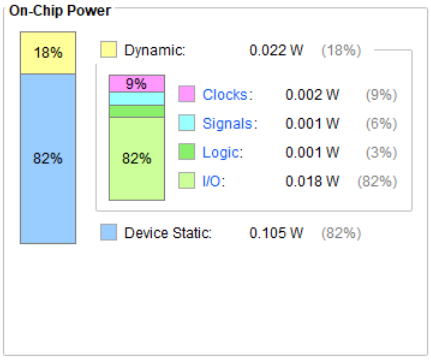


Figure 9: Implementazione n=8

Tramite un attenta analisi dei Reports è possibile verificare che il Constraint introdotto, non solo è rispettato, ma essendo il Worst Negative Slack una quantità positiva, è possibile ridurlo ulteriormente:

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

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)  THS Failing Endpoints  THS Total Endpoints     WPWS(ns)     TPWS(ns)
    -------      -------  ---------------------  -------------------      -------      -------  ---------------------  -------------------     --------     --------
      0.688        0.000                      0                   29        0.118        0.000                      0                   29        2.000        0.000


All user specified timing constraints are met.
```

2.5 Sintesi con n posto a 16 bit

Come svolto con n posto a 8 bit, svolgiamo la sintesi del circuito, analizzando anche le risorse occupate:

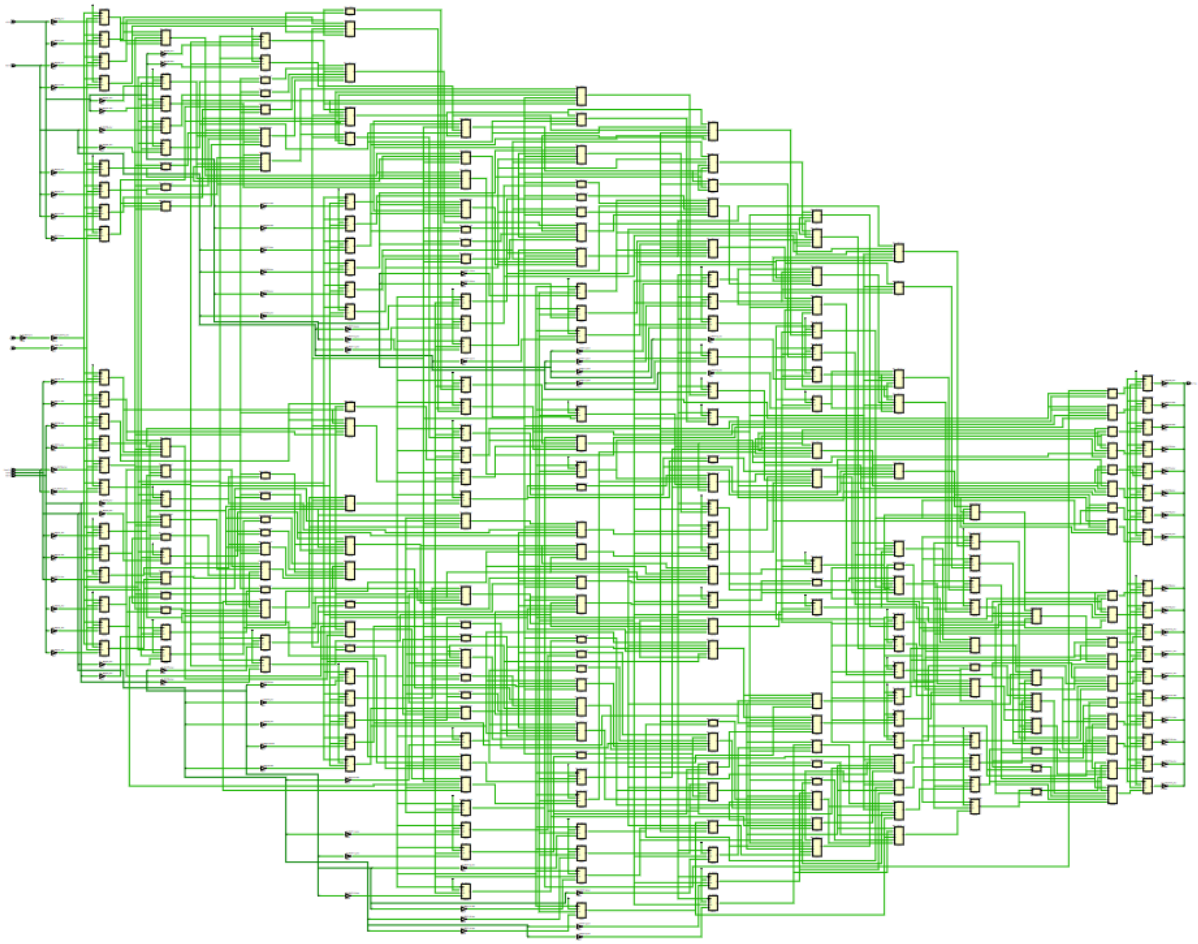


Figure 10: Sintesi Circuitale con n=16 bit

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	104	0	53200	0.20
LUT as Logic	104	0	53200	0.20
LUT as Memory	0	0	17400	0.00
Slice Registers	119	0	106400	0.11
Register as Flip Flop	119	0	106400	0.11
Register as Latch	0	0	106400	0.00
F7 Muxes	0	0	26600	0.00
F8 Muxes	0	0	13300	0.00

Figure 11: Risorse Occupate

2.5.1 Implementazione con n posto a 16bit

Considerato il costraint sul clock di 5ns, viene eseguita l'implementazione: Osservando la Fig.12

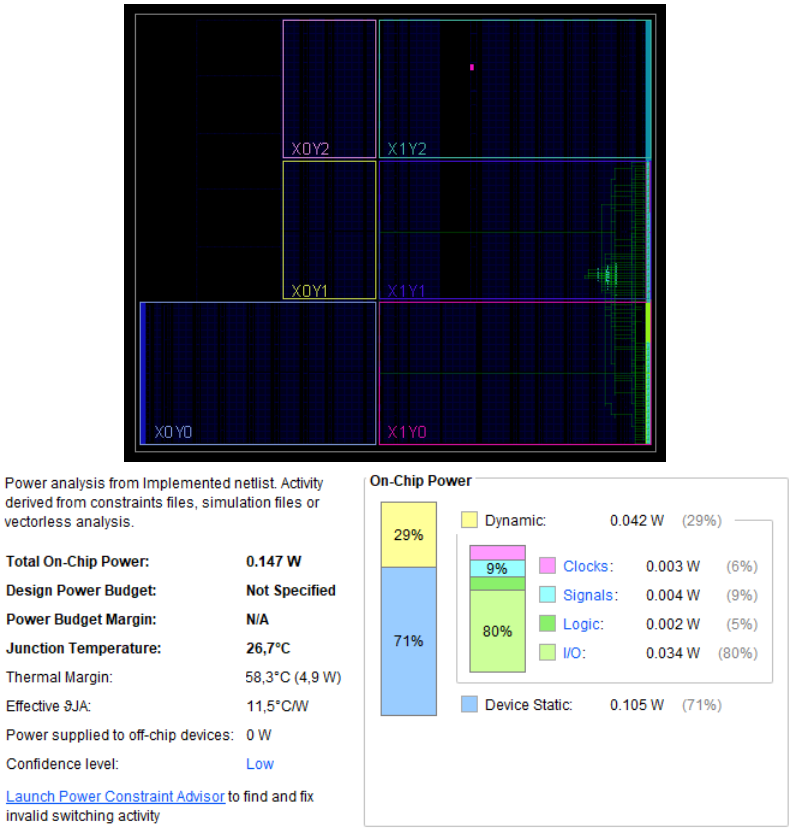


Figure 12: Implementazione n=16

è possibile notare che non vi sono problemi di dissipazione di potenza. Ma purtroppo non viene rispettato il vincolo di frequenza:

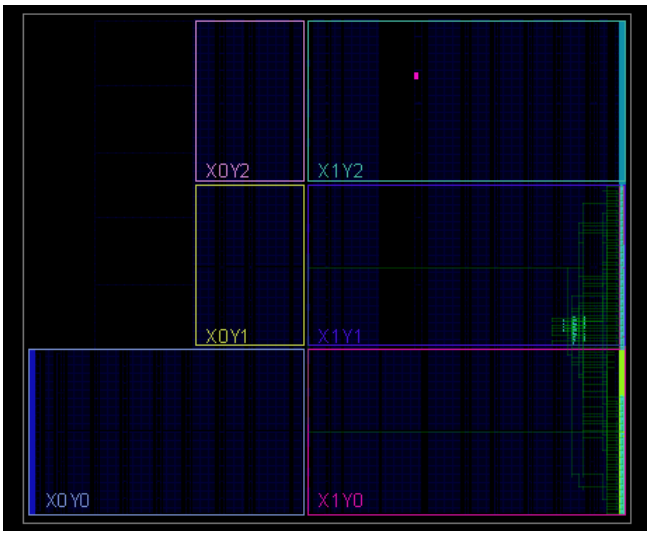
```
------------------------------------------------------------------------------------------------
| Design Timing Summary
| ---------------------
------------------------------------------------------------------------------------------------

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)  THS Failing Endpoints  THS Total Endpoints     WPWS(ns)     TPWS(ns)
    -------      -------  ---------------------  -------------------      -------      -------  ---------------------  -------------------     --------     --------
     -0.597       -1.430                      7                   53        0.152        0.000                      0                   53        2.000        0.000


Timing constraints are not met.
```

Perciò portiamo il vincolo di frequenza sul clock a 6ns.

Portando il vincolo di frequenza a 6ns, è garantito il corretto funzionamento del circuito come dimostrato dai dati presenti nelle figure sotto riportate.



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

Total On-Chip Power:	**0.14 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

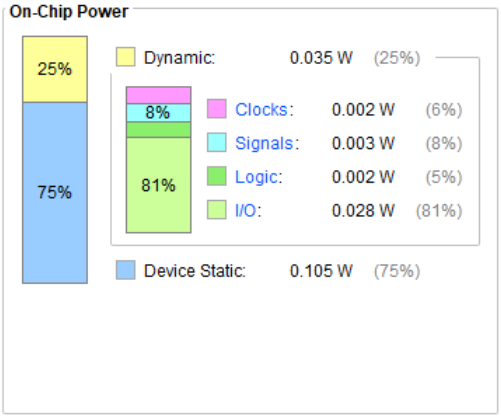


Figure 13: Implementazione n=16

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

    WNS(ns)      TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints      WHS(ns)      THS(ns)  THS Failing Endpoints  THS Total Endpoints     WPWS(ns)     TPWS(ns)
    -------      -------  ---------------------  -------------------      -------      -------  ---------------------  -------------------     --------     --------
      0.587        0.000                      0                   53        0.167        0.000                      0                   53        2.500        0.000


All user specified timing constraints are met.
```

2.6 Analisi simulazione con Timing-Post Implementazione

Effettuiamo ora la simulazione, tenendo conto di tutti i comportamenti reali del circuito (Ritardo LUT e Ritardo Collegamenti), introducendo quindi una **non idealità totale**. Si osservi che è stato inserito nel Test Bench il comando

$$wait \quad for \quad Tclk + 100ns;$$

Per mettere in risalto che, dato il general reset attuato nei circuiti nei primi 100ns, l'uscita non segue una valida variazione degli ingressi.

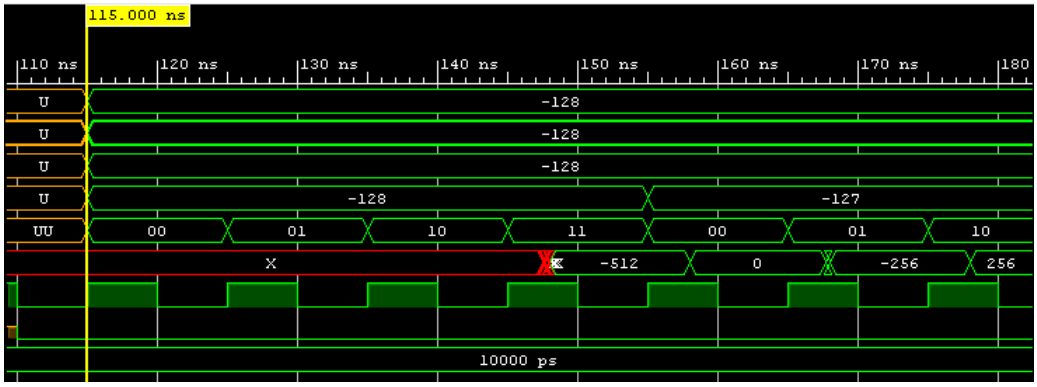


Figure 14: Grafico Simulazione con Timing n=8bit

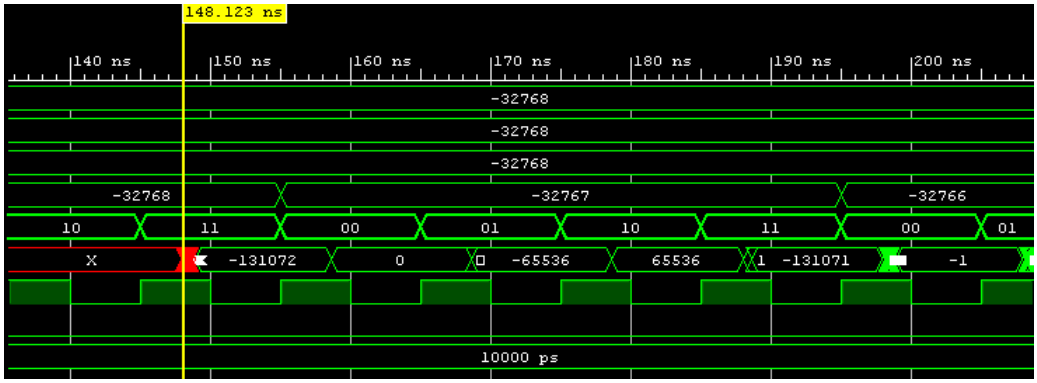


Figure 15: Grafico Simulazione con Timing n=16bit

È possibile osservare che in alcuni intervalli di tempo, determinati dal **tempo di assestamento** del dispositivo, l'uscita non è "valida". Tale situazione si verifica, poiché gli n bit in uscita, vengono elaborati ognuno con un tempo differente e solo che dopo tutti i bit si assestano, allora, l'uscita diventa valida.

2.7 Analisi simulazione con Timing-Post Sintesi

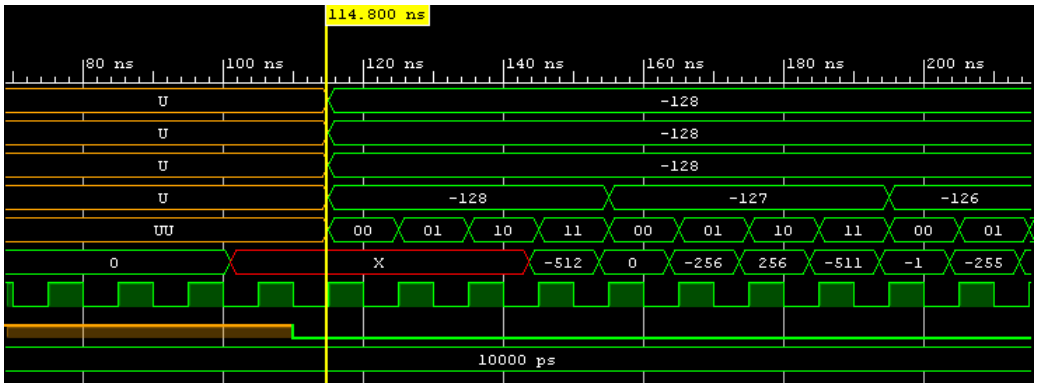


Figure 16: Grafico Simulazione con Timing n=8bit

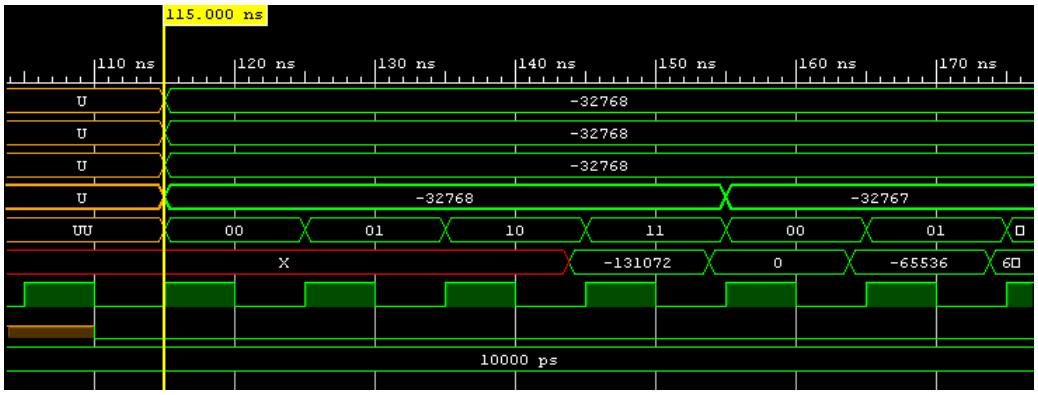


Figure 17: Grafico Simulazione con Timing n=16bit

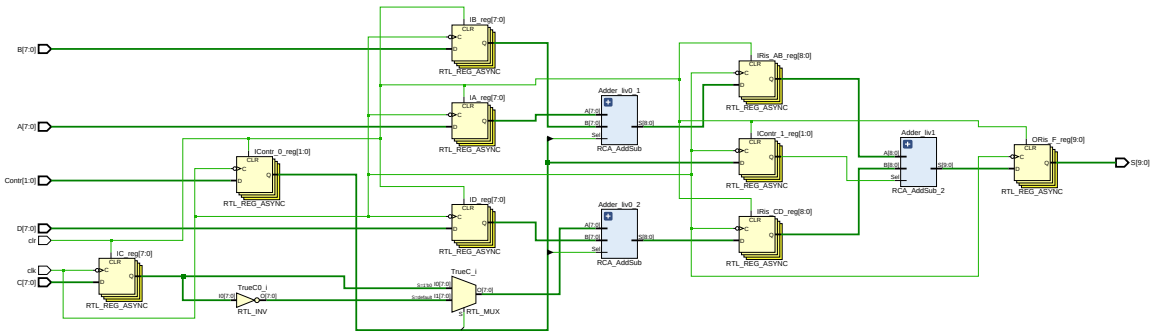


Figure 18: Design n=8bit

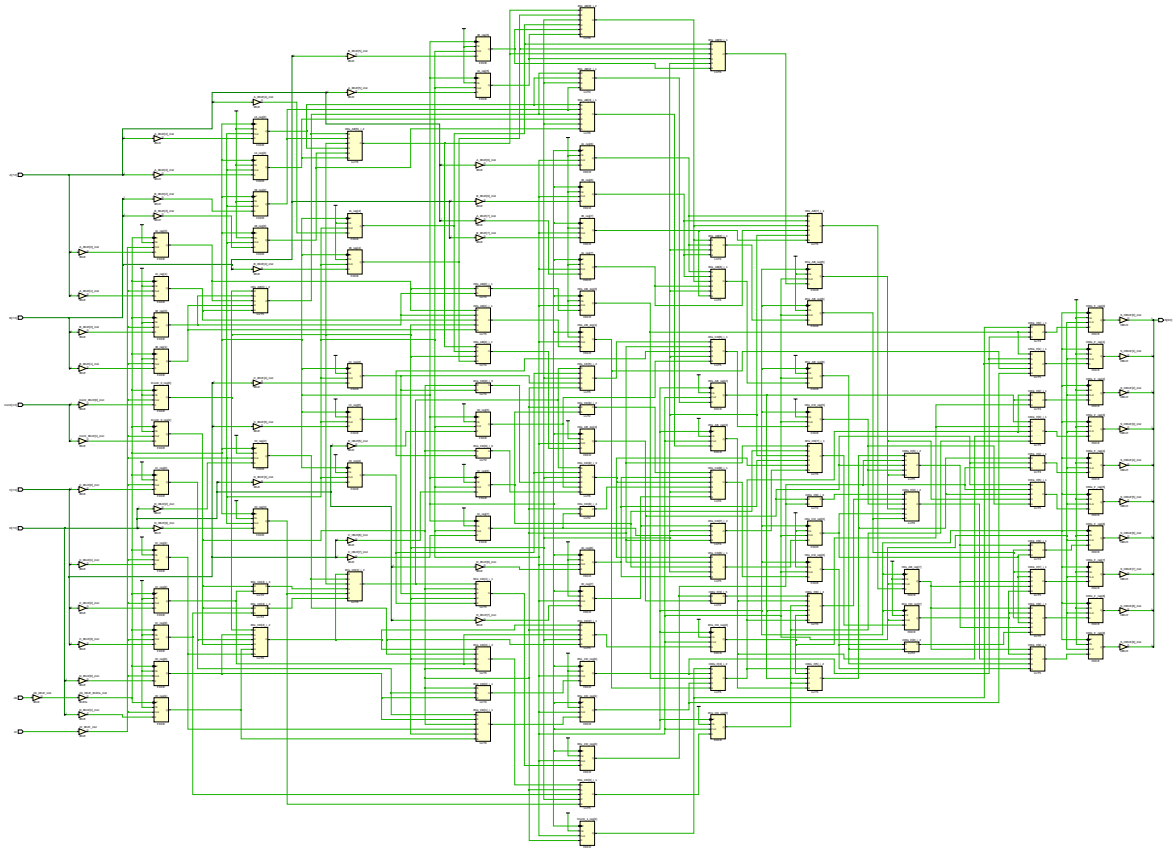


Figure 19: Sintesi Circuitale n=8bit

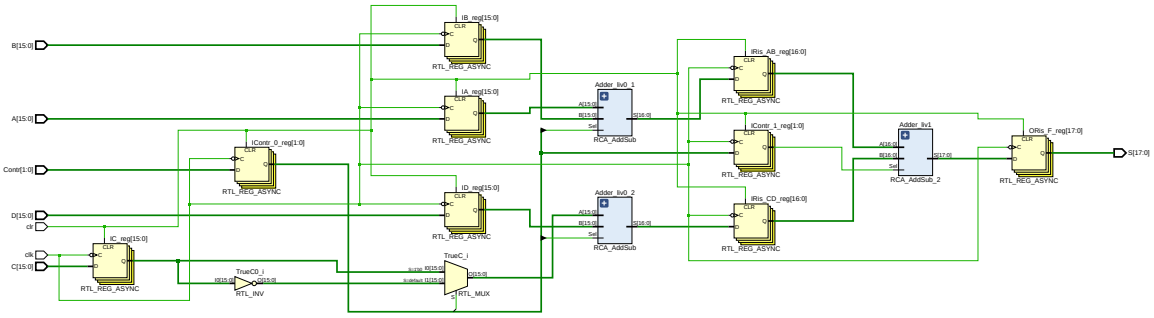


Figure 20: Design n=16bit

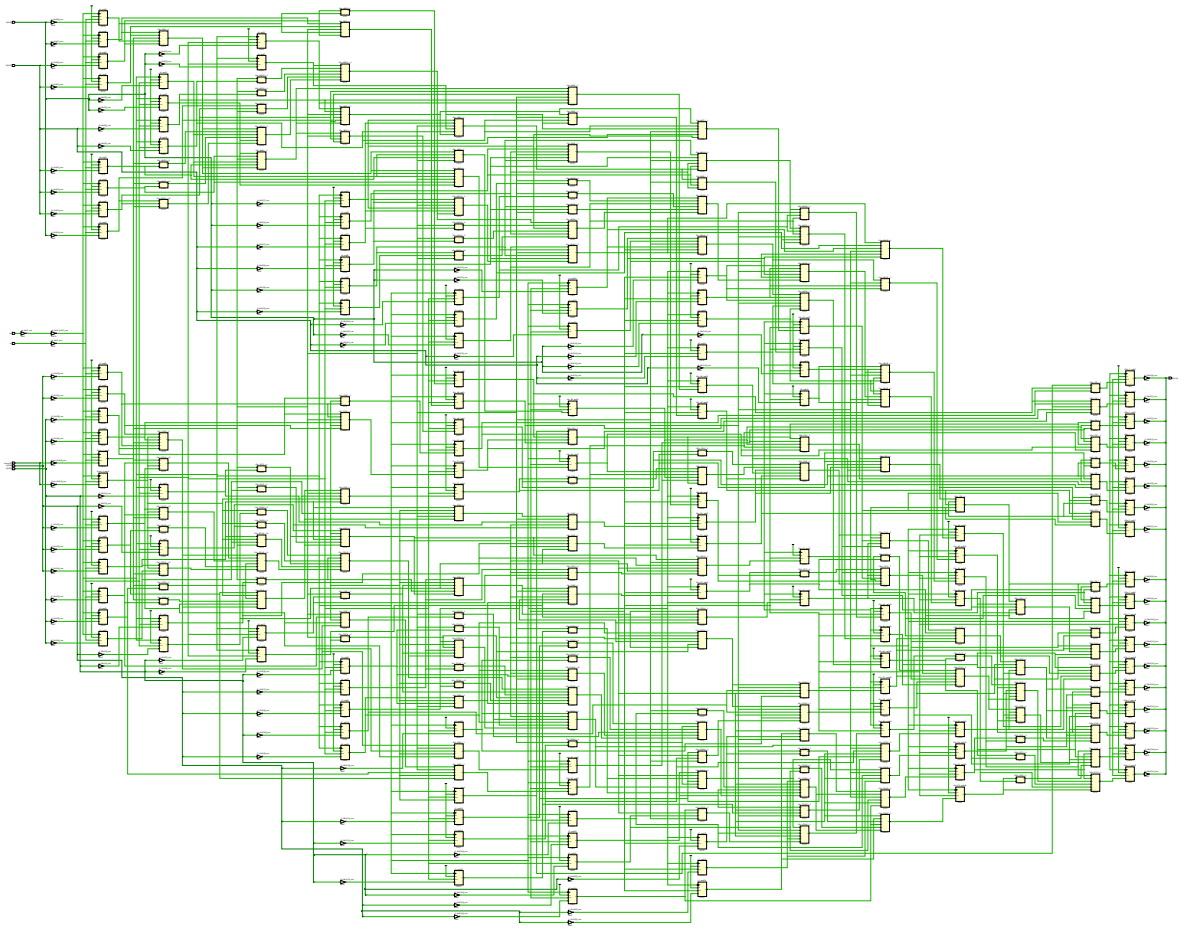


Figure 21: Sintesi Circuitale n=16bit