

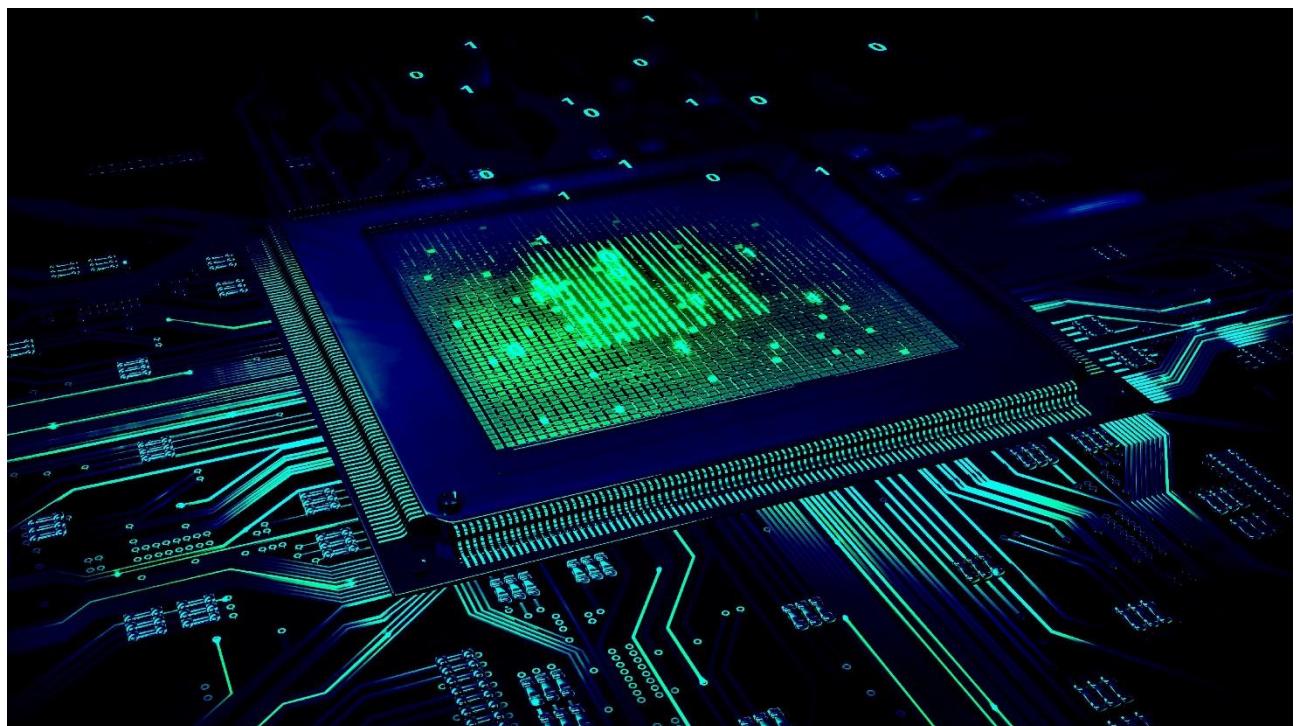
Adder-Subtracter a n bit

$$Ris = A+B-C$$

In questo progetto si è realizzato l'**Adder-Subtracter a n bit** implementando inizialmente i blocchi che lo costituiscono, dunque facendo uso di una gerarchia.

La strategia adoperata è stata quella di sviluppare i blocchi più semplici per arrivare progressivamente a quelli più complessi.

È stato approfondito di volta in volta lo studio di ciascun componente: si è iniziato con i **Registri a n bit** proseguendo poi con un **Adder algebrico a n bit** (in grado di effettuare addizioni e sottrazioni).



DEFINIZIONI

RETI LOGICHE

In elettronica digitale, una **rete logica sequenziale** è un tipo di circuito logico, la cui uscita dipende non solo dal valore presente dei suoi segnali in ingresso, ma anche dalla sequenza degli ingressi passati, la cosiddetta "storia degli ingressi".

Quanto detto la contraddistingue da una **rete logica combinatoria**, la cui uscita è funzione solo degli ingressi presenti.

Per tale motivo, le reti sequenziali sono dotate di "stati" (**la memoria**), a differenza delle reti combinatorie.

La **logica sequenziale** è usata per costruire *macchine a stato finito* (termine usato anche in teoria dell'automazione), un blocco di base per tutta la circuiteria digitale.

In teoria, tutti i circuiti usati sui dispositivi sono un mix di reti combinatorie e sequenziali.

I circuiti digitali **logico-sequenziali** vengono suddivisi in **sincroni** e **asincroni**.

Nei **circuiti sequenziali sincroni**, lo stato del dispositivo cambia solo in tempi discreti in risposta a un segnale di clock.

Nei **circuiti sequenziali asincroni** invece, lo stato del dispositivo può variare in qualunque istante in risposta alla variazione degli ingressi.

Il termine **clock**, in elettronica, indica un segnale periodico, generalmente un'onda quadra, utilizzato per sincronizzare il funzionamento dei **dispositivi elettronici digitali**; scansiona, dunque, gli eventi nel tempo.

Può essere generato da qualsiasi oscillatore e si usa generalmente il tipo a quarzo per la sua alta stabilità di oscillazione.

In elettronica digitale, il **latch** (letteralmente "serratura", "chiavistello") è un circuito elettronico bistabile, caratterizzato quindi da almeno due stati stabili, in grado di memorizzare un bit di informazione nei sistemi a logica sequenziale *asincrona*.

Il clock per i *latch* funge da porta tra l'ingresso D e l'uscita Q:

- nel latch negativo, la porta si apre quando il clock è pari a 0
- nel latch positivo, quando il clock è pari a 1

Nei *flip-flop* l'uscita Q è determinata dai fronti del clock:

- fronte di salita = *rising edge*
- fronte di discesa = *falling edge*

Il **flip-flop** è un circuito sequenziale molto semplice, utilizzato, per esempio, come dispositivo di memoria elementare.

Il nome deriva dal rumore che facevano i primi circuiti elettronici di questo tipo, costruiti con dei relè che realizzavano il cambiamento di stato.

Ne esistono di vari tipi: flip-flop SR, flip-flop JK ecc.

Il termine **pipeline** indica una tecnica per l'implementazione del *parallelismo* a livello di istruzione all'interno di un singolo processore.

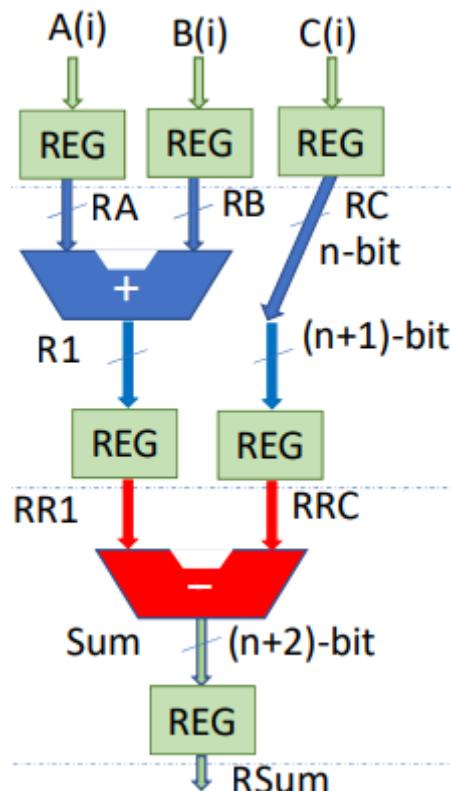
Il pipelining cerca di mantenere ogni *core* del processore occupato con alcune istruzioni da svolgere, dividendo le istruzioni in arrivo in una serie di passaggi sequenziali eseguiti da diverse unità del processore con diverse parti di istruzioni elaborate in parallelo.

Il lavoro svolto da un processore con *pipelining*, per eseguire un'istruzione, è diviso in passi (stadi della pipeline), che richiedono una frazione del tempo necessario all'esecuzione dell'intera istruzione.

Gli stadi sono connessi in maniera seriale per formare la pipeline. Le istruzioni:

1. entrano da un'estremità della pipeline;
2. vengono elaborate dai vari stadi secondo l'ordine previsto;
3. escono dall'altra estremità della pipeline.

Il tempo necessario per fare avanzare un'istruzione di uno stadio lungo la pipeline corrisponde ad un ciclo di clock di pipeline, per questo motivo, poiché gli stadi della pipeline sono collegati in sequenza, devono operare in modo sincrono.



esempio di Pipeline a basso livello del circuito del progetto

Il fenomeno dell'overflow per somme algebriche di due numeri rappresentati in complemento a due si verifica quando:

1. la somma tra due numeri entrambi positivi restituisce un numero negativo;
2. la somma tra due numeri entrambi negativi restituisce un numero positivo.

LINGUAGGIO VHDL

TIPO di input/output

Il tipo STANDARD LOGIC amplia la possibilità di rappresentazione e, per poterlo utilizzare, bisogna importare ad inizio pagina la libreria in cui è presente:

```
library IEEE; --rende visibile la libreria IEEE
```

```
use IEEE.STD_LOGIC_1164.ALL; --rende visibili i contenuti del package STD_LOGIC_1164
```

Per questo motivo quelli che sono stati chiamati bit per comodità, in realtà vengono definiti come STD_LOGIC nel codice scritto.

COMPONENT

Il **VHDL** permette una modellazione gerarchica, ovvero assemblare un modulo attraverso sotto-moduli; ciò avviene con l'utilizzo della parola chiave **COMPONENT** che serve per richiamare i codici già scritti in altri file di testo.

PROCESS

Il **PROCESS** aiuta a gestire la sequenzialità delle istruzioni, ovvero la loro dipendenza dal tempo.

SCHEMATIC

Eseguendo la **sintesi**, si è ottenuta una mappa (**SCHEMATIC**) che rappresenta le vere porte logiche, messe a disposizione dalla piattaforma scelta ad inizio progetto: *ZedBoard Zynq Evaluation and Development Kit*. Quanto descritto rappresenta l'hardware.

Sono state ricavate tabelle di verità sotto forma di *LUT* (che usano *FPGA*).

Dopo aver eseguito la sintesi, si è proceduto con l'implementazione per poter fissare i collegamenti.

GENERIC

Generic supporta le informazioni statiche nei blocchi in modo simile alle costanti, ma a differenza di esse, i valori di **generic** possono essere forniti esternamente (es. *nbitin* in **support**).

Analogamente alle porte, anch'essi possono essere dichiarati nell'*entity* e nei *component*, definite prima delle porte.

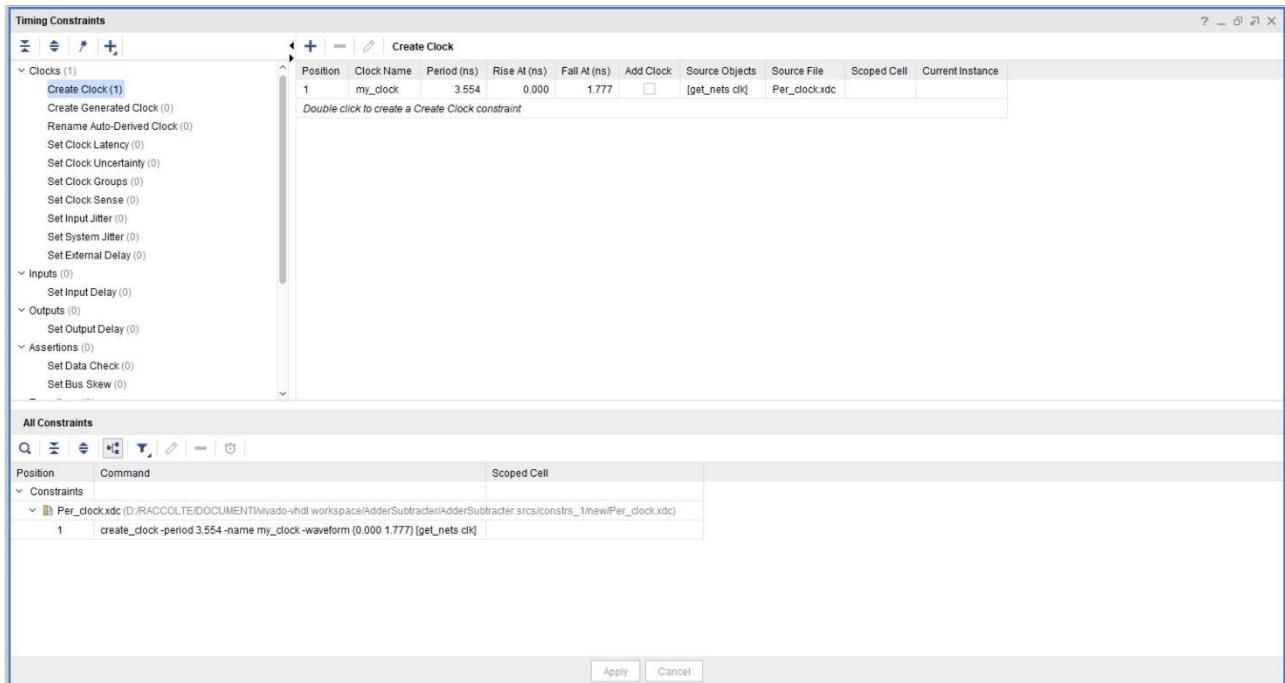
I valori di **generic** dichiarati nelle *entity* possono essere letti nell'*entity* stessa o nell'*architecture* associata.

In particolare, è possibile utilizzare un **generic** per:

- specificare la dimensione delle porte
- il numero di sottocomponenti all'interno di un blocco
- le caratteristiche temporali di un blocco
- le caratteristiche fisiche di un progetto
- la larghezza dei vettori all'interno di un'architecture
- numero di iterazioni di loop, ecc.

clock

TIMING CONSTRAINTS nb = 8 bit

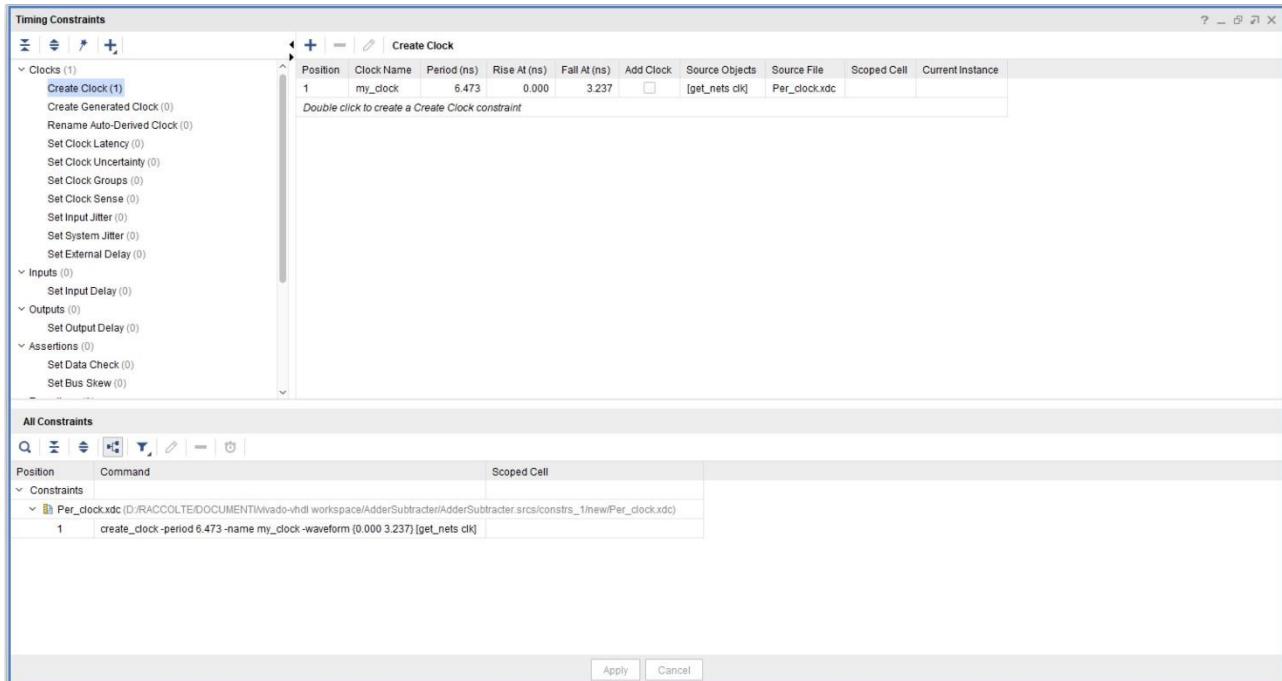


Constraint Per_clock nb = 8 bit

The screenshot shows the 'Per_clock.xdc' file in a code editor. The file contains a single line of timing constraint code:

```
create_clock -period 3.554 -name my_clock -waveform {0.000 1.777} [get_nets clk]
```

TIMING CONSTRAINTS nb = 16 bit

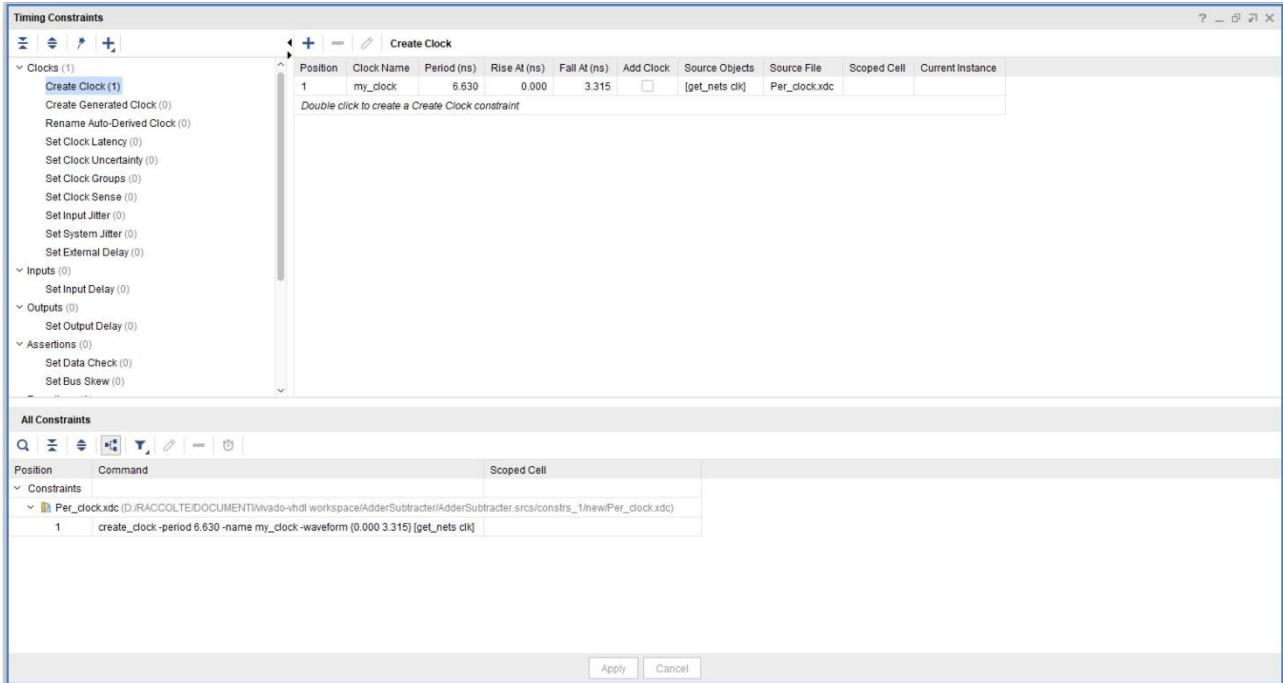


Constraint Per_clock nb = 16 bit

```
Per_clock.xdc
D:/RACCOLTE/DOCUMENTI/vivado-vhdl workspace/AdderSubtractor/AdderSubtractor.srscs/consts_1/new/Per_clock.xdc

1: create_clock -period 6.473 -name my_clock -waveform {0.000 3.237} [get_nets clk]
2:
3:
4:
5:
6:
7:
8:
9:
10:
11:
12:
```

TIMING CONSTRAINTS nb = 32 bit



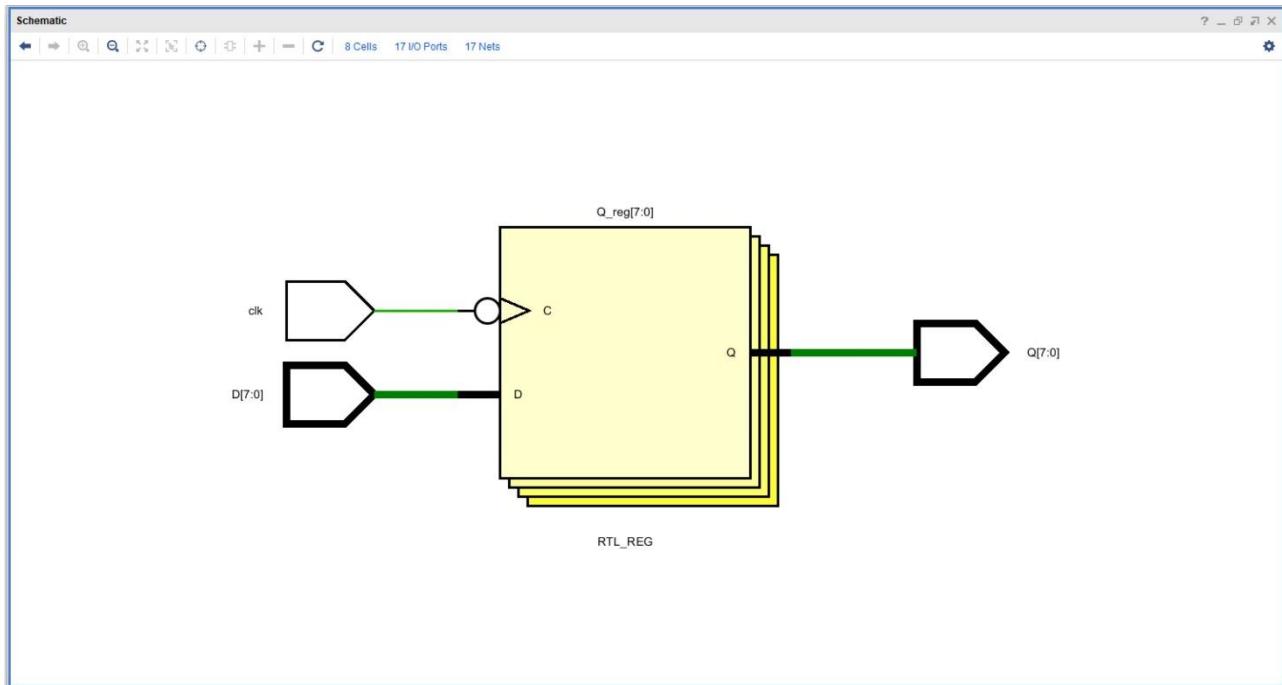
Constraint Per_clock nb = 32 bit

The screenshot shows the 'Per_clock.xdc' text editor window. The file path is D:/RACCOLTE/DOCUMENTI/vivado-vhdl workspace/AdderSubtractor/AdderSubtractor.srcs/constrs_1/new/Per_clock.xdc. The code area contains the following line:

```
create_clock -period 6.630 -name my_clock -waveform {0.000 3.315} [get_nets clk]
```

Registro

Schematic



Definizione

In elettronica digitale, i **registri hardware** sono un tipo di circuito tipicamente composto da **flip-flop**, spesso con molte caratteristiche simili alla memoria, come:

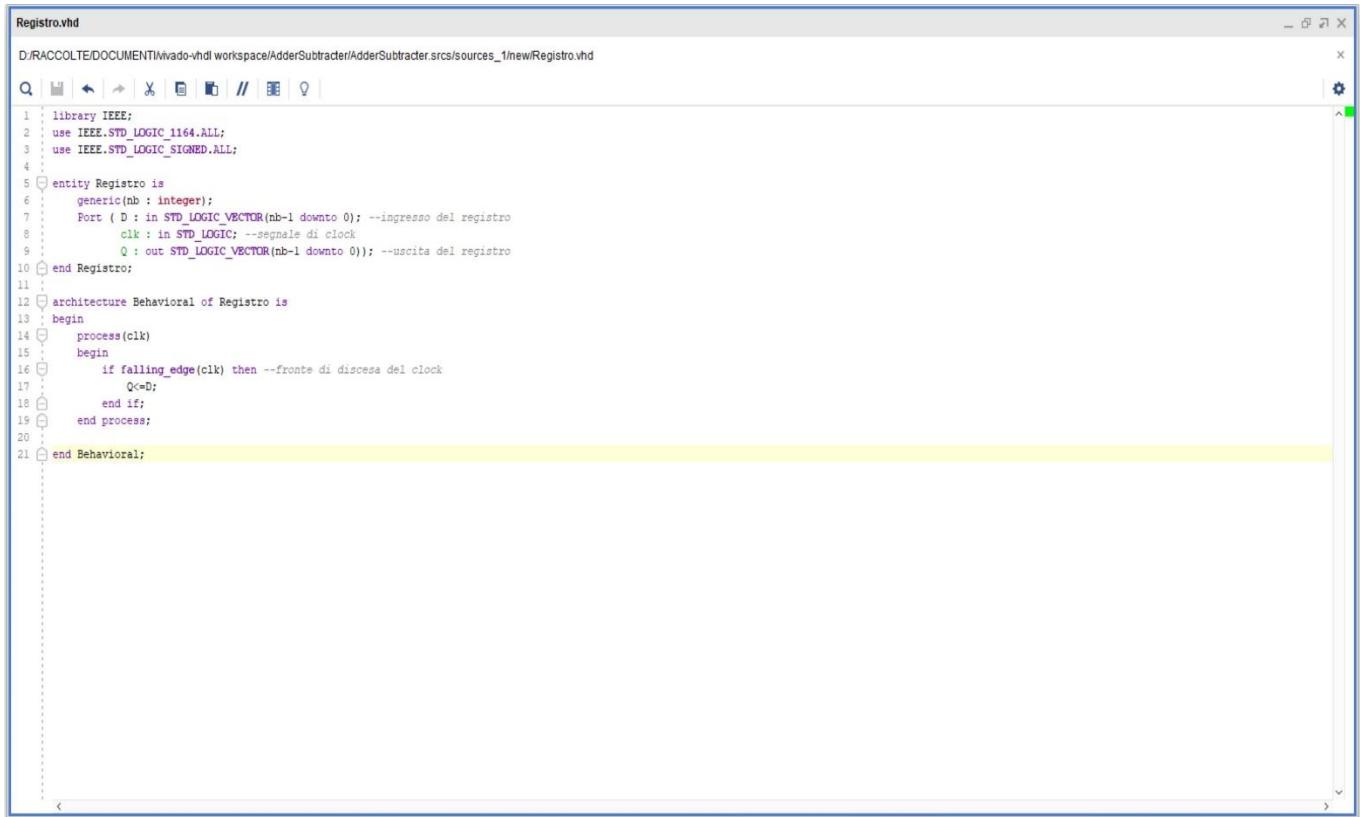
- L'abilità di leggere (*read*) o scrivere (*write*) multipli *bit* simultaneamente.
- Usare un indirizzo per selezionare un particolare registro in maniera simile a un indirizzo di memoria.

Nella pratica, i registri hardware sono usati come interfaccia tra il lato software e le periferiche. I software scrivono sui registri per inviare informazioni al dispositivo, e viceversa.

Alcuni dispositivi hardware inoltre includono registri utili per il loro uso interno, ma non visibili al software.

Nel progetto realizzato, il **Registro** è stato implementato seguendone la definizione: in particolare è stato reso sensibile al fronte di discesa del segnale di clock (***falling_edge***), in modo da memorizzare il valore in ingresso **D** (uno ***STD_LOGIC_VECTOR*** generico in *n* bit) e renderlo disponibile come uscita **Q** (sempre ***STD_LOGIC_VECTOR*** generico in *n* bit).

Codice



The screenshot shows a VHDL code editor window titled "Registro.vhd". The code is as follows:

```
Registro.vhd
D:/RACCOLTE/DOCUMENTI/vivado-vhdl workspace/AdderSubtractor/AdderSubtractor.srsc/sources_1/newRegistro.vhd

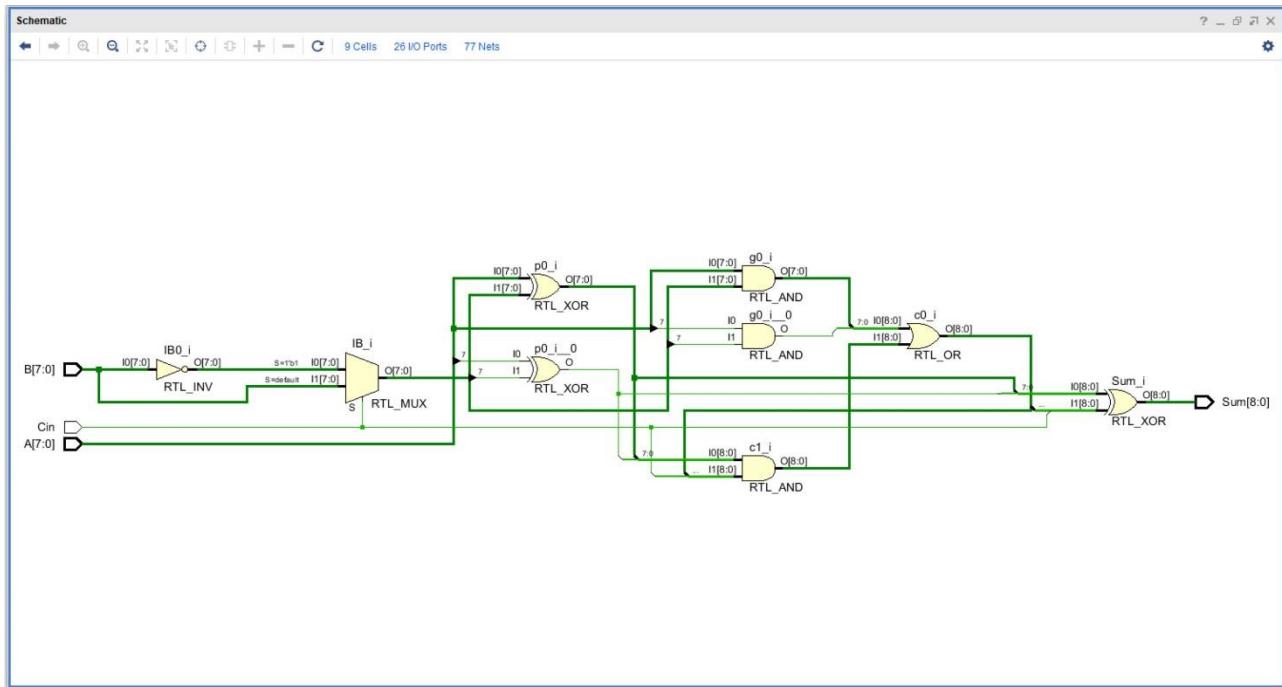
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Registro is
    generic(nb : integer);
    Port ( D : in STD_LOGIC_VECTOR(nb-1 downto 0); --ingresso del registro
            clk : in STD_LOGIC; --segnale di clock
            Q : out STD_LOGIC_VECTOR(nb-1 downto 0)); --uscita del registro
end Registro;

architecture Behavioral of Registro is
begin
    process(clk)
    begin
        if falling_edge(clk) then --fronte di discesa del clock
            Q<=D;
        end if;
    end process;
end Behavioral;
```

Adder (Sommatore Algebrico)

Schematic



Definizione

Il metodo più diretto per realizzare un generico addizionatore algebrico ad n bit è rappresentato dal **Ripple-Carry Adder** (RCA) o addizionatore a propagazione del riporto.

Il **Ripple-Carry** è stato realizzato utilizzando la descrizione *Behavioral*, senza fare uso della gerarchia che implica l'utilizzo del **Full-Adder**.

L'addizionatore RCA non fa altro che calcolare la somma così come verrebbe calcolata secondo il metodo **carta e penna**.

In particolare, sono stati utilizzati:

- Due ingressi (*operandi*) a n bit (**A** e **B**).
- Un bit di riporto (CARRY) in ingresso (**C_{in}**), in grado di determinare l'operazione da svolgere: con '0' l'addizione tra gli operandi, con '1' la sottrazione.
- Un'uscita somma algebrica (**Sum**), sempre a n bit.

L'uso delle *signal* e le fasi successive della progettazione derivano dalla seguente definizione:

Per rappresentare l'opposto di un numero binario in complemento a due se ne invertono, o negano, i singoli bit: si applica cioè l'operazione logica NOT. Si aggiunge infine 1 al valore del numero trovato con questa operazione. In formula con un generico numero binario N:

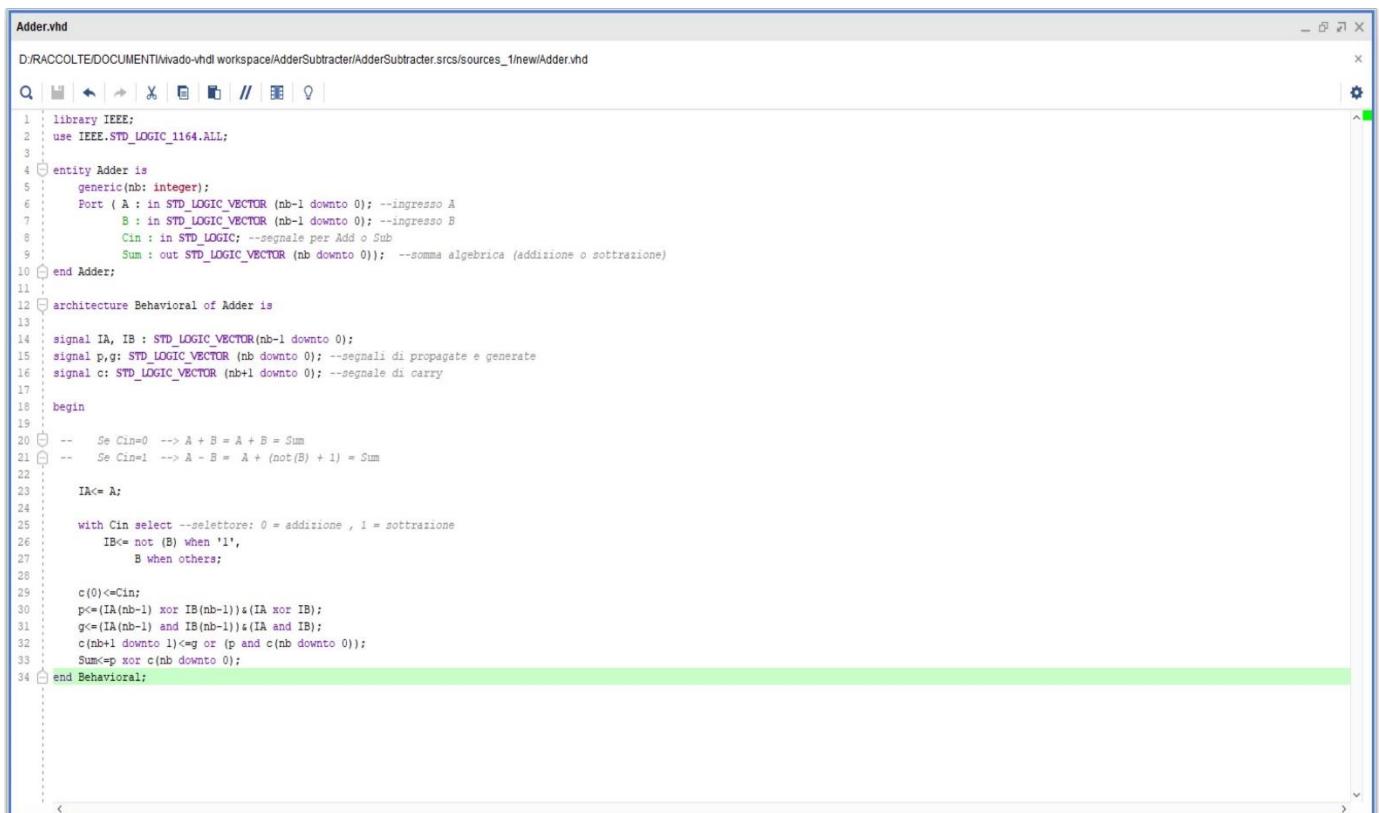
$$-N = \bar{N} + 1.$$

Oltre all'implementazione generale, sono state quindi utilizzate delle *signal*:

- Due segnali **IA** e **IB**, creati come **STD_LOGIC_VECTOR** generici a *n* bit, usati per gestire gli ingressi ed eventualmente negare (operazione logica *not*) il secondo operando **B**.
- Due segnali **p** e **g**, rispettivamente segnale di *propagate* e *generate*, definiti come **STD_LOGIC_VECTOR** generici a *n+1 bit*, con il compito di gestire le somme fra singoli *bit* e con un *bit* in più per evitare la problematica dell'overflow.
- Un segnale **c**, **STD_LOGIC_VECTOR** a *n+2* bit, usato per gestire i riporti tra i singoli *bit* (quindi *n carry*), il *bit* di riporto in uscita (un ipotetico *C_{out}*) e il bit di riporto in ingresso (il **C_{in}** visto sopra).

Infine, con l'utilizzo del costrutto **with/select** disponibile in **VHDL**, al variare di **C_{in}** sceglieremo di lavorare con l'operando **B** o con l'opposto di **B** (ovvero -B).

Codice



```

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity Adder is
    generic(nb: integer);
    port (A : in STD_LOGIC_VECTOR (nb-1 downto 0); --ingresso A
          B : in STD_LOGIC_VECTOR (nb-1 downto 0); --ingresso B
          Cin : in STD_LOGIC; --segnale per Add o Sub
          Sum : out STD_LOGIC_VECTOR (nb downto 0)); --somma algebrica (addizione o sottrazione)
end Adder;

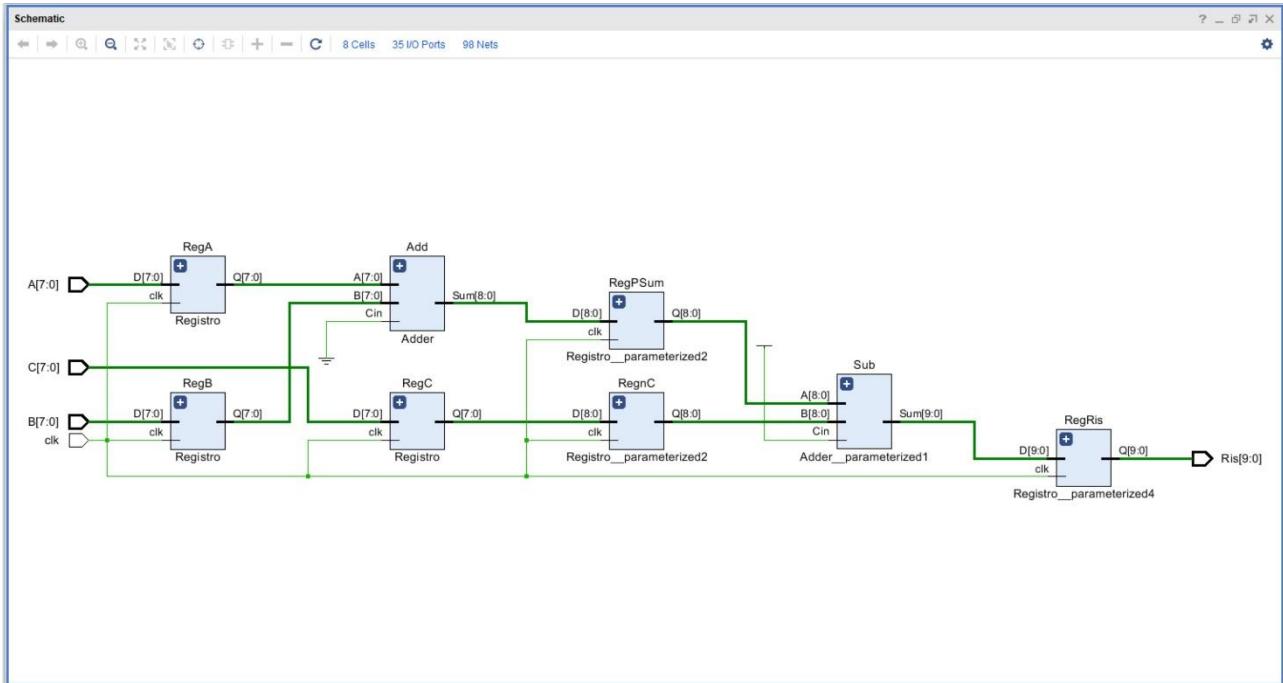
architecture Behavioral of Adder is
begin
    IA<= A;
    IB<= B;
    with Cin select
        Sum<= not (B) when '1',
        B when others;
    c(0)<=Cin;
    p<=(IA(nb-1) xor IB(nb-1))&(IA xor IB);
    g<=(IA(nb-1) and IB(nb-1))&(IA and IB);
    c(nb+1 downto 1)<=g or (p and c(nb downto 0));
    Sum<=p xor c(nb downto 0);
end Behavioral;

```

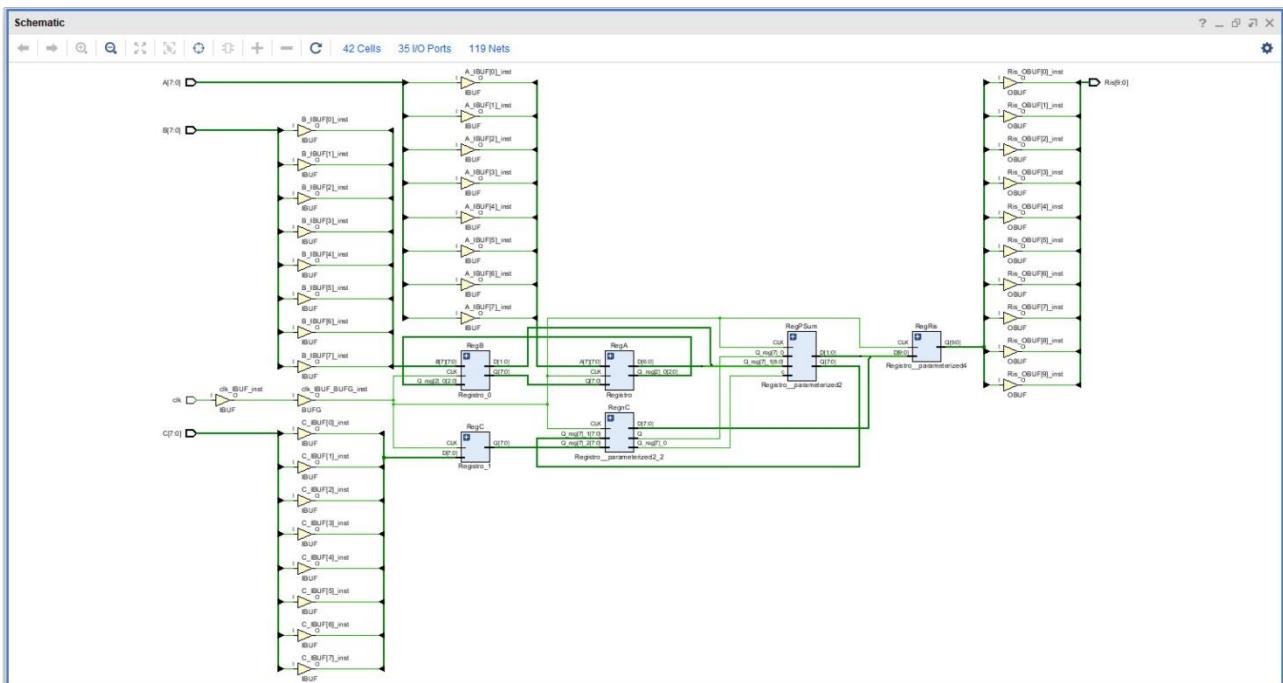
AddSub

Schematic 8 bit

RTL ANALYSIS > Schematic

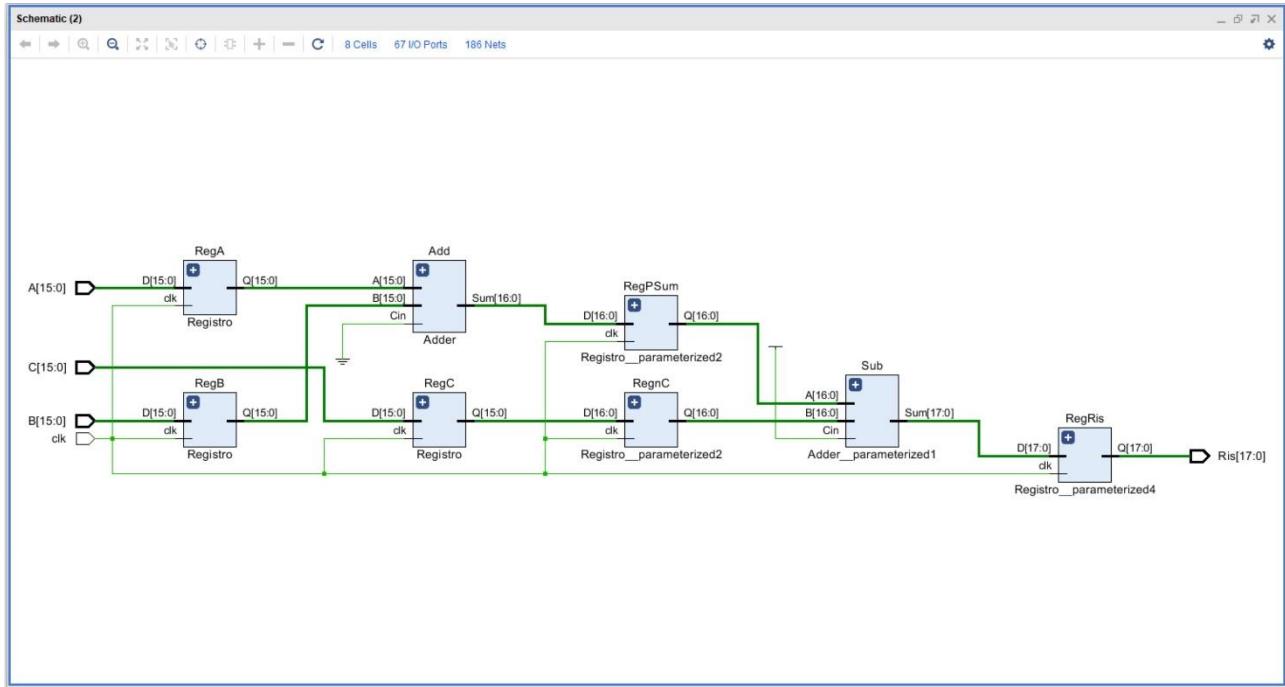


IMPLEMENTATION > Schematic

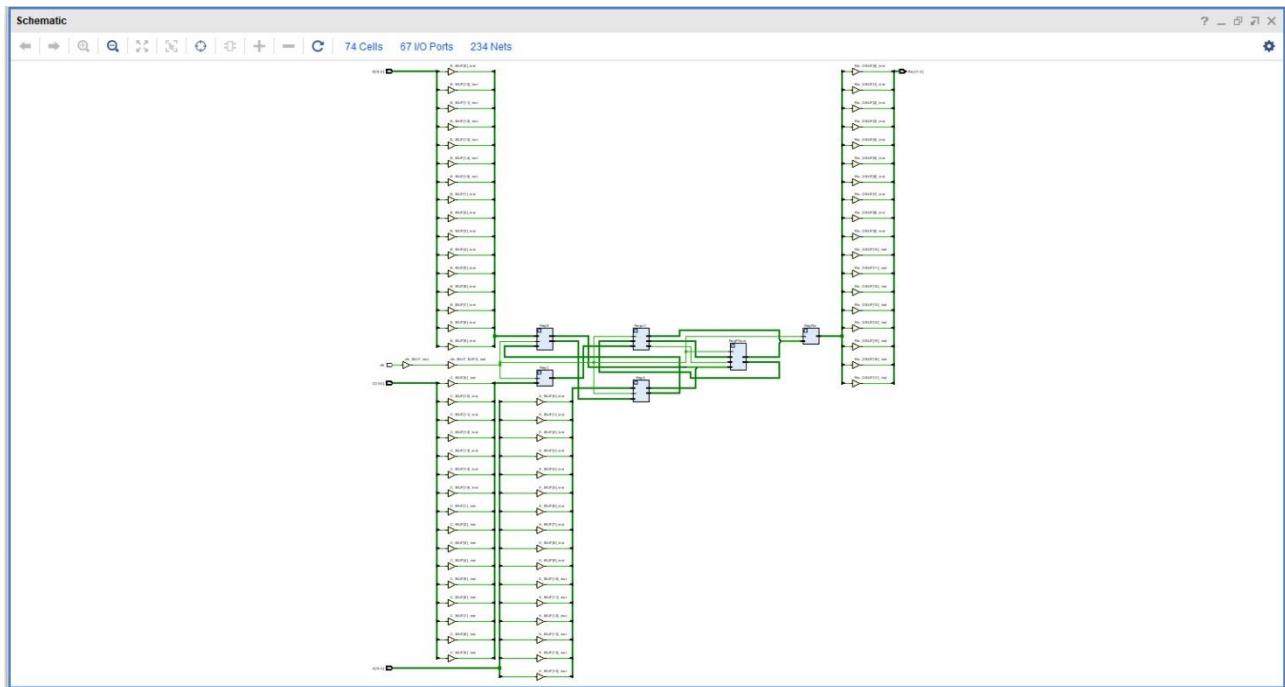


Schematic 16 bit

RTL ANALYSIS > Schematic

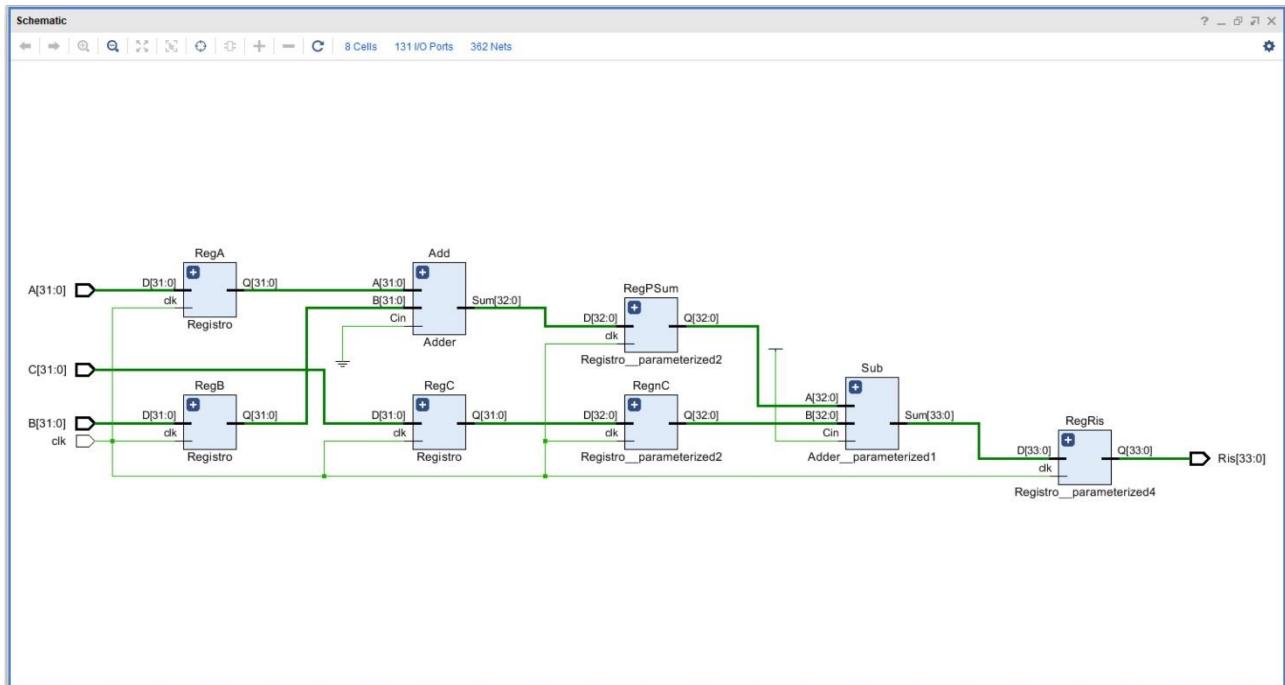


IMPLEMENTATION > Schematic

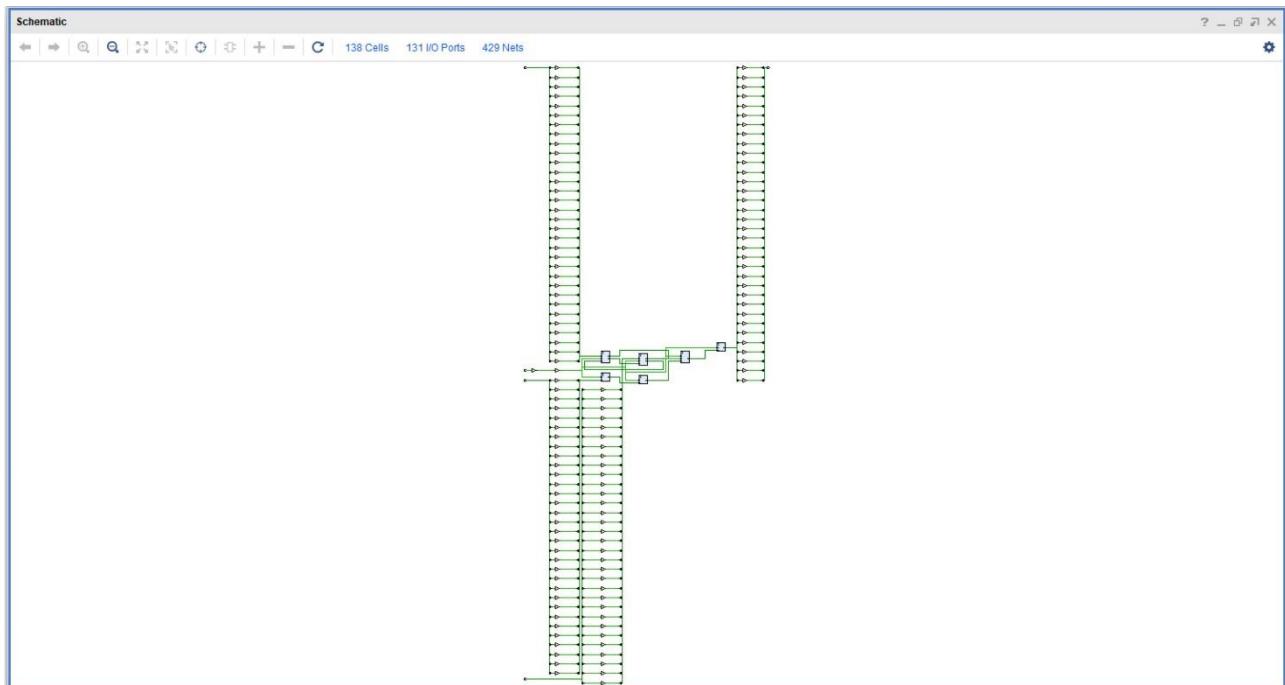


Schematic 32 bit

RTL ANALYSIS > Schematic



IMPLEMENTATION > Schematic



Definizione

Il circuito discusso in questo paragrafo, che abbiamo scelto di chiamare **AddSub**, può essere visto come composizione di circuiti più semplici già trattati in precedenza: sei **registri** e due **Adder** (*un sommatore e un sottrattore*).

Considerato che il circuito dovrà svolgere l'operazione algebrica $A + B - C$ con gli operandi generici in n bit, e valutando che una somma algebrica di tre operandi può essere svolta ricorsivamente come due somme elementari di soli due operandi, l'implementazione scelta consiste di:

- Tre ingressi (*operandi*) generici a n bit (**A**, **B** e **C**), da destinare ai rispettivi registri.
- Un segnale di clock (**clk**) per gestire le operazioni in modo sequenziale sincrono.
- Un'uscita risultato (**Ris**), questa volta a $n+2$ bit, per gestire il problema dell'*overflow* nelle possibili somme di bit espressi in *complemento a 2* (se **A** e **B** sono di segno concorde e **C**, al contrario, non lo è, l'operazione $A + B - C$ diventa una addizione di tre operandi e può provocare perciò ben 2 bit di *overflow* da controllare).

Come nel caso dell'**Adder** algebrico generico a n bit, sono state usate delle *signal*:

- Tre segnali **RA**, **RB** ed **RC**, **STD_LOGIC_VECTOR** generici a n bit, usati per comunicare i valori di **A**, **B** e **C** ai rispettivi registri *RegA*, *RegB* e *RegC*.
- Due segnali **nC** e **nRC**, questa volta **STD_LOGIC_VECTOR** generici a $n+1$ bit. La denominazione **nC** significa “*nuovo C*” ed è da intendere come estensione di un bit dell'operando **C** (secondo la definizione per un generico numero N a *nb* bit espresso in complemento a 2 vale:

$$N_{esteso} = (N(nb-1) \& N)$$

nRC invece, come per **RA** e **RB**, comunica il valore di **C** esteso al suo registro *nRegC*.

- Due segnali **PSum** e **RPSum**, come sopra **STD_LOGIC_VECTOR** generici a $n+1$ bit, definiti per gestire l'uscita dal primo *Adder* (un nuovo operando $A + B$) da inviare nel nuovo registro *RegPSum*.
- Segnale **ORis**, dato da inviare all'ultimo registro *RegRis*, che comunicherà infine il *risultato finale* all'uscita **Ris** dell'intera operazione $A + B - C$.

Codice

```

AddSub.vhd
? - D X
D:\RACCOLTE\DOCUMENTI\vivado-vhd\workspace\AdderSubtractor\AdderSubtractor.scs\sources_1\new\kddSub.vhd
Q [ ] < > X E D // [ ] V
AddSub.vhd

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.all;
3 library work; --directory in cui si trova il progetto attuale
4 use work.yourinst.all; --per la costante nbitin (dichiarsata in support)
5
6 entity AddSub is
7     generic(nb : integer:=nbitin);
8     Port ( A : in STD.LOGIC_VECTOR(nb-1 downto 0); --ingresso A
9             B : in STD.LOGIC_VECTOR(nb-1 downto 0); --ingresso B
10            C : in STD.LOGIC_VECTOR(nb-1 downto 0); --ingresso C
11            clk : in STD.LOGIC; --segnale di clock
12            Ris : out STD.LOGIC_VECTOR(nb-1 downto 0); --somma algebrica (addizione e sottrazione)
13    end AddSub;
14
15 architecture Behavioral of AddSub is
16
17 signal RA,RB,RC : STD.LOGIC_VECTOR(nb-1 downto 0); --segnali dei registri A, B e C
18 signal nC,nBC : STD.LOGIC_VECTOR(nb downto 0); -- segnale di C esteso (nuovo C) e del registro nC
19 signal FSum, RFSum : STD.LOGIC_VECTOR(nb downto 0); --segnali della Somma Parziale e del rispettivo registro
20 signal ORis : STD.LOGIC_VECTOR(nb-1 downto 0); --segnale del risultato finale
21
22 component Registro is
23     generic(nb : integer);
24     Port (D : in STD.LOGIC_VECTOR(nb-1 downto 0); -- segnale di C esteso (nuovo C)
25             clk : in STD.LOGIC;
26             Q : out STD.LOGIC_VECTOR(nb-1 downto 0));
27 end component;
28
29 component Adder is
30     generic(nb : integer);
31     Port(A : in STD.LOGIC_VECTOR(nb-1 downto 0);
32           B : in STD.LOGIC_VECTOR(nb-1 downto 0);
33           Cin : in STD.LOGIC;
34           Sum : out STD.LOGIC_VECTOR(nb downto 0));
35 end component;
36
37 begin
38
39     --Registro A
40     RegA : Registro
41         generic map(nb)
42         port map(A,clk,RA); --Operando A
43
44     --Registro B
45     RegB : Registro
46         generic map(nb)
47         port map(B,clk,RB); --Operando B
48
49     --Registro C
50     RegC : Registro
51         generic map(nb)
52         port map(C,clk,RC); --Operando C
53         generic map(nb-1) --estensione di C di un bit
54         port map(nC,clk,nBC); --Operando nC
55
56         --Addition
57         Add : Adder
58             generic map(nb)
59             port map(RA,RB,"0",FSum); -- A+B= SommaParziale
60
61         --Registro della somma parziale
62         RegFSum : Registro
63             generic map(nb+1)
64             port map(FSum,clk,RFSum);
65
66         --Sottrazione
67         Sub : Adder
68             generic map(nb+1)
69             port map(RFSum,nBC,"1",ORis); --SommaParziale + (-C)
70
71         --Registro del risultato
72         RegRis : Registro
73             generic map(nb+2)
74             port map(ORis,clk,Ris); --Risultato finale
75
76 end Behavioral;
    
```

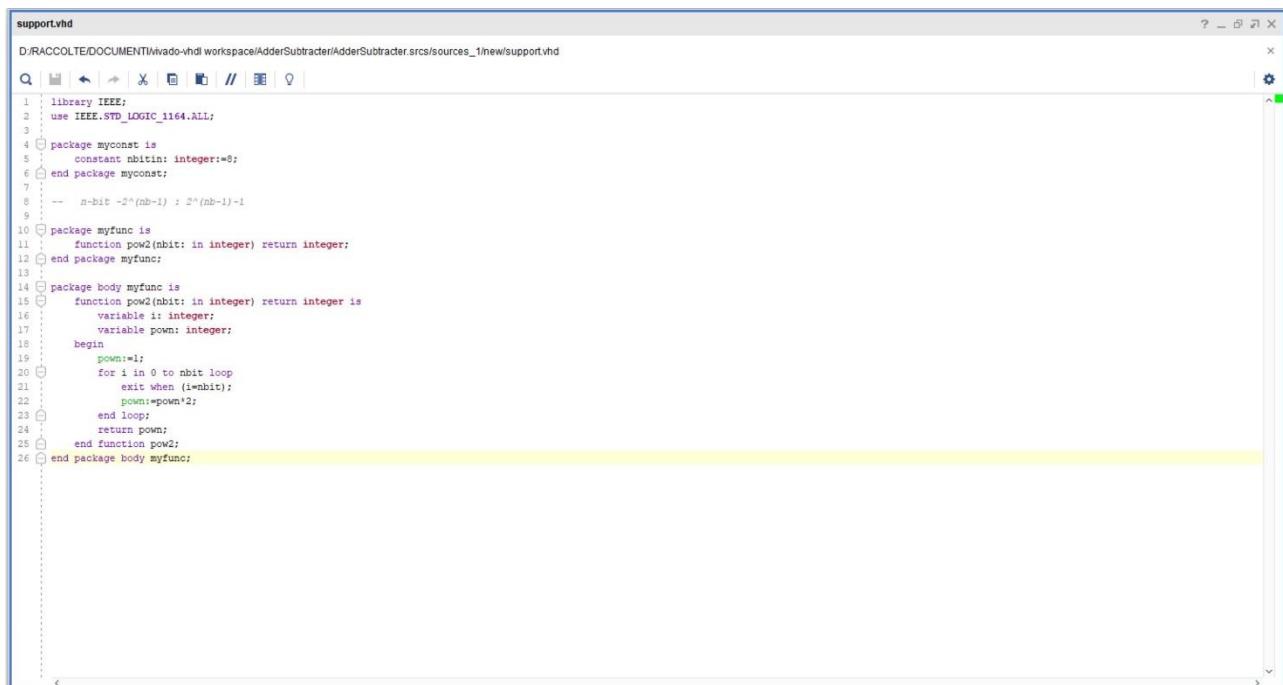
support

È stata definita, all'interno del package ***myconst***, una costante **nbitin** di tipo *integer*, usata in ogni file precedente, che ci consente di modificare il *generic nb* agendo solo su questo parametro presente in support; tale operazione è possibile inserendo ad inizio codice **use work.myconst.all;**

In VHDL non esiste una funzione che calcoli 2^n , dunque è stata definita con il nome di ***myfunc*** dentro il file di testo *support*.

All'interno della *begin* si utilizza il **for loop**, il quale permette di eseguire per *n* volte un gruppo di statement sequenziali.

CODICE con nbitin = 8 bit



```

support.vhd
D:/RACCOLTE/DOCUMENTI/mvado-vhdl workspace/AdderSubracter/AdderSubracter.srca/sources_1/new/support.vhd

1 library IEEE;
2 use IEEE.STD_LOGIC_1164.ALL;
3
4 package myconst is
5   constant nbitin: integer:=8;
6 end package myconst;
7
8 -- n-bit -2^(nb-1) : 2^(nb-1)-1
9
10 package myfunc is
11   function pow2(nbit: in integer) return integer;
12 end package myfunc;
13
14 package body myfunc is
15   function pow2(nbit: in integer) return integer is
16     variable i: integer;
17     variable pown: integer;
18   begin
19     pown:=1;
20     for i in 0 to nbit loop
21       exit when (i=nbit);
22       pown:=pown*2;
23     end loop;
24   return pown;
25 end function pow2;
26 end package body myfunc;

```

CODICE con nbitin = 16 bit

```

support.vhd
D:/RACCOLTE/DOCUMENTI/vivado-vhdl workspace/AdderSubtractor/AdderSubtractor.scs/sources_1/new/support.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
package myconst is
  constant nbitin: integer:=16;
end package myconst;
-- n-bit -2^(nb-1) : 2^(nb-1)-1
package myfunc is
  function pow2(nbit: in integer) return integer;
end package myfunc;
package body myfunc is
  function pow2(nbit: in integer) return integer is
    variable i: integer;
    variable pown: integer;
  begin
    pown:=1;
    for i in 0 to nbit loop
      exit when (i=nbit);
      pown:=pown*2;
    end loop;
    return pown;
  end function pow2;
end package body myfunc;

```

CODICE con nbitin = 32 bit

```

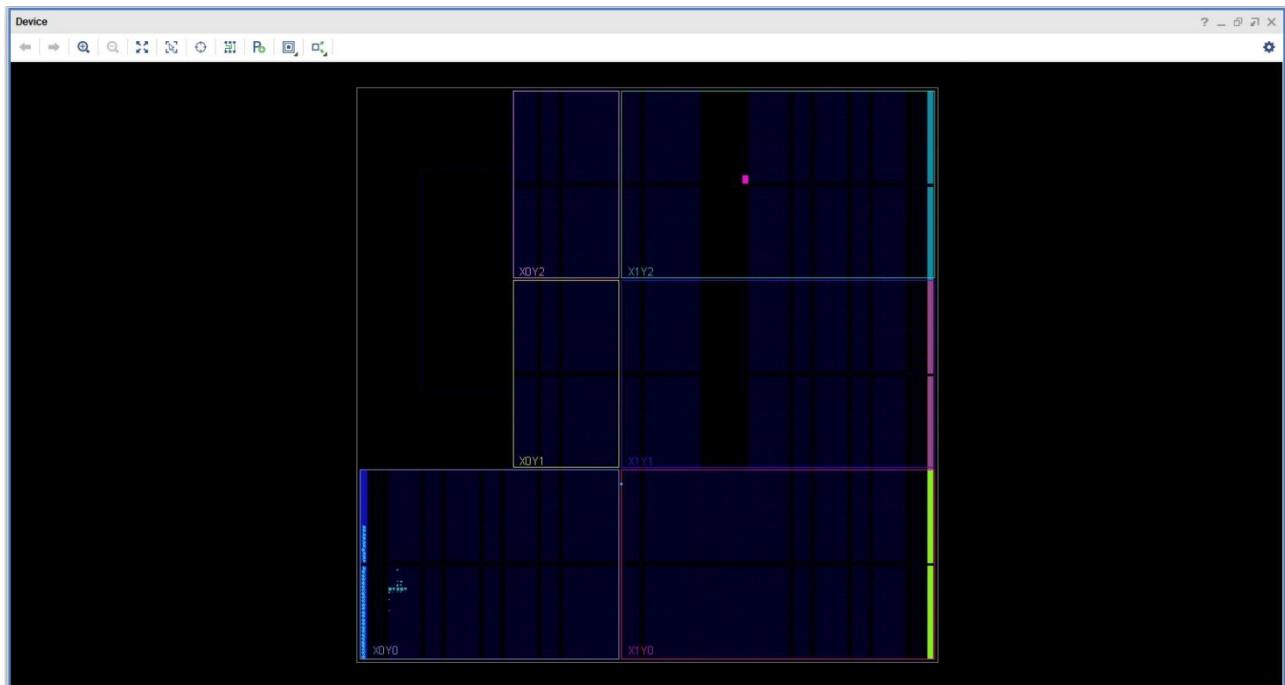
support.vhd
D:/RACCOLTE/DOCUMENTI/vivado-vhdl workspace/AdderSubtractor/AdderSubtractor.scs/sources_1/new/support.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
package myconst is
  constant nbitin: integer:=32;
end package myconst;
-- n-bit -2^(nb-1) : 2^(nb-1)-1
package myfunc is
  function pow2(nbit: in integer) return integer;
end package myfunc;
package body myfunc is
  function pow2(nbit: in integer) return integer is
    variable i: integer;
    variable pown: integer;
  begin
    pown:=1;
    for i in 0 to nbit loop
      exit when (i=nbit);
      pown:=pown*2;
    end loop;
    return pown;
  end function pow2;
end package body myfunc;

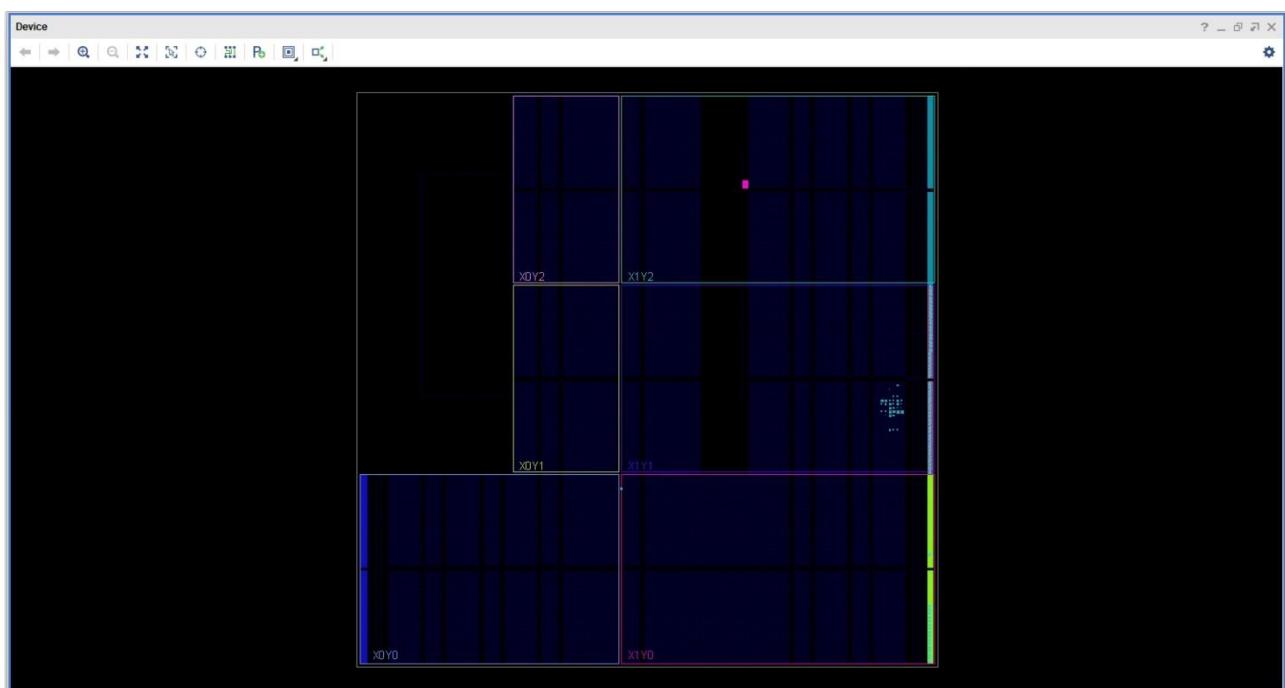
```

Device (*Implemented Design*)

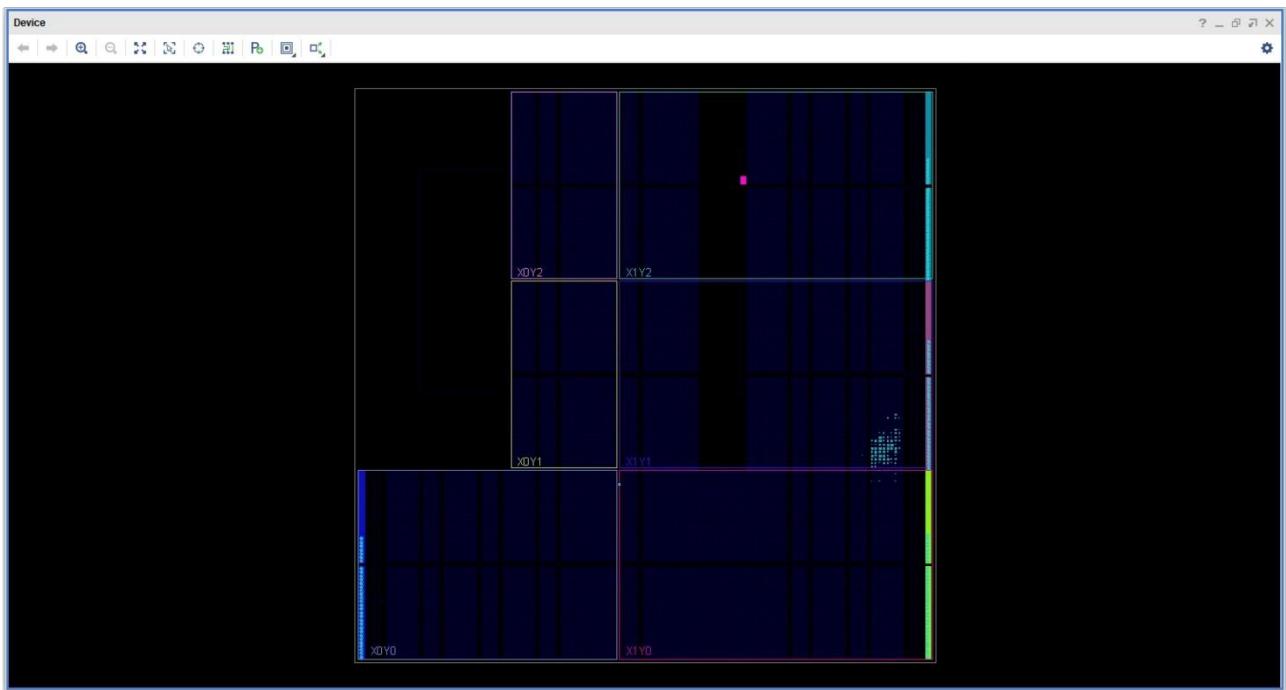
DEVICE 8 bit



DEVICE 16 bit



DEVICE 32 bit



Frequenza di funzionamento (*Report Timing Summary*)

IMPL_ROUTE_REPORT_TIMING_SUMMARY NB = 8 BIT

```
impl_1_route_report_timing_summary_0 - impl_1
D:\RACCOLTE\DOCUMENTI\wado\vhdl\workspace\AdderSubtractor\AdderSubtractor\run\impl_1\AddSub_timing_summary_routed.rpt
Q | + | - | X | B | // | E | ? | Read-only | X |
116
116 12. checking latch_loops
117 -----
118 There are 0 combinational latch loops in the design through latch input
119
120
121
122 -----
123 Design Timing Summary
124 -----
125
126
127 WNS(ns) TNS(ns) TNS Failing Endpoints TNS Total Endpoints WNS(ns) TNS(ns) TNS Failing Endpoints TNS Total Endpoints WWS(ns) TWS(ns) TWS Failing Endpoints TWS Total Endpoints
128 0.281 0.000 0 27 0.198 0.000 0 27 1.277 0.000 0 52
129
130
131
132 All user specified timing constraints are met.
133
134
135 -----
136 Clock Summary
137 -----
138
139
140 Clock Waveform(ns) Period(ns) Frequency(MHz)
141 -----
142 my_clock [0.000 1.777] 3.554 201.373
143
144
145 -----
146 Initie Clock Table
147 -----
148
149
150 Clock WNS(ns) TNS(ns) TNS Failing Endpoints TNS Total Endpoints WNS(ns) TNS(ns) TNS Failing Endpoints TNS Total Endpoints WWS(ns) TWS(ns) TWS Failing Endpoints TWS Total Endpoints
151 -----
152 my_clock 0.281 0.000 0 27 0.198 0.000 0 27 1.277 0.000 0 52
153
154
155 -----
156 Timing Details
157 -----
158
159
160 From Clock: my_clock
161 To Clock: my_clock
162
163 Setup : 0 Failing Endpoints, Worst Slack 0.281ns, Total Violation 0.00ns
164 Hold : 0 Failing Endpoints, Worst Slack 0.198ns, Total Violation 0.00ns
165 PW : 0 Failing Endpoints, Worst Slack 1.277ns, Total Violation 0.00ns
166
167
168 Max Delay Paths
169
170
171 Slack (MET) : 0.201ns (required time - arrival time)
172 Source: RegnQ/Q_req[0]/C
173 (falling edge-triggered cell FIRE clocked by my_clock (rise@0.000ns fall@1.777ns period=3.554ns))
174 Destination: RegnQ/Q_req[?]/D
175 (falling edge-triggered cell FIRE clocked by my_clock (rise@0.000ns fall@1.777ns period=3.554ns))
176 Path Group: my_clock
177 Path Type: Setup (Max at SLE Process Corner)
178 Requirements: 3.554ns (my_clock fall@6.331ns - my_clock fall@1.777ns)
179 Data Path Delay: 3.257ns (logic@3.857ns (41.461%) route 1.900ns (58.33%))
180 Long Path: 4. Lut(0x100)->RegnQ/Q_req[0]
181 Clock Path Skew: -0.000ns (CLK_RCD CLK_CWD)
182 Destination Clock Delay (DCD): 4.585ns = ( 9.913 - 5.331 )
183 Source Clock Delay (SCD): 5.104ns = ( 6.881 - 1.777 )
184 Clock Pessimism Removal (CPW): 0.459ns
185 Clock Uncertainty: 0.000ns ((TCD * (TID^2))^(1/2) + DJ) / 2 + PE
186 Clock System Jitter (TSJ): 0.071ns
187 Total Input Jitter (TIJ): 0.000ns
188 Discrete Jitter (DJ): 0.000ns
189 Phase Error (PE): 0.000ns
190
191 Location Delay type Inc(ns) Path(ns) Netlist Resource(s)
192
193 (clock my_clock fall edge)
194
195 AA9 net (F0=0) 0.000 1.777 f clk (IN)
196 AA9 net (F0=1) 0.000 1.777 f clk
197 IB0F (Prop_ibuf_I_0) 0.124 0.336 z RegnQ/Q[4]_3_2_0/0
198 net (F0=1, routed) 0.419 9.456 RegnS/S[0]/S[1]
199 IB0F (Prop_ibuf_I_0) 0.171 4.956 clk_IBUF_inst/0
200 net (F0=1, routed) 0.281 9.158 RegnQ/Q[4]_3_2_0/0
201 SLICE_X3V18 LUTS (Prop_lut3_I_0) 0.101 5.057 f clk_IBUF_BTFG_inst/0
202 net (F0=1, routed) 1.624 6.081 RegnS/CLK
203
204 SLICE_X3V18 FIRE z RegnQ/Q_req[0]/C (IS_INVERTED)
205
206 SLICE_X3V18 FIRE (Prop_fddr_C_0) 0.459 7.340 f RegnQ/Q_req[0]/Q
207 net (F0=1, routed) 0.873 8.212 RegnQ/Q[0]/C[0]
208 SLICE_X3V18 LUTS (Prop_lut3_I_0) 0.124 0.336 z RegnQ/Q[4]_3_2_0/0
209 net (F0=1, routed) 0.419 9.456 RegnS/S[0]/S[1]
210 SLICE_X3V18 LUTS (Prop_lut3_I_0) 0.171 4.956 RegnQ/Q[4]_3_2_0/0
211 net (F0=1, routed) 0.281 9.158 RegnQ/Q[4]_3_2_0/0
212 SLICE_X3V18 LUTS (Prop_lut3_I_0) 0.327 10.138 z RegnQ/Q[7]_3_1_0/0
213 net (F0=1, routed) 0.000 10.138 RegnQ/Q[7]
214 SLICE_X3V18 FIRE z RegnQ/Q_req[?]/D
215
216
217 (clock my_clock fall edge)
218 AA9 0.331 5.331 f
219 net (F0=0) 0.000 5.331 f clk (IN)
220 AA9 0.000 5.331 clk
221 IB0F (Prop_ibuf_I_0) 0.074 6.205 f clk_IBUF_inst/0
222 net (F0=1, routed) 1.972 8.177 clk_IBUF
223 IB0F (Prop_ibuf_I_0) 0.091 0.268 f clk_IBUF_BTFG_inst/0
224 net (F0=1, routed) 1.645 9.913 RegnS/CLK
225 SLICE_X4V18 FIRE z RegnQ/Q_req[?]/C (IS_INVERTED)
226
227 clock pessimism 0.458 10.371
228 clock uncertainty -0.035 10.335
229 SLICE_X4V18 FIRE (Setup_fddr_C_0) 0.084 10.415 RegnQ/Q_req[?]
230
231 required time 10.415
232 arrival time -10.138
233 slack 0.281
234
235 Slack (MET) : 0.301ns (required time - arrival time)
236 Convise: RegnQ/Q_req[?]/C
237
```

IMPL_ROUTE_REPORT_TIMING_SUMMARY NB = 16 BIT

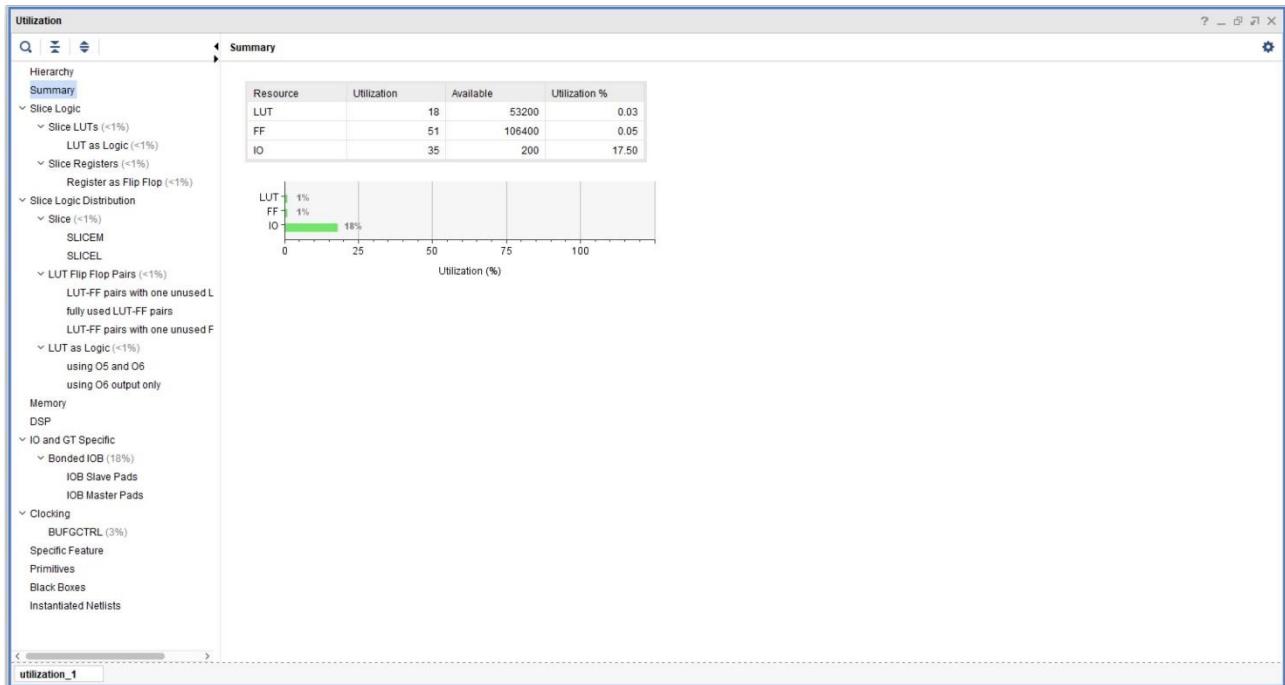
IMPL_ROUTE_REPORT_TIMING_SUMMARY NB = 32 BIT

Risorse occupate (*Report Utilization*)

IMPLEMENTATION_1_PLACE_REPORT_UTILIZATION NB = 8 BIT

```
impl_1_place_report_utilization_0 - impl_1
D:/RACCOLTE/DOCUMENTI/Nvado-vhd/workspace/HeaderSubtrader/HeaderSubtrader/runs/impl_1/AddSub_utilization_placed.rpt
Read-only

28: 1. Slice Logic
29: -----
30: -----
31: +-----+ Site Type + Used | Fixed | Available | Util% +-----+
32: | Slice LUTs | 18 | 0 | 53200 | 0.03 |
33: | LUT as Logic | 15 | 0 | 53200 | 0.03 |
34: | LUT as Memory | 0 | 0 | 53200 | 0.00 |
35: | Shift Registers | 51 | 0 | 104400 | 0.05 |
36: | Register as Flip Flop | 51 | 0 | 104400 | 0.05 |
37: | Register as Latch | 0 | 0 | 104400 | 0.00 |
38: | F7 Muxes | 0 | 0 | 26600 | 0.00 |
39: | FB Muxes | 0 | 0 | 13300 | 0.00 |
40: +-----+
41: -----
42: -----
43: -----
44: -----
45: 1.1 Summary of Registers by Type
46: -----
47: -----
48: -----
49: +-----+ Total | Clock Enable | Synchronous | Asynchronous +-----+
50: | 0 | 0 | - | - |
51: | 0 | 0 | - | - |
52: | 0 | 0 | - | Set |
53: | 0 | 0 | - | Reset |
54: | 0 | 0 | - | Set |
55: | 0 | 0 | - | Reset |
56: | 0 | 0 | Yes | - |
57: | 0 | 0 | Yes | Set |
58: | 0 | 0 | Yes | Reset |
59: | 0 | 0 | Yes | Set |
60: | 0 | 0 | Yes | Reset |
61: +-----+
62: -----
63: -----
64: 2. Slice Logic Distribution
65: -----
66: -----
67: +-----+ Site Type + Used | Fixed | Available | Util% +-----+
68: | 0 | Site Type + Used | Fixed | Available | Util% |
69: +-----+
70: | Slice | 16 | 0 | 13300 | 0.12 |
71: | SLICE | 6 | 0 | 1 | 1 |
72: | SLICES | 10 | 0 | 1 | 1 |
73: | LUT as Logic | 18 | 0 | 53200 | 0.03 |
74: | using CT output only | 0 | 0 | 1 | 1 |
75: | using CE output only | 11 | 1 | 1 | 1 |
76: | using OS and OE | 7 | 1 | 1 | 1 |
77: | using OS and distributed RAM | 0 | 0 | 1 | 1 |
78: | LUT as Memory | 0 | 0 | 17400 | 0.00 |
79: | LUT as a distributed RAM | 0 | 0 | 1 | 1 |
80: | LUT as Shift Register | 0 | 0 | 1 | 1 |
81: | LUT Flip Flop Pairs | 13 | 0 | 53200 | 0.02 |
82: | fully used LUT-FF pairs | 3 | 1 | 1 | 1 |
83: | LUT-FF pairs with one unused LUT output | 9 | 1 | 1 | 1 |
84: | LUT-FF pairs with one unused Flip Flop | 5 | 1 | 1 | 1 |
85: | Unique Control Sets | 1 | 1 | 1 | 1 |
86: +-----+
87: * Note: Review the Control Sets Report for more information regarding control sets.
88: -----
89: -----
90: 3. Memory
91: -----
92: -----
93: +-----+ Site Type + Used | Fixed | Available | Util% +-----+
94: | 0 | Site Type + Used | Fixed | Available | Util% |
95: +-----+
96: | Block RAM Tile | 0 | 0 | 140 | 0.00 |
97: | RAMB34/FIFO* | 0 | 0 | 140 | 0.00 |
98: | RAMB16 | 0 | 0 | 280 | 0.00 |
99: +-----+
100: * Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36E1 or one FIFO16E1. However, if a FIFO16E1 occupies a Block RAM Tile, that tile can still accommodate a RAMB16E1.
101: -----
102: -----
103: 4. Primitives
104: -----
105: -----
106: +-----+ Ref Name + Used | Functional Category +-----+
107: | 0 | Ref Name + Used | Functional Category |
108: +-----+
109: | I_FORE | 51 | Flop & Latch |
110: | I_BUF | 25 | IO |
111: | I_BUFG | 10 | IO |
112: | LUT5 | 8 | LUT |
113: | LUT6 | 7 | LUT |
114: | LUT7 | 4 | LUT |
115: | LUT8 | 4 | LUT |
116: | LUT9 | 2 | LUT |
117: | I_BOTG | 1 | CLOCK |
118: +-----+
119: -----
120: -----
121: -----
122: -----
123: -----
124: -----
125: -----
126: -----
127: -----
128: -----
129: -----
130: -----
131: -----
132: -----
133: -----
134: -----
135: -----
136: -----
137: -----
138: -----
139: -----
140: -----
141: -----
142: -----
143: -----
144: -----
145: -----
146: -----
147: -----
148: -----
149: -----
150: -----
151: -----
152: -----
153: -----
154: -----
155: -----
156: -----
157: -----
158: -----
159: -----
160: -----
161: -----
162: -----
163: -----
164: -----
165: -----
166: -----
167: -----
168: -----
169: -----
170: -----
171: -----
172: -----
173: -----
174: -----
175: -----
176: -----
177: -----
178: -----
179: -----
180: -----
181: -----
182: -----
183: -----
184: -----
185: -----
186: -----
187: -----
188: -----
189: -----
190: -----
191: -----
192: -----
193: -----
194: -----
195: -----
196: -----
197: -----
198: -----
199: -----
200: -----
201: -----
202: -----
203: -----
204: -----
205: -----
206: -----
207: -----
208: -----
209: -----
210: -----
211: -----
212: -----
213: -----
214: -----
215: -----
216: -----
217: -----
218: -----
219: -----
220: -----
221: -----
222: -----
223: -----
224: -----
225: -----
226: -----
227: -----
228: -----
229: -----
230: -----
231: -----
232: -----
233: -----
234: -----
235: -----
236: -----
237: -----
238: -----
239: -----
240: -----
241: -----
242: -----
243: -----
244: -----
245: -----
246: -----
247: -----
248: -----
249: -----
250: -----
251: -----
252: -----
253: -----
254: -----
255: -----
256: -----
257: -----
258: -----
259: -----
260: -----
261: -----
262: -----
263: -----
264: -----
265: -----
266: -----
267: -----
268: -----
269: -----
270: -----
271: -----
272: -----
273: -----
274: -----
275: -----
276: -----
277: -----
278: -----
279: -----
280: -----
281: -----
282: -----
283: -----
284: -----
285: -----
286: -----
287: -----
288: -----
289: -----
290: -----
291: -----
292: -----
293: -----
294: -----
295: -----
296: -----
297: -----
298: -----
299: -----
300: -----
301: -----
302: -----
303: -----
304: -----
305: -----
306: -----
307: -----
308: -----
309: -----
310: -----
311: -----
312: -----
313: -----
314: -----
315: -----
316: -----
317: -----
318: -----
319: -----
320: -----
321: -----
322: -----
323: -----
324: -----
325: -----
326: -----
327: -----
328: -----
329: -----
330: -----
331: -----
332: -----
333: -----
334: -----
335: -----
336: -----
337: -----
338: -----
339: -----
340: -----
341: -----
342: -----
343: -----
344: -----
345: -----
346: -----
347: -----
348: -----
349: -----
350: -----
351: -----
352: -----
353: -----
354: -----
355: -----
356: -----
357: -----
358: -----
359: -----
360: -----
361: -----
362: -----
363: -----
364: -----
365: -----
366: -----
367: -----
368: -----
369: -----
370: -----
371: -----
372: -----
373: -----
374: -----
375: -----
376: -----
377: -----
378: -----
379: -----
380: -----
381: -----
382: -----
383: -----
384: -----
385: -----
386: -----
387: -----
388: -----
389: -----
390: -----
391: -----
392: -----
393: -----
394: -----
395: -----
396: -----
397: -----
398: -----
399: -----
400: -----
401: -----
402: -----
403: -----
404: -----
405: -----
406: -----
407: -----
408: -----
409: -----
410: -----
411: -----
412: -----
413: -----
414: -----
415: -----
416: -----
417: -----
418: -----
419: -----
420: -----
421: -----
422: -----
423: -----
424: -----
425: -----
426: -----
427: -----
428: -----
429: -----
430: -----
431: -----
432: -----
433: -----
434: -----
435: -----
436: -----
437: -----
438: -----
439: -----
440: -----
441: -----
442: -----
443: -----
444: -----
445: -----
446: -----
447: -----
448: -----
449: -----
450: -----
451: -----
452: -----
453: -----
454: -----
455: -----
456: -----
457: -----
458: -----
459: -----
460: -----
461: -----
462: -----
463: -----
464: -----
465: -----
466: -----
467: -----
468: -----
469: -----
470: -----
471: -----
472: -----
473: -----
474: -----
475: -----
476: -----
477: -----
478: -----
479: -----
480: -----
481: -----
482: -----
483: -----
484: -----
485: -----
486: -----
487: -----
488: -----
489: -----
490: -----
491: -----
492: -----
493: -----
494: -----
495: -----
496: -----
497: -----
498: -----
499: -----
500: -----
501: -----
502: -----
503: -----
504: -----
505: -----
506: -----
507: -----
508: -----
509: -----
510: -----
511: -----
512: -----
513: -----
514: -----
515: -----
516: -----
517: -----
518: -----
519: -----
520: -----
521: -----
522: -----
523: -----
524: -----
525: -----
526: -----
527: -----
528: -----
529: -----
530: -----
531: -----
532: -----
533: -----
534: -----
535: -----
536: -----
537: -----
538: -----
539: -----
540: -----
541: -----
542: -----
543: -----
544: -----
545: -----
546: -----
547: -----
548: -----
549: -----
550: -----
551: -----
552: -----
553: -----
554: -----
555: -----
556: -----
557: -----
558: -----
559: -----
560: -----
561: -----
562: -----
563: -----
564: -----
565: -----
566: -----
567: -----
568: -----
569: -----
570: -----
571: -----
572: -----
573: -----
574: -----
575: -----
576: -----
577: -----
578: -----
579: -----
580: -----
581: -----
582: -----
583: -----
584: -----
585: -----
586: -----
587: -----
588: -----
589: -----
590: -----
591: -----
592: -----
593: -----
594: -----
595: -----
596: -----
597: -----
598: -----
599: -----
600: -----
601: -----
602: -----
603: -----
604: -----
605: -----
606: -----
607: -----
608: -----
609: -----
610: -----
611: -----
612: -----
613: -----
614: -----
615: -----
616: -----
617: -----
618: -----
619: -----
620: -----
621: -----
622: -----
623: -----
624: -----
625: -----
626: -----
627: -----
628: -----
629: -----
630: -----
631: -----
632: -----
633: -----
634: -----
635: -----
636: -----
637: -----
638: -----
639: -----
640: -----
641: -----
642: -----
643: -----
644: -----
645: -----
646: -----
647: -----
648: -----
649: -----
650: -----
651: -----
652: -----
653: -----
654: -----
655: -----
656: -----
657: -----
658: -----
659: -----
660: -----
661: -----
662: -----
663: -----
664: -----
665: -----
666: -----
667: -----
668: -----
669: -----
670: -----
671: -----
672: -----
673: -----
674: -----
675: -----
676: -----
677: -----
678: -----
679: -----
680: -----
681: -----
682: -----
683: -----
684: -----
685: -----
686: -----
687: -----
688: -----
689: -----
690: -----
691: -----
692: -----
693: -----
694: -----
695: -----
696: -----
697: -----
698: -----
699: -----
700: -----
701: -----
702: -----
703: -----
704: -----
705: -----
706: -----
707: -----
708: -----
709: -----
710: -----
711: -----
712: -----
713: -----
714: -----
715: -----
716: -----
717: -----
718: -----
719: -----
720: -----
721: -----
722: -----
723: -----
724: -----
725: -----
726: -----
727: -----
728: -----
729: -----
730: -----
731: -----
732: -----
733: -----
734: -----
735: -----
736: -----
737: -----
738: -----
739: -----
740: -----
741: -----
742: -----
743: -----
744: -----
745: -----
746: -----
747: -----
748: -----
749: -----
750: -----
751: -----
752: -----
753: -----
754: -----
755: -----
756: -----
757: -----
758: -----
759: -----
760: -----
761: -----
762: -----
763: -----
764: -----
765: -----
766: -----
767: -----
768: -----
769: -----
770: -----
771: -----
772: -----
773: -----
774: -----
775: -----
776: -----
777: -----
778: -----
779: -----
780: -----
781: -----
782: -----
783: -----
784: -----
785: -----
786: -----
787: -----
788: -----
789: -----
790: -----
791: -----
792: -----
793: -----
794: -----
795: -----
796: -----
797: -----
798: -----
799: -----
800: -----
801: -----
802: -----
803: -----
804: -----
805: -----
806: -----
807: -----
808: -----
809: -----
810: -----
811: -----
812: -----
813: -----
814: -----
815: -----
816: -----
817: -----
818: -----
819: -----
820: -----
821: -----
822: -----
823: -----
824: -----
825: -----
826: -----
827: -----
828: -----
829: -----
830: -----
831: -----
832: -----
833: -----
834: -----
835: -----
836: -----
837: -----
838: -----
839: -----
840: -----
841: -----
842: -----
843: -----
844: -----
845: -----
846: -----
847: -----
848: -----
849: -----
850: -----
851: -----
852: -----
853: -----
854: -----
855: -----
856: -----
857: -----
858: -----
859: -----
860: -----
861: -----
862: -----
863: -----
864: -----
865: -----
866: -----
867: -----
868: -----
869: -----
870: -----
871: -----
872: -----
873: -----
874: -----
875: -----
876: -----
877: -----
878: -----
879: -----
880: -----
881: -----
882: -----
883: -----
884: -----
885: -----
886: -----
887: -----
888: -----
889: -----
890: -----
891: -----
892: -----
893: -----
894: -----
895: -----
896: -----
897: -----
898: -----
899: -----
900: -----
901: -----
902: -----
903: -----
904: -----
905: -----
906: -----
907: -----
908: -----
909: -----
910: -----
911: -----
912: -----
913: -----
914: -----
915: -----
916: -----
917: -----
918: -----
919: -----
920: -----
921: -----
922: -----
923: -----
924: -----
925: -----
926: -----
927: -----
928: -----
929: -----
930: -----
931: -----
932: -----
933: -----
934: -----
935: -----
936: -----
937: -----
938: -----
939: -----
940: -----
941: -----
942: -----
943: -----
944: -----
945: -----
946: -----
947: -----
948: -----
949: -----
950: -----
951: -----
952: -----
953: -----
954: -----
955: -----
956: -----
957: -----
958: -----
959: -----
960: -----
961: -----
962: -----
963: -----
964: -----
965: -----
966: -----
967: -----
968: -----
969: -----
970: -----
971: -----
972: -----
973: -----
974: -----
975: -----
976: -----
977: -----
978: -----
979: -----
980: -----
981: -----
982: -----
983: -----
984: -----
985: -----
986: -----
987: -----
988: -----
989: -----
990: -----
991: -----
992: -----
993: -----
994: -----
995: -----
996: -----
997: -----
998: -----
999: -----
9999: -----
```



IMPLEMENTATION_1_PLACE_REPORT_UTILIZATION_NB = 16 BIT

```

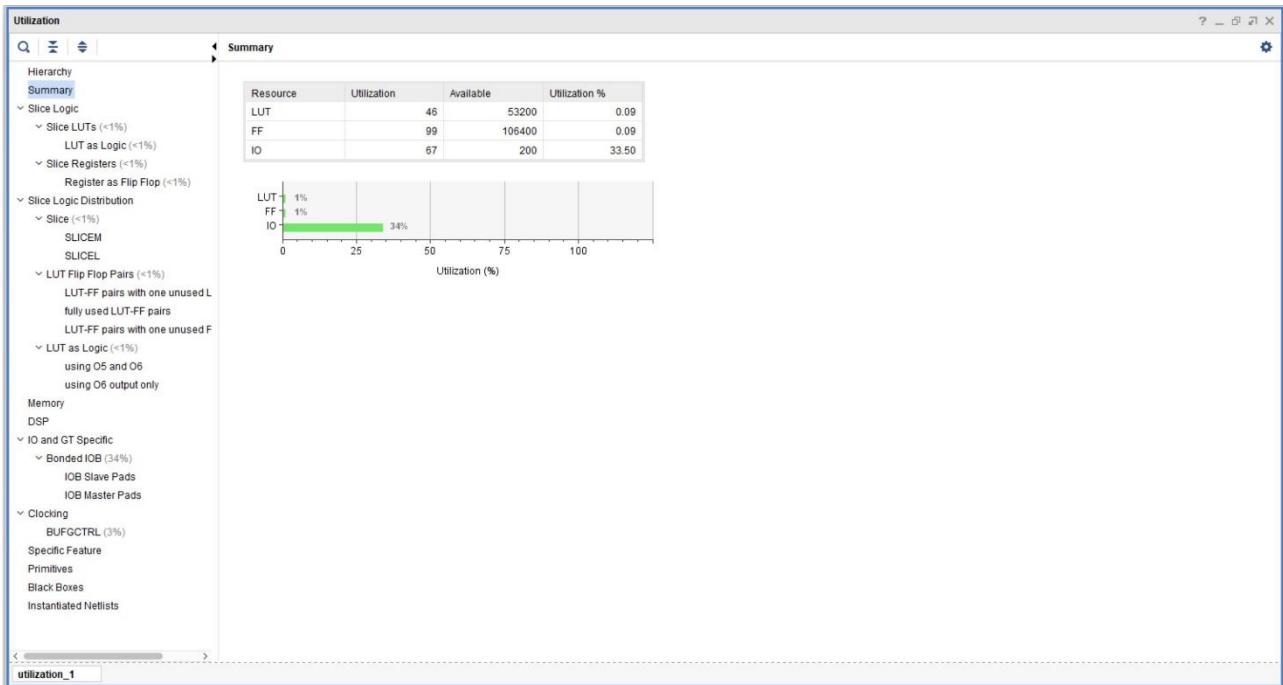
impl_1_place_report_utilization_0 - impl_1

D:\RACCOLTE\DOCUMENTI\lvado\vhdl\workspace\AdderSubtractor\AdderSubtractor.rns\impl_1\AddSub_utilization_place.rpt

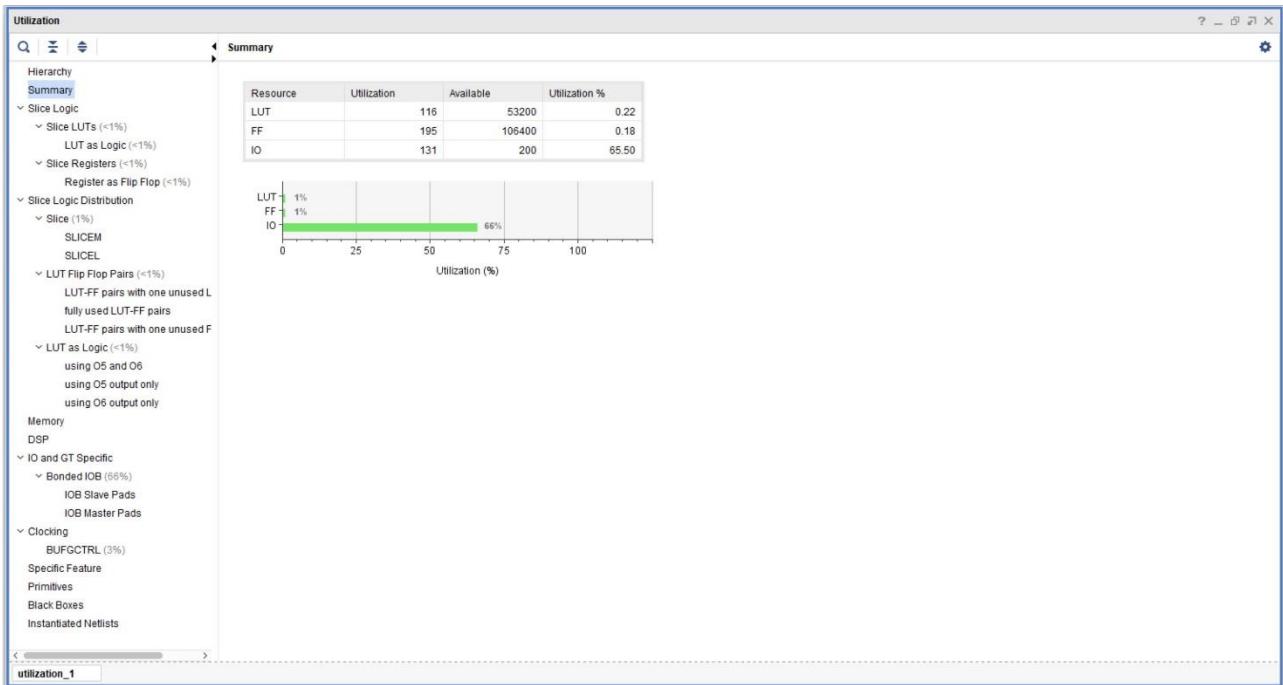
Q | < | > | X | E | File | Help | // | I | Q | Read-only | Settings | X

28 1. Slice Logic
29 -----
30
31 +-----+-----+-----+-----+
32 | Site Type | Used | Fixed | Available | Utiliz |
33 +-----+-----+-----+-----+
34 | SLICE LUT's | 44 | 0 | 53200 | 0.09 |
35 | LUT as Logic | 44 | 0 | 53200 | 0.09 |
36 | LUT as Memory | 0 | 0 | 17400 | 0.00 |
37 | Slices Registers | 90 | 0 | 106400 | 0.09 |
38 | Registers as Flip Flop | 90 | 0 | 106400 | 0.09 |
39 | Registers as Latch | 0 | 0 | 106400 | 0.00 |
40 | FT Muxes | 0 | 0 | 26600 | 0.00 |
41 | T8 Muxes | 0 | 0 | 13300 | 0.00 |
42 +-----+-----+-----+-----+
43
44
45 1.1 Summary of Registers by Type
46 -----
47
48 +-----+-----+-----+-----+
49 | Total | Clock Enable | Synchronous | Asynchronous |
50 +-----+-----+-----+-----+
51 | 0 | - | - | - |
52 | 0 | - | - | Set |
53 | 0 | - | - | Reset |
54 | 0 | - | Set | - |
55 | 0 | - | Reset | - |
56 | 0 | Yes | - | - |
57 | 0 | Yes | - | Set |
58 | 0 | Yes | - | Reset |
59 | 0 | Yes | Set | - |
60 | 99 | Yes | Reset | - |
61 +-----+-----+-----+-----+
62
63
64 2. Slice Logic Distribution
65 -----
66
67 +-----+-----+-----+-----+
68 | Site Type | Used | Fixed | Available | Utiliz |
69 +-----+-----+-----+-----+
70 | SLICE | 32 | 0 | 13300 | 0.24 |
71 | SLICEL | 22 | 0 | 0 | 0 |
72 | SLICEM | 10 | 0 | 0 | 0 |
73 | LUT as Logic | 44 | 0 | 53200 | 0.09 |
74 | using 04 output only | 0 | 0 | 0 | 0 |
75 | using 04 output only | 31 | 0 | 0 | 0 |
76 | using 05 and 06 | 15 | 0 | 0 | 0 |
77 | LUT as Memory | 0 | 0 | 17400 | 0.00 |
78 | LUT as Distributed RAM | 0 | 0 | 0 | 0 |
79 | LUT as Shift Register | 0 | 0 | 0 | 0 |
80 | LUT RAM Pairs | 22 | 0 | 53200 | 0.04 |
81 | fully used LUT-FF pairs | 4 | 0 | 0 | 0 |
82 | LUT-FF pairs with one unused LUT output | 16 | 0 | 0 | 0 |
83 | LUT-FF pairs with one unused Flip Flop | 6 | 0 | 0 | 0 |
84 | Unique Control Sets | 1 | 0 | 0 | 0 |
85 +-----+-----+-----+-----+
86 * Note: Review the Control Sets Report for more information regarding control sets.
87
88
89 3. Memory
90 -----
91
92 +-----+-----+-----+-----+
93 | Site Type | Used | Fixed | Available | Utiliz |
94 +-----+-----+-----+-----+
95 | Block RAM Tile | 0 | 0 | 140 | 0.00 |
96 | RAM36x1FIFO | 0 | 0 | 140 | 0.00 |
97 | RAM24 | 0 | 0 | 240 | 0.00 |
98 +-----+-----+-----+-----+
99 * Note: Each Block RAM Tile only has one FIFO logic available and therefore can accommodate only one FIFO36El or one FIFO18El. However, if a FIFO18El occupies a Block RAM Tile, that tile can still accommodate a RAMB18El
100
101
102 4. Primitives
103 -----
104
105 +-----+-----+-----+
106 | Ref Name | Used | Functional Category |
107 +-----+-----+-----+
108 | FDRE | 99 | Flip flop & Latch |
109 | IBRD | 49 | IO |
110 | LUT5 | 31 | LUT |
111 | LUT4 | 31 | LUT |
112 | LUT3 | 12 | LUT |
113 | LUT6 | 10 | LUT |
114 | LUT2 | 6 | LUT |
115 | LUT4 | 2 | LUT |
116 | BORG | 1 | Clock |
117 +-----+-----+-----+
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187 <

```



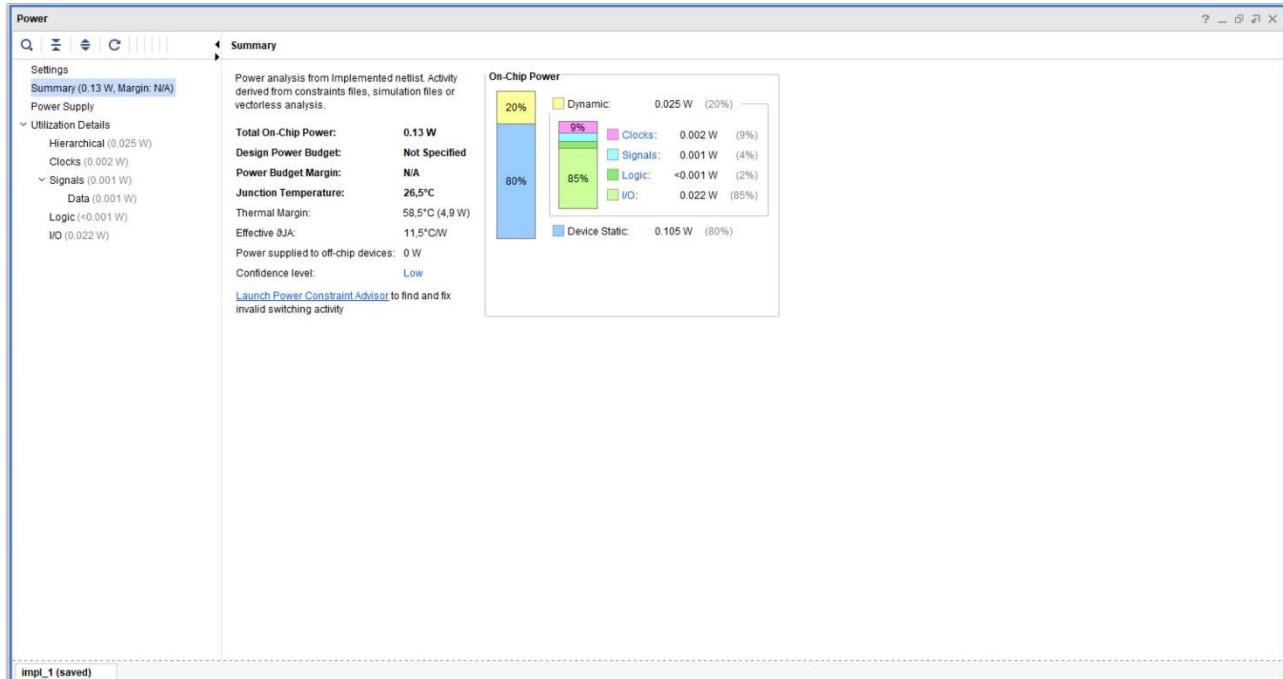
IMPLEMENTATION_1_PLACE_REPORT_UTILIZATION NB = 32 BIT



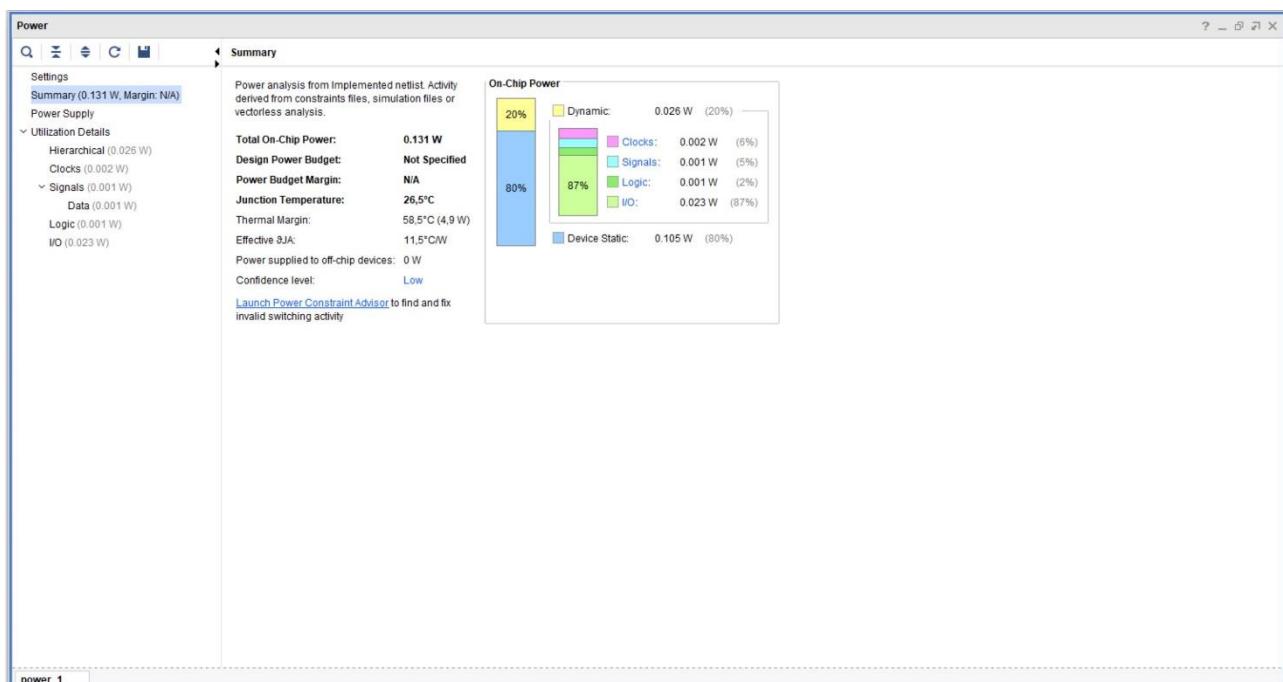
Dissipazione di potenza

(Implementation > Report Power)

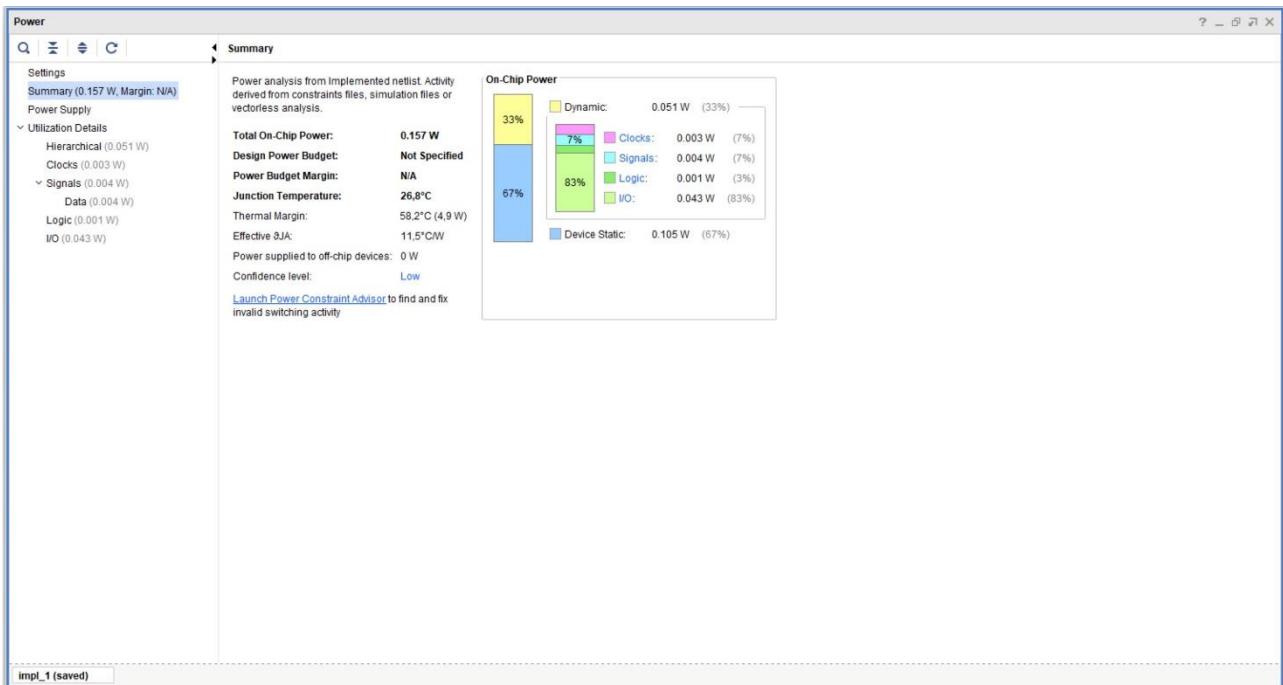
NB = 8 BIT



NB = 16 BIT



NB = 32 BIT



SimAddSub

Per eseguire un TEST bisogna scrivere un altro file VHDL che prende il nome di **TEST BENCH**, ovvero un file dedicato alle simulazioni che non è né sintetizzabile né implementabile (nel nostro caso è chiamato **SimAddSub**).

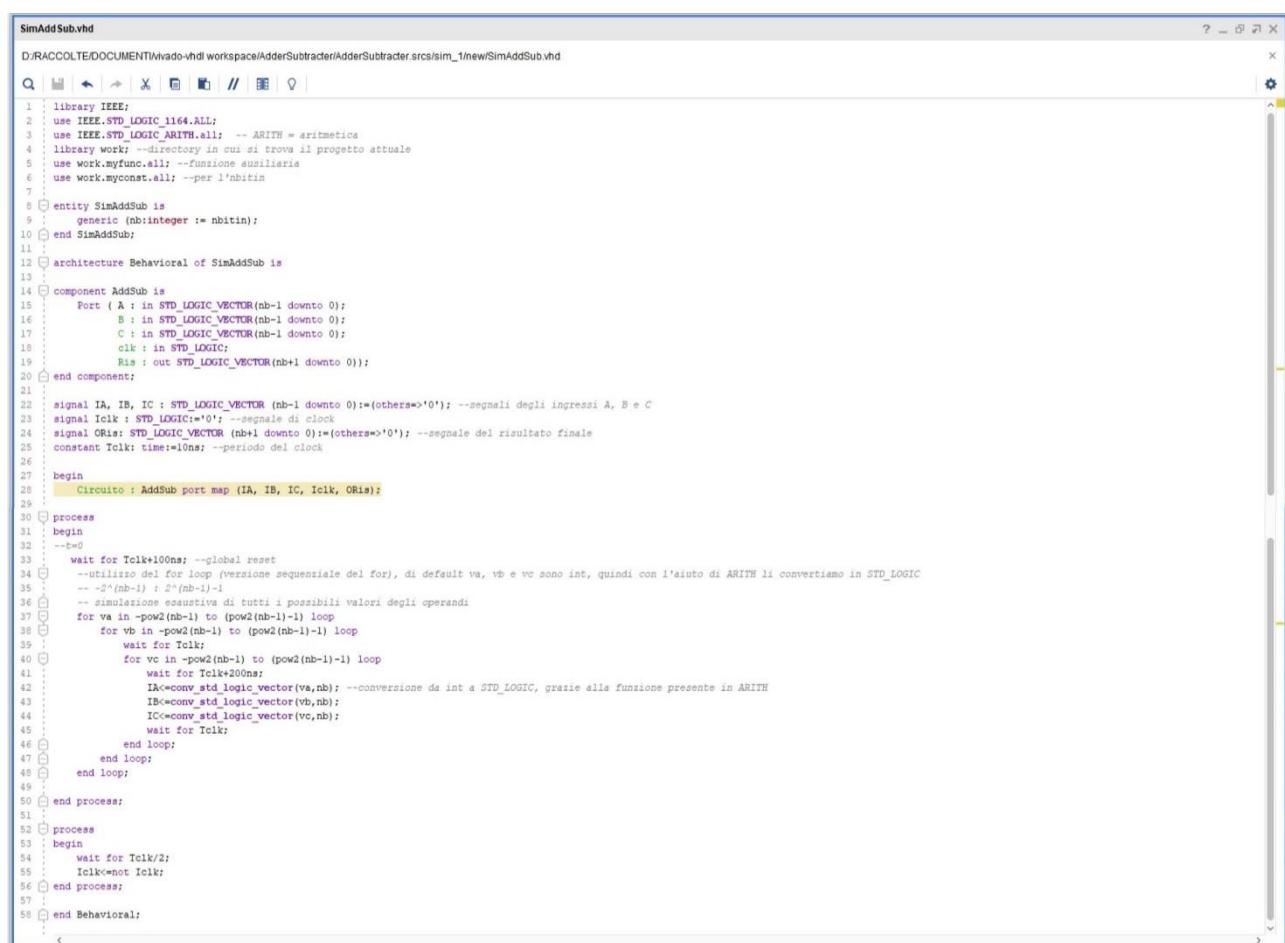
Il **TEST BENCH** gestisce la variazione degli ingressi nel tempo.

La keyword **wait for** si utilizza per attendere un determinato tempo prima di proseguire con le istruzioni.

La keyword **others** si utilizza per assegnazioni indipendenti dalla grandezza del vettore.

I tre **for** innestati coprono tutte le combinazioni possibili degli operandi.

Codice



```

SimAddSub.vhd
D:\RACCOLTE\DOCUMENTI\ivado-vhdl\workspace\AdderSubtractor\AdderSubtractor.srsc\sim_1\new\SimAddSub.vhd

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_UNSIGNED.all; -- ARITH = aritmetica
library work; --directory in cui si trova il progetto attuale
use work.myfunc.all; --funzione ausiliaria
use work.myconst.all; --per l'abitazione dei costanti
entity SimAddSub is
    generic (nb:integer := nbbitin);
end SimAddSub;
architecture Behavioral of SimAddSub is
component AddSub is
    Port ( A : in STD_LOGIC_VECTOR(nb-1 downto 0);
           B : in STD_LOGIC_VECTOR(nb-1 downto 0);
           C : in STD_LOGIC_VECTOR(nb-1 downto 0);
           clk : in STD_LOGIC;
           Ris : out STD_LOGIC_VECTOR(nb+1 downto 0));
end component;
begin
    Circuito : AddSub port map (IA, IB, IC, Iclk, ORis);
process
begin
    --t=0
    wait for Tclk+100ns; --global reset
    --utilizzo del for loop (versione sequenziale del for), di default va, vb e vc sono int, quindi con l'aiuto di ARITH li convertiamo in STD_LOGIC
    -- -2^(nb-1) : 2^(nb-1)-1
    -- simulazione esauritiva di tutti i possibili valori degli operandi
    for va in -pow2(nb-1) to (pow2(nb-1)-1) loop
        for vb in -pow2(nb-1) to (pow2(nb-1)-1) loop
            wait for Tclk;
            for vc in -pow2(nb-1) to (pow2(nb-1)-1) loop
                wait for Tclk400ns;
                IA=>conv_std_logic_vector(va,nb); --conversione da int a STD_LOGIC, grazie alla funzione presente in ARITH
                IB=>conv_std_logic_vector(vb,nb);
                IC=>conv_std_logic_vector(vc,nb);
                wait for Tclk;
            end loop;
        end loop;
    end loop;
end process;
process
begin
    wait for Tclk/2;
    Iclk=>not Iclk;
end process;
end Behavioral;

```

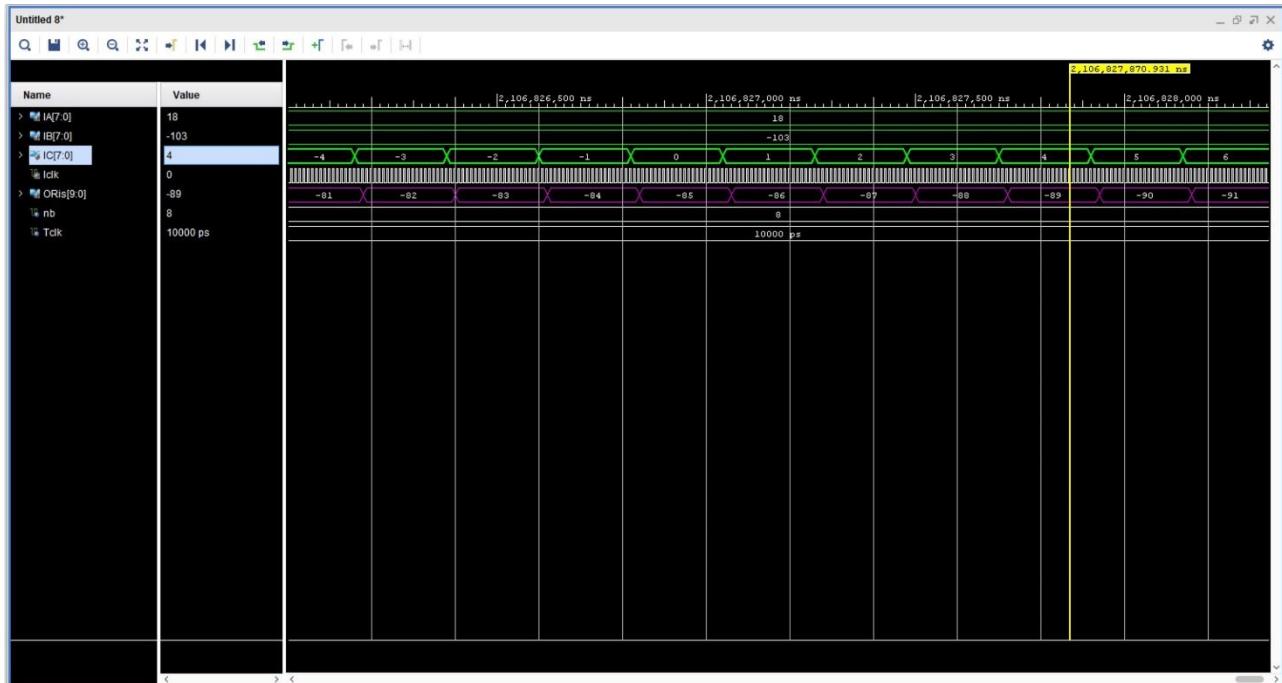
N.B. Per quanto riguarda il Testbench riferito al caso nb = 32, si è dovuto ridurre il range del ciclo **for**, poiché il tipo **integer** può assumere valori da $-(2^{31} - 1)$ a $(2^{31} + 1)$.

Pertanto, con il numero di bit pari a 32 questo range definito non viene rispettato.

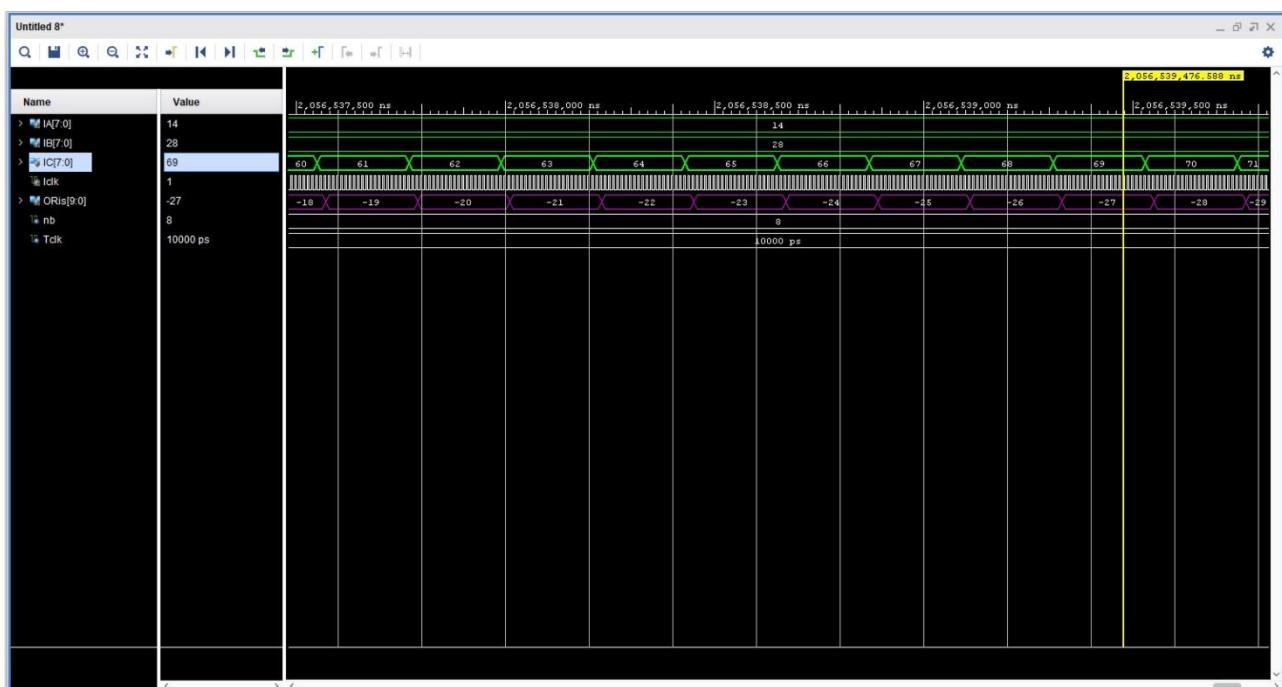
SCREENSHOTS DEI TEST

$nb = 8$ bit

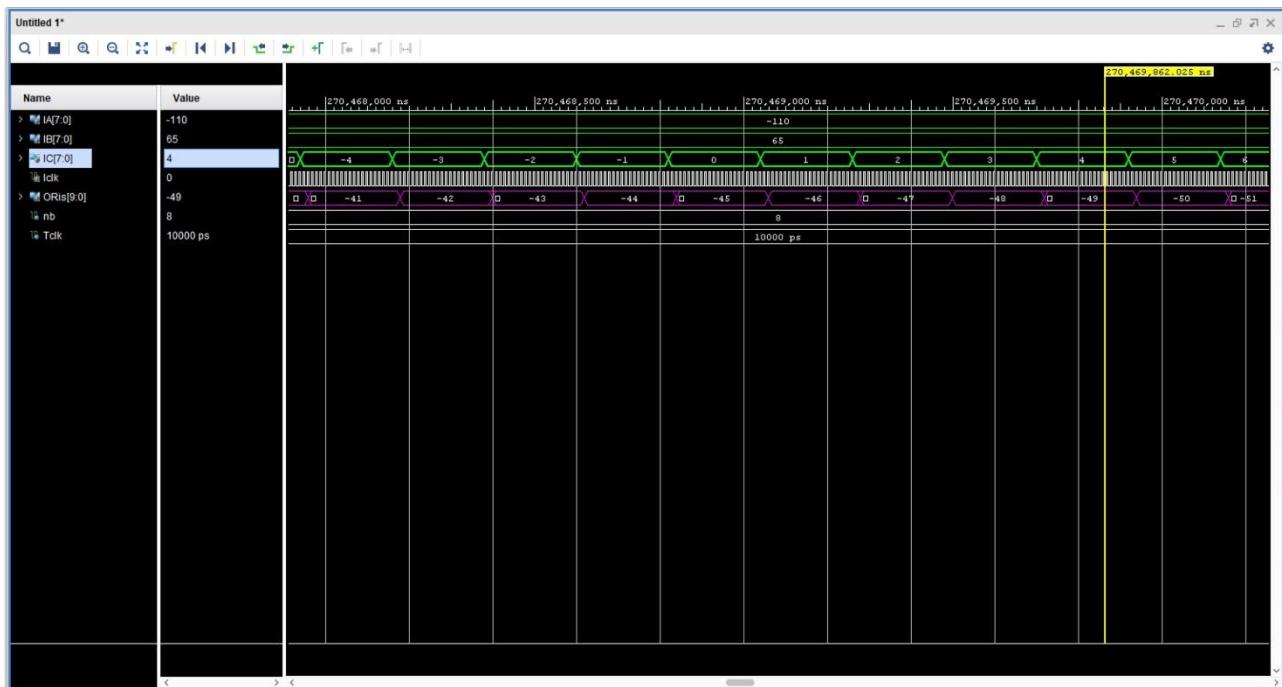
*Behavioral SIMULATION
 $(A) + (-B) - (-C)$ e $(A) + (-B) - (C)$*



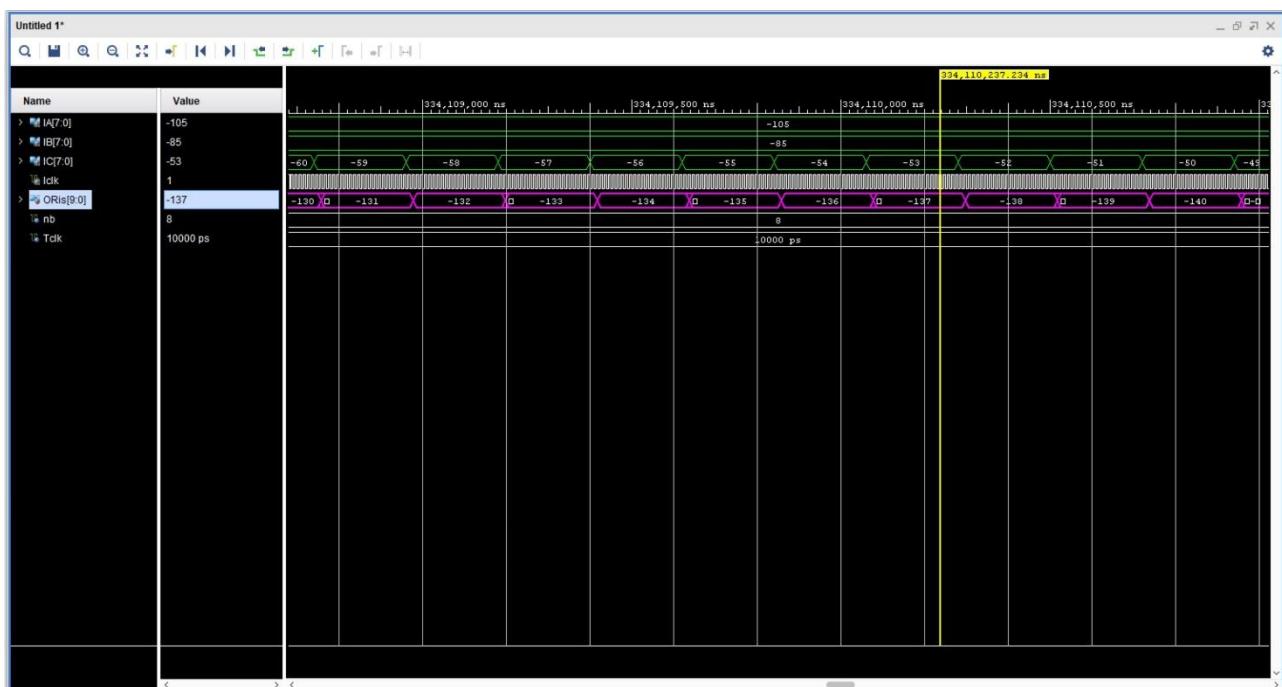
$(A) + (B) - (C)$



Post-Implementation Timing SIMULATION (-A)+(B)-(-C) e (-A)+(B)-(C)

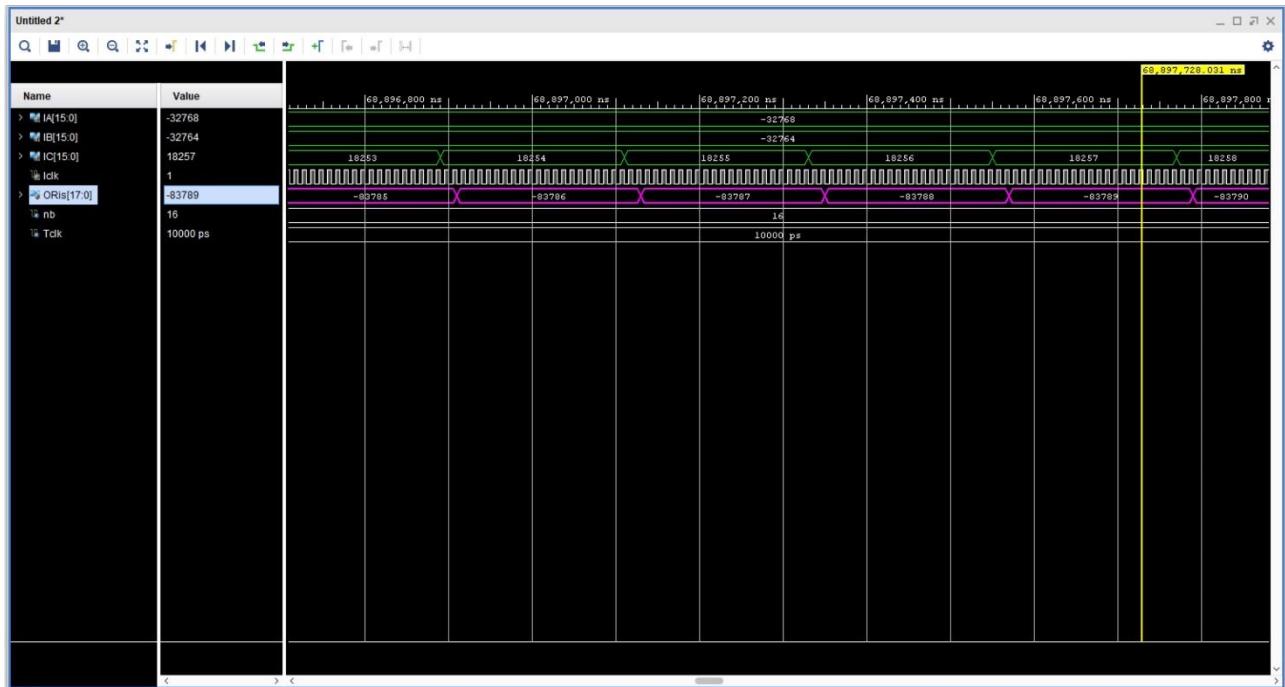


(-A)+(B)-(-C)

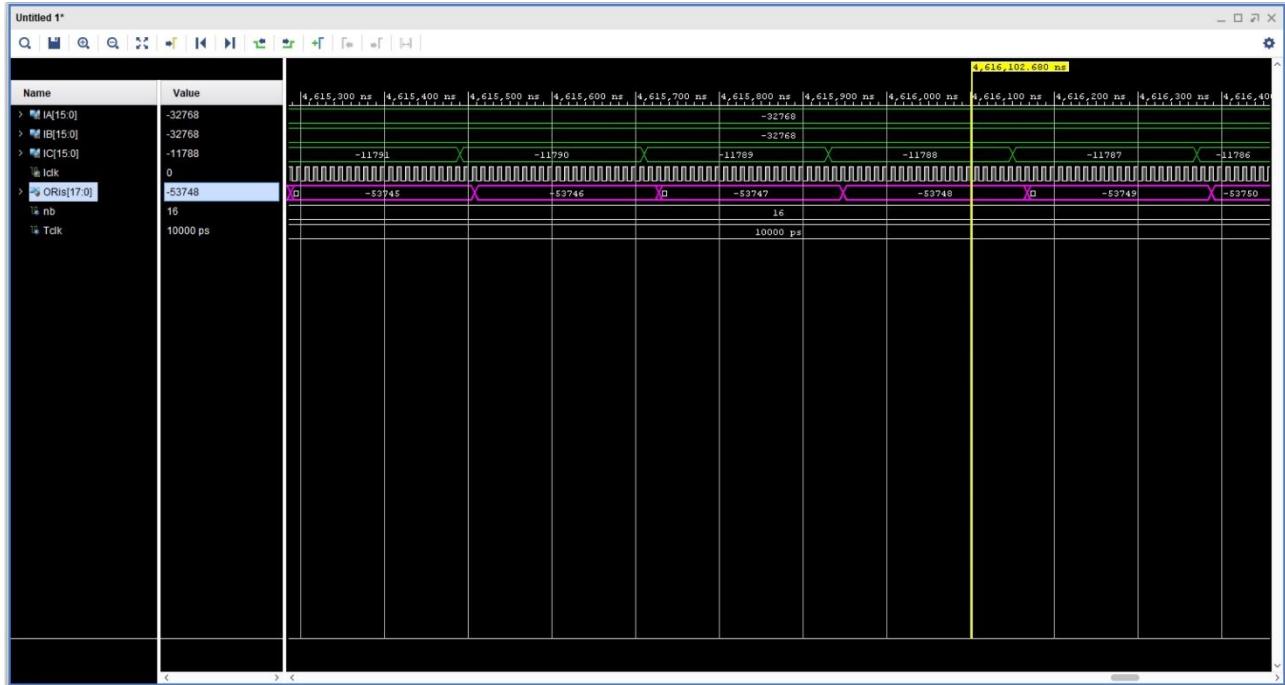


$n_b = 16 \text{ bit}$

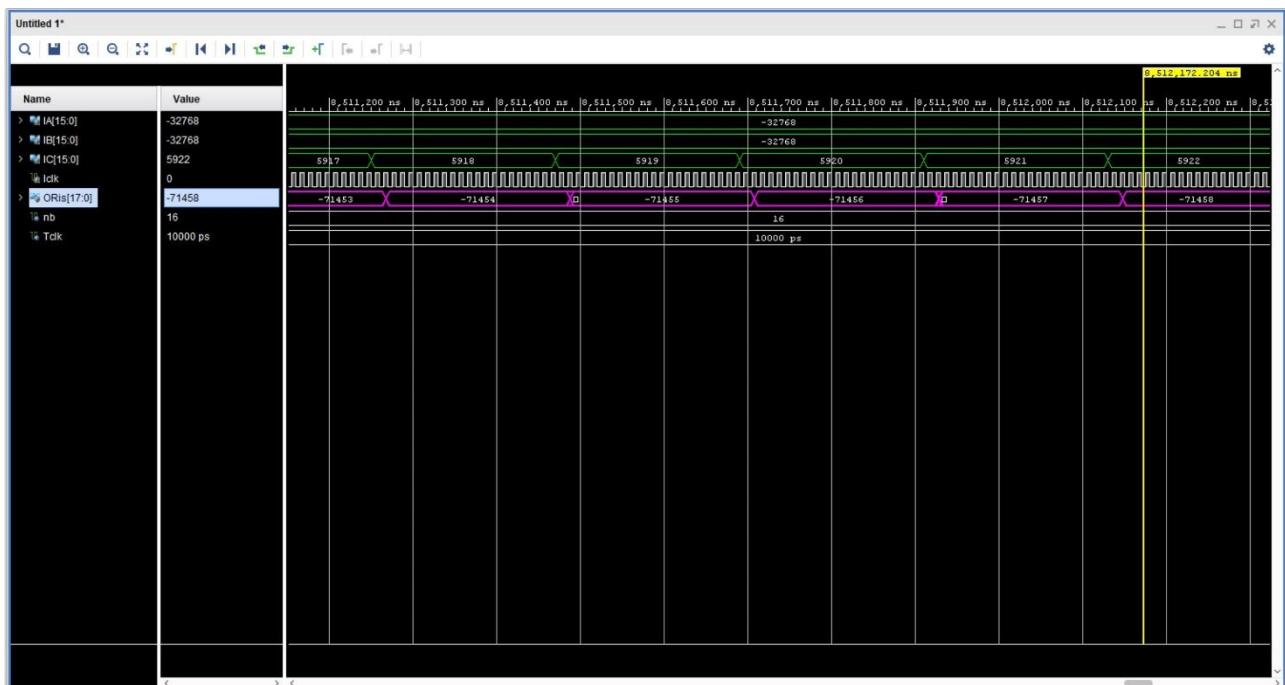
*Behavioral SIMULATION
(-A)+(-B)-(C)*



Post-Implementation Timing SIMULATION (-A)+(-B)-(-C)

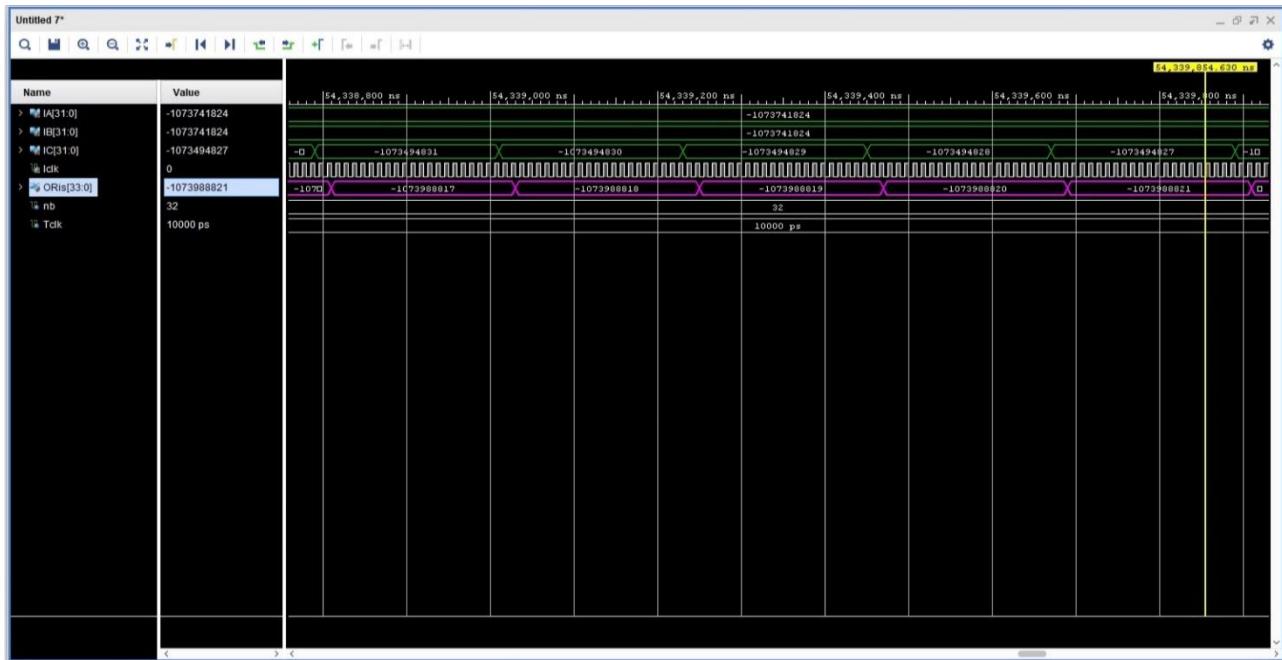


(-A)+(-B)-(C)



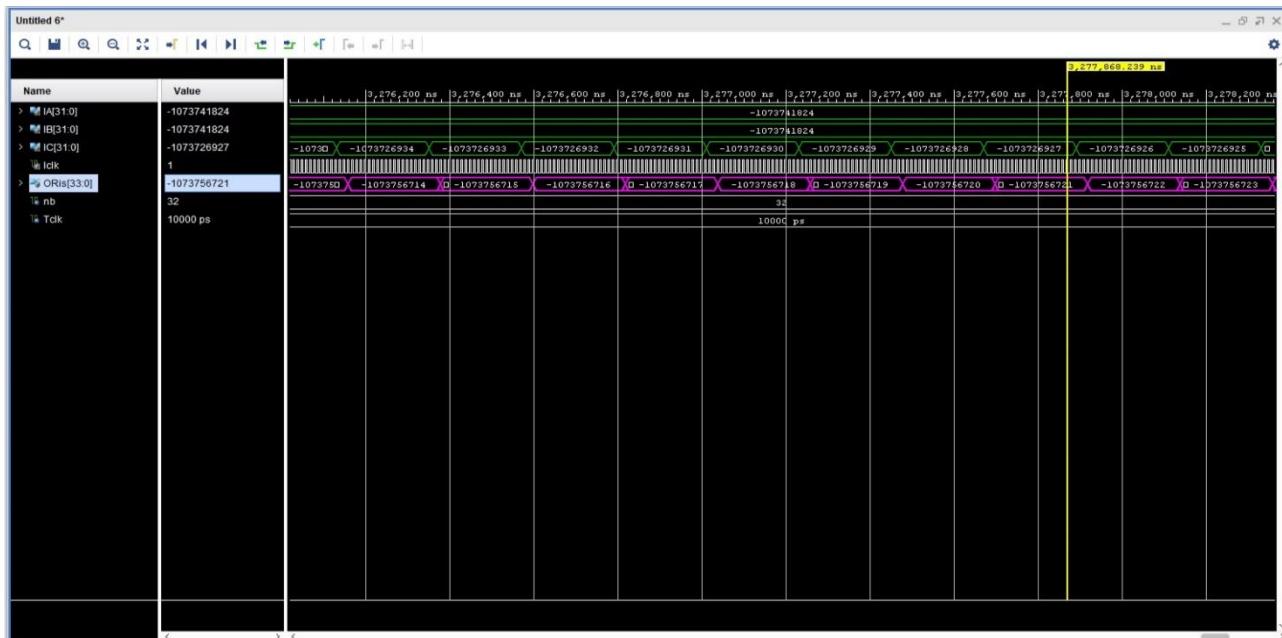
$nb = 32$ bit

Behavioral SIMULATION (-A)+(-B)-(-C)



Post-Implementation Timing SIMULATION

(-A)+(-B)-(-C)



Progetto realizzato da:
Michele Purrone
Antonino Vaccarella