



Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Ingegneria Informatica

***RILEVATORE DI SPIKES
ALL'INTERNO DI UN
SEGNALE PROVENIENTE DA
UN SENSORE***

Anno Accademico 2024/2025

Studenti
Colaleo Corrado matr. M63001762
De Rosa Francesca matr. M63001811

Indice

1	Frontend	1
1.1	IIR	1
1.2	Enfasi	8
1.3	Detector	19
2	Backend	28
2.1	IIR	30
2.1.1	Sintesi	30
2.1.2	Floorplan	34
2.1.3	Placement	37
2.1.4	Clock Tree	39
2.1.5	Routing	40
2.1.6	Controlli finali	42
2.2	Enfasi	43
2.2.1	Sintesi	43
2.2.2	Floorplan	47
2.2.3	Placement	48
2.2.4	Clock Tree	50
2.2.5	Routing	52

2.2.6	Controlli finali	53
2.3	Detector	55
2.3.1	Sintesi	55
2.3.2	Floorplan	58
2.3.3	Placement	59
2.3.4	Clock Tree	61
2.3.5	Routing	62
2.3.6	Controlli finali	63

Chapter 1

Frontend

1.1 IIR

Il primo modulo del nostro progetto è un filtro IIR passa basso, progettato per attenuare il rumore presente nel segnale di ingresso proveniente da un sensore esterno. Assumiamo che i dati siano campionati a una frequenza di 100 MHz, che corrisponderà quindi anche alla frequenza del clock che governa il sistema. Inoltre, assumiamo che la frequenza di taglio del filtro sia pari a 10 MHz. I dati in ingresso sono rappresentati in virgola fissa e assumono valori compresi tra $[-1 \text{ e } 1]$. Il bit meno significativo ha un peso di 2^{-10} . Un filtro IIR è descritto dalla seguente equazione:

$$y(n) + \sum_{k=1}^N a_k * y(n - k) = \sum_{k=0}^M b_k * x(n - k) \quad (1.1)$$

Possiamo notare che questo filtro può essere implementato utilizzando un'architettura ricorsiva, in cui l'uscita $y(n)$ dipende dalle uscite precedenti. Il comportamento del filtro (passa alto, passa basso ecc...) è determinato dalla selezione dei coefficienti a_k e b_k . Il filtro è stato implementato utilizzando sezioni del secondo ordine, ossia tramite blocchi che seguono la seguente equazione:

$$y(n) + a_1 y(n-1) + a_2 y(n-2) = b_0 x(n) + b_1 x(n-1) + b_2 x(n-2) \quad (1.2)$$

Utilizzeremo un sistema composto dalla cascata di due sezioni del secondo ordine, poiché una potrebbe non essere sufficiente. L'architettura che implementa l'equazione sopra riportata è illustrata in figura:

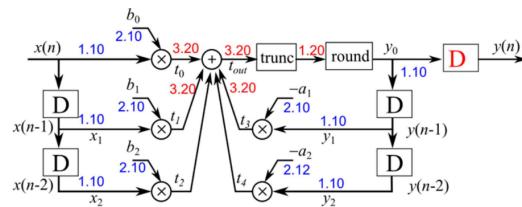


Figure 1.1: Architettura Filtro IIR

La sezione mostrata in figura è solo una delle due presenti nel nostro sistema. Utilizzeremo infatti due sezioni in cascata, identiche dal punto di vista architettonale, ma con valori differenti per i coefficienti a_k e b_k . I coefficienti, forniti da MATLAB, devono essere opportunamente modificati prima di essere utilizzati nella realizzazione hardware. In particolare:

- **Scalatura dei coefficienti:** tutti i coefficienti devono essere

moltiplicati per una stessa costante affinché il formato dei dati in ingresso e in uscita alle due sezioni del filtro sia lo stesso. Nel nostro caso, sia l'ingresso che l'uscita del filtro sono rappresentati nel formato **Q1.10**.

- **Quantizzazione dei coefficienti:** i coefficienti devono essere quantizzati. Per la rappresentazione dei coefficienti utilizziamo il formato **Q2.10**. La scalatura è stata scelta in modo tale che l'uscita y sia rappresentabile nello stesso formato dell'ingresso x , ossia:

$$x \rightarrow Q1.10$$

$$y \rightarrow Q1.10$$

$$a_i, b_i \rightarrow Q2.10$$

L'uscita dei moltiplicatori e dell'addizionatore, tuttavia, è rappresentata nel formato **Q3.20**, che differisce dal formato di y . Di conseguenza, dobbiamo applicare due operazioni fondamentali per adattare il formato dell'uscita:

- **Troncamento:** vengono eliminati i due bit più significativi a sinistra di *tout*. In uscita al blocco trunc abbiamo ancora una quantità con 20 cifre decimali, a noi servono soltanto 10, quindi è necessario effettuare un arrotondamento.
- **Arrotondamento:** viene effettuato in modo tale che il bit meno

significativo (LSB) corrisponda a 2^{-10} .

Esaminiamo ora il codice SystemVerilog del blocco IIR riportato di seguito.

```
1  `timescale 1ns / 1ps
2
3  module iir_section1 (input logic clk, rst,
4    input logic signed [10:0] x,
5    output logic signed [10:0] y);
6
7  localparam logic signed [11:0] b0=12'sb0000000011100;
8  localparam logic signed [11:0] b1=12'sb0000000100001;
9  localparam logic signed [11:0] b2=12'sb0000000011100;
10 localparam logic signed [11:0] ma1=12'sb011000101001; //m sta per meno
11 localparam logic signed [11:0] ma2=12'sb110101101110; //m sta per meno
12
13 localparam logic signed [20:0] round_const=21'sb00000000000010000000000;
14 logic signed [10:0] x1;
15 logic signed [10:0] x2;
16 logic signed [10:0] y0;
17 logic signed [10:0] y1;
18 logic signed [10:0] y2;
19 logic signed [22:0] t0;
20 logic signed [22:0] t1;
21 logic signed [22:0] t2;
22 logic signed [22:0] t3;
23 logic signed [22:0] t4;
24 logic signed [22:0] tout;
25 logic signed [20:0] t_troncato;
26 logic signed [20:0] temp;
27
```

Figure 1.2: iir_section1.sv

La figura mostra i valori dei coefficienti e della costante di arrotondamento, che dovrà essere sommata al segnale in uscita dal blocco di troncamento. Inoltre, vengono definiti alcuni segnali ausiliari, che saranno utilizzati per realizzare i collegamenti interni al modulo.

```
28  always_ff @(posedge clk)
29  begin
30      if (rst)
31          begin
32              x1<='0;
33              x2<='0;
34              y<='0;
35              y1<='0;
36              y2<='0;
37          end
38      else
39          begin
40              x1<=x;
41              x2<=x1;
42              y2<=y1;
43              y1<=y0;
44              y<=y0;
45          end
46  end
47
```

Figure 1.3: iir_section1.sv

In SystemVerilog, i registri sono definiti utilizzando il costrutto `always_ff`. Il blocco viene attivato sul fronte di salita del clock (`@posedge clk`) ed è dotato di un reset sincrono. Le operazioni eseguite dal modulo sono illustrate in figura.

```
48 assign t0 = b0*x;
49 assign t1= x1 * b1;
50 assign t2 = x2 * b2;
51 assign t3 = y1 * ma1;
52 assign t4 = y2 * ma2;
53 assign tout = t0 + t1 + t2 + t3 + t4;
54 assign t_troncato = signed '(tout[20:0]);
55 assign temp = t_troncato + round_const;
56 assign y0 = signed '(temp[20:10]);
57
58 endmodule
```

Figure 1.4: iir_section1.sv

Alla riga 54 viene eseguita l'operazione di troncamento, rimuovendo i due bit più significativi del segnale tout. Alle righe 55 e 56 viene applicata l'operazione di arrotondamento. Nella Figura è mostrato il top level del sistema.

```

1  `timescale 1ns / 1ps
2  module iir (input logic clk, rst,
3      input logic signed [10:0] x,
4      output logic signed [10:0] z);
5
6  // completare
7  logic signed [10:0] yreg;
8  logic signed [10:0] y_intermedia;
9
10 always_ff @(posedge clk)
11 begin
12     if (rst)
13         yreg <= '0;
14     else
15         yreg <= x;
16 end
17
18 iir_section1 iir1 (.clk(clk),.rst(rst),.x(yreg),.y(y_intermedia));
19
20 iir_section2 iir2 (.clk(clk),.rst(rst),.x(y_intermedia),.y(z));
21
22 endmodule

```

Figure 1.5: iir.sv

L'ingresso x del modulo IIR deve essere prima memorizzato in un registro, come illustrato nella Figura successiva. L'uscita di questo registro fungerà da ingresso per il modulo $iir1$. Successivamente, l'uscita del modulo $iir1$ (denominata $y_intermedia$) sarà utilizzata come ingresso per il blocco $iir2$.

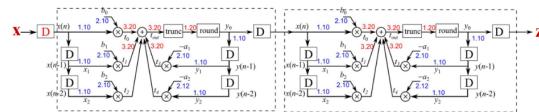


Figure 1.6:

1.2 Enfasi

Il secondo componente del nostro sistema è il blocco ENFASI, un circuito progettato per enfatizzare gli impulsi. Esso è composto dalla cascata di due sottoblocchi. Il primo blocco implementa l'operatore ASO, mentre il secondo effettua il filtraggio con una finestra di HAMMING.

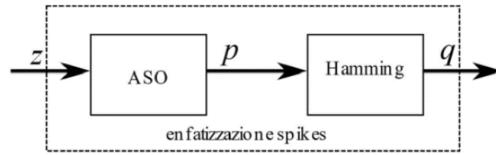


Figure 1.7: Struttura Enfatizzazione degli Spikes

L'operatore **ASO** genera l'uscita $p(n)$ a partire dalla sequenza $z(n)$ secondo la seguente relazione:

$$p(n) = z(n) \times [z(n) - z(n - k)]$$

Il parametro k deve essere scelto in base alla durata attesa degli *spikes* e alla frequenza di campionamento; nel nostro caso, utilizziamo $k = 4$. Di seguito è riportata l'implementazione hardware.

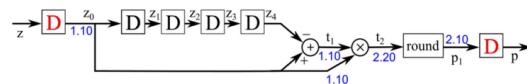


Figure 1.8: Schema a blocchi blocco ASO

Notiamo che l'uscita viene arrotondata per adattarsi al formato

Q2.10. Il codice SystemVerilog relativo al blocco ASO è riportato di seguito.

```
1  `timescale 1ns / 1ps
2
3  module aso (input logic clk, rst,
4    |      input logic signed [10:0] z,
5    |      output logic signed [11:0] p);
6
7  localparam logic signed [21:0] round_const=signed'({12'b0,1'b1, 9'b0});
8  logic signed [10:0] z0,z1,z2,z3,z4,t1;
9  logic signed [21:0] t2,temp;
10 logic signed [11:0] p1;
11
```

Figure 1.9: aso.sv

Il blocco ha come ingressi il segnale di **clock**, di **reset** e il segnale z , mentre il segnale q rappresenta l'uscita. Vengono inoltre definiti i seguenti segnali intermedi:

- **round_const**: una costante composta da 12 zeri, 1 uno e 9 zeri.
- $z_0, z_1, z_2, z_3, z_4, t_1$: segnali di tipo `logic` a 11 bit.
- $t_2, temp$: segnali di tipo `logic` a 22 bit.
- p_1 : segnale di tipo `logic` a 12 bit.

```
12  always_ff @ (posedge clk)
13  begin
14    if (rst)
15      begin
16        z0<='0;
17        z1<='0;
18        z2<='0;
19        z3<='0;
20        z4<='0;
21        p<='0;
22      end
23    else
24      begin
25        z0<=z;
26        z1<=z0;
27        z2<=z1;
28        z3<=z2;
29        z4<=z3;
30        p<=p1;
31      end
32  end
```

Figure 1.10: aso.sv

Abbiamo implementato il registro utilizzando un blocco `always_ff` con reset sincrono. Quando il segnale di reset è attivo e si verifica un

fronte attivo del clock, tutte le uscite dei flip-flop e dell'intero sistema vengono azzerate. Altrimenti, sul fronte attivo del segnale di clock, i valori vengono aggiornati come segue:

$$z_0 \leftarrow z$$

$$z_1 \leftarrow z_0$$

$$z_2 \leftarrow z_1$$

$$z_3 \leftarrow z_2$$

$$z_4 \leftarrow z_3$$

$$p \leftarrow p_1$$

```
34  always_comb
35  begin
36      t1 = z0-z4;
37      t2 = t1 * z0;
38      temp = t2 + round_const;
39      p1 = signed'(temp[21:10]);
40  end
41  endmodule
```

Figure 1.11: aso.sv

Il blocco `always_comb` mostrato in Figura implementa una serie di operazioni combinatorie per il calcolo del segnale di uscita. Il suo funzionamento può essere descritto nei seguenti passi:

1. **Calcolo della differenza:** Il segnale t_1 viene ottenuto come differenza tra z_0 e z_4 .
2. **Moltiplicazione:** Il segnale t_2 è calcolato moltiplicando t_1 per z_0 , producendo un valore intermedio.
3. **Somma con la costante di arrotondamento:** Il segnale temp viene ottenuto sommando t_2 con una costante di arrotondamento, che introduce un 1 nella decima posizione.
4. **Estrazione dei bit significativi:** Il segnale di uscita p_1 è derivato estraendo specifici bit da temp , selezionando quelli compresi tra la posizione 10 e la posizione 21.

L'uscita $p(n)$ generata dall'operatore ASO viene successivamente filtrata mediante un filtro FIR, che implementa la seguente funzione:

$$q(n) = b_0 p(n) + b_1 p(n-1) + b_2 p(n-2) + \cdots + b_L p(n-L+1)$$

I coefficienti b_i corrispondono a una finestra di lunghezza L . In questo caso, utilizziamo una finestra di Hamming con $L = 8$. I coefficienti della finestra di Hamming possono essere ottenuti in MATLAB. Questi coefficienti sono simmetrici, ovvero il primo valore è uguale all'ultimo, il secondo è uguale al penultimo e così via. Per garantire che l'uscita del filtro FIR sia rappresentabile nello stesso formato dell'ingresso, i coefficienti vengono opportunamente scalati e quantizzati. In partico-

lare, utilizziamo il formato numerico $Q0.10$, in cui tutti i coefficienti sono minori di uno. La realizzazione del filtro FIR avviene nel seguente modo:

- I valori come $p(n - 1)$, $p(n - 2)$, ecc., vengono memorizzati nei registri.
- Ogni valore memorizzato viene moltiplicato per il corrispondente coefficiente b_i .
- I prodotti ottenuti vengono infine sommati per ottenere l'uscita $q(n)$.

La Figura di seguito illustra la struttura del sistema. I coefficienti sono rappresentati nel formato $Q0.10$, mentre il risultato finale viene arrotondato al formato $Q2.10$. Si noti la presenza dei registri in ingresso e in uscita, che garantiscono la corretta sincronizzazione dei dati nel filtro FIR.

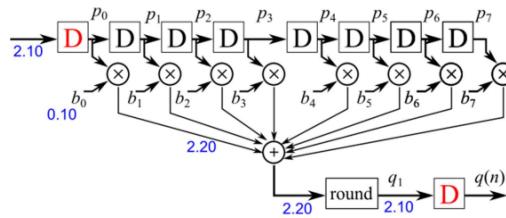


Figure 1.12: Filtro FIR

Il codice SystemVerilog del Filtro FIR è il seguente:

```
1  `timescale 1ns / 1ps
2
3  module fir (input logic clk, rst,
4      input logic signed [11:0] p,
5      output logic signed [11:0] q);
6
7  localparam logic signed [9:0] b0=10'sb0000010101;
8  localparam logic signed [9:0] b1=10'sb0001000011;
9  localparam logic signed [9:0] b2=10'sb0010101010;
10 localparam logic signed [9:0] b3=10'sb0011111101;
11 localparam logic signed [9:0] b4=10'sb0011111101;
12 localparam logic signed [9:0] b5=10'sb0010101010;
13 localparam logic signed [9:0] b6=10'sb0001000011;
14 localparam logic signed [9:0] b7=10'sb0000010101;
15
16 localparam logic signed [21:0] round_const=signed'({12'b0,1'b1, 9'b0});
17
18 logic signed [11:0] p0,p1,p2,p3,p4,p5,p6,p7,q1;
19 logic signed [21:0] r0,r1,r2,r3,r4,r5,r6,r7,t_sum,temp;
20
```

Figure 1.13: fir.sv

Il blocco di Hamming ha come ingressi il segnale di *clock*, *reset* e *p*, mentre in uscita fornisce il segnale *q*. Per realizzare le moltiplicazioni, sono stati definiti come parametri locali i coefficienti necessari. Inoltre, è stata introdotta una costante di arrotondamento composta da 12 zeri, seguiti da un 1 e da altri 9 zeri. Le seguenti grandezze sono state definite con specifiche dimensioni:

- I segnali p_0-p_7 e q_1 sono rappresentati con 12 bit.
- I segnali t_0-t_7 , t_sum e $temp$ sono rappresentati con 22 bit.

```
21  always_ff @ (posedge clk)
22  begin
23      if (rst)
24          begin
25              p0<='0;
26              p1<='0;
27              p2<='0;
28              p3<='0;
29              p4<='0;
30              p5<='0;
31              p6<='0;
32              p7<='0;
33              q<='0;
34          end
35      else
36          begin
37              p0<=p;
38              p1<=p0;
39              p2<=p1;
40              p3<=p2;
41              p4<=p3;
42              p5<=p4;
43              p6<=p5;
44              p7<=p6;
45              q<=q1;
46          end

```

Figure 1.14: fir.sv

In Figura è illustrata l'implementazione del registro mediante il costrutto

`always_ff`. Il comportamento del registro è il seguente:

- Se il segnale di *reset* è attivo (alto) e c'è il fronte attivo del *clock*, tutte le uscite dei *flip-flop* vengono azzerate.
- Altrimenti, sul fronte attivo del segnale di *clock*, i valori vengono aggiornati in modo sequenziale:
 - p_0 assume il valore di p .
 - p_1 assume il valore di p_0 .
 - p_2 assume il valore di p_1 , e così via fino a p_7 .
 - Infine, il segnale di uscita q assume il valore di q_1 .

```
49  always_comb
50  begin
51      r0 = b0 * p0;
52      r1 = b1 * p1;
53      r2 = b2 * p2;
54      r3 = b3 * p3;
55      r4 = b4 * p4;
56      r5 = b5 * p5;
57      r6 = b6 * p6;
58      r7 = b7 * p7;
59      t_sum = r0+r1+r2+r3+r4+r5+r6+r7;
60      temp = t_sum + round_const;
61      q1 = signed '(temp[21:10]);
62  end
63
64 endmodule
```

Figure 1.15: fir.sv

Abbiamo poi definito il blocco `always_comb`, nel quale vengono eseguite le seguenti operazioni:

- Si calcolano i prodotti $t_i = p_i \times b_i$ per $i = 0, \dots, 7$, dove p_i rappresenta i segnali di ingresso e b_i sono i coefficienti predefiniti.
- Si sommano tutti i prodotti ottenuti, assegnando il risultato al segnale t_sum .
- Si aggiunge a t_sum la costante di arrotondamento, ottenendo il segnale intermedio $temp$.

- Infine, il segnale di uscita q_1 è ottenuto estraendo i bit di $temp$ compresi tra la posizione 10 e la posizione 21.

```
1  `timescale 1ns / 1ps
2  module enfasi (input logic clk, rst,
3  |   |   |   |   input logic signed [10:0] z,
4  |   |   |   |   output logic signed [11:0] q);
5
6  logic signed [11:0] p;
7
8  aso D1 (.clk(clk),.rst(rst),.p(p),.z(z));
9  fir D2 (.clk(clk),.rst(rst),.p(p),.q(q));
10
11 endmodule
```

Figure 1.16: enfasi.sv

Il blocco di enfasi finale ha i seguenti segnali di ingresso e uscita:

- **Ingressi:**

- `clk` (segnale di clock)
- `rst` (segnale di reset)
- `z` (segnale di ingresso a 11 bit)

- **Uscita:**

- `q` (segnale di uscita a 12 bit)

All'interno del blocco viene dichiarato un segnale intermedio `p` di 12 bit, che rappresenta l'uscita del blocco ASO e l'ingresso del blocco HAMMING.

Il blocco enfasi è composto da due moduli principali:

1. **Blocco ASO:** viene instanziato con il nome D1. I terminali dell’istanza del modulo ASO sono collegati ai segnali interni del blocco enfasi utilizzando la corrispondenza per nome, ovvero la sintassi `.nome_terminale(nome_segnale)`.
2. **Blocco FIR:** viene instanziato con il nome D2. Anche in questo caso, i terminali dell’istanza del modulo FIR sono collegati ai segnali interni del blocco enfasi utilizzando la corrispondenza per nome.

1.3 Detector

Il blocco Detector rappresenta il terzo e ultimo modulo del sistema in esame. Il suo compito principale è rilevare la presenza di spikes nel segnale di ingresso q , proveniente dalle elaborazioni precedenti. La rilevazione avviene confrontando il valore del segnale di ingresso con una soglia predefinita. Quando il sistema identifica la presenza di uno spike, attiva il segnale di uscita spike. La soglia di confronto è fornita come ingresso al blocco e viene caricata in modo seriale quando il segnale di enable (en) è alto. Per eseguire correttamente il confronto tra il valore della soglia e il segnale di ingresso, è necessario un convertitore seriale-parallelo. Di seguito, viene illustrato lo schema a blocchi del modulo Detector.

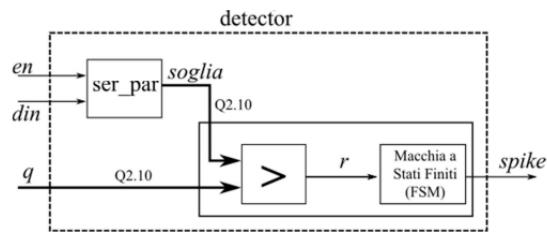


Figure 1.17: Struttura Detector

Partiamo dalla descrizione SystemVerilog del sottoblocco ser_par riportata di seguito.

```
1 `timescale 1ns/1ps
2 module ser_par(input logic clk, rst,
3 | | | | input logic enable, din,
4 | | | | output logic signed [11:0] soglia);
5
6 // Inserire la soglia MSB first
7 logic signed [11:0] temp;
8
9 always_ff @(posedge clk)
10 begin
11     if (rst)
12         begin
13             | temp <='0;
14         end
15     else if (enable)
16         begin
17             | temp[11:1]<=temp[10:0];
18             | temp[0]<=din;
19         end
20     end
21
22 assign soglia = temp;
23
24 endmodule
```

Figure 1.18: ser_par.sv

Il modulo implementa un **registro a scorrimento** (*shift register*) a 12 bit, utilizzato per caricare in modalità **seriale** il valore della soglia.

- **Ingressi:**

- `clk`: segnale di clock.
- `rst`: segnale di reset, che azzerà il registro quando attivo.
- `enable`: segnale di controllo per l'inserimento del valore di soglia.
- `din`: ingresso seriale (MSB first).

- **Uscita:**

- `soglia`: valore a 12 bit, inserito serialmente attraverso `din` (MSB first) quando `enable` è alto.

Il blocco utilizza un **registro a scorrimento** composto da **12 flip-flop** (rappresentati dall'array `temp`), in cui il valore di `din` viene **caricato serialmente** quando `enable` è alto. Il caricamento avviene nel seguente modo:

1. **Reset attivo** (`rst == 1`): tutti i bit del registro vengono azzerati.
2. **Enable attivo** (`enable == 1`):
 - Il valore di `din` viene memorizzato nel **flip-flop meno significativo** (`temp[0]`).

- Gli altri bit del registro vengono **shiftati di una posizione** verso sinistra ($\text{temp}[11:1] = \text{temp}[10:0]$).

3. Dopo **12 cicli di clock**, il valore di **soglia** è completamente caricato e può essere letto parallelamente.

Passiamo ora al sottoblocco FSM.

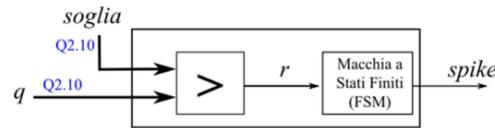


Figure 1.19: Schema a blocchi Comparatore-FSM

La **Macchina a Stati Finiti** viene introdotta per prevenire false rilevazioni di *spikes* dovute alla presenza di rumore additivo sul segnale *q*. Il segnale di uscita *spike* viene attivato solo se, dopo aver individuato un '1' nel segnale *r*, lo stesso segnale resta alto per almeno due volte nei tre campioni successivi. La figura di seguito descrive il funzionamento desiderato in funzione del segnale di ingresso.

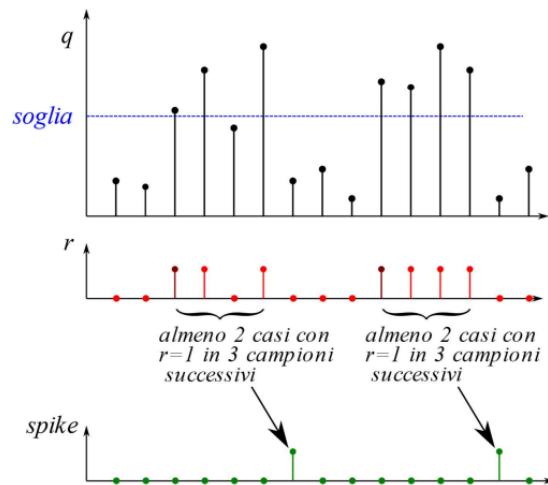


Figure 1.20: Comportamento FSM

È quindi fondamentale descrivere il funzionamento della Macchina a Stati Finiti attraverso un automa. Di seguito viene riportato il diagramma dell'automa.

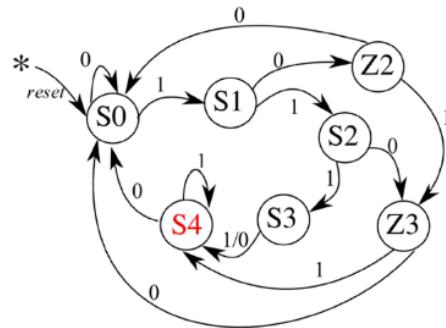


Figure 1.21: Automa FSM

Dopo il reset iniziale, la Macchina a Stati Finiti (FSM) si posiziona nello stato S_0 e vi rimane finché l'ingresso r non assume valore 1. Al verificarsi della condizione $r = 1$, la FSM transita nello stato S_1 e, se r continua a rimanere alto, procede verso gli stati S_2 e S_3 . Il raggiungimento dello stato S_3 avviene solo se nel segnale di ingresso sono stati rilevati almeno due valori consecutivi pari a 1, oltre al primo. Una volta in S_3 , il sistema passa automaticamente allo stato S_4 , indipendentemente dal valore assunto da r . In questo stato, l'uscita spike viene attivata. Successivamente, quando il segnale r si abbassa, la FSM ritorna nello stato iniziale S_0 . Gli stati intermedi Z_2 e Z_3 permettono di attivare l'uscita spike in tutti i casi in cui il segnale r risulti alto almeno due volte nei tre campioni successivi.

```
1  `timescale 1ns/1ps
2  module fsm(input logic clk, rst,
3  |     |     input logic signed [11:0] q, soglia,
4  |     |     output logic spike);
5
6  typedef enum logic [2:0] {S0, S1, S2, S3, S4, Z2, Z3} stato_t;
7
8  stato_t stato_corrente, stato_futuro;
9
10 logic r;
11
12 always_comb
13 begin
14     if (q>=soglia)
15         r = 1'b 1;
16     else
17         r = 1'b 0;
18 end
19
```

Figure 1.22: fsm.sv

Il modulo fsm include, oltre ai consueti ingressi clk e rst, due ulteriori ingressi a 12 bit: soglia e q. Il primo rappresenta l'uscita del blocco ser_par, mentre q corrisponde al segnale elaborato nelle fasi precedenti. L'uscita del modulo, denominata spike, è un segnale a singolo bit. Per implementare la macchina a stati finiti, è stato definito un tipo enumerativo denominato stato_t, che può assumere i valori S0, S1, S2, S3, S4, Z2 e Z3. Successivamente, sono stati dichiarati due segnali di tipo stato_t, denominati stato_corrente e stato_futuro. Successivamente, è stato implementato il blocco combinatorio del comparatore. Questo modulo confronta il valore di q con quello di soglia: se q risulta maggiore di soglia, il segnale r viene impostato a 1; in caso contrario, viene posto a 0.

```
20  always_comb
21      begin
22          case (stato_corrente)
23              S0: stato_futuro = (r)?S1:S0;
24              S1: stato_futuro = (r)?S2:Z2;
25              S2: stato_futuro = (r)?S3:Z3;
26              S3: stato_futuro = S4;
27              S4: stato_futuro = (r)?S4:S0;
28              Z2: stato_futuro = (r)?S0:Z3;
29              Z3: stato_futuro = (r)?S4:S0;
30          default:
31              stato_futuro = S0;
32      endcase
33  end
34
35  always_ff @ (posedge clk)
36      begin
37          if (rst)
38              stato_corrente <= S0;
39          else
40              stato_corrente <= stato_futuro;
41      end
42
43      assign spike = stato_corrente == S4;
44
45  endmodule
```

Figure 1.23: fsm.sv

Successivamente, è stata descritta la FSM in Verilog. In particolare, è presente un blocco combinatorio che determina lo stato futuro della FSM in base allo stato corrente e al valore dell'ingresso *r*. L'operatore condizionale '?' viene utilizzato per valutare *r*: se *r* è 1, *stato_futuro*

assume il primo valore specificato; se r è 0, assume il secondo valore. Infine, un blocco `always_ff` si occupa di aggiornare `stato_corrente` in base al `clock`. L'uscita `spike` viene assegnata tramite un'istruzione `assign`: se lo stato corrente è `S4`, il segnale `spike` viene impostato a 1. Ora possiamo combinare i due sottoblocchi, `FSM` e `ser_par`, in un unico modulo denominato `detector`. Come illustrato nel diagramma a blocchi all'inizio di questo paragrafo, è sufficiente concatenare i due sottoblocchi.

Figure 1.24: detector.sv

L'uscita del modulo `ser_par`, denominata `soglia`, diventa un nodo interno che funge da ingresso per il sottoblocco `FSM`, insieme al segnale `q`. L'uscita complessiva del modulo `detector` è il segnale `spike`, che viene attivato quando viene rilevato uno `spike`. Per l'istanza dei due sottoblocchi è stata utilizzata la sintassi `.*`, che consente di assegnare automaticamente a ciascun terminale il segnale corrispondente avente

lo stesso nome.

Chapter 2

Backend

Durante la fase di backend, si procede con l’implementazione fisica dei moduli. Il sistema viene suddiviso in parti e le celle di libreria vengono disposte sul chip (placement) per poi essere collegate tra loro tramite interconnessioni metalliche (routing). Una volta ottenuto il layout definitivo, si esegue una verifica per assicurarsi che il circuito funzioni correttamente, tenendo conto degli effetti dei parametri parassiti delle interconnessioni. I parametri parassiti del circuito hanno un impatto significativo sulle prestazioni complessive. Poiché durante la sintesi logica il layout non è ancora definito, si possono solo stimare gli effetti delle interconnessioni sui parametri prestazionali. La fase di piazzamento inizia con il *floorplanning*, in cui si stabilisce la disposizione dei macroblocchi che compongono il sistema, allocando adeguatamente lo spazio tra di essi. Durante il *floorplanning*:

- Si definisce la posizione dei terminali di I/O,

- Si organizza la disposizione dei blocchi all'interno del chip,
- Si decide il tipo di distribuzione delle alimentazioni.

Gli obiettivi principali di questa fase includono:

- Posizionare opportunamente i macroblocchi (in modo da minimizzare la lunghezza delle successive interconnessioni che in questa fase può essere solo approssimativamente stimata)
- Organizzare le linee di alimentazione in modo tale da ridurre al minimo le cadute resistive

Dopo il *floorplanning*, si passa alla fase di *placement*, che consiste nel disporre le celle standard all'interno dell'area loro assegnata. L'obiettivo è posizionarle in modo da ridurre la lunghezza delle interconnessioni che verranno successivamente realizzate. Questa fase si articola in due passaggi:

1. **Global placement:** le celle vengono posizionate in maniera preliminare, cercando di stimare e ridurre la lunghezza complessiva dei collegamenti. Tuttavia, il piazzamento in questa fase non è ancora definitivo e potrebbero verificarsi sovrapposizioni tra le celle.
2. **Detailed placement:** si affinano le posizioni delle celle spostandole leggermente per ottenere un layout fisicamente valido e correggere eventuali sovrapposizioni.

Successivamente, si procede con la generazione dell'albero di clock, in cui viene creato il *clock tree* e inseriti i buffer necessari per bilanciare i ritardi all'interno del sistema. Infine, si passa alla fase di *routing*, anch'essa suddivisa in due passaggi:

- **Global routing:** si determina in modo preliminare il percorso delle connessioni, ma il routing non è ancora completamente risolto e potrebbero esserci violazioni, come cortocircuiti o sovrapposizioni.
- **Detailed routing:** si affinano progressivamente i collegamenti per correggere le violazioni e ottenere un instradamento corretto.

2.1 IIR

2.1.1 Sintesi

```
1 Timing report:  
2 10.000 10.000  clock my_clock (rise edge)  
3 0.000 10.000  clock network delay (ideal)  
4 -0.100 9.900  clock uncertainty  
5 0.000 9.900  clock reconvergence pessimism  
6 9.900 ^ z[9]$_SDFF_PP0_/CK (DFF_X1)  
7 -0.029 9.871  library setup time  
8 9.871  data required time  
9 --
```

```
10          9.871    data required time
11          -3.227   data arrival time
12 --
13          6.644    slack (MET)
14
15 Area report:
16
17 Number of wires:          3089
18 Number of wire bits:      3109
19 Number of public wires:    81
20 Number of public wire bits: 101
21 Number of ports:          4
22 Number of port bits:      24
23 Number of memories:       0
24 Number of memory bits:    0
25 Number of processes:      0
26 Number of cells:          3008
27     AND2_X1                157
28     AND3_X1                52
29     AND4_X1                4
30     AOI211_X1              9
31     AOI21_X1                242
32     AOI221_X1              4
33     AOI22_X1                24
34     DFF_X1                  88
```

```

35      INV_X1           191
36      MUX2_X1           6
37      NAND2_X1          332
38      NAND3_X1          56
39      NAND4_X1           7
40      NOR2_X1          322
41      NOR3_X1          65
42      NOR4_X1           5
43      OAI211_X1          20
44      OAI21_X1          202
45      OAI221_X1           3
46      OAI22_X1          29
47      OR2_X1           95
48      OR3_X1           42
49      OR4_X1           2
50      XNOR2_X1          647
51      XOR2_X1          404
52
53      Chip area for module '\iir': 3850.084000
54      of which used for sequential elements: 397.936000
55      (10.34%)

```

In questa fase, non essendo ancora noto il layout del circuito, è possibile solo effettuare una stima dell'effetto delle interconnessioni sulle prestazioni. Possiamo osservare che, nella figura in alto, è stato calcolato lo *slack* per il vincolo sul tempo di setup, risultando pari a 6.644 ns. Questo implica la possibilità di aumentare la frequenza di clock, che inizialmente era fissata a 100 MHz. Il periodo minimo del clock

aggiornato sarà:

$$T_{\min} = T - 6.644 = 3.356 \text{ ns}$$

Di conseguenza, la frequenza massima raggiungibile sarà:

$$f_{\max} = \frac{1}{T_{\min}} \approx 298 \text{ MHz}$$

Tuttavia, è fondamentale sottolineare che questi vincoli sono stati calcolati senza avere ancora il layout definitivo del circuito. Di conseguenza, si tratta solo di una stima preliminare dei valori reali, che verranno confrontati con quelli ottenuti nei controlli finali.

2.1.2 Floorplan

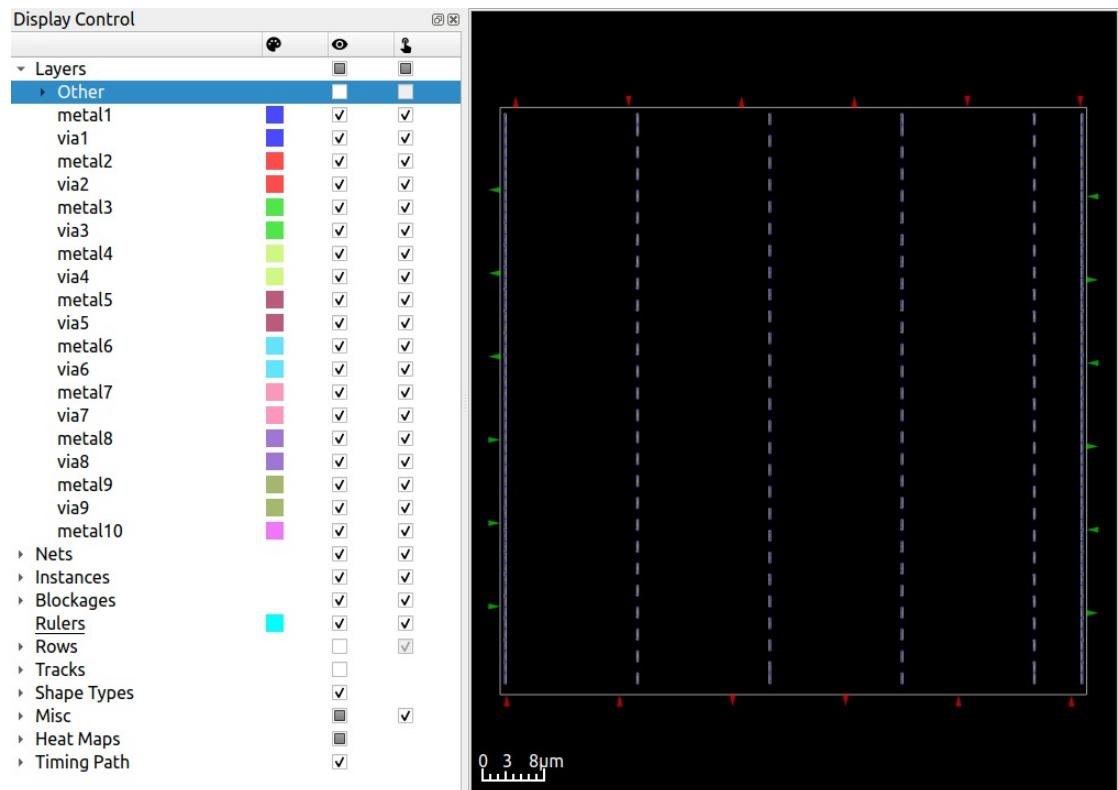


Figure 2.1: Floorplan IIR

Con il comando `tapcell` inseriamo su ogni riga delle celle particolari che non hanno una funzione logica, ma sono fondamentali per realizzare i collegamenti con il substrato.

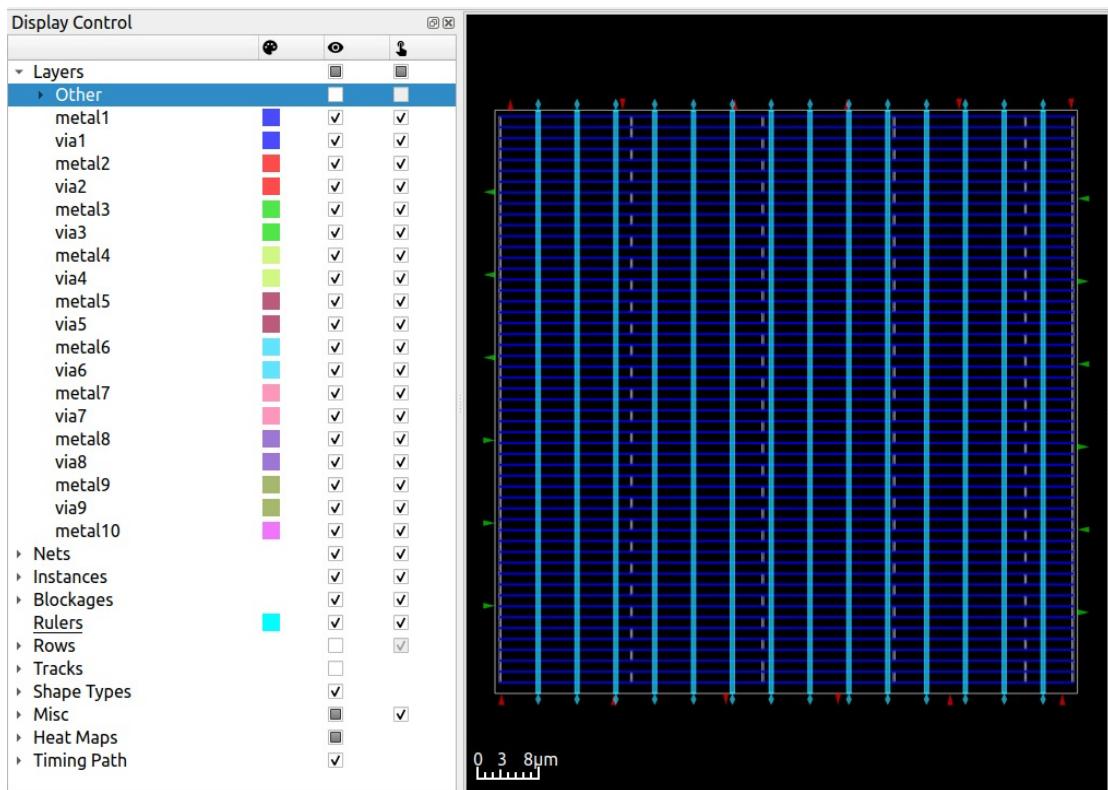


Figure 2.2: Floorplan IIR

In questa fase è stata definita la griglia di alimentazione. Le linee di alimentazione verticali sono state realizzate al livello metal6, mentre le linee di alimentazione delle celle, disposte orizzontalmente, si trovano in metal1. Inoltre, sono stati posizionati i terminali di I/O: i terminali di input sono evidenziati in verde, mentre quelli di output sono indicati in rosso.

Cell type report:

Tap cell	212
Inverter	191
Sequential cell	88
Multi-Input combinational cell	2729
Total	3220

Cell instance report:

AND2_X1	157
AND3_X1	52
AND4_X1	4
AOI211_X1	9
AOI21_X1	242
AOI221_X1	4
AOI22_X1	24
DFF_X1	88
TAPCELL_X1	212
INV_X1	191
MUX2_X1	6
NAND2_X1	332
NAND3_X1	56
NAND4_X1	7
NOR2_X1	322
NOR3_X1	65
NOR4_X1	5
OAI211_X1	20
OAI21_X1	202
OAI221_X1	3
OAI22_X1	29
OR2_X1	95
OR3_X1	42
OR4_X1	2
XNOR2_X1	647
XOR2_X1	404

>>> report_design_area
Design area 3906 μ^2 73% utilization.

Figure 2.3: Cell type report e cell instance report IIR

La design area utilization è pari a 0.73, ovvero il 73 per cento di utilizzo.

2.1.3 Placement

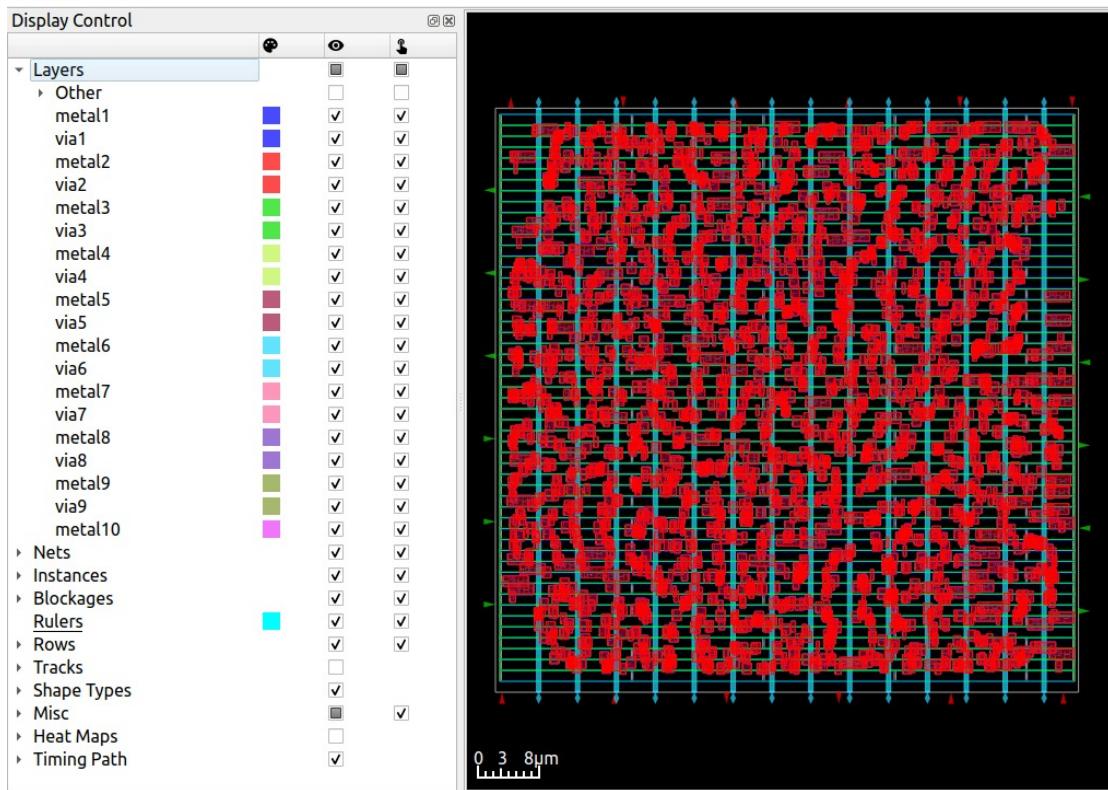


Figure 2.4: Placement IIR

La figura mostra il risultato della fase di **detailed placement**, durante la quale le celle vengono spostate in un intorno della posizione definita dal **global placement**, in modo da "legalizzare" il loro piazzamento. Al termine di questa fase, il tool fornisce un **report sintetico**, che include il parametro **HPWL** (*Half-Perimeter Wirelength*), ovvero una stima della lunghezza complessiva dei futuri collegamenti. Un valore più basso di **HPWL** indica un placement più efficiente. Di seguito,

i valori ottenuti:

- **Total displacement:** $3590.0 \mu m$
- **Average displacement:** $1.1 \mu m$
- **Max displacement:** $4.3 \mu m$
- **Original HPWL:** $14505.6 \mu m$
- **Legalized HPWL:** $18355.9 \mu m$
- **Delta HPWL:** $+27\%$

Dall'analisi del report, si osserva che il processo di legalizzazione del **placement** ha comportato un incremento dell'HPWL. Per migliorare il risultato, si può eseguire un'**ottimizzazione** (*optimize*), che sposta le celle in un intorno più ristretto di **10 μm** (rispetto ai **20 μm** usati nella fase di legalizzazione). Questa tecnica permette di ridurre l'HPWL, anche se ulteriori iterazioni non porterebbero benefici significativi. I risultati dopo l'ottimizzazione sono:

- **Original HPWL:** $17,854.4 \mu m$
- **Final HPWL:** $15993.1 \mu m$
- **Delta HPWL:** -10.4%

2.1.4 Clock Tree

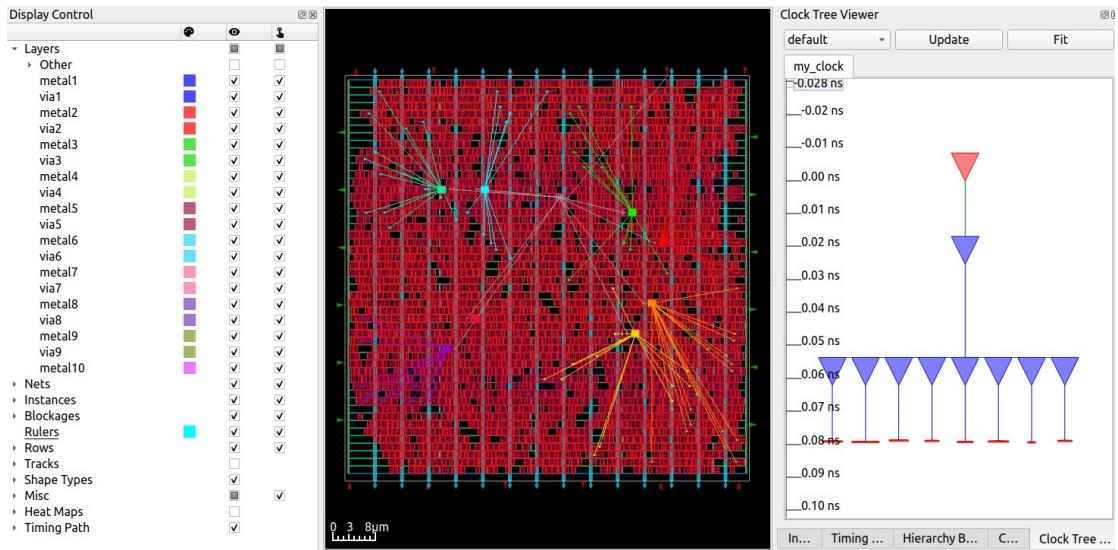


Figure 2.5: Clock Tree IIR

Il **clock tree** ottenuto dopo l'esecuzione del comando `repair_timing -hold` è mostrato in figura. Questa operazione ha comportato l'inserimento di ulteriori **buffer** per garantire che lo **slack** raggiunga almeno **100 ps**. Di seguito, alcuni parametri significativi del clock tree:

- **Numero totale di Clock Roots:** 1
- **Numero totale di Buffer inseriti:** 9
- **Numero totale di Clock Subnets:** 9
- **Numero totale di Sinks:** 88

Dopo la generazione del **clock tree**, è necessario eseguire nuovamente la **legalizzazione del placement**, poiché sono stati aggiunti **buffer**

sia per la gestione del **clock** che per il rispetto del **vincolo di hold**.

I nuovi valori di **HPWL** (*Half-Perimeter Wirelength*) risultano:

- **HPWL iniziale:** $17,215.5 \mu m$
- **HPWL finale:** $16,305.3 \mu m$
- **Variazione HPWL:** -5.3%

2.1.5 Routing

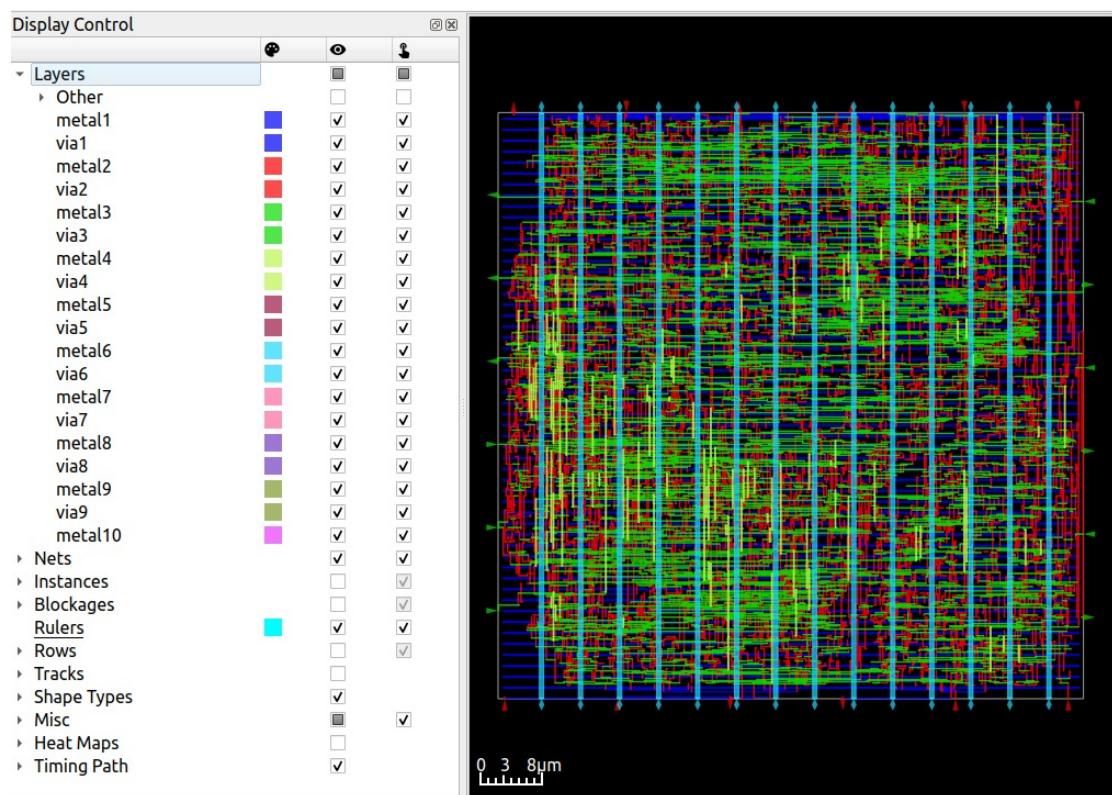
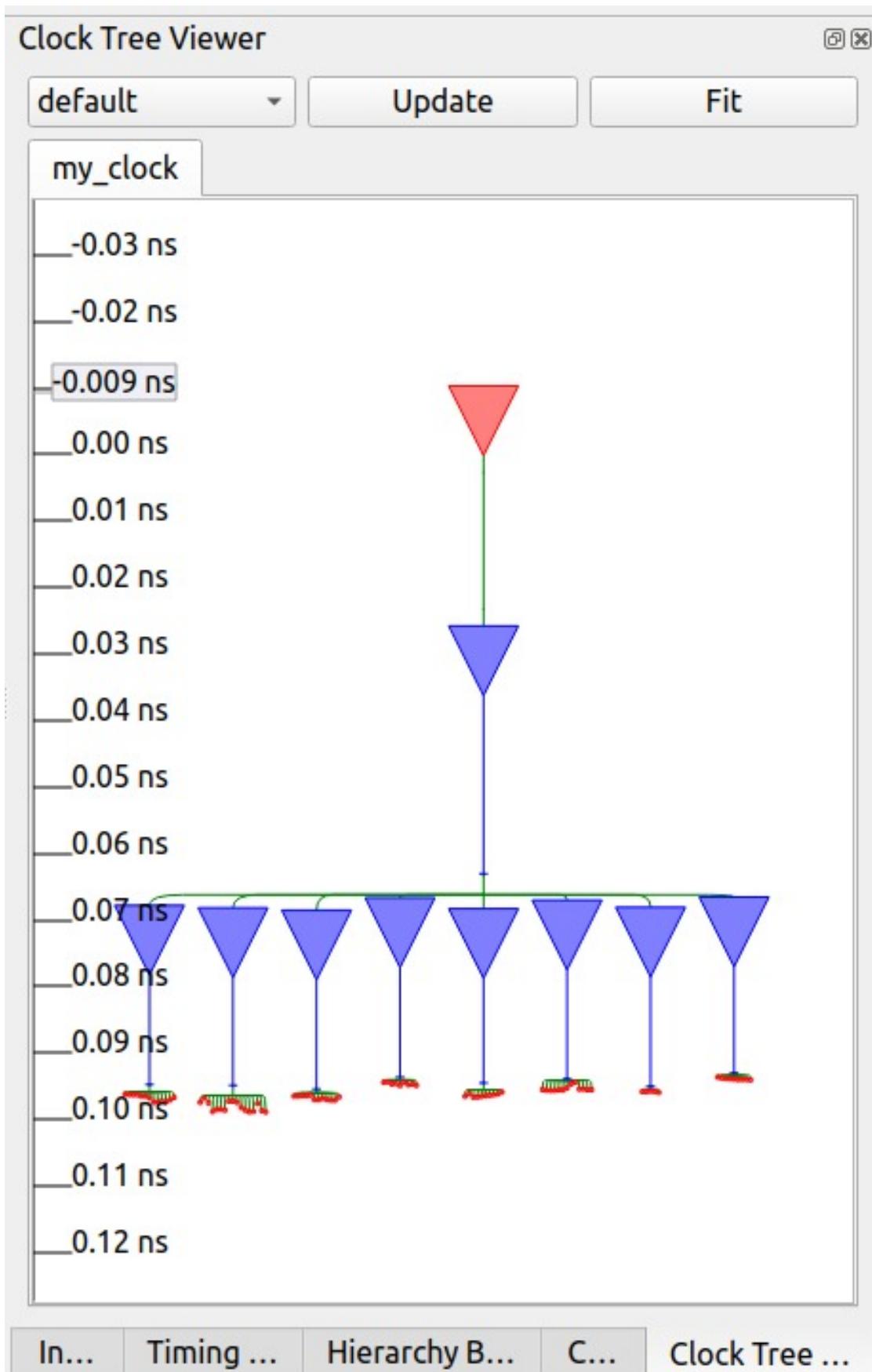


Figure 2.6: Routing IIR

Figure 2.7: A_1 Clock Tree IIR

Nella figura superiore è mostrato il risultato della fase di detailed routing, mentre nella figura inferiore è rappresentato il nuovo albero di clock, influenzato dai disturbi introdotti dalle interconnessioni.

2.1.6 Controlli finali

```
10.000  10.000  clock my_clock (rise edge)
 0.000  10.000  clock network delay (ideal)
-0.100  9.900  clock uncertainty
 0.000  9.900  clock reconvergence pessimism
         9.900 ^ z[9]$_SDFF_PP0_/CK (DFF_X1)
-0.030  9.870  library setup time
         9.870  data required time
-----
         9.870  data required time
-3.401  data arrival time
-----
 6.469  slack (MET)
```

Figure 2.8: Controlli Finali IIR

In figura è mostrato lo **slack** calcolato rispetto al vincolo sul **tempo di setup**. Si può notare che il valore ottenuto differisce da quello stimato inizialmente durante la fase di sintesi. Possiamo ora determinare la massima frequenza operativa del sistema:

$$T_{\min} = T - 6,467 = 3,351 \text{ ns}$$

$$f_{\max} = \frac{1}{T_{\min}} \approx 283 \text{ MHz}$$

Si osserva che l'influenza dei **parametri parassiti** introdotti dalle interconnessioni provoca un **leggero degrado delle prestazioni**,

riducendo la frequenza massima operativa rispetto alla stima effettuata in fase di sintesi.

0.104 slack (MET)					
Group	Internal Power	Switching Power	Leakage Power	Total Power (Watts)	
Sequential	3.20e-03	7.67e-05	6.87e-06	3.28e-03	1.5%
Combinational	1.12e-01	1.01e-01	8.20e-05	2.13e-01	98.4%
Clock	5.38e-05	2.91e-05	2.44e-06	8.53e-05	0.0%
Macro	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.0%
Pad	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.0%
Total	1.15e-01	1.01e-01	9.13e-05	2.16e-01	100.0%
	53.3%	46.7%	0.0%		

Figure 2.9: Controlli Finali IIR

Per quanto riguarda lo slack relativo al vincolo sul tempo di hold, si ottiene un valore di 0.104 ns. È possibile fare alcune osservazioni riguardo al consumo di potenza. In particolare, possiamo distinguere tra potenza interna, potenza esterna e potenza di leakage. Il principale contributo alla dissipazione di potenza proviene dalla potenza dinamica, sia interna che esterna, generata dai circuiti combinatori.

2.2 Enfasi

2.2.1 Sintesi

```

1 Timing report:
2   10.000  10.000  clock my_clock (rise edge)
3   0.000  10.000  clock network delay (ideal)

```

```
4      -0.100    9.900    clock uncertainty
5      0.000    9.900    clock reconvergence pessimism
6      9.900 ^ q[11]$_SDFF_PP0_/CK (DFF_X1)
7      -0.029    9.871    library setup time
8      9.871    data required time
9      --
10
10      9.871    data required time
11      -3.858    data arrival time
12      --
13
13      6.013    slack (MET)
14
15 Area report:
16
17 === enfasi ===
18
19 Number of wires:          3969
20 Number of wire bits:       3990
21 Number of public wires:    167
22 Number of public wire bits: 188
23 Number of ports:          4
24 Number of port bits:       25
25 Number of memories:        0
26 Number of memory bits:      0
27 Number of processes:        0
```

```
28 Number of cells: 3802
29     AND2_X1 290
30     AND3_X1 58
31     AND4_X1 4
32     AOI211_X1 4
33     AOI21_X1 284
34     AOI221_X1 3
35     AOI22_X1 31
36     DFF_X1 175
37     INV_X1 191
38     MUX2_X1 6
39     NAND2_X1 491
40     NAND3_X1 87
41     NAND4_X1 10
42     NOR2_X1 441
43     NOR3_X1 55
44     NOR4_X1 5
45     OAI211_X1 12
46     OAI21_X1 234
47     OAI221_X1 4
48     OAI22_X1 28
49     OR2_X1 96
50     OR3_X1 46
51     OR4_X1 4
52     XNOR2_X1 658
53     XOR2_X1 585
54
55 Chip area for module '\enfasi': 5026.602000
```

In questa fase il layout del circuito non è ancora noto, quindi è possibile solo stimare l'impatto delle interconnessioni sui parametri prestazionali. Dalla figura in alto, si osserva che lo slack relativo al vincolo sul tempo di setup è pari a 6.013 Ciò implica che sia possibile incrementare la frequenza di clock, determinando il periodo minimo ammissibile:

$$T_{\min} = T - 6,013 = 3,987 \text{ ns}$$

Di conseguenza, la frequenza massima teorica raggiungibile è:

$$f_{\max} = \frac{1}{T_{\min}} \approx 251 \text{ MHz}$$

Tuttavia, è fondamentale sottolineare che questi vincoli sono stati calcolati senza considerare il layout effettivo del circuito. Pertanto, rappresentano solo una stima preliminare dei valori reali. Un confronto più accurato sarà effettuato nelle fasi di verifica finale.

2.2.2 Floorplan

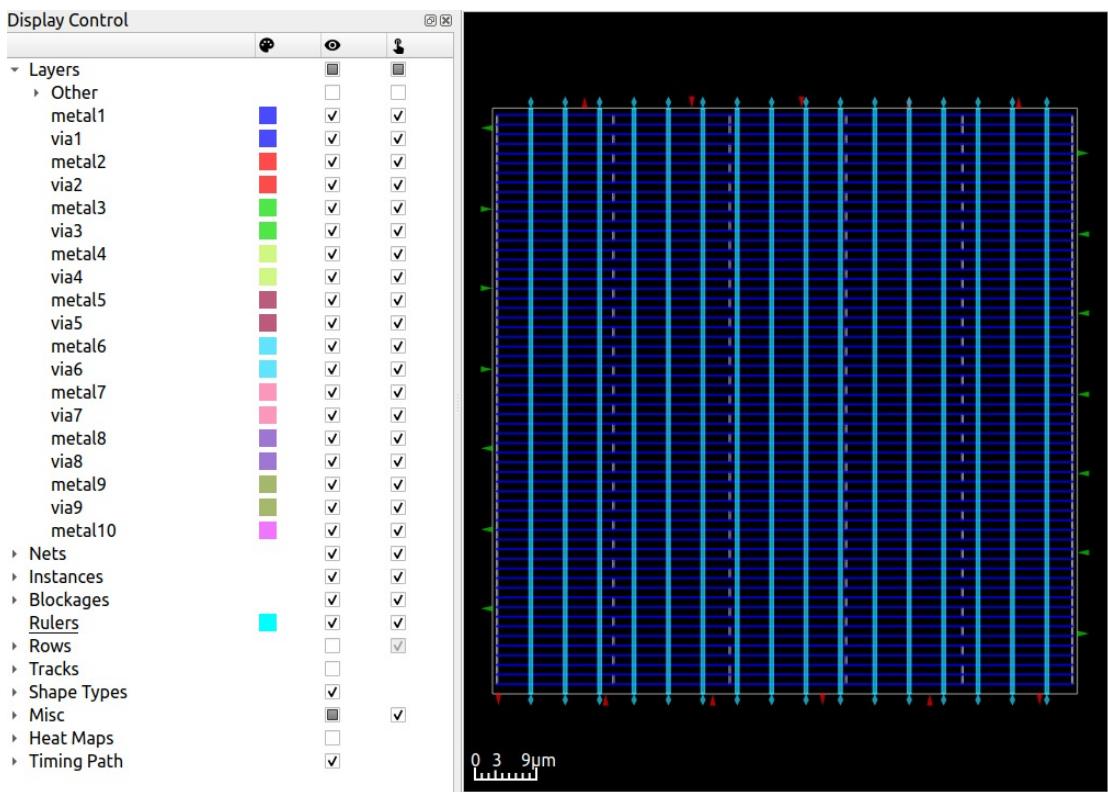


Figure 2.10: Floorplan ENFASI

In questa fase è stata definita la griglia di alimentazione. Le linee di alimentazione verticali sono state realizzate al livello metal6, mentre le linee di alimentazione delle celle, disposte orizzontalmente, si trovano in metal1. Inoltre, sono stati posizionati i terminali di I/O: i terminali di input sono evidenziati in verde, mentre quelli di output sono indicati in rosso.

```

Cell type report:
  Tap cell           240
  Inverter          191
  Sequential cell   175
  Multi-Input combinational cell 3436
  Total             4042

>>> report_design_area
Design area 5090  $\mu^2$  74% utilization.

```

Figure 2.11: Cell type report ENFASI

La design area utilization è proprio 0.74, ovvero il 74 di utilizzo.

2.2.3 Placement

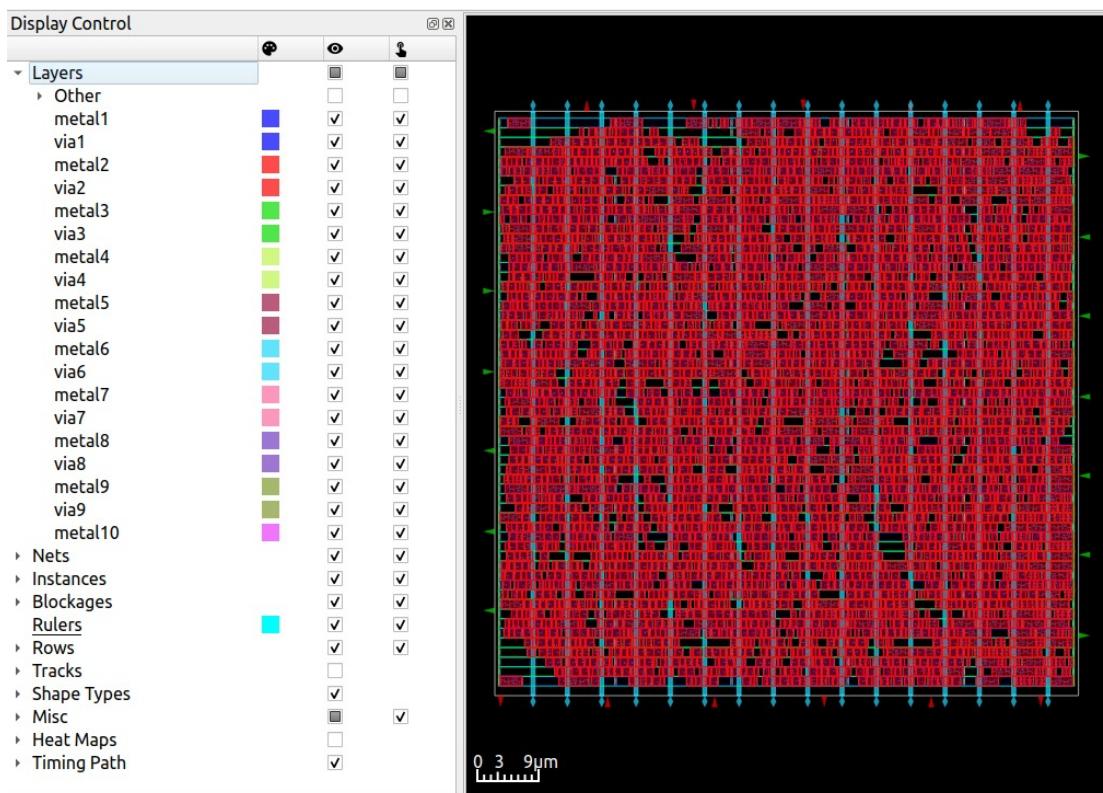


Figure 2.12: Placement ENFASI

In figura è illustrato il risultato ottenuto dopo la fase di *detailed placement*, in cui le celle vengono spostate in un intorno della posizione assegnata durante il *global placement*, con l’obiettivo di rendere il piazzamento legalmente valido. Dopo questa fase, il tool fornisce un report sintetico che include il valore dell’HPWL (*Half-Perimeter Wirelength*), che rappresenta una stima della lunghezza totale dei collegamenti futuri. Un valore di HPWL più basso indica un piazzamento più efficiente.

- **Displacement totale:** 5629.4 u
- **Displacement medio:** 1.3 u
- **Displacement massimo:** 10.3 u
- **HPWL iniziale:** 17820.5 u
- **HPWL legalizzato:** 23782.2 u
- **Variazione HPWL:** +33%

Dall’analisi dei risultati, si nota che il processo di legalizzazione del piazzamento ha comportato un aumento significativo dell’HPWL. Per migliorare questo aspetto, è possibile eseguire un’ottimizzazione (*optimize*), che tenta di riposizionare le celle in un intorno di 10 μm , rispetto ai 20 μm utilizzati nella legalizzazione. Questo approccio ha prodotto un effettivo miglioramento dell’HPWL. Tuttavia, ulteriori iterazioni non sarebbero vantaggiose poiché i benefici sarebbero trascurabili.

- **HPWL iniziale:** 22857.1 μ
- **HPWL finale dopo ottimizzazione:** 19576.5 μ
- **Variazione HPWL:** -14.4%

2.2.4 Clock Tree

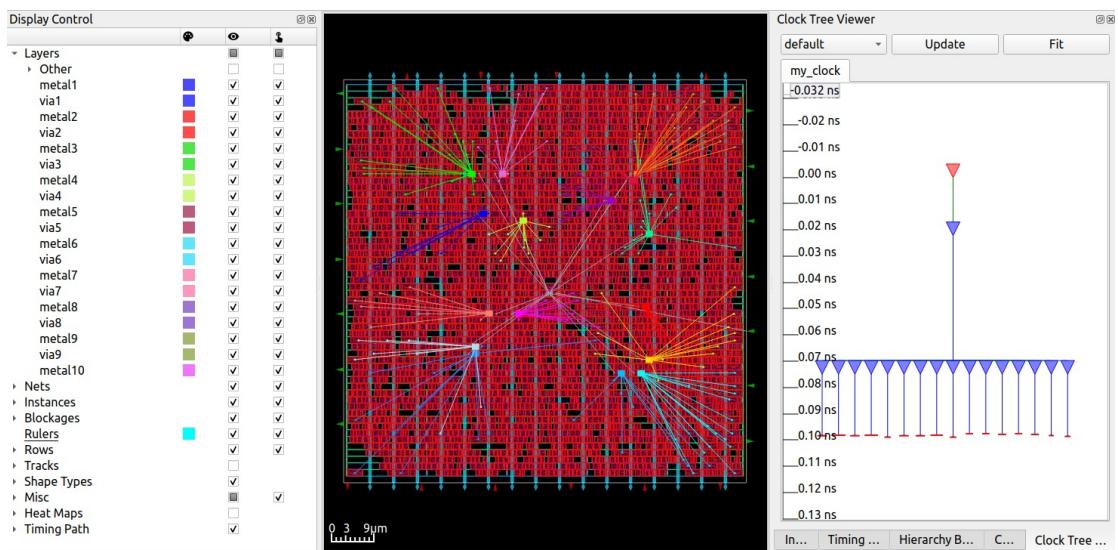


Figure 2.13: Clock tree ENFASI

Durante il processo di generazione dell'albero di clock, sono stati ottenuti i seguenti risultati:

- **Numero totale di radici del clock:** 1
- **Numero complessivo di buffer inseriti:** 17
- **Numero totale di subnet del clock:** 17
- **Numero complessivo di sink:** 175

La figura mostra la struttura dell’albero di clock risultante dopo l’esecuzione del comando `repair_timing -hold`, il quale ha introdotto ulteriori buffer con l’obiettivo di garantire uno slack minimo di 100 ps. Terminata la costruzione dell’albero di clock, è stato necessario effettuare una nuova legalizzazione del piazzamento per tenere conto dei buffer aggiunti per il clock e per il rispetto del vincolo di *hold*. I nuovi valori di HPWL (*Half-Perimeter Wirelength*) mostrano una riduzione rispetto alla configurazione iniziale:

- **HPWL originale:** 22373.1 *u*
- **HPWL dopo la legalizzazione:** 20154.4 *u*
- **Variazione percentuale dell’HPWL:** -9.9%

2.2.5 Routing

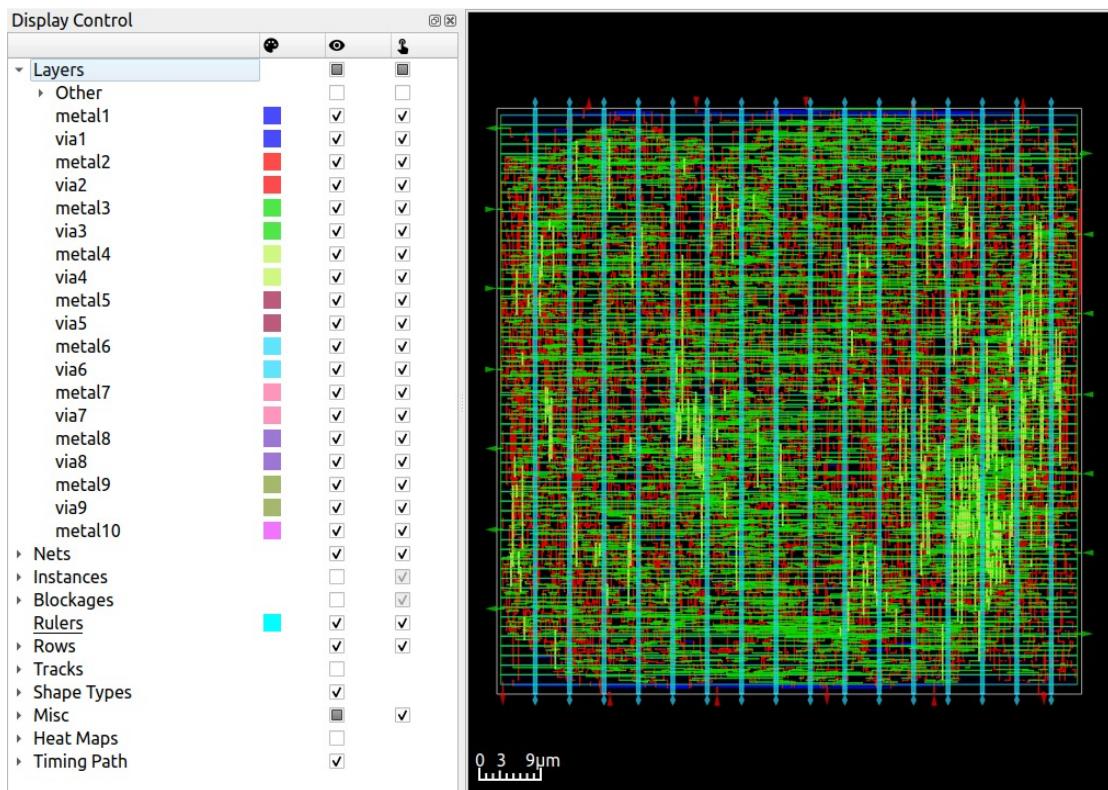


Figure 2.14: Routing ENFASI

2.2.6 Controlli finali

```

10.000  10.000  clock my_clock (rise edge)
  0.000  10.000  clock network delay (ideal)
 -0.100   9.900  clock uncertainty
  0.000   9.900  clock reconvergence pessimism
           9.900 ^ q[11]$_SDFF_PP0_/_CK (DFF_X1)
 -0.035   9.865  library setup time
   9.865  data required time
-----
           9.865  data required time
 -4.081  data arrival time
-----
      5.783  slack (MET)

```

Figure 2.15: Controlli Finali ENFASI

La figura mostra lo slack calcolato in relazione al vincolo sul tempo di setup. Si può osservare che tale valore differisce da quello stimato inizialmente durante la fase di sintesi. Procediamo ora al calcolo della massima frequenza operativa del sistema:

$$T_{\min} = T - 5.783 = 4.217 \text{ ns}$$

Da cui otteniamo la frequenza massima di funzionamento:

$$f_{\max} = \frac{1}{T_{\min}} \approx 237 \text{ MHz}$$

Si nota che la presenza di effetti parassiti introdotti dalle interconnessioni ha causato un leggero degrado delle prestazioni del sistema. Di conseguenza, la frequenza massima ottenibile risulta inferiore rispetto

a quella stimata nella fase di sintesi.

Delay	Time	Description
0.000	0.000	clock my_clock (rise edge)
0.000	0.000	clock network delay (ideal)
0.000	0.000	\wedge D2.p0[9]\$_SDFF_PP0_/CK (DFF_X1)
0.108	0.108	\vee D2.p0[9]\$_SDFF_PP0_/Q (DFF_X1)
0.030	0.138	\vee hold409/Z (CLKBUF_X1)
0.025	0.164	\vee hold126/Z (CLKBUF_X1)
0.027	0.191	\vee _6551_/ZN (AND2_X1)
0.026	0.217	\vee hold127/Z (CLKBUF_X1)
0.000	0.217	\vee D2.p1[9]\$_SDFF_PP0_/D (DFF_X1)
	0.217	data arrival time
0.000	0.000	clock my_clock (rise edge)
0.000	0.000	clock network delay (ideal)
0.100	0.100	clock uncertainty
0.000	0.100	clock reconvergence pessimism
	0.100	\wedge D2.p1[9]\$_SDFF_PP0_/CK (DFF_X1)
0.011	0.111	library hold time
	0.111	data required time
	0.111	data required time
	-0.217	data arrival time
	0.106	slack (MET)

Figure 2.16: Controlli Finali ENFASI

Per quanto riguarda lo slack sul vincolo del tempo di hold si ha che è pari a 0.106 ns.

Group	Internal Power	Switching Power	Leakage Power	Total Power (Watts)	
<hr/>					
Sequential	1.60e-04	2.17e-05	1.35e-05	1.95e-04	5.0%
Combinational	1.84e-03	1.64e-03	1.04e-04	3.58e-03	90.9%
Clock	9.98e-05	6.01e-05	4.54e-06	1.64e-04	4.2%
Macro	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.0%
Pad	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.0%
<hr/>					
Total	2.10e-03	1.72e-03	1.22e-04	3.94e-03	100.0%
	53.3%	43.6%	3.1%		

Figure 2.17: Controlli Finali ENFASI

Possiamo fare alcune considerazioni sulla potenza; distinguiamo la potenza interna, esterna e quella dovuta al leakage. Il contributo maggiore di dissipazione è dovuto alla potenza dinamica (interna ed esterna) dei sistemi combinatori.

2.3 Detector

2.3.1 Sintesi

```

1 Timing report:
2
3 10.000 10.000  clock my_clock (rise edge)
4
5 0.000 10.000  clock network delay (ideal)
6
7 -0.100 9.900  clock uncertainty
8
9 0.000 9.900  clock reconvergence pessimism
10
11 9.900 ^ blocco2/clk (enfasi)
12
13 -0.859 9.041  library setup time
14
15 9.041  data required time
16
17 --
```

```
10          9.041    data required time
11          -1.017    data arrival time
12  --
13          8.023    slack (MET)
14
15 Area report:
16
17  Number of wires:          154
18  Number of wire bits:       164
19  Number of public wires:    44
20  Number of public wire bits: 54
21  Number of ports:          6
22  Number of port bits:       16
23  Number of memories:        0
24  Number of memory bits:     0
25  Number of processes:       0
26  Number of cells:          113
27    AND2_X1                  1
28    AND3_X1                  2
29    AOI21_X1                 21
30    AOI221_X1                1
31    DFF_X1                   15
32    INV_X1                   17
33    MUX2_X1                  2
34    NAND2_X1                 4
```

```

35      NAND4_X1           1
36      NOR2_X1           10
37      NOR3_X1           6
38      NOR4_X1           1
39      OAI211_X1          1
40      OAI21_X1           16
41      OR2_X1             1
42      XOR2_X1            12
43      enfasi             1
44      iir                 1
45
46      Chip area for module '\top': 12135.448000
47      of which used for sequential elements: 67.830000
          (0.56%)

```

In questa fase il layout del circuito non è ancora noto, quindi è possibile solo stimare l'impatto delle interconnessioni sui parametri prestazionali. Dalla figura in alto, si osserva che lo slack relativo al vincolo sul tempo di setup è pari a Ciò implica che sia possibile incrementare la frequenza di clock, determinando il periodo minimo ammissibile:

$$T_{\min} = T - 8.023 = 1.977 \text{ ns}$$

Di conseguenza, la frequenza massima teorica raggiungibile è:

$$f_{\max} = \frac{1}{T_{\min}} \approx 506 \text{ MHz}$$

Tuttavia, è fondamentale sottolineare che questi vincoli sono stati calcolati senza considerare il layout effettivo del circuito. Pertanto, rappresentano solo una stima preliminare dei valori reali. Un confronto più accurato sarà effettuato nelle fasi di verifica finale.

2.3.2 Floorplan

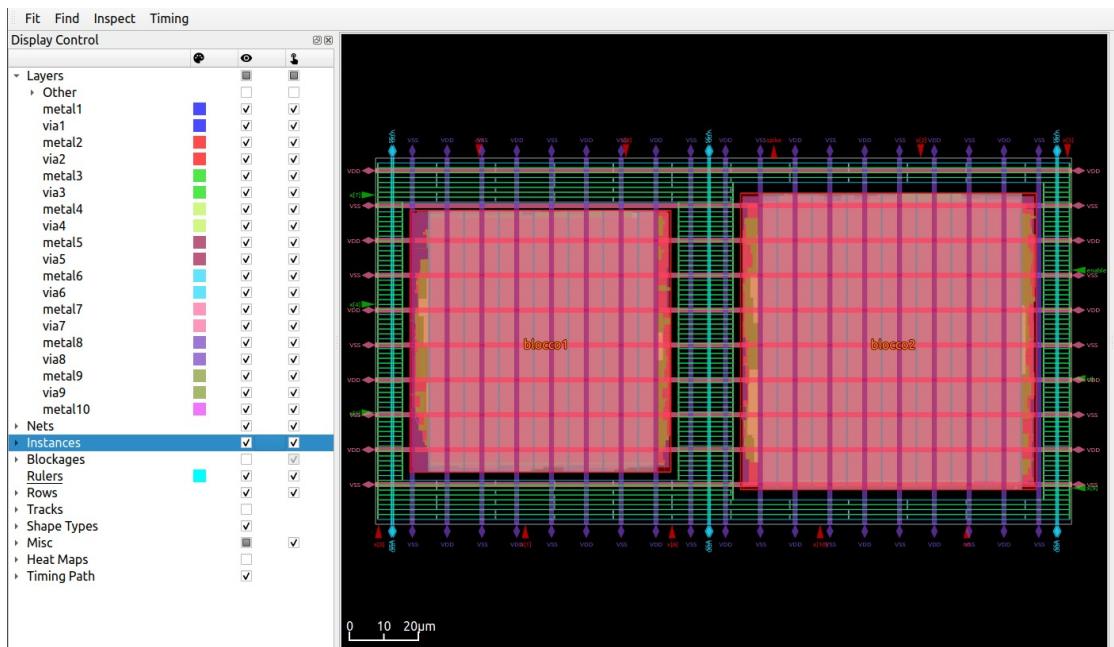


Figure 2.18: Floorplan DETECTOR

La figura fa direttamente riferimento alla fase di floorplanning 2. In questa fase, è stata definita la griglia di alimentazione, realizzando una *power grid* più articolata. Per garantire un'adeguata connessione all'alimentazione delle righe situate a sinistra e a destra dei due blocchi, sono state impiegate delle *stripes* in *metal6*. Tuttavia, all'interno dei blocchi, l'uso di *metal6* non è possibile, poiché già impiegato dalle

macro. Per risolvere questa limitazione, sono state quindi aggiunte ulteriori *stripes* verticali in *metal8* e orizzontali in *metal7*. Di seguito, si riporta la percentuale di utilizzo del chip:

Design area = $0 \mu\text{m}^2$, 65% utilization.

2.3.3 Placement

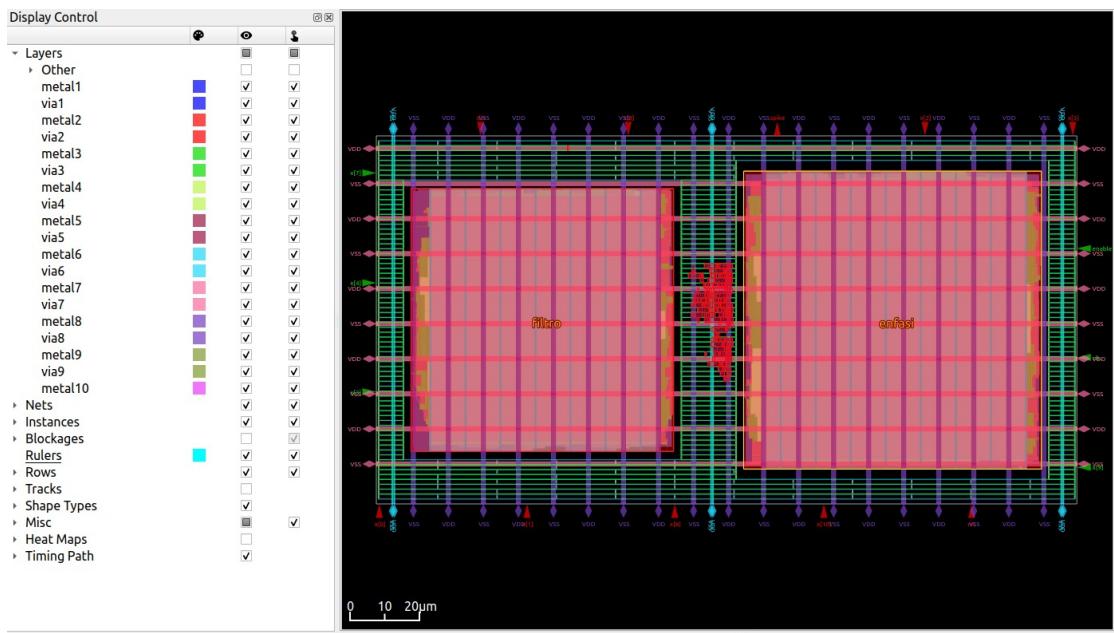


Figure 2.19: Placement DETECTOR

In figura è illustrato il risultato della fase di *detailed placement*, durante la quale le celle vengono leggermente spostate rispetto alla posizione determinata dal *global placement*, con l'obiettivo di rendere il piazzamento delle celle conforme ai vincoli fisici del layout. Al termine del *detailed placement*, il tool fornisce un report sintetico che include un

parametro denominato *HPWL* (*Half Perimeter Wire Length*), il quale rappresenta una stima della lunghezza totale dei collegamenti futuri. Un valore inferiore di *HPWL* indica un *placement* più efficiente. Di seguito sono riportati alcuni dati significativi:

- **Spostamenti delle celle:**
 - total displacement: $233.6 \mu m$
 - average displacement: $0.4 \mu m$
 - max displacement: $6.4 \mu m$
- **HPWL:**
 - *HPWL* originale: $5256.1 \mu m$
 - *HPWL* dopo la legalizzazione: $5363.2 \mu m$
 - Variazione di *HPWL*: $+2\%$

Si osserva che, a seguito della legalizzazione del *placement*, si è verificato un incremento del valore di *HPWL*. Per migliorare ulteriormente la qualità del *placement*, è possibile applicare un'ottimizzazione (*optimize*), che consiste nello spostare le celle in un intorno di $10 \mu m$ (mentre durante la legalizzazione venivano spostate fino a $20 \mu m$). L'ottimizzazione ha prodotto un miglioramento tangibile nel valore di *HPWL*, sebbene ulteriori iterazioni risulterebbero poco efficaci.

- *HPWL* originale: $5328.1 \mu m$

- *HPWL* finale: $5017.3 \mu m$
- Variazione di *HPWL*: -5.8%

2.3.4 Clock Tree

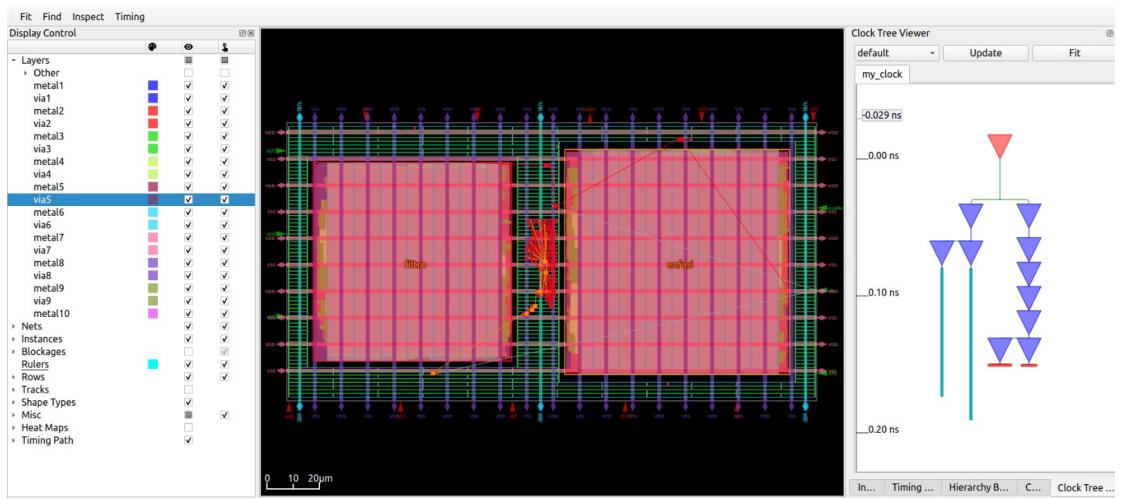


Figure 2.20: Clock tree DETECTOR

Di seguito sono riportati i dati relativi alla struttura dell'albero di clock generato:

- Numero totale di *Clock Roots*: 2
- Numero totale di *Buffer* inseriti: 6
- Numero totale di *Clock Subnets*: 6
- Numero totale di *Sinks*: 17

In figura è illustrato l'albero di clock risultante dopo l'esecuzione del comando `repair_timing -hold`, il quale ha introdotto ulteriori

buffer con l'obiettivo di garantire uno *slack* minimo di 100 ps . Una volta completata la generazione dell'albero di clock, si rende necessario un nuovo processo di legalizzazione del *placement*, in quanto sono stati aggiunti sia i *buffer* del clock che quelli per il vincolo di *hold*. I nuovi valori di *HPWL* risultano essere:

- *HPWL* originale: $5624.7\text{ }\mu\text{m}$
- *HPWL* finale: $5441.7\text{ }\mu\text{m}$
- Variazione di *HPWL*: -3.3%

2.3.5 Routing

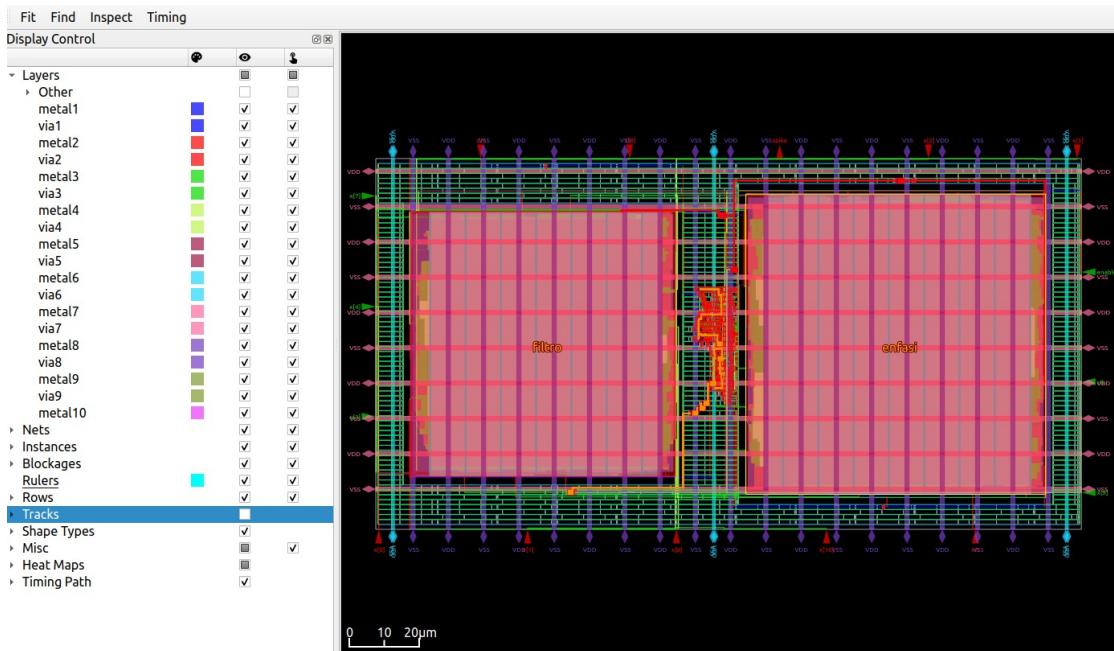


Figure 2.21: Routing DETECTOR

2.3.6 Controlli finali

0.112 slack (MET)					
Group	Internal Power	Switching Power	Leakage Power	Total Power (Watts)	
Sequential	1.09e-05	5.95e-07	1.17e-06	1.27e-05	15.5%
Combinational	6.35e-06	4.66e-06	2.91e-06	1.39e-05	17.0%
Clock	3.89e-05	1.46e-05	1.77e-06	5.53e-05	67.5%
Macro	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.0%
Pad	0.00e+00	0.00e+00	0.00e+00	0.00e+00	0.0%
Total	5.61e-05	1.99e-05	5.86e-06	8.19e-05	100.0%
	68.6%	24.3%	7.2%		

Figure 2.22: Controlli finali DETECTOR

Possiamo fare alcune considerazioni sulla potenza; distinguiamo la potenza interna, esterna e quella dovuta al leakage. Il contributo maggiore di dissipazione è dovuto alla potenza dinamica (interna ed esterna) del clock.

0.000	0.000	clock my_clock (rise edge)
0.000	0.000	clock network delay (ideal)
0.500	0.500	^ input external delay
0.541	1.041	^ rst (in)
0.000	1.041	^ blocco2/rst (enfasi)
	1.041	data arrival time
10.000	10.000	clock my_clock (rise edge)
0.000	10.000	clock network delay (ideal)
-0.100	9.900	clock uncertainty
0.000	9.900	clock reconvergence pessimism
	9.900	^ blocco2/clk (enfasi)
-0.859	9.041	library setup time
	9.041	data required time
	9.041	data required time
	-1.041	data arrival time
	8.000	slack (MET)

Figure 2.23: Controlli finali DETECTOR

In figura è mostrato il valore di *slack* calcolato in relazione al vincolo sul tempo di *setup*. Si osserva che esso differisce da quanto inizialmente stimato nella fase di sintesi. Possiamo determinare la massima frequenza operativa del sistema come segue:

$$T_{\min} = T - 8.00 = 2 \text{ ns}$$

$$f_{\max} = \frac{1}{T_{\min}} \approx 500 \text{ MHz}$$

Si nota che la presenza di effetti parassiti dovuti alle interconnessioni introduce una lieve degradazione delle prestazioni complessive. Infatti, la frequenza massima ottenibile risulta inferiore rispetto a quella

stimata nella fase di sintesi.