POLITECNICO
MILANO 1863

DIPARTIMENTO DI ELETTRONICA
INFORMAZIONE E BIOINGEGNERIA

Prova Finale

Progetto di Reti Logiche

Autori: **Stefano Ungaro** (****** | ********)
Alessandro Ferdinando Verrengia (****** | ********)

Prof. Gianluca Palermo
Anno accademico: 2024-25

Indice

1. Introduzione

1.1. Scopo del progetto

Lo scopo del progetto è la realizzazione di un componente hardware descritto tramite VHDL che si interfacci con una memoria RAM contenente un messaggio di K parole, applichi un filtro differenziale su di esse e salvi in memoria la sequenza di K parole ottenute come risultato.

1.2. Descrizione generale della specifica

Tutti i dati da utilizzare sono memorizzati in memoria, a partire dall'indirizzo ADD, nel modo seguente:

- 17 byte di **preambolo**, che forniscono i valori con cui lavorare. In particolare,
  - K_1 e K_2, due byte, che indicano la lunghezza della sequenza (K) dei dati (W_1...W_k); K_1 è il byte più significativo; **la lunghezza minima è di 7 byte**,
  - S, un byte, che indica quale filtro selezionare. Se il bit meno significativo di S è 0 allora il filtro da utilizzare è quello di ordine 3, altrimenti è quello di ordine 5,
  - da C_1 a C_{14}, 14 byte, che memorizzano i coefficienti dei due filtri. In particolare:
    - ordine 3, c = [0, -1, 8, 0, -8, 1, 0], con normalizzazione $n = 12$,
    - ordine 5, c = [1, -9, 45, 0, -45, 9, -1], con normalizzazione $n = 60$,
- K byte da elaborare, che contengono valori da -128 a +127.

I K elementi ottenuti dopo il filtraggio (R_1...R_k) vengono inseriti in memoria subito dopo i K elementi da elaborare, cioè a partire dall'indirizzo ADD + 17 + K .

Il filtro utilizza la funzione

$$f'(i) = \frac{1}{n} \cdot \sum_{j=-l}^{l} C_j \cdot f[j+i]$$

per calcolare i valori filtrati. In particolare:

- l varia tra -2 e 2 nel caso di filtro di ordine 3 e varia tra -3 e 3 nel caso di filtro di ordine 5,
- C_j è il j-esimo coefficiente del filtro da applicare,
- n è il coefficiente di normalizzazione unico a ciascun filtro.

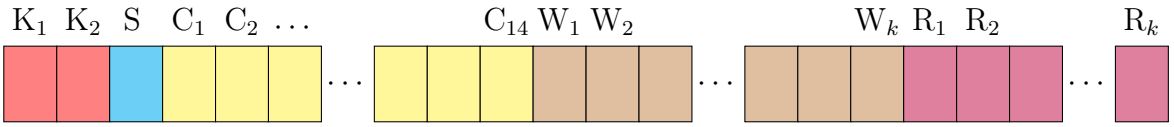

Figura 1: Rappresentazione grafica della memoria RAM.

Si considerano pari a 0 i valori precedenti a W_1 e successivi a W_k durante le fasi di applicazione del filtro.

La divisione viene approssimata attraverso gli *shift* logici a destra, rispettivamente:

- la normalizzazione a $\frac{1}{12}$ è approssimata con $\frac{1}{16} + \frac{1}{64} + \frac{1}{256} + \frac{1}{1024}$, cioè con *shift* di 4, 6, 8 e 10 bit,
- la normalizzazione a $\frac{1}{60}$ è approssimata con $\frac{1}{64} + \frac{1}{1024}$, cioè con *shift* di 6 e 10 bit.

Nel caso della normalizzazione di numeri negativi, a ciascuno dei valori dopo lo *shift* si aggiunge 1 per compensare l'errore di troncamento durante lo *shift*.

1.3. Esempio di funzionamento

Si consideri la lista di valori in ingresso

[47, -2, 39, -71, -108, -111, 37].

Si vuole calcolare la sequenza di valori filtrati attraverso il filtro di ordine 3. Nella memoria, a partire dall'indirizzo esposto su `i_add`, il preambolo sarà così composto:

- $K_1 = 0$,
- $K_2 = 7$,
- S un qualsiasi numero pari (il bit meno significativo vale 0),
- $[C_1, \ldots, C_{14}] = [0, -1, 8, 0, -8, 1, 0, 1, -9, 45, 0, -45, 9, -1]$,
- $[W_1, \ldots, W_7] = [47, -2, 39, -71, -108, -111, 37]$.

Il componente calcolerà i valori filtrati nel seguente modo, ricordando che per il filtro di ordine 3 la normalizzazione è $\frac{1}{12}$ (si utilizza per semplicità il prodotto scalare tra vettori):

- $R_1 = [-1, 8, 0, -8, 1]^T \cdot [0, 0, 47, -2, 39]^T = 55$, normalizzato a 3,
- $R_2 = [-1, 8, 0, -8, 1]^T \cdot [0, 47, -2, 39, -71]^T = -7$, normalizzato a 0,
- $R_3 = [-1, 8, 0, -8, 1]^T \cdot [47, -2, 39, -71, -108]^T = 397$, normalizzato a 31,
- $R_4 = [-1, 8, 0, -8, 1]^T \cdot [-2, 39, -71, -108, -111]^T = 1067$, normalizzato a 87,
- $R_5 = [-1, 8, 0, -8, 1]^T \cdot [39, -71, -108, -111, 37]^T = 318$, normalizzato a 24,
- $R_6 = [-1, 8, 0, -8, 1]^T \cdot [-71, -108, -111, 37, 0]^T = -1089$, normalizzato a -90,
- $R_7 = [-1, 8, 0, -8, 1]^T \cdot [-108, -111, 37, 0, 0]^T = -780$, normalizzato a -63.

1.4. Interfaccia del componente

Il componente progettato deve rispettare un'interfaccia standard definita dal seguente frammento di codice VHDL

```
ENTITY project_reti_logiche IS
    PORT (
        i_clk      : IN  std_logic;
        i_rst      : IN  std_logic;
        i_start    : IN  std_logic;
        i_add      : IN  std_logic_vector (15 DOWNTO 0);

        o_done     : OUT std_logic;

        o_mem_addr : OUT std_logic_vector (15 DOWNTO 0);
        i_mem_data : IN  std_logic_vector (7 DOWNTO 0);
        o_mem_data : OUT std_logic_vector (7 DOWNTO 0);
        o_mem_we   : OUT std_logic;
        o_mem_en   : OUT std_logic;
    );
END project_reti_logiche;
```

Dove:

- `i_clk` è il segnale di CLOCK in ingresso generato dal Test Bench,
- `i_rst` è il segnale di RESET asincrono che inizializza la macchina pronta per ricevere il primo segnale di START,
- `i_start` è il segnale di START generato dal Test Bench,
- `i_add` è il segnale (vettore) ADD generato dal Test Bench che rappresenta l'indirizzo dal quale parte la sequenza da elaborare,
- `o_done` è il segnale DONE di uscita che comunica la fine dell'elaborazione,
- `o_mem_addr` è il segnale (vettore) di uscita che manda l'indirizzo alla memoria,
- `i_mem_data` è il segnale (vettore) che arriva dalla memoria e contiene il dato in seguito a una richiesta di lettura,
- `o_mem_data` è il segnale (vettore) che va verso la memoria e contiene il dato che verrà successivamente scritto,
- `o_mem_en` è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che scrittura),
- `o_mem_we` è il segnale di WRITE ENABLE da mandare alla memoria (=1) per poter scrivere su di essa. Per leggere da memoria deve invece essere 0.

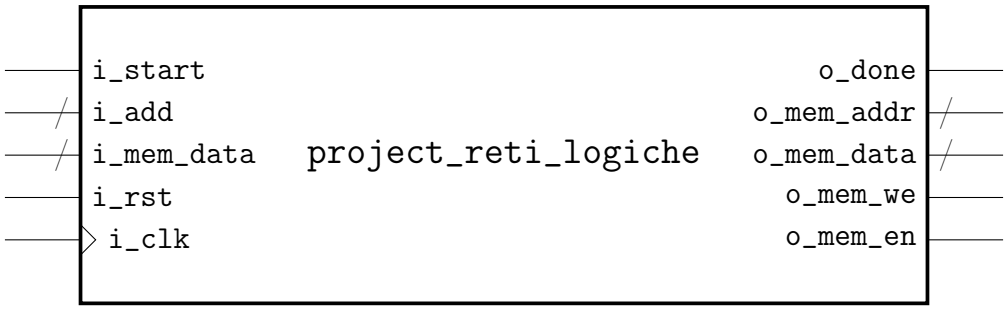

Figura 2: Schema del componente da implementare.

2. Architettura

2.1. Scelte progettuali

Il componente è stato progettato attraverso un **unico modulo**, oltre a quello della memoria RAM, costituito da un **unico processo** VHDL che si occupa di elaborare tutti i segnali di input e output e che implementa la **Macchina a Stati Finiti** (completamente specificata): tale macchina gestisce la logica di funzionamento del componente.

Non è stato reputato necessario suddividere ulteriormente la logica in più processi o moduli diversi per via della sua compattezza e per mantenere il codice sintetico e lineare.

Si è scelto inoltre di unificare gli stati relativi ai due diversi filtri, che si presentano simili nella logica, con piccole variazioni, effettuando le diverse scelte intermedie grazie all'osservazione del segnale `s`, illustrato di seguito. Questo significa che le fasi di calcolo si presentano uguali per entrambi i filtri, con l'accortezza di utilizzare il filtro di ordine 3 esteso in lunghezza da 5 a 7 elementi, con le due estremità poste a 0.

2.2. Segnali utilizzati

Per realizzare il componente sono stati utilizzati i seguenti segnali interni ausiliari:

- `current`, indica qual è lo stato corrente della macchina,
- `lunghezza`, è di tipo `INTEGER` e si occupa di memorizzare la lunghezza K della sequenza da analizzare,
- `s`, è un bit che assume valore 0 quando il filtro da utilizzare è quello di ordine 3 e 1 quando è quello di ordine 5,
- `i`, è il segnale `INTEGER` che indica l'avanzamento relativo della lettura in memoria rispetto a `i_add`; attraverso dei confronti su di esso è possibile distinguere quale parte di memoria si sta leggendo e quale sia l'azione successiva da eseguire. In particolare:
  - se `i` è compreso tra 0 e 19 si stanno leggendo i valori del preambolo e i primi tre valori della sequenza; non è ancora necessario passare agli stati di calcolo,

  - se `i` è compreso tra 19 e K + 16 si stanno leggendo i valori della sequenza e sarà necessario passare agli stadi di calcolo del valore filtrato,
  - se `i` è compreso tra K + 17 e K + 19 si stanno calcolando gli ultimi tre valori filtrati e non sarà necessario richiedere nuovi valori alla memoria,
  - se `i` è pari a K + 20 si è conclusa l'elaborazione.

- `filtro`, è un vettore di `INTEGER` che memorizza i sette valori del filtro da utilizzare (nel caso di filtro di ordine 3 il primo e l'ultimo valore sono forzati a 0).
- `valori`, è il vettore di `INTEGER` contenente i sette valori della sequenza da utilizzare per calcolare il valore filtrato. Quando l'elaborazione coinvolge i primi tre elementi della sequenza o gli ultimi tre il vettore è completato con degli 0 come da specifica.
- `pre_norm`, è un segnale intermedio tra lo stato di pre-normalizzazione e lo stato di normalizzazione e scrittura in memoria.

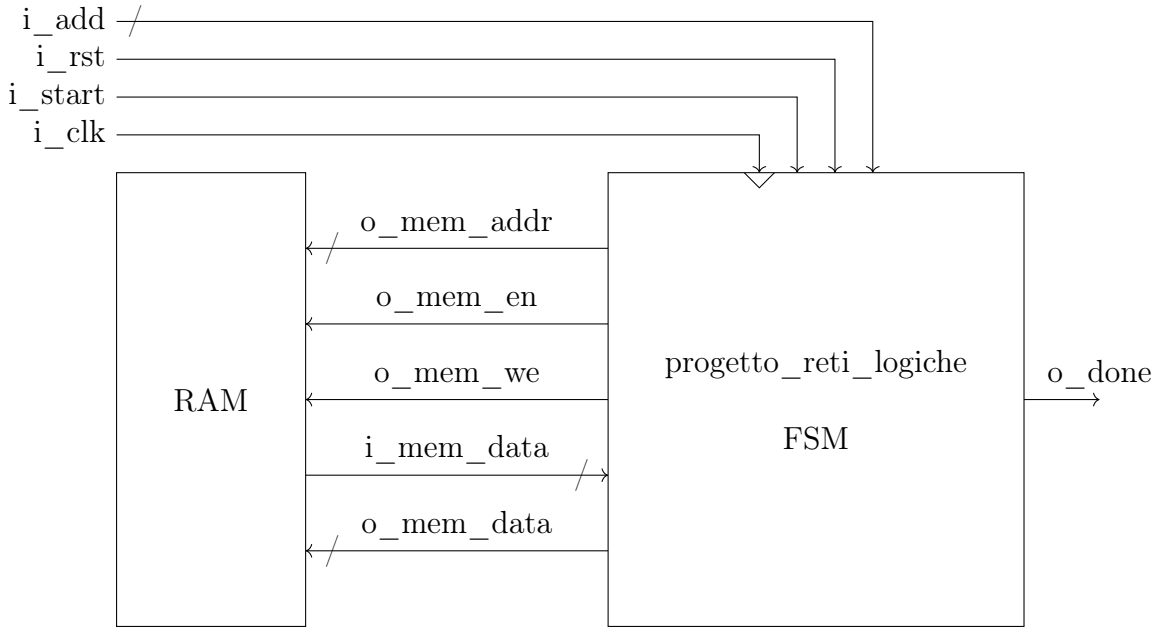

Figura 3: Schema del modulo e della RAM.

2.3. Macchina a Stati Finiti

La **Macchina a Stati Finiti** (completamente specificata) si occupa di gestire la logica e il funzionamento del componente ed è l'unico modulo del progetto. Si compone di **sette stati**, descritti in seguito. Sia il flusso di esecuzione che il passaggio da uno stato a un altro sono gestiti con l'aiuto del segnale ausiliario `i`, come illustrato nel paragrafo precedente.

Si è scelto di mantenere il numero di stati relativamente basso, raggruppando le fasi di elaborazione in alcuni macro-gruppi. Rispetto a una prima versione del progetto, che prevedeva sei stati (raggruppando pre-normalizzazione, normalizzazione e scrittura in memoria in un unico stato), si è scelto di suddividere il calcolo dei valori filtrati in due stati, uno di pre-normalizzazione e in uno successivo di normalizzazione e scrittura.

Questa scelta di progetto è frutto di un'analisi dei percorsi critici e dei *report* post sintesi che evidenziavano una logica eccessivamente complessa nello stato unificato che produceva un successivo *slack* troppo basso per essere reputato soddisfacente. Aggiungendo uno stato si è quindi resa la macchina più stabile a scapito di un tempo di elaborazione totale leggermente più alto.

La macchina possiede inoltre uno stato, denominato `IDLE`, dove viene riportata ogni volta che viene ricevuto un segnale di RESET (alto).

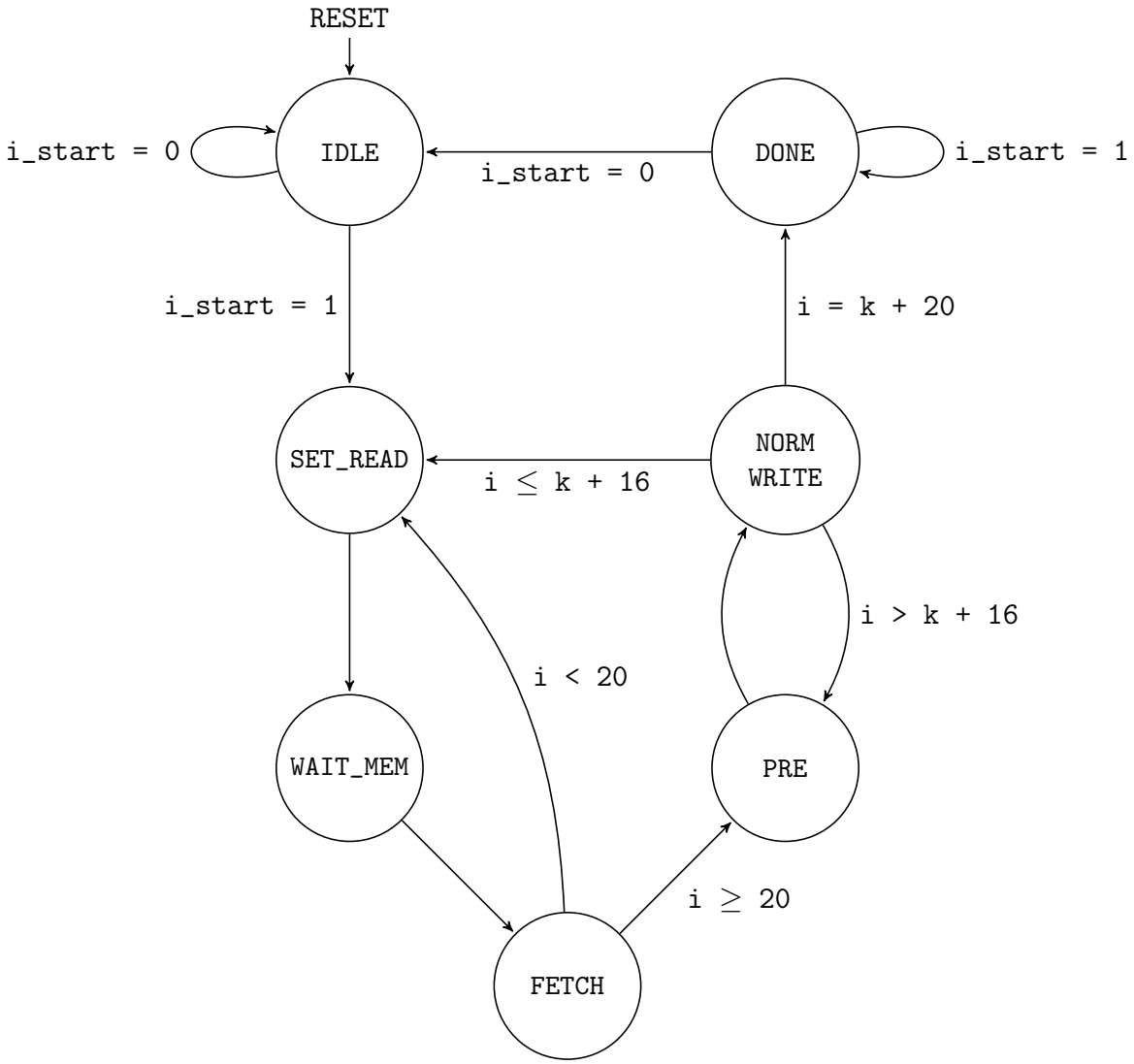

Figura 4: Diagramma degli stati della Macchina a Stati Finiti. Si noti che ogni stato ha una transizione implicita, qui non rappresentata, verso lo stato di `IDLE` nel caso di ricezione di un segnale di RESET.

Segue ora una descrizione degli stati e del loro comportamento.

2.3.1. Stato di IDLE

Si tratta dello stato in cui la macchina si posiziona subito dopo aver ricevuto il RESET e dove resta in attesa che `i_start` venga alzato a 1. Se ciò avviene, lo stato successivo è `SET_READ`, altrimenti la macchina rimane in `IDLE`. In questo stato vengono inizializzati tutti i registri della macchina.

2.3.2. Stato di SET_READ

In questo stato la macchina espone sull'uscita `o_mem_addr` l'indirizzo a cui si desidera accedere in lettura, alzando `o_mem_en` e abbassando `o_mem_we`. Poiché è necessario scorrere la memoria sequenzialmente, viene utilizzato il segnale interno `i` per determinare l'indirizzo di lettura a partire da `i_add`. La macchina si porta poi nello stato di attesa `WAIT_MEM`.

2.3.3. Stato di WAIT_MEM

Si tratta di uno stato intermedio in cui la macchina non esegue alcuna azione ma resta in attesa per un ciclo di clock che la memoria recepisca l'indirizzo e fornisca i dati richiesti. La macchina si porta quindi nello stato di `FETCH`.

2.3.4. Stato di FETCH

Lo stato di `FETCH` si occupa di salvare nei registri interni alla macchina i valori che sono stati richiesti alla memoria dagli stati precedenti. In base al valore di `i` la macchina ha modo di interpretare la tipologia di dato appena ricevuto per salvarlo nel registro corretto:

- se `i` < 17 si stanno leggendo i valori del preambolo, che vengono salvati, dopodiché la macchina torna nello stato di `SET_READ` incrementando anche il segnale `i` in quanto non è ancora iniziata la fase di calcolo,
- se `i` ≥ 17 si stanno leggendo, uno alla volta, i dati da filtrare, che vengono quindi inseriti in ultima posizione nel registro interno `valori`. Inoltre, la macchina attende che vengano letti tutti i primi valori della sequenza tornando in `SET_READ` prima di passare allo stato di `PRE`.

Lo stato di `FETCH` è inoltre ottimizzato per salvare unicamente i valori del filtro richiesto all'inizio attraverso il byte S, dato che non verrà richiesto un cambio di tipologia di filtro durante l'esecuzione.

2.3.5. Stato di PRE

Lo stato di `PRE` è uno dei due stati relativi al calcolo del valore filtrato e si occupa di elaborare il valore pre-normalizzazione come

$$\texttt{pre\_norm} = \sum_{j=-l}^{l} C_j \cdot f[j+i].$$

Si tratta dello stato con il percorso critico più lungo e con l'esecuzione più lenta: ciò è dovuto alla presenza di numerose somme e moltiplicazioni. Per cercare di ottimizzarne i tempi di elaborazione, sono stati implementati alberi delle somme che spezzano il percorso critico in un numero di passi logaritmico rispetto a quello iniziale.

Dopo il calcolo del valore pre-normalizzazione la macchina passa allo stato di `NORM_WRITE`.

2.3.6. Stato di `NORM_WRITE`

In questa seconda e ultima fase di calcolo la macchina normalizza il valore calcolato nello stato precedente utilizzando il coefficiente richiesto da ciascun filtro, ossia

$$\texttt{norm} = \frac{1}{n} \cdot \texttt{pre\_norm}.$$

Lo stato si avvale pertanto della variabile `norm`, di tipo `INTEGER`, che contiene il valore dopo la normalizzazione; ciò è necessario per controllare che appartenga al range di valori rappresentabili in 8 bit ed eventualmente saturarlo, prima di esporlo in scrittura.

Per la divisione vengono utilizzati degli *shift* logici verso destra come illustrato nella descrizione della specifica [1.2]. Anche in questo stadio vengono utilizzati alberi delle somme per ottimizzare i calcoli.

Dopo aver ottenuto il valore normalizzato `norm`, la macchina si occupa di predisporre la scrittura, alzando `o_mem_we` e `o_mem_en`, esponendo su `o_mem_data` il valore da salvare e su `o_mem_addr` l'indirizzo in cui salvarlo (ottenuto con `i_add` + `i` - 4).

Al termine delle operazioni di scrittura la macchina verifica se sia necessario richiedere alla memoria un nuovo valore della sequenza valutando il segnale `i`, passando a `SET_READ`, oppure inserire direttamente degli 0 in coda al vettore, restando in `CALC` (ultimi tre valori); se l'elaborazione è invece conclusa la macchina passa allo stato di `DONE`.

2.3.7. Stato di `DONE`

Lo stato di `DONE` si occupa di alzare l'uscita `o_done` e di disabilitare la memoria. La macchina permane in questo stato finché il segnale `i_start` non viene abbassato, riportando in tal caso il componente in `IDLE` e abbassando `o_done`, pronto per una nuova elaborazione.

3. Risultati sperimentali

Il componente è stato sintetizzato correttamente e ha superato le **simulazioni *Behavioural*, *Post-Synthesis Functional*** e anche ***Post-Synthesis Timing***. Si riportano i *report* dei *tool* di sintesi e i risultati dei Test Bench eseguiti.

3.1. Sintesi

È stato utilizzato, come da specifica, lo strumento **Xilinx Vivado Webpack** con FPGA target Artix 7 FPGA xc7a200tfbg484-1.

3.1.1. `report_utilization`

La sintesi ha generato il seguente `report_utilization`.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	773	0	134600	0.57
LUT as Logic	773	0	134600	0.57
LUT as Memory	0	0	46200	0.00
Slice Registers	171	0	269200	0.06
Register as Flip Flop	171	0	269200	0.06
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

I valori più importanti da osservare sono il numero di **Flip Flop** e di **Latch**. Rispetto a una prima stesura del progetto, in seguito a diverse ottimizzazioni, il numero di Flip Flop utilizzati è diminuito circa del 15%. Si può notare anche l'assenza di Latch, indice di una buona riuscita dell'implementazione.

3.1.2. `report_timing`

La sintesi ha prodotto anche il *report* sul *timing*, generato considerando come *constraint* un **periodo di clock pari a 20 ns**.

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

 WNS(ns)  TNS(ns)  TNS Failing Endpoints  TNS Total Endpoints
 -------  -------  ---------------------  -------------------
 6.995      0.000                      0                 1189

All user specified timing constraints are met.
```

Come si può notare, il *Worst Negative Slack* (WNS) è di poco inferiore a 7 ns. Ciò significa che, su un ciclo di clock di 20 ns, il componente impiega al più circa 13 ns per l'elaborazione più lunga.

Come menzionato nel paragrafo concernente le scelte progettuali, una prima versione del progetto prevedeva l'unione degli stati di `PRE` e `NORM_WRITE` in un unico stato `CALC`; questa versione presentava però uno *slack* di soli 0.5 ns, ritenuto troppo basso per mantenere un soddisfacente margine rispetto al periodo di clock, in un'ottica di reale implementazione fisica in cui i *delay* sono anche maggiori di quelli simulati e in cui sono presenti rumori.

Un'analisi approfondita ha permesso di comprendere come il percorso critico si presenti nella fase di somma pre-normalizzazione, ovvero le sei moltiplicazioni succedute da sei somme che, sebbene ottimizzate attraverso un albero delle somme, mantenevano ancora un elevato tempo di elaborazione: isolando questa fase del calcolo in uno stato a sé stante è stato possibile ridurre drasticamente i tempi di elaborazione.

Si riporta anche l'estratto dal `report_timing` che descrive il percorso critico.

```
Max Delay Paths
--------------------------------------------------------------------
Slack (MET) :             6.995ns (required time - arrival time)
  Source:                 pre_norm2__0/CLK
                            (rising edge-triggered cell DSP48E1
                            clocked by clock  {rise@0.000ns
                            fall@10.000ns period=20.000ns})
  Destination:            pre_norm_reg[29]/D
                            (rising edge-triggered cell FDRE
                            clocked by clock  {rise@0.000ns
                            fall@10.000ns period=20.000ns})
  Path Group:             clock
  Path Type:              Setup (Max at Slow Process Corner)
  Requirement:            20.000ns  (clock rise@20.000ns -
                                      clock rise@0.000ns)
  Data Path Delay:        12.887ns  (logic 9.327ns (72.376%)
                                      route 3.560ns (27.624%))
  Logic Levels:           11  (CARRY4=5 DSP48E1=1 LUT2=1
                                LUT3=2 LUT4=2)
```

Come si può notare, il percorso critico è situato lungo il segnale `pre_norm` che deve attraversare undici livelli di logica prima di terminare la sua elaborazione.

3.2. Simulazioni

Le simulazioni hanno permesso di testare il componente sia in regime di funzionamento normale, verificando la correttezza dei calcoli e degli accessi in memoria, sia in casi di funzionamento nei *corner case* della specifica, come la sequenza di lunghezza minima e i valori normalizzati eccedenti il range rappresentabile su 8 Byte.

Tutti i Test Bench hanno superato le simulazioni ***Behavioural***, **funzionali post-sintesi** e anche **temporali post-sintesi** (non richieste dalla specifica ma verificate ugualmente).

3.2.1. Test Bench 1 - lunghezza minima

Il primo Test Bench ha verificato il corretto funzionamento del componente nel caso in cui la stringa di ingresso (escluso il preambolo) fosse della lunghezza minima consentita (7 Byte). Questo test ha inoltre permesso di verificare la correttezza dei calcoli effettuati.

È stato utilizzato l'esempio 1 fornito con la specifica come fonte dei dati da testare.

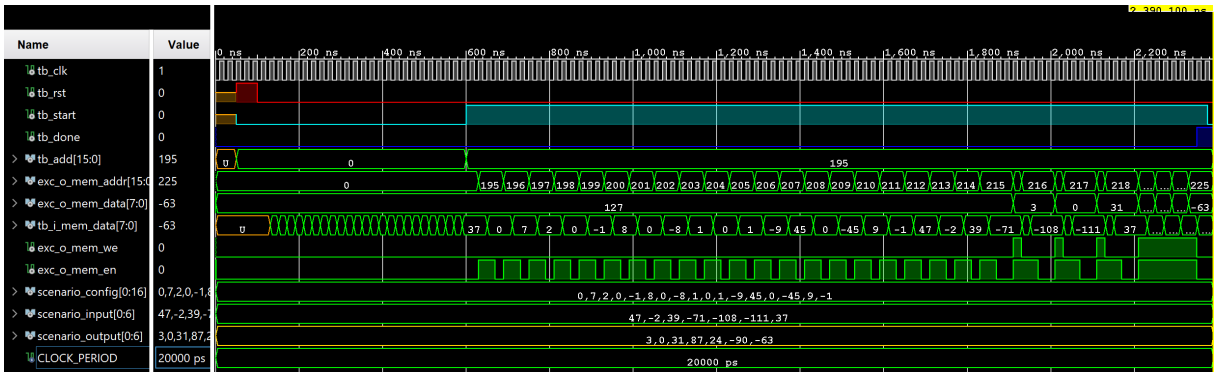

Figura 5: Test Bench 1.

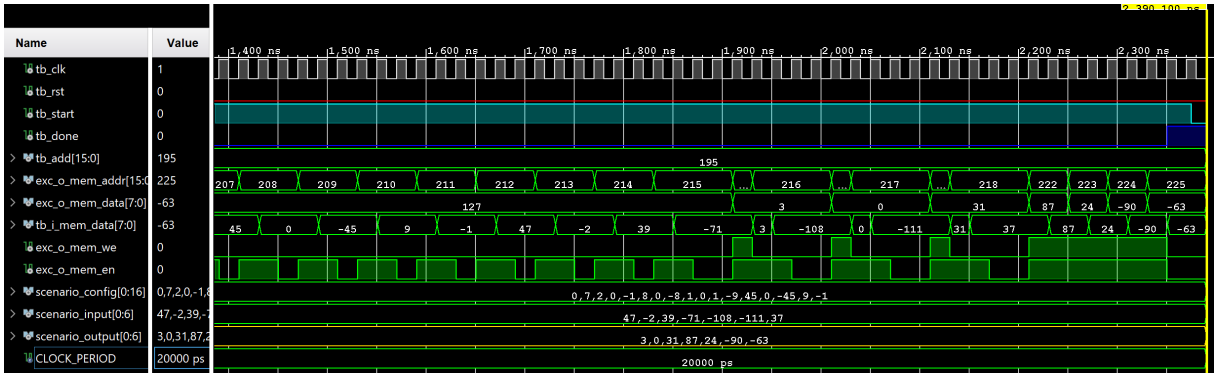

Figura 6: Test Bench 1 - dettaglio della fase di calcolo.

Come si può notare nelle figure 5 e 6, la macchina risponde correttamente all'input e produce i risultati sperati (nella riga evidenziata in giallo sono presenti i valori di output attesi), alzando il segnale DONE (blu) al termine.

3.2.2. Test Bench 2 - segnale di reset e seconda elaborazione

Il secondo Test Bench si è occupato di verificare che la macchina si azzeri correttamente qualora riceva un segnale asincrono di RESET durante l'elaborazione di una sequenza, in qualsiasi fase essa sia. Al termine dell'elaborazione della prima sequenza (prima interrotta e poi rielaborata da capo), il test ha inoltre richiesto alla macchina l'elaborazione di una seconda sequenza, per verificare il corretto funzionamento della macchina per più elaborazioni sequenziali.

Il test ha anche verificato che la macchina ignori i coefficienti C_1 e C_7, che non devono essere utilizzati dal filtro di ordine 3.

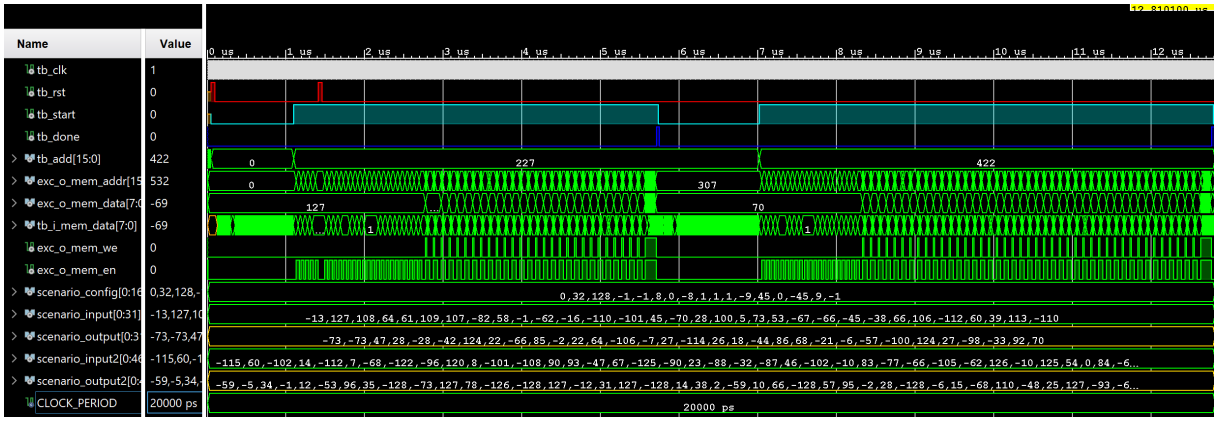

Figura 7: Test Bench 2.

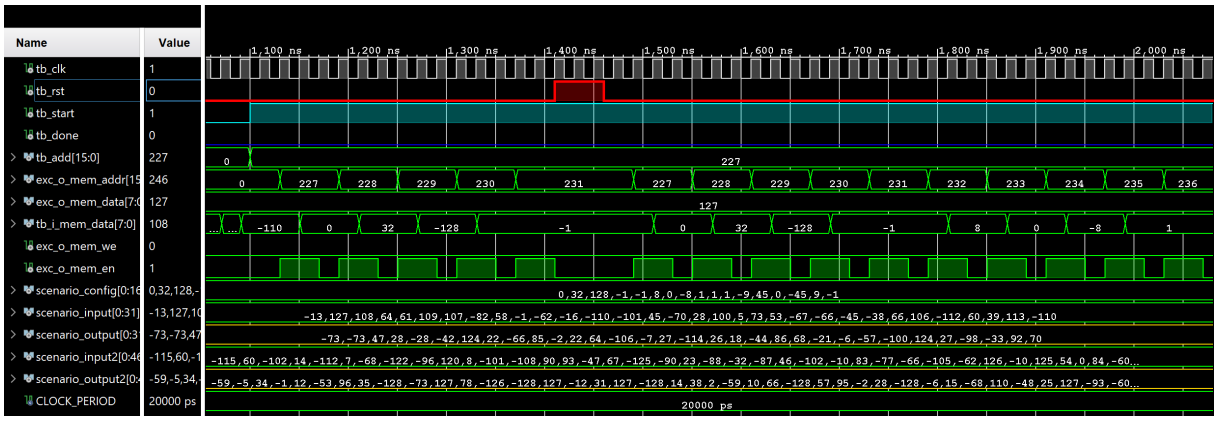

Figura 8: Test Bench 2 - dettaglio della ricezione del RESET asincrono.

Come si può notare in figura 7 e in figura 8, la macchina riceve il RESET (in rosso) dopo circa 1400 ns e ricomincia da capo la lettura del preambolo (stava leggendo C_1).

Dopo l'elaborazione della prima stringa (che ha inoltre testato il filtro di ordine 3) la macchina alza il segnale di DONE (blu) e lo riabbassa quando START viene posto a 0. Quando viene richiesta la seconda elaborazione (che testa il filtro di ordine 5) la macchina funziona correttamente.

3.2.3. Test Bench 3 - lunghezza massima

Un terzo Test Bench si è reso necessario per controllare che la macchina interpretasse in modo corretto i dati forniti dalla memoria per quanto riguarda la lunghezza della sequenza da elaborare (due byte consecutivi). In questo test si è proposta al componente l'elaborazione di una sequenza di lunghezza 32759 (la massima che può essere contenuta nella memoria istanziata dal Test Bench). In questo modo si è testata anche la persistenza temporale del componente.

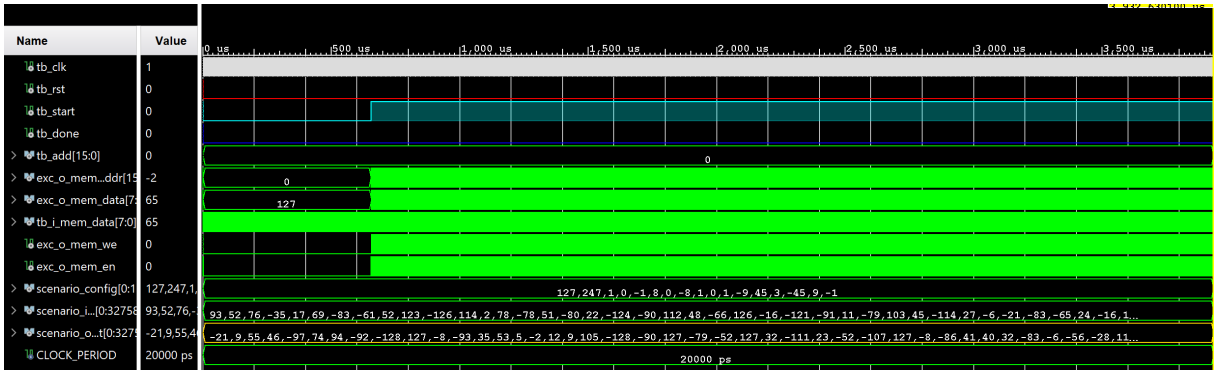

Figura 9: Test Bench 3.

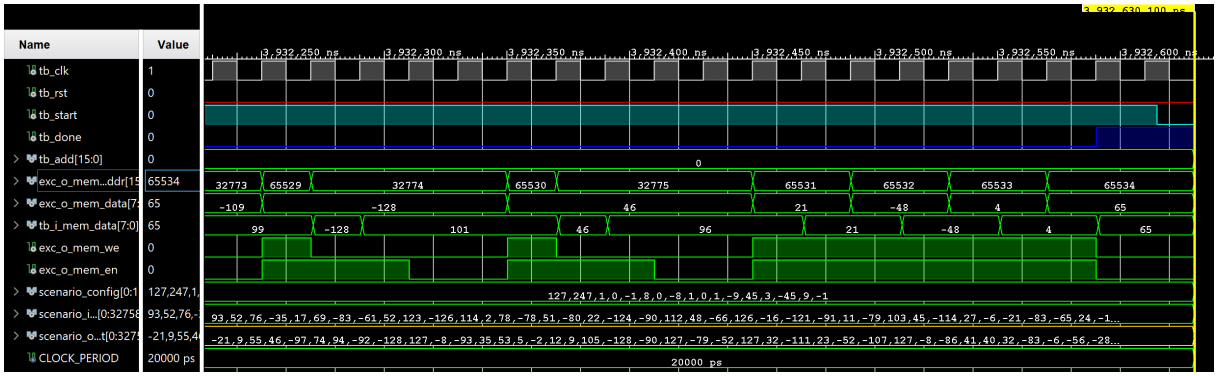

Figura 10: Dettaglio della fase finale del Test Bench 3.

L'elaborazione avviene in maniera corretta sino alla fine. I segnali che rappresentano gli indirizzi e gli indici interni non subiscono *overflow* di alcun tipo.

3.2.4. Test Bench 4 e 5 - valori limite

Sono stati creati e utilizzati alcuni Test Bench che verificano la corretta elaborazione dei segnali qualora il valore pre-normalizzazione raggiunga il massimo (rispettivamente il minimo) valore rappresentabile, cioè $128 \cdot 128 \cdot 7$ (rispettivamente $-128 \cdot 128 \cdot 7$). Inoltre viene valutata la corretta saturazione dei risultati qualora eccedano il range di valori rappresentabili su 8 bit in complemento a 2, cioè [-128, 127].

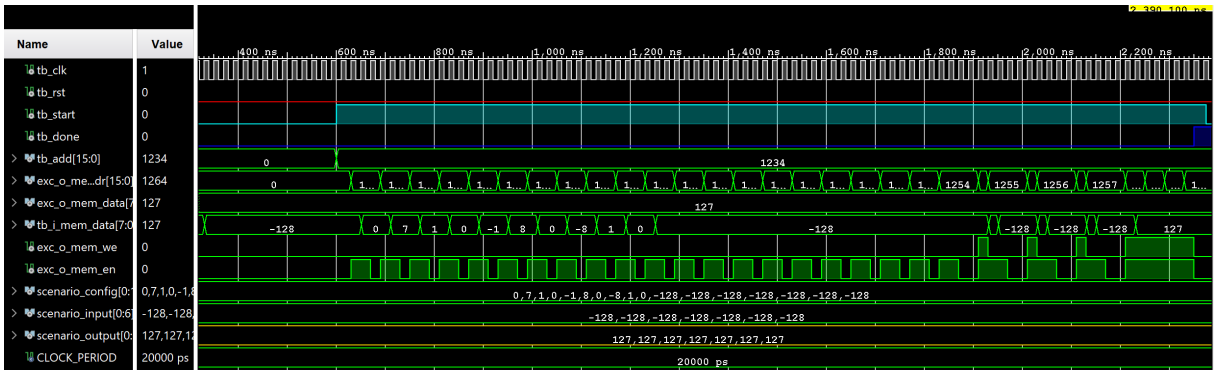

Figura 11: Test Bench 4 - valori massimi rappresentabili, in giallo gli output attesi.

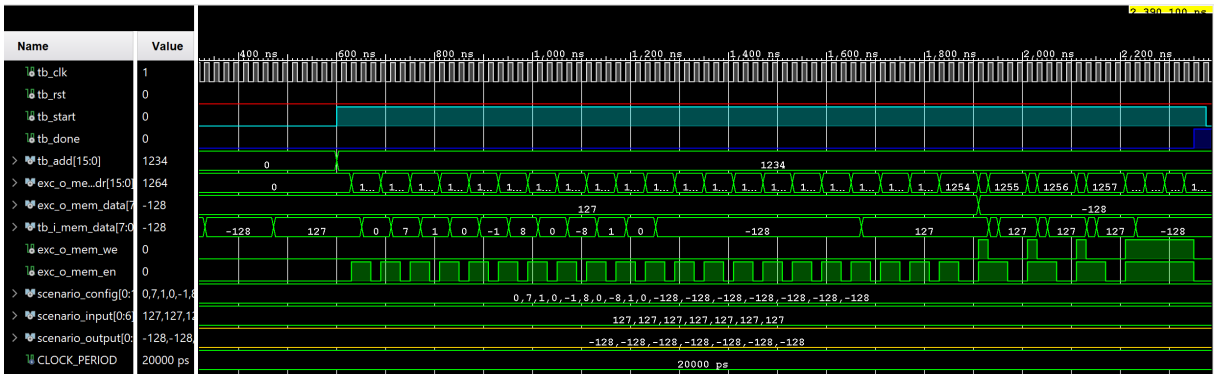

Figura 12: Test Bench 5 - valori minimi rappresentabili, in giallo gli output attesi.

3.2.5. Test Bench aggiuntivi

Sono stati inoltre eseguiti ulteriori Test Bench con i dati provenienti dalla specifica (esempi dal 2 al 6) e con dati manualmente generati per verificare la correttezza dell'elaborazione sia con il filtro di ordine 3 che con il filtro di ordine 5, tutti correttamente superati, ma non se ne riportano i grafici dei segnali perché non rilevanti.

I test hanno anche verificato che il componente elabori correttamente la sequenza anche quando i coefficienti del filtro cambiano (ignorando sempre C_1 e C_7).

4. Conclusioni

Il risultato ottenuto risponde alle specifiche fornite in modo soddisfacente ed è resiliente sia nel tempo che nelle funzioni. Le scelte progettuali (come la separazione degli stati di calcolo e l'ottimizzazione delle somme) e il corretto funzionamento post-sintesi mostrano il componente adatto anche a una possibile implementazione hardware.

Grazie alla scelta di mantenere il componente in un unico modulo e processo è possibile scalare facilmente il progetto, per via della sua linearità e immediatezza.