POLITECNICO
MILANO 1863

Progetto Finale di Reti Logiche

Claudio De Blasio - 10854381
Matteo Enut - 10828090

8 giugno 2025

1 Introduzione

Lo scopo del progetto è quello di descrivere ed implementare, in linguaggio VHDL, un modulo hardware che legga i dati dalla memoria, applichi un filtro differenziale ed infine scriva il risultato in memoria

1.1 Descrizione generale

Il sistema elabora una sequenza di dati memorizzata in memoria, seguendo una precisa struttura di lettura e scrittura. Le operazioni si svolgono secondo i seguenti passaggi:

1. **Lettura dei dati di configurazione:**
   A partire da un indirizzo specifico in memoria, indicato con *ADD*, il sistema legge 17 byte consecutivi così organizzati:
   a. **K1 e K2** (2 byte) rappresentano, combinati, la lunghezza K della sequenza da elaborare. Il valore *K* viene ottenuto interpretando i due byte come un intero a 16 bit con *K1* il byte più significativo.
   b. **S** (1 byte): definisce il tipo di filtro da applicare. In particolare, il bit meno significativo determina la scelta tra due filtri disponibili:
      i. Se è 0 il filtro è di ordine 3
      ii. Se è 1 il filtro è di ordine 5
   c. **Da C1 a C14**(14 byte): contengono i coefficienti utilizzati per i filtri.
2. **Lettura della sequenza di dati**
   Una volta acquisiti i parametri di configurazione, il sistema legge dalla memoria *K* byte di dati a partire dall'indirizzo *ADD + 17*. Ogni byte rappresenta un valore compreso tra -128 e +127.
3. **Applicazione filtro**
   Il filtro selezionato (di ordine 3 o 5) viene applicato alla sequenza dei K valori letti, utilizzando i coefficienti memorizzati.

4.Scrittura del risultato in memoria

I valori risultanti dall'elaborazione vengono scritti in memoria immediatamente dopo la sequenza originale, ovvero a partire dall'indirizzo *ADD + 17 + K*.

1.2 Il filtro

Il filtro calcola per ciascun elemento della sequenza d'ingresso una media pesata di valori vicini secondo la formula :$f'(i) = \frac{1}{n} \cdot \sum_{j=-l}^{+l} Cj \cdot f[i+j]$

Dove

- f(i) è il valore della sequenza in i
- f'(i) è il valore filtrato corrispondente
- Cj rappresenta il j-esimo coefficiente del filtro
- l è uguale a 2 per il filtro di ordine 3 e uguale a 3 per il filtro di ordine 5
- n è il fattore di normalizzazione: 12 per l ordine 3 e 60 per l ordine 5

La sommatoria ci indica la "finestra" su cui il filtro viene applicato in quella determinata posizione della sequenza.

Poiché il filtro utilizza anche valori vicini a quello corrente, è necessario gestire i margini della sequenza (inizio e fine) dove alcuni valori richiesti non esistono. In questi casi, il sistema assume valori pari a 0.

Il risultato della somma pesata viene diviso per il coefficiente di normalizzazione n. Tale divisione non viene effettuata direttamente, ma è approssimata mediante somme di shift bit a destra.

Dopo l'applicazione del filtro e della normalizzazione, il risultato viene saturato all'interno dell'intervallo [-128, +127] per garantire che ogni valore risultante possa essere rappresentato su 8 bit con segno.

1.3 Esempio di funzionamento

All'avvio, il componente riceve il segnale di START e legge i dati dalla memoria a partire da un indirizzo specificato, ad esempio *ADD = 0x0000.*

I primi 17 byte contengono le informazioni necessarie per configurare l'elaborazione della sequenza. In particolare i primi due byte all'indirizzo *0x0000* e *0x0001* saranno *K1* e *K2* e indicano che la sequenza da elaborare è di K=24.

Il terzo byte all indirizzo *0x0002* sarà S il cui bit meno significativo è 0, quindi viene selezionato il filtro differenziale di ordine 3.

I successivi 14 byte contengono i coefficienti da utilizzare per i filtri di ordine 3 e 5. In questo caso, per il filtro di ordine 3, i coefficienti utilizzati sono: -1, 8, 0, -8, 1.

Una volta caricata la configurazione, il componente accede ai K byte successivi in memoria *(da 0x0011 a 0x0028*) che contengono la sequenza di input su cui applicare il filtro.
Il filtro di ordine 3 viene quindi applicato su tutta la sequenza, un elemento alla volta.

Dopo il calcolo, il valore ottenuto viene saturato all'intervallo [-128, +127], poiché il formato dei dati è su 8 bit con segno. Ad esempio, se il risultato grezzo è -151, il valore memorizzato sarà -128.

Infine, i 24 risultati filtrati vengono scritti consecutivamente in memoria a partire dall'indirizzo (*0x0029*), occupando 24 byte.

1) VALORI DI BASE

Variabile	Valore
ADD	0x0000
K1	0 (dec)
K2	24 (dec)
S	0 (dec)

C1	**C2**	**C3**	**C4**	**C5**	**C6**	**C7**	**C8**	**C9**	**C10**	**C11**	**C12**	**C13**	**C14**
0	-1	8	0	-8	1	0	1	-9	45	0	-45	9	-1

2) SEQUENZA ORIGINALE

-19	-5	76	-118	-13	124
9	-18	-89	-120	-74	-85
-128	-51	73	-106	-92	-3
-79	121	24	-12	-30	4

3) SEQUENZA FILTRATA

10	-52	-17	21	-128	-7
94	40	17	-14	-26	6
-10	-101	28	110	-66	16
-85	-66	102	40	-10	91

2 Architettura

L'interfaccia del componente è la seguente:

```
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;
```

Figura 1: Interfaccia del componente

Segnali di input e output del componente

- *i_clk*: Segnale di clock.
- *i_rst*: Segnale di reset.
- *i_start*: Segnale di start per l'elaborazione.
- *i_add*: Indirizzo dell'inizio del preambolo.
- *o_done*: Segnale di fine elaborazione.
- *o_mem_addr*: Indirizzo di lettura/scrittura in memoria.
- *i_mem_data*: Valore letto dalla memoria.
- *o_mem_data*: Valore da scrivere in memoria.
- *o_mem_we*: Flag per scrivere in memoria.
- *o_mem_en*: Flag per l'accesso alla memoria.

Funzionamento di base del componente

Il comportamento del componente è gestito da una macchina a stati, schematizzata nella *Figura 2*.

Il componente si trova inizialmente in uno stato di *IDLE*, nel quale attende che gli input *i_start* e *i_rst* siano rispettivamente *1* e *0*.

L'elaborazione comincia con la lettura in sequenza dei primi 17 byte.
Vengono letti i byte *K1* e *K2* per calcolare *K*, dal quale dipende, ad esempio, l'indirizzo di inizio scrittura dei risultati: *ADD + 17 + K*. *K1* viene letto e salvato in un registro in attesa di *K2* ed al suo arrivo *K* viene calcolato ed utilizzato, senza bisogno di salvarlo.

Successivamente viene letto il byte S e salvato per poi applicare il filtro selezionato nello stato *COMPUTE_R*.

In base ad S nello stato *STORE_C* si leggono i 7 coefficienti del filtro che si sta applicando salvandoli in un array.
In seguito, nello stato *STORE_W*, avviene l'inizializzazione della finestra con il caricamento dei primi 4 valori *W*.

Al passo successivo comincia l'applicazione del filtro, che avviene secondo la seguente logica: *COMPUTE_R* calcola il risultato in 3 step: applicazione del filtro, normalizzazione e saturazione. Fatto ciò, lo carica in memoria e successivamente effettua una scelta in base ai due indirizzi correnti: *current_addr* e *w_addr*.

- **current_addr >= r_end_addr**, l'indirizzo di scrittura ha superato l'ultimo byte disponibile per la scrittura dei risultati, quindi si è arrivati alla fine dell'elaborazione e si passa allo stato DONE.
- ***current_addr < r_end_addr***, l'elaborazione è ancora in corso e si esegue il seguente controllo:
  - ***w_addr < r_start_addr***, ci sono ancora valori W da leggere, quindi si passa allo stato *GET_NEXT*, il quale legge dalla memoria il prossimo valore.
  - ***w_addr >= r_start_addr***, si passa direttamente allo stato *SHIFT_WINDOW*, che carica uno 0 al posto del valore corrispondente.

Nello stato *SHIFT_WINDOW* viene effettuato lo slittamento a sinistra della finestra. Viene quindi eliminato il primo valore e l'ultimo valore diventa *W* letto dalla memoria oppure *0* se viene terminata la sequenza iniziale.
Nello stato *DONE* si alza il segnale *o_done,* che segnala la fine dell'elaborazione, e si attende che *i_start* venga abbassato per tornare allo stato iniziale *IDLE*.

Ad ogni momento dell'elaborazione, se viene alzato il segnale *i_rst* si ritorna allo stato di *IDLE*.

La memoria
La memoria è una *Single-Port Block RAM*. Questa permette di eseguire le seguenti operazioni:

- Lettura:
  o_mem_en = 1,
  o_mem_we = 0,
  o_mem_addr = indirizzo_di_lettura
- Scrittura:
  o_mem_en = 1,
  o_mem_we = 1,
  o_mem_addr = indirizzo_di_scrittura,
  o_mem_data = dato da scrivere

Questo tipo di memoria introduce delle limitazioni. Essendo single-port, si può leggere o scrivere un solo valore alla volta, ed inoltre lettura e scrittura non possono avvenire nello stesso ciclo di clock. Questo comporta, ad esempio, la divisione in cicli diversi della lettura del preambolo: bisogna leggere K1 e solo al ciclo successivo è possibile leggere K2.

2.1 Macchina a stati

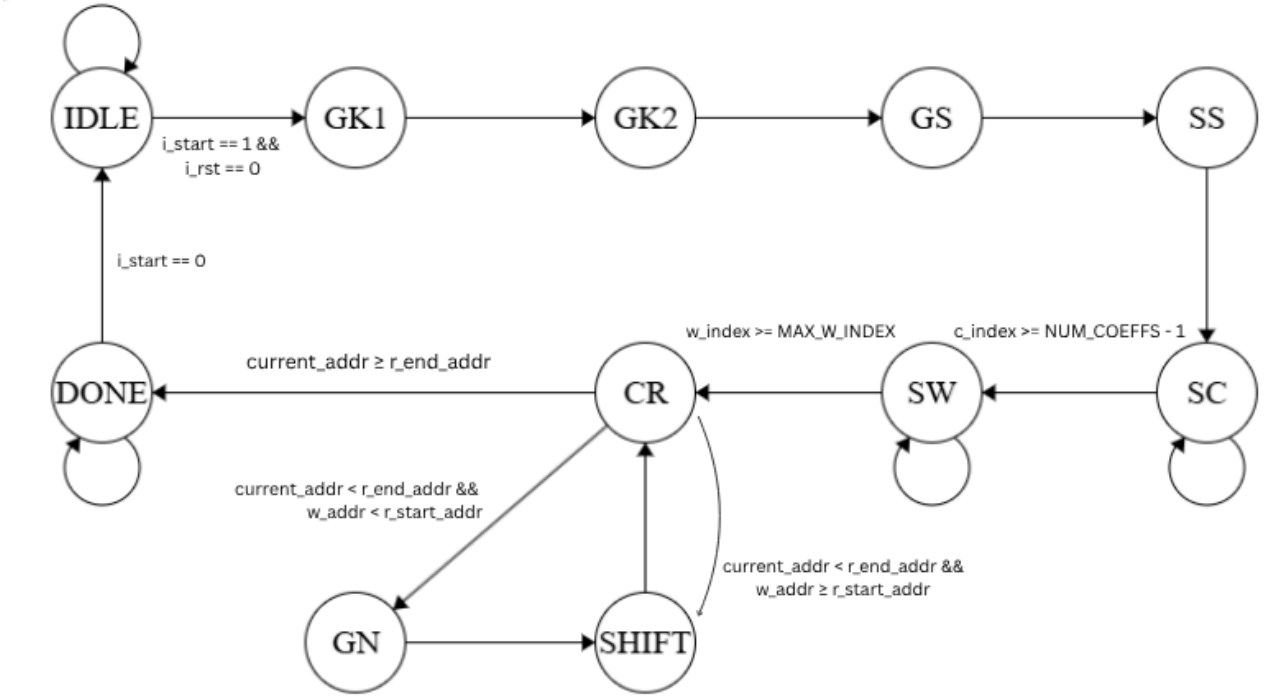

Figura 2: Rappresentazione della macchina a stati

Stati utilizzati

- ***IDLE***: Stato iniziale e di reset, inizializza i registri, attende i_start = 1 e i_rst = 0.
- ***GET_K1 (GK1)***: Configura l'ouput per la lettura di K1 all'indirizzo in i_add, inizializza current_addr e w_start_addr, che dipendono da i_add.
- ***GET_K2 (GK2)***: Salva K1 e legge K2.
- ***GET_S (GS)***: Usa K1 e K2 presente in i_mem_data per calcolare k e di conseguenza r_start_addr e r_end_addr, i quali dipendono da k.
- ***STORE_S (SS)***: Salva S per poi utilizzarlo nella selezione del filtro, imposta current_addr all'indirizzo corrispondente al primo coefficiente del filtro e lo legge.
- ***STORE_C (SC)***: Salva il coefficiente e legge il prossimo. Quando *c_index* >= *NUM_COEFFS - 1* legge il primo valore da salvare in finestra e passa allo stato successivo.
- ***STORE_W (SW)***: Salva il valore in finestra e legge il prossimo. Quando *w_index* >= *MAX_W_INDEX* passa all'applicazione del filtro in *COMPUTE_R*.
- ***COMPUTE_R (CR)***: Applica il filtro, normalizza, corregge lo shift, satura e scrive in memoria il risultato.
- ***GET_NEXT (GN)***: Legge dalla memoria all'indirizzo *w_addr*.
- ***SHIFT_WINDOW (SW)***: Effettua lo slittamento della finestra ed inserisce come ultimo elemento i_mem_data oppure 0.
- ***DONE***: Pone o_done = 1 e attende che i_start = 0 per tornare allo stato iniziale.

Segnali utilizzati

- ***current_addr***: Indirizzo corrente usato per lettura/scrittura in memoria a seconda dello stato.
- ***w_addr***: Indirizzo usato per la lettura dei valori da elaborare.
- ***r_start_addr***: Indirizzo al quale comincia a scrivere i risultati in memoria, ovvero *ADD + 17 + K*.
- *r_end_addr*: Primo indirizzo successivo all'ultimo risultato che verrà scritto in memoria, ovvero *ADD + 17 + 2K*.
- ***w_index***: Indice usato per il caricamento iniziale in finestra. Partendo dal centro della finestra con indice 3, vengono caricati 4 valori, quindi: *0 <= w_index <= MAX_W_INDEX = 3*.
- ***c_index***: Indice usato per il caricamento dei coefficienti in un array di lunghezza *NUM_COEFFS = 7*. Il caricamento si ferma quando viene caricato l'ultimo coefficiente e quindi *c_index >= NUM_COEFFS - 1*.

2.1 Process: reg_update

Il componente separa la logica in tre processi distinti:

reg_update: process(i_clk, i_rst)

reg_update è un processo sincrono che effettua il reset quando i_rst = 1 e ad ogni nuovo ciclo, in base allo stato, aggiorna il contenuto dei registri.

2.2 Process: state_transition

state_transition: process(current_state, i_rst, i_start, c_index, w_addr, r_start_addr, r_end_addr, current_addr, w_index)

state_transition è un processo combinatorio che calcola lo stato successivo in base alle informazioni presenti nei registri e allo stato corrente.

2.3 Process: output_logic

output_logic: process(current_state, i_add, current_addr, i_mem_data, coeffs, window, s, w_addr, r_start_addr)

output_logic è un processo combinatorio che imposta le uscite del componente per la lettura/scrittura e applica il filtro nello stato *COMPUTE_R*.

Esempio di applicazione del filtro di ordine 3 all'interno di COMPUTE_R

coeffs

0	-1	8	0	-8	1	0

window

0	0	0	32	-24	-35	0

risultati parziali (per chiarezza)

0	0	0	0	192	-35	0

temp_res = 192 - 35 = 157

temp_res	**right_shift**	**t**
157	4	t1 = 9
157	6	t2 = 2
157	8	t3 = 0
157	10	t4 = 0
temp_res > 0	**temp_res (normalization)**	11
-128 <= temp_res < =127	**temp_res (saturation)**	11

temp_res (da scrivere in memoria) = 11

2.4 Finestra a scorrimento

La soluzione della finestra a scorrimento ovvia al problema di salvare tutti i valori *W* insieme. Provando a salvare tutti i valori insieme, considerando il tipo di memoria utilizzato, non si risparmia né tempo né spazio.
Infatti, il tempo è il medesimo per entrambe le soluzioni: k cicli di clock. Inoltre, se si effettua il salvataggio di tutti i valori insieme, si ha bisogno di 2^{16} byte, poiché la lunghezza *K* della sequenza è un numero a 16 bit.

La soluzione della finestra a scorrimento utilizza un array di 7 byte, poiché l'applicazione del filtro richiede al massimo 7 valori: il valore per i=0, i 3 precedenti ed i 3 successivi.

18	**32**	**-3**	**-12**	**89**	**-1**	52

Filtro di ordine 3: valori utilizzati

18	**32**	**-3**	**-12**	**89**	**-1**	**52**

Filtro di ordine 5: valori utilizzati

Inizialmente vengono salvati i primi 4 valori, successivamente la finestra viene slittata verso sinistra e viene inserito un nuovo valore dopo ogni applicazione del filtro.

Inizializzazione della finestra

t = t0

0	0	0	0	0	0	0

t = t0 + 1

0	0	0	**18**	0	0	0

t = t0 + 4

0	0	0	18	32	-3	**-12**

Scorrimento della finestra

t = t1

0	0	0	18	32	-3	-12

t = t1 + 1

0	0	18	32	-3	-12	**89**

Valori fuori dalla sequenza

Il filtro viene applicato ad ogni valore della sequenza e, di conseguenza, vengono richiesti valori precedenti al primo e successivi all'ultimo, per la natura del filtro.
In particolare, se si applica il filtro di ordine 5, vengono richiesti tutti i valori presenti tra gli indirizzi *ADD + 17 - 3* e *ADD + 17 + K + 2*.
Da specifica, questi vanno interpretati come 0.

All'inizializzazione della finestra vengono quindi caricati soltanto i primi 4 valori, in modo da lasciare nei primi 3 posti il valore 0. Quando si finiscono i valori all'interno della sequenza, ovvero quando *w_addr >= r_start_addr*, si smettono di leggere valori dalla memoria e si passa dallo stato *COMPUTE_R* direttamente a *SHIFT_WINDOW*, che utilizzerà come nuovo valore 0.

3 Risultati sperimentali

3.1 Sintesi

Il risultato della sintesi del componente è il seguente:

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	943	0	134600	0.70
LUT as Logic	943	0	134600	0.70
LUT as Memory	0	0	46200	0.00
Slice Registers	210	0	269200	0.08
Register as Flip Flop	210	0	269200	0.08
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

Come evidenziato dalla tabella, non sono presenti latch non voluti, che potrebbero essere generati, ad esempio, dalla mancata inizializzazione di un segnale.

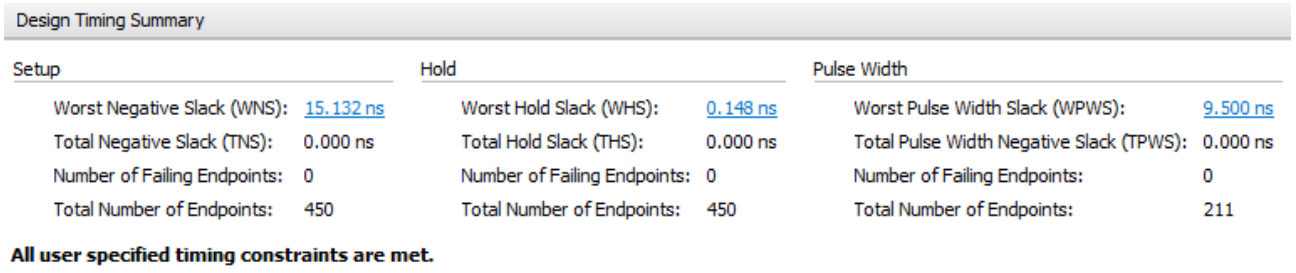

Figura 3: Report timing del componente

Il report del timing mostra un Worst Negative Slack di circa 15 ns. Il tempo di clock che il componente deve rispettare è di 20 ns e, di conseguenza, il componente potrebbe funzionare a cicli di clock più stringenti.

3.2 Simulazioni

Durante lo sviluppo del componente sono stati effettuati vari test bench per provare il suo corretto funzionamento di base e nei casi limite. I seguenti test bench vengono tutti superati sia in Behavioral Simulation che in Post-Synthesis Functional Simulation.

3.2.1 Funzionamento base

Il primo passo dello sviluppo è implementare il funzionamento di base del componente, testato dal seguente test bench. Il componente applica correttamente un filtro di ordine 3 (S = 0), interfacciandosi con la memoria e configurando correttamente le uscite.

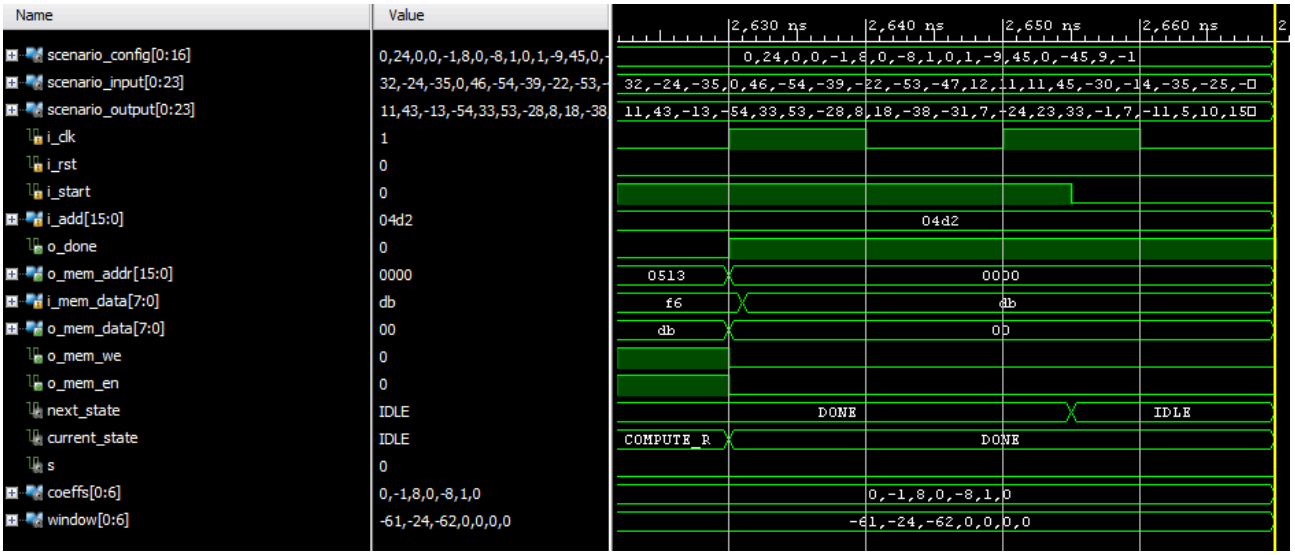

Figura 4: Simulazione test bench base

3.2.2 Filtro di ordine 5

Successivamente viene testato il filtro di ordine 5 (S = 1). Il componente gestisce il caso in cui viene selezionato il seguente filtro utilizzando i coefficienti giusti e calcolando il risultato correttamente.

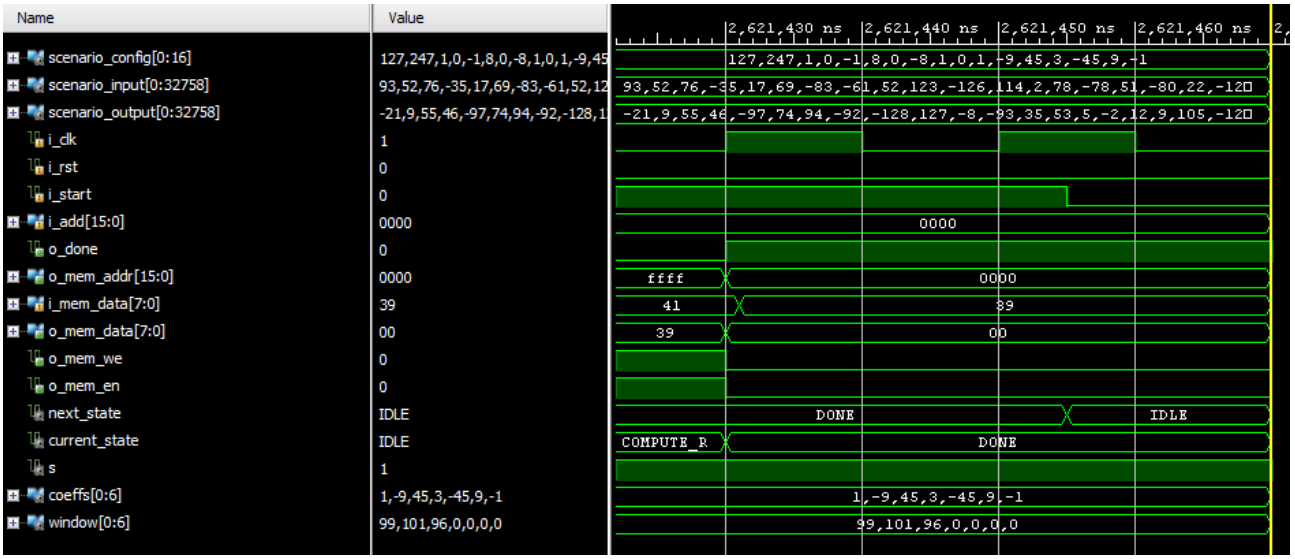

Figura 5: Simulazione test bench filtro di ordine 5

3.2.1 Massimo numero positivo e minimo numero negativo

Il calcolo del risultato necessita di un registro intermedio in cui salvare il risultato dell'applicazione del filtro.

Il massimo risultato positivo raggiungibile si ha nel caso in cui viene applicato un filtro di ordine 5 con tutti i coefficienti pari a -128 e tutti i valori in finestra uguali a -128.
Il risultato è quindi:

$(-128)^2 * 7 = 114688$

Il minimo risultato negativo raggiungibile si ha nel caso in cui viene applicato un filtro di ordine 5 con tutti i coefficienti pari a -128 e tutti i valori in finestra uguali a 127.
Il risultato è quindi:

$(-128) * (127) * 7 = -113792$

Per rappresentare correttamente questi risultati intermedi ho bisogno del seguente numero di bit:

$log_2(114688 + 113792) = 17.8 => 18\,bit$

Un numero a 18 bit codifica interi da -131072 a 131071 e quindi la variabile *temp_res* è a 18 bit. Questo test bench ha portato ad aumentare la capacità della variabile *temp_res*.

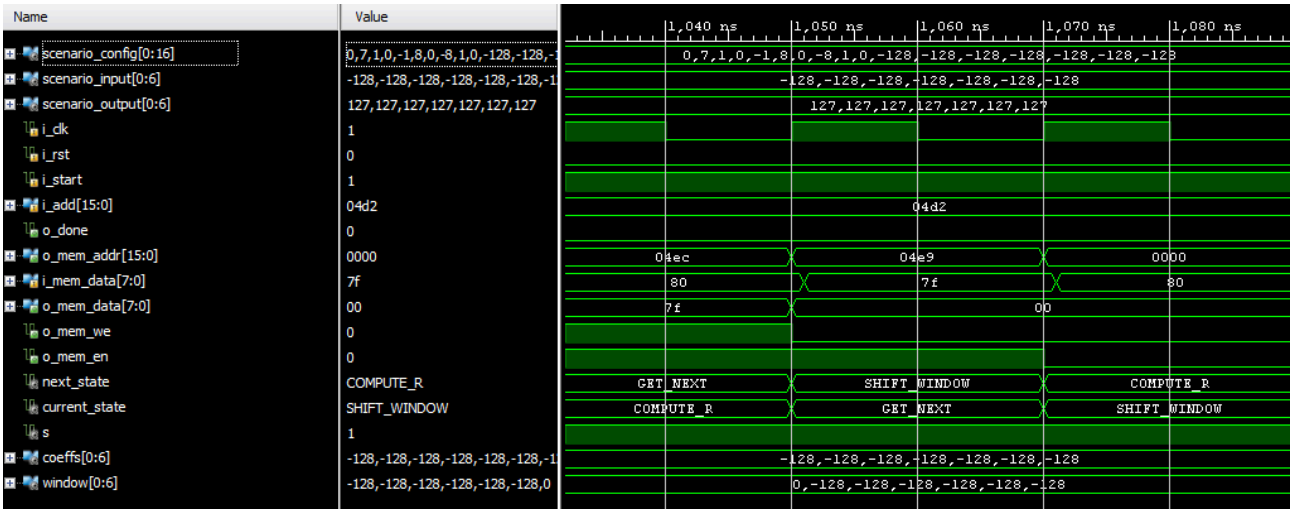

Figura 6: Simulazione test bench massimo numero positivo

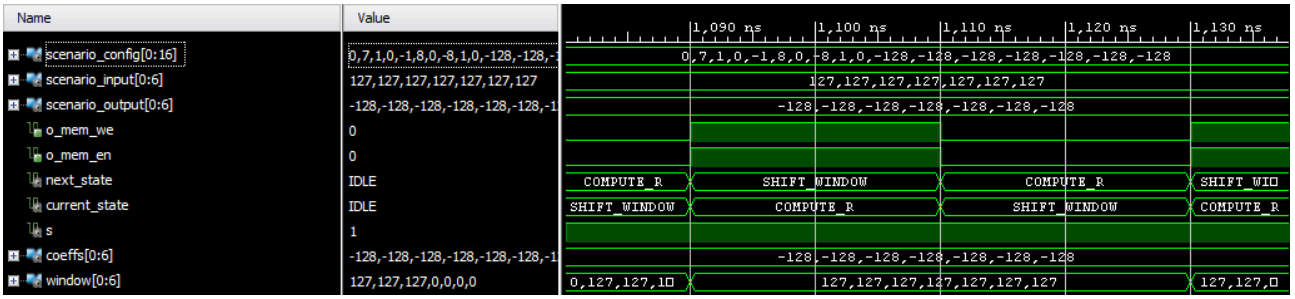

Figura 7: Simulazione test bench minimo numero negativo

3.2.3 Stress test

Il componente è stato, infine, sottoposto ad uno stress test. Questo test bench raggruppa più elaborazioni con filtri di ordine 3 e 5, vari coefficienti e reset intermedi.

Il test bench viene superato e, controllando il valore finale dei segnali interni, si può appurare il giusto funzionamento del componente, senza comportamenti inaspettati.

- ***current_addr = r_end_add***: come da definizione, l'ultimo valore viene scritto a *r_end_addr - 1* e al successivo ciclo di clock ci si trova nello stato *DONE*.
- ***c_index = 6***: il caricamento dei coefficienti si ferma all'ultimo indice disponibile.
- ***w_index = 3***: il caricamento iniziale dei valori nella finestra si ferma all'ultimo indice disponibile (3 + 3 = 6).

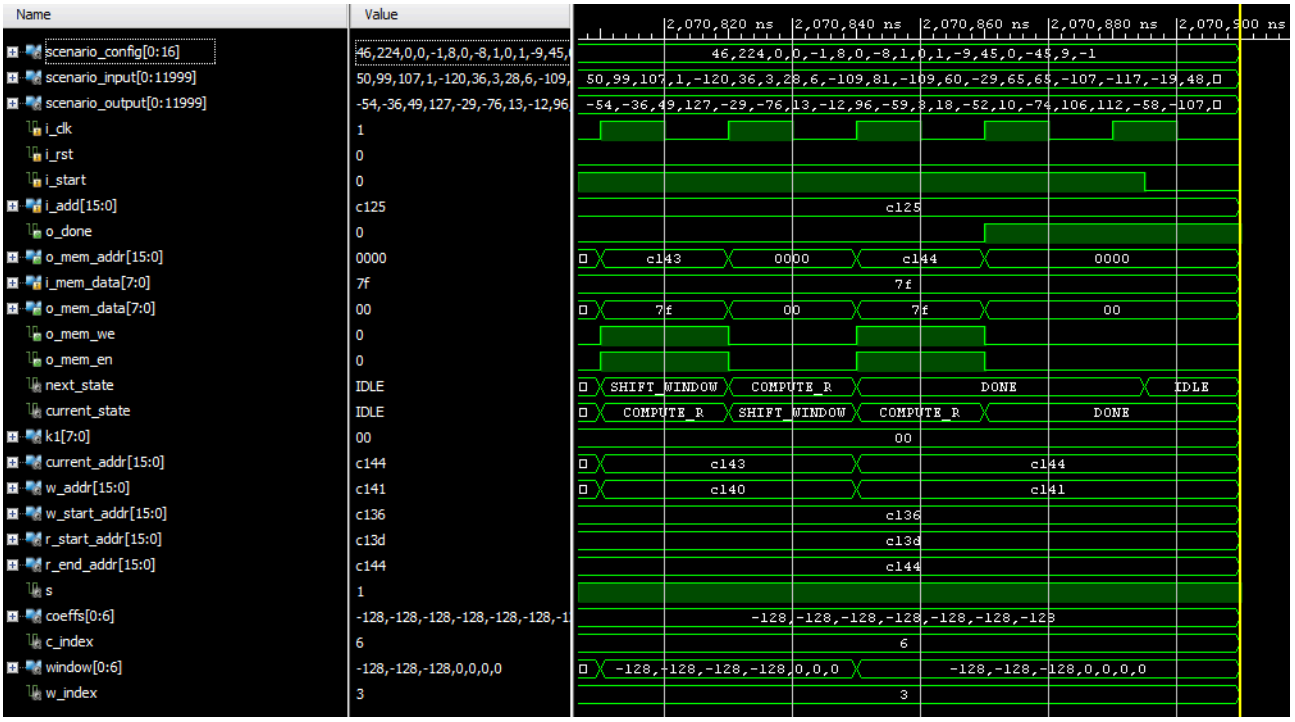

Figura 8: Simulazione test bench stress test

4 Conclusioni

L' architettura del componente è stata progettata rispettando totalmente le specifiche ed è stata verificata la correttezza dai vari casi di test effettuati. Infine l' architettura è stata realizzata utilizzando il minimo numero di stati e ottimizzando l'impiego delle risorse.