Prova Finale di Reti Logiche

Raffaella Corcione, 10924817, mat. 215144

Politecnico di Milano, a.a. 2024/2025 - Prof. William Fornaciari

1 Introduzione

La Prova Finale di Reti Logiche richiede di implementare un modulo hardware, descritto in linguaggio VHDL, che applichi un filtro differenziale ad una sequenza di byte letta da un modulo di memoria, scrivendo il risultato dell'elaborazione nella stessa memoria. In particolare, il sistema legge, un byte alla volta, una sequenza di K parole W (rappresentate in complemento a 2 e di valore compreso tra -128 e 127), applica il filtro a ciascun byte e scrive in memoria la sequenza dei K risultati R. La funzione che definisce il filtro è la seguente:

$$R_i = f(W_i) = \frac{1}{n} * \sum_{-l}^{l} C_j * W_{j+i}$$

dove C_j rappresentano i coefficienti del filtro, l vale 2 nel caso di filtro di ordine 3 e 3 nel caso di filtro di ordine 5 e n è il coefficiente di normalizzazione, di valore 12 per il filtro di ordine 3 e 60 per il filtro di ordine 5. I dati presenti in memoria hanno la seguente struttura. Essi si trovano situati in 17+K byte consecutivi a partire da un indirizzo ADD fornito in input al sistema e sono salvati nel seguente ordine: i primi due byte rappresentano K1 e K2, che indicano la lunghezza K della sequenza di dati; all'indirizzo ADD+2 si trova S, un byte il cui bit meno significativo indica l'ordine del filtro da utilizzare (se vale 0 va applicato il filtro di ordine 3, se vale 1 quello di ordine 5); successivamente sono memorizzati i 14 coefficienti per i due ordini del filtro; infine si trova la sequenza di K byte che costituiscono i valori su cui bisogna applicare il filtro. I K valori R del risultato devono essere scritti in memoria a partire dall'indirizzo successivo a quello che contiene l'ultimo valore della sequenza W. La struttura della memoria alla fine dell'elaborazione sarà quindi quella indicata in Figura 1.

Indirizzo	Valore
ADD	K1
ADD+1	K2
ADD+2	S
ADD+3	C1
...	...
ADD+16	C14
ADD+17	W1
...	...
ADD+16+K	Wk
ADD+17+K	R1
...	...
ADD+16+2K	Rk

Figura 1: Schema della memoria.

1.1 Interfaccia del componente

Il modulo implementato ha la seguente interfaccia, espressa in 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;
```

Tutti i segnali sono sincroni ed interpretati sul fronte di salita del clock, ad eccezione del reset che invece è asincrono. Il sistema segue il seguente protocollo: All'istante iniziale di reset, l'uscita `o_done` vale 0. Il modulo inizia l'elaborazione quando rileva che l'ingresso `i_start` è stato portato a 1. Si assume che finché il segnale `o_done` non sarà portato a 1 per indicare la fine dell'elaborazione, `i_start` rimarrà alto e `i_add` manterrà il suo valore valido di indirizzo iniziale. Il segnale `o_done` rimane alto finché `i_start` non viene portato a 0 e `i_start` non può essere alzato finché `o_done` non è stato riportato a 0. Si assume inoltre che prima del primo start verrà sempre dato un reset, ma per le elaborazioni successive alla prima non sarà necessario portare `i_rst` a 1. Ogniqualvolta viene rilevato `i_rst` pari a 1, il modulo viene re-inizializzato.

2 Architettura

Il componente è diviso in diversi moduli, raffigurati in Figura 2, relativi alle varie fasi della memorizzazione dei valori e del calcolo del filtro.

Il sistema è suddiviso in varie parti, ciascuna preposta ad una funzione specifica dell'elaborazione.

2.1 Registri

Ai fini della memorizzazione dei dati, sono presenti quattro diversi registri:

1. Registro per la memorizzazione di K (K REG): registro da 16 bit preposto alla memorizzazione del valore K, a caricamento parallelo 8 bit alla volta e lettura parallela. Possiede un ingresso per i dati, i segnali di clock e reset ed infine due segnali di enable che indicano l'abilitazione del registro e la selezione del gruppo da 8 bit in cui memorizzare il segnale in ingresso. Questo è necessario poiché K è un valore da 16 bit, ma il segnale in arrivo dalla memoria ha dimensione 8 bit.

2. Registro per la memorizzazione di S0 (S0 REG): registro da 1 bit per la memorizzazione del valore del bit meno significativo del byte S. Possiede gli ingressi di clock, reset ed enable.

3. Gruppo da 7 registri da 8 bit per la memorizzazione dei coefficienti C: registro a scorrimento con lettura parallela di 7 byte. Sincrono al clock, ma con reset asincrono. Possiede un segnale di enable e l'ingresso proviene dalla memoria.

4. Gruppo da 7 registri da 8 bit per la memorizzazione dei dati W: registro a scorrimento con lettura parallela di 7 byte. Sincrono al clock, ma con reset asincrono. L'ingresso è collegato all'uscita di

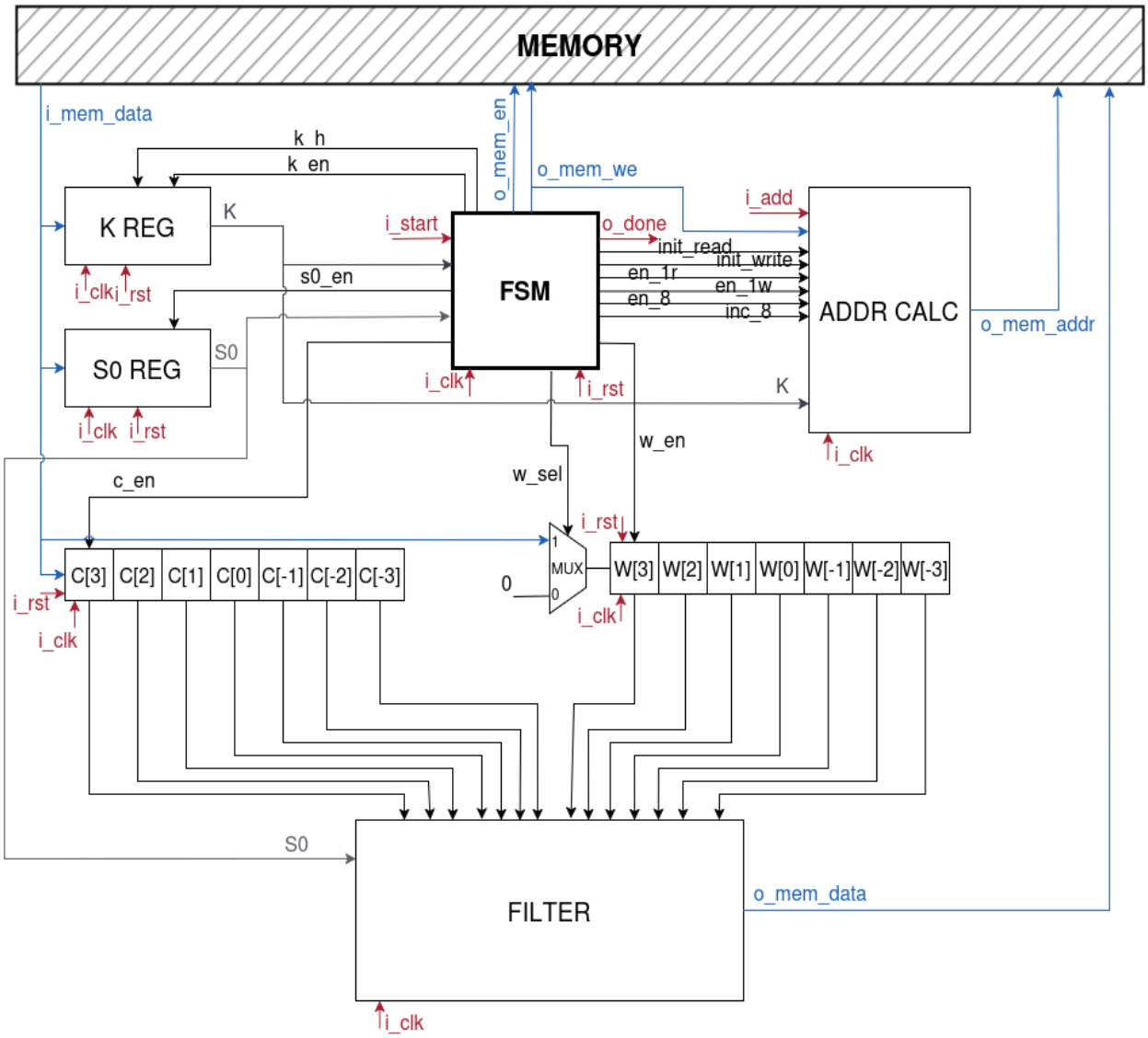


Figura 2: Diagramma generale del modulo.

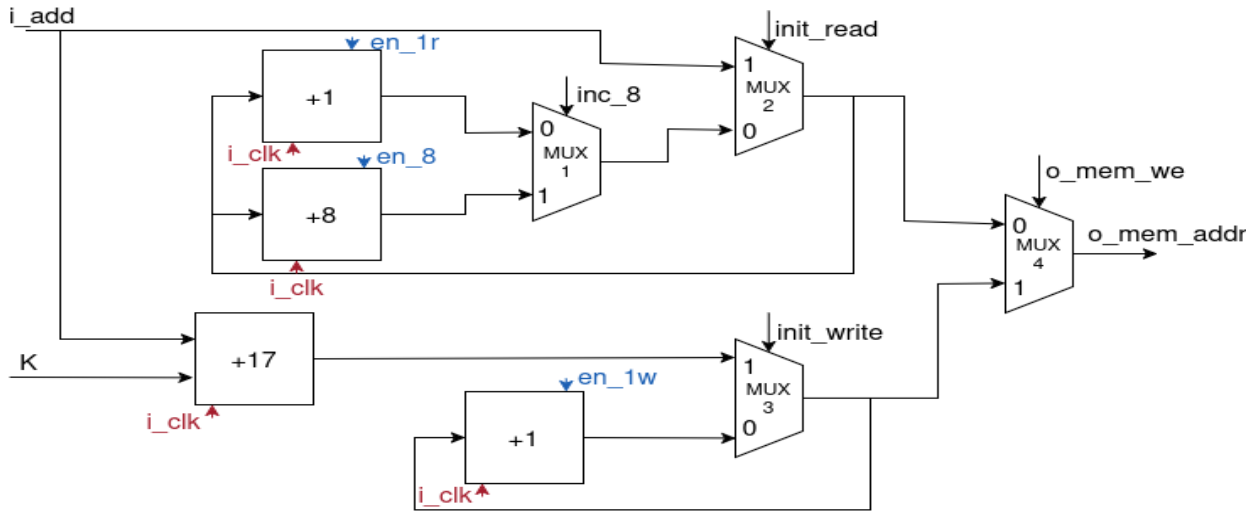


Figura 3: Modulo di calcolo indirizzi.

un multiplexer controllato dal segnale `w_sel` che indica se nel registro deve essere inserito il valore proveniente dalla memoria oppure il valore 0 [1].

2.2 Calcolo degli indirizzi

Per il calcolo degli indirizzi da porre sull'uscita `o_mem_addr`, il sistema utilizza un modulo apposito, controllato da alcuni segnali impostati dalla macchina a stati. Il modulo ha la struttura riportata in Figura 3.

Questo componente è formato da due parti distinte: la prima, composta dai due sommatori +1 e +8 e dai mux 1 e 2, si occupa di calcolare e fornire gli indirizzi per la lettura, che vanno da ADD a ADD+16. La seconda, formata dai due sommatori +1 e +17 e dal mux 3, calcola e fornisce gli indirizzi per la scrittura in memoria. Tutti i sommatori sono sincroni al clock e sono controllati da appositi segnali di enable e select provenienti dalla macchina a stati. In particolare, i due segnali `init_read` e `init_write` vengono posti a 1 quando il modulo viene inizializzato. In tale momento, l'indirizzo di lettura vale ADD, mentre quello di scrittura vale ADD+17. Tutti gli ingressi, i segnali interni e l'uscita di questo modulo hanno dimensione 16 bit [2].

2.3 Calcolo del filtro

Il modulo che effettua il calcolo del filtro ha la seguente struttura (riportata in Figura 4): riceve in ingresso i sette coefficienti C_j e sette valori W_j rappresentati su 8 bit, esegue sette moltiplicazioni $Cj * Wj$ con risultato a 16 bit, somma i sette valori ottenuti su 20 bit per evitare overflow, esegue la divisione approssimata per 12 o per 60 in base all'ordine del filtro da applicare e infine riporta il risultato ad un valore compreso tra -128 e 127 su 8 bit. L'uscita del componente viene collegata al segnale `o_mem_data`. I due componenti `NORM 12` e `NORM 60` sono a loro volta composti da moduli di base che eseguono i singoli shift e la correzione dell'errore. Poichè sono composti da una fase di shift seguita da una fase di somma, richiedono due cicli di clock per eseguire il calcolo. I multiplexer che effettuano la correzione dell'errore hanno come ingresso di selezione il bit più significativo del byte in ingresso (che determina il segno del numero da normalizzare, espresso in complemento a 2). I componenti che effettuano la moltiplicazione sono forniti di un segnale di enable che è posto sempre a 1 per i moduli dal secondo al sesto, mentre viene collegato a `s0` per il primo e l'ultimo, poiché il filtro di ordine 3 utilizza soltanto cinque valori.

[1] In accordo con la specifica del progetto, per il calcolo del filtro sui primi e ultimi tre valori della sequenza, mancando i dati agli estremi, il filtro utilizza degli zeri.

[2] Si assume che le somme non possano provocare overflow, in quanto la specifica del progetto precisa che tutti i testbench utilizzano valori validi per ADD e K in modo da rimanere nei limiti delle dimensioni della memoria.

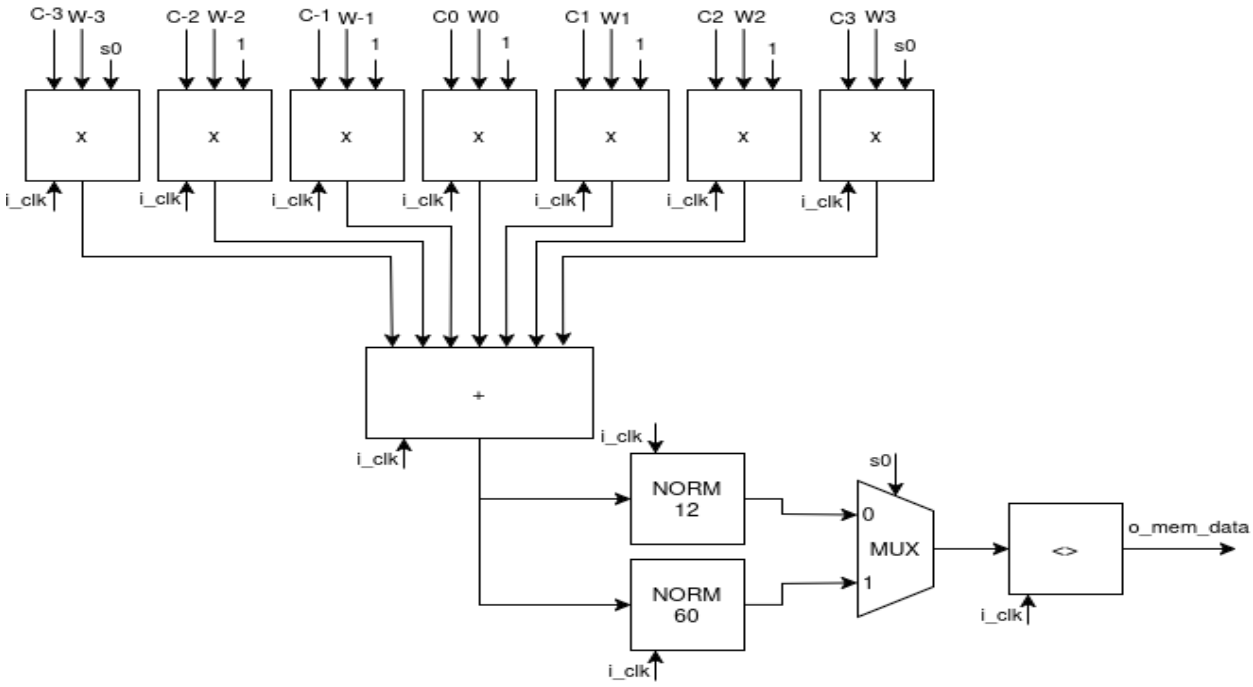
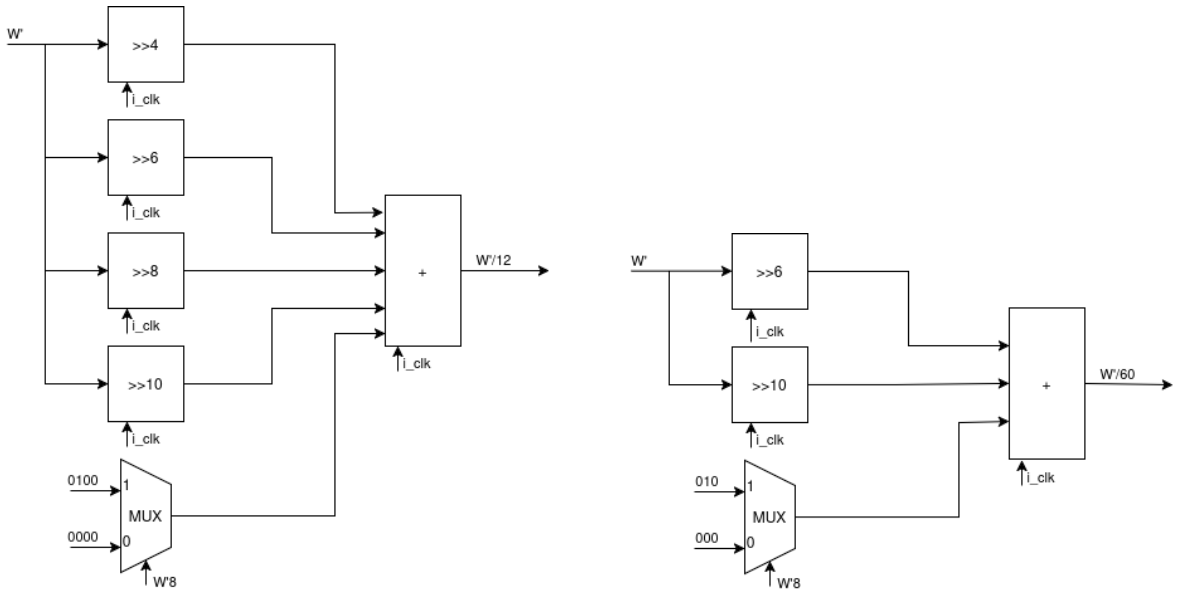


Figura 4: Modulo di calcolo filtro.



(a) n=12

(b) n=60

Figura 5: Elementi di normalizzazione.

2.4 Macchina a Stati Finiti

La macchina a stati finiti che costituisce il controllore centralizzato del modulo è una macchina di Mealy composta da 24 stati, 5 ingressi, 15 uscite e due segnali interni facenti funzione di contatori. Le tabelle degli stati e delle uscite sono rappresentate in Figura 6 e Figura 7 [3]. Gli ingressi della fsm sono i segnali `start`, `i_clk`, `rst`, `S0`, `K`. Le uscite sono invece le seguenti: `o_done` indica la fine dell'elaborazione; `mem_en` abilita l'accesso alla memoria; `mem_we` abilita la scrittura in memoria; `k_en` abilita il registro in cui è salvato il valore K; `k_h` seleziona quale metà del registro da 16 bit viene abilitata alla scrittura in fase di lettura e salvataggio di K1 e K2; `s0_en` abilita il registro in cui è salvato S0; `c_en` abilita il registro a scorrimento in cui sono salvati i coefficienti del filtro; `w_en` abilita il registro a scorrimento in cui viene salvata, sette valori per volta, la sequenza di dati; `w_sel` è il segnale che determina se nel registro W viene salvato il valore letto dalla memoria o uno zero; `init_r` viene utilizzato nel modulo di calcolo indirizzi per salvare il valore da cui iniziare a leggere la memoria; analogamente, `init_w` indica la selezione del valore iniziale dell'indirizzo per la scrittura; `en_1r` abilita l'incremento unitario dell'indirizzo di lettura; allo stesso modo, `en_1w` abilita l'incremento unitario dell'indirizzo di scrittura; `inc_8` è il segnale di controllo che indica se l'indirizzo di lettura va incrementato di 1 oppure di 8 (utilizzato nel calcolo dell'indirizzo di lettura dei coefficienti del filtro e nel successivo re-allineamento per la lettura dei dati, in quanto i due gruppi di sette coefficienti si trovano in zone adiacenti in memoria); `en_8` abilita l'incremento di 8 unità dell'indirizzo di lettura. Vengono inoltre utilizzati due contatori interni: `counter` viene utilizzato per controllare i loop su di un singolo stato, mentre `counter_k` gestisce la ripetizione per K volte del ciclo di lettura dato, calcolo filtro e scrittura in memoria. Nota sul segnale di start e sui contatori: sebbene essi determinino le transizioni di stato e siano pertanto presenti come ingressi nella tabella degli stati, non sono stati inseriti nella sensitivity list del processo *lambda* poiché (a differenza del reset) sono segnali che vanno interpretati in modo sincrono al fronte di salita del clock. Per stati diversi da `WAIT_START` e `FINAL`, poiché da specifica il segnale di start deve rimanere alto fino a che il segnale di done non è stato portato a 1, il comportamento della macchina per l'ingresso `i_start = 0` non è specificato.

2.4.1 Comportamento della FSM

Si riporta di seguito una descrizione del comportamento della macchina negli stati definiti per essa.

1. `WAIT_START`: si giunge in questo stato in seguito ad un reset (che viene sempre garantito prima della prima richiesta di elaborazione) oppure dopo che il segnale di start è stato riportato a 0 alla fine dell'elaborazione. Nessun segnale di ingresso o proveniente dalla memoria è valido finchè start non è portato a 1 e reset non è tornato a 0.

2. `INIT`: stato di inizializzazione della macchina. Vengono inizializzati i registri e i componenti addetti al calcolo degli indirizzi. In uscita su `o_mem_addr` è presente il valore ADD.

3. `S1`: viene abilitata la memoria in lettura portando `o_mem_en` a 1. Si richiede alla memoria il valore di K1. Viene abilitato l'incremento dell'indirizzo di lettura per poter richiedere al prossimo ciclo di clock il valore di K2.

4. `S2`: poiché sul fronte di salita del prossimo ciclo di clock sarà disponibile su `i_mem_data` il valore K1, viene abilitato il registro corrispondente portando `k_en` a 1. Viene richiesto alla memoria K2.

5. `S3`: K1 è pronto per essere salvato nel suo registro. Viene richiesto alla memoria S. Viene disabilitato l'incremento degli indirizzi di lettura, poiché i prossimi dati da leggere sono i coefficienti del filtro e per sapere quale gruppo deve essere memorizzato è necessario aver salvato il valore di S0. Poiché al ciclo di clock precedente è stato chiesto alla memoria K2, il suo registro viene abilitato ponendo `k_h` a 1 in modo che possa essere salvato.

6. `S4`: K1 è correttamente salvato nel suo registro. Viene abilitato il registro di S e disabilitato quello di K.

7. `S5`: K è correttamente memorizzato. S è disponibile su `i_mem_data`.

[3] La notazione X indica una condizione di don't care.

S	rst/start	counter	counter_k	S*
X	1X	X	X	WAIT_START
WAIT_START	00	X	X	WAIT_START
WAIT_START	01	X	X	INIT
INIT	01	X	X	S1
S1	01	X	X	S2
S2	01	X	X	S3
S3	01	X	X	S4
S4	01	X	X	S5
S5	01	X	X	S6
S6	01	X	X	S7
S7	01	X	X	S8
S8	01	DA 0 A 7	X	S8
S8	01	8	X	S9
S9	01	X	X	S10
S10	01	X	X	FP
FP	01	X	X	P1
P1	01	X	X	P2
P2	01	X	X	P3
P3	01	X	X	P4
P4	01	X	X	P4_bis
P4_bis	01	X	X	P5
P5	01	DA 0 A 3	X	P5
P5	01	4	X	P6
P6	01	X	X	P7
P7	01	X	DA 1 A K-4	P4
P7	01	X	DA K-3 A K-1	P8
P7	01	X	K	FINAL
P8	01	X	X	P9
P9	01	X	X	P5
FINAL	01	X	X	FINAL
FINAL	00	X	X	WAIT_START

Figura 6: Tabella degli stati della FSM (funzione δ).

8. `S6`: S0 è correttamente memorizzato nel suo registro e si può leggere sull'ingresso `s0` della fsm. In base al suo valore, vengono abilitati i componenti per il calcolo del prossimo indirizzo di lettura in memoria. Se `s0 = 0`, il filtro è di ordine 3 e i coefficienti si trovano a partire dall'indirizzo `ADD+3`. Se `s0 = 1`, il filtro è di ordine 5 e i coefficienti si trovano a partire dall'indirizzo `ADD+10`.

9. `S7`: il nuovo indirizzo di lettura è stato calcolato e viene posto in uscita su `o_mem_addr`.

10. `S8`: Si itera in questo stato per nove volte (tramite il segnale contatore `counter`) per leggere i coefficienti (indirizzi da `ADD+3` a `ADD+9` nel caso di filtro di ordine 3, da `ADD+10` a `ADD+16` nel caso di filtro di ordine 5). Alla prima iterazione viene chiesto alla memoria il valore del primo coefficiente e viene abilitato l'incremento dell'indirizzo di lettura. Nelle successive sette, viene abilitato il registro C ponendo `c_en` a 1. Alla settima iterazione (`counter` = 6), poiché sono stati richiesti alla memoria tutti i coefficienti, viene disabilitato l'incremento degli indirizzi di lettura (`en_1r`=0). All'ultima iterazione (`counter` = 9), viene disabilitato il registro C.

11. `S9`: sono disponibili alle uscite del loro registro tutti i coefficienti C. In base all'ordine del filtro viene calcolato l'indirizzo di lettura in memoria perché sia pari ad `ADD+17`, dove si trova il primo dato W1.

12. `S10`: su `o_mem_addr` si trova il valore `ADD+17`. La memoria è disabilitata.

S	s0	counter	done/ mem_en/mem_we	k_en/k_h/s0_en/ c_en/w_en/w_sel	init_r/init_w/en_1r/ en_1w/inc_8/en_8
WAIT_START	X	X	000	000000	000000
INIT	X	X	000	000000	110000
S1	X	X	010	000000	111000
S2	X	X	010	100000	011000
S3	X	X	010	110000	010000
S4	X	X	010	001000	010000
S5	X	X	010	000000	010000
S6	0	X	010	000000	011000
S6	1	X	010	000000	010011
S7	0	X	010	000000	010000
S7	1	X	010	000000	010010
S8	X	0	010	000000	011000
S8	X	DA 1 A 5	010	000100	011000
S8	X	DA 6 A 7	010	000100	010000
S8	X	8	010	000000	010000
S9	0	X	010	000000	010011
S9	1	X	010	000000	011000
S10	0	X	000	000000	000010
S10	1	X	000	000000	000000
FP	X	X	010	000010	001000
P1	X	X	010	000011	001000
P2	X	X	010	000011	001000
P3	X	X	010	000011	001000
P4	X	X	010	000011	000000
P4_bis	X	X	010	000001	000000
P5	X	X	010	000000	000000
P6	X	X	011	000000	000000
P7	X	X	010	000000	001100
P8	X	X	010	000010	000000
P9	X	X	010	000000	000000
FINAL	X	X	100	000000	000000

Figura 7: Tabella delle uscite della FSM (funzione λ).

13. `FP`: tutti i parametri del filtro sono stati letti e salvati correttamente. Viene richiesto alla memoria il valore di W1 abilitandola in lettura e viene abilitato il registro W.

14. `P1`: `w_sel = 1` per selezionare in ingresso al registro W il dato proveniente dalla memoria. Su `o_mem_addr` è presente l'indirizzo di W2. Per poter applicare il filtro su W1, è necessario aver salvato anche W2, W3 e W4.

15. `P2`: `w_sel = 1` per selezionare in ingresso al registro W il dato proveniente dalla memoria. Su `o_mem_addr` è presente l'indirizzo di W3.

16. `P3`: `w_sel = 1` per selezionare in ingresso al registro W il dato proveniente dalla memoria. Su `o_mem_addr` è presente l'indirizzo di W4.

17. `P4`: W4 è stato chiesto alla memoria, quindi viene disabilitato `en_1r` perché non vanno chiesti altri valori alla memoria finché non sarà stato calcolato e scritto R_1.

18. `P4_bis`: viene disabilitato il registro W. Al prossimo ciclo di clock tutti i valori necessari per il calcolo saranno correttamente salvati nei loro registri e saranno quindi disponibili agli ingressi del filtro.

19. `P5`: al fronte di salita del clock che porta in questo stato, i valori di W e C si trovano nell'ordine corretto all'ingresso dei moltiplicatori, s0 si trova all'ingresso del filtro e della fsm, e così K. In questo stato si svolge il calcolo del filtro relativo al valore Wi. La fsm deve aspettare per 5 cicli di clock (contati tramite il segnale `counter`). Tutti i registri e i segnali di controllo sono disabilitati, con l'eccezione di `o_mem_en`.

20. `P6`: la memoria deve essere abilitata in scrittura (`o_mem_we = 1`), poiché al prossimo fronte di salita sarà disponibile il risultato da scrivere. `counter` viene azzerato.

21. `P7`: il valore del risultato viene caricato in memoria. Vengono incrementati sia l'indirizzo di lettura sia quello di scrittura per chiedere alla memoria il valore del prossimo W e calcolare il risultato successivo. Viene riportato `o_mem_we` a 0. Al prossimo ciclo dovrà essere abilitato il registro W, quindi si ritorna allo stato P4. Il loop P4-P7 viene eseguito per K-3 volte, poiché per poter calcolare il filtro sugli ultimi tre valori della sequenza è necessario utilizzare degli zeri che non vanno letti dalla memoria. Per questo controllo, viene utilizzato il segnale interno `counter_k`. Quando questo segnale assume valore K, tutti i valori *Ri* sono stati calcolati e caricati in memoria e si può passare allo stato FINAL.

22. `P8`: in questo stato, l'interazione in lettura con la memoria è finita ed è necessario inserire uno zero nel registro W per poter calcolare gli ultimi valori di R.

23. `P9`: è stato caricato uno zero nel registro W ed è necessario aspettare che sia disponibile sull'uscita per poter tornare in P5 e iniziare il calcolo del filtro.

24. `FINAL`: l'elaborazione è finita. `o_done` viene portato a 1, tutti i registri, i componenti e i segnali per l'interazione con la memoria sono disabilitati e si attende che start torni a 0 per iniziare una nuova elaborazione.

3 Risultati sperimentali

Il modulo progettato è stato simulato e sintetizzato tramite il software Vivado. Il clock impostato per il sistema è pari a 20ns con duty-cycle pari al 50%.

3.1 Simulazioni

Il sistema è stato simulato tramite diversi testbench: innanzitutto sono stati eseguiti quelli forniti assieme alla specifica, sia per il filtro di ordine 3 sia per quello di ordine 5, che esprimevano esempi di normale funzionamento del componente. Successivamente, sono stati effettuati i seguenti test:

1. Verifica del comportamento del modulo con valore di K minimo tra quelli ammissibili: K=7, S=126, C=(74, -1, 8, 0, -8, 1, 24, 1, -9, 45, 0, -45, 9, -1), W=(1, 12, 15, -1, -4, -6, -3), R=(-6, -8, 7, 10, 1, 0, -2). Questo test verifica il corretto funzionamento della fsm nella gestione dei loop tra gli stati, in particolare il corretto inserimento degli zeri iniziali e finali. In aggiunta, verifica la corretta applicazione del filtro di ordine 3 anche nel caso in cui il primo e l'ultimo coefficiente siano diversi da 0;

2. Verifica del comportamento del modulo in caso di reset: dopo 597ns dal primo start, il segnale di reset viene portato a 1. Si attendono 50 ns prima di riportare start a 0, altri 15 ns per riportare reset a 0 e infine dopo 25 ns start viene riportato nuovamente a 1. Il componente esegue correttamente il reset, facendo ripartire l'elaborazione dallo stato iniziale e rileggendo nuovamente tutti i dati dalla memoria;

3. Verifica del comportamento del modulo in caso di seconda richiesta di start senza aver prima ricevuto un reset: alla rilevazione del segnale di done alla fine della prima elaborazione, si riporta start a 0 e poi nuovamente a 1 dopo 32 ns. Il componente riparte correttamente nell'elaborazione dall'inizio in modo sincrono al clock;

4. Verifica dell'assenza di overflow nei segnali interni al modulo di calcolo del filtro: utilizzando il filtro di ordine 5 e K=7, si esegue l'elaborazione con i seguenti valori: C=(-128, -128, -128, -128, -128, -128, -128), W=(-128, -128, -128, -128, -128, -128, -128), R=(127, 127, 127, 127, 127, 127, 127) e C=(-128, -128, -128, -128, -128, -128, -128), W=(127, 127, 127, 127, 127, 127, 127), R=(-128, -128, -128, -128, -128, -128, -128). I valori finali risultano correttamente saturati ai limiti dell'intervallo e i segnali interni al filtro riportano i valori corretti all'ingresso dei moduli di normalizzazione, dimostrando che i segnali preposti alla trasmissione dei risultati intermedi delle somme e moltiplicazioni sono stati correttamente dimensionati.

3.2 Sintesi

Tutte le simulazioni eseguite in pre-sintesi sono state eseguite anche in post-sintesi ottenendo i medesimi risultati. La sintesi è stata eseguita per la FPGA xc7a200tfbg484-1. Inoltre, la sintesi ha riscontrato le seguenti caratteristiche:
Slack time pari a 13.714ns: questo valore indica la differenza tra il periodo di clock e il ritardo introdotto dal circuito per il passaggio dei segnali. Un valore positivo indica che il componente rispetta la frequenza di clock impostata (pari a 50Mhz).
Tabella *Slice Logic*:

Site Type	Used	Fixed	Available	Util%
Slice LUTs	768	0	134600	0.57
LUT as Logic	768	0	134600	0.57
LUT as Memory	0	0	46200	0.00
Slice Registers	429	0	269200	0.16
Register as Flip Flop	429	0	269200	0.16
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

L'assenza di registri di tipo latch indica che il progetto non ha richiesto al tool di sintesi di introdurre elementi di memoria aggiuntivi per la memorizzazione dei segnali interni.

4 Considerazioni conclusive

Visti i risultati dei test effettuati e le informazioni ottenute dalla sintesi, visti i requisiti di progetto e le scelte di design compiute in conseguenza di essi, si può concludere che il modulo realizzato rispetta le specifiche fornite. Il componente esegue correttamente i calcoli e rispetta i protocolli di start/reset e di

comunicazione con la memoria. L'algoritmo implementato ha complessità temporale lineare rispetto a K e costante rispetto agli altri valori. Inoltre l'area occupata sulla FPGA di riferimento risulta molto ridotta, come si può dedurre dalla tabella Slice Logic osservando la percentuale di registri utilizzati rispetto a quelli disponibili. Applicando la regola di Paull-Unger, si può dimostrare che la fsm progettata è minima. Infine, la sintesi effettuata da Vivado utilizza la codifica one-hot per gli stati.