POLITECNICO
MILANO 1863

Prova Finale Di Reti Logiche
Anno Accademico 2025/2026

Brusa Paolo
Codice Persona: 10835369 - Matricola: 211863

1.Introduzione

Lo scopo del progetto consiste nel leggere da memoria diversi dati tra cui k1, k2, che concatenati fanno la lunghezza k la quale corrisponde a ai numeri totali della sequenza, s, il bit meno significativo di un numero che corrisponde al filtro da dover usare (ordine 3 oppure 5), 14 coefficienti, 7 che servono al filtro di

ordine 3 e 7 per quello di ordine 5, e, infine, W numeri da convertire con il filtro specifico salvando poi il nuovo dato nuovamente sulla memoria.

ESEMPIO

Si consideri la sequenza +101 +5 0 +41 +87 -123 -127 +87 +108. Utilizzando un filtro di ordine 3, il primo valore ottenuto è 0*-1 +0*8 +101*0 +5*-8 +0*1 cioè -40, che dopo viene normalizzato come : (-40 >> 4 +1)+(-40 >> 6+1)+(-40 >> 8+1) + (-40 >> 10+1).

Interfaccia Memoria

```
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;
```

Di seguito vengono descritti i segnali dell'interfaccia del componente da descrivere:

*Segnali di **INPUT:***

- *i_clk :* segnale di CLOCK fornito dal TestBench;

- *i_rst :* segnale di RESET fornito dal TestBench;
- *i_start :* segnale di START fornito dal TestBench che regola l'inizio di una nuova elaborazione;
- *i_add:* segnale (vettore) generato dal TestBench che indica l'indirizzo di memoria da cui parte la sequenza da elaborare;
- *i_mem_data:* segnale (vettore) proveniente dalla memoria che indica il valore contenuto in una determinata cella.

*Segnali di **OUTPUT:***

- *o_done :* segnale che comunica la fine dell'elaborazione dati e del processo dei dati;
- *o_mem_addr :* segnale (vettore) che comunica alla memoria l'indirizzo della cella sulla quale si vuole leggere/scrivere;
- *o_mem_data :* segnale (vettore) che comunica alla memoria il valore da scrivere in una determinata cella;
- *o_mem_en :* segnale che, se alto, permette l'operazione di lettura in memoria;
- *o_mem_we :* segnale che, se alto insieme ad *o_mem_en,* permette l'operazione di scrittura in memoria.

Il segnale di clock e' stato impostato a 20ns come da specifica.

Signal Utilizzati:

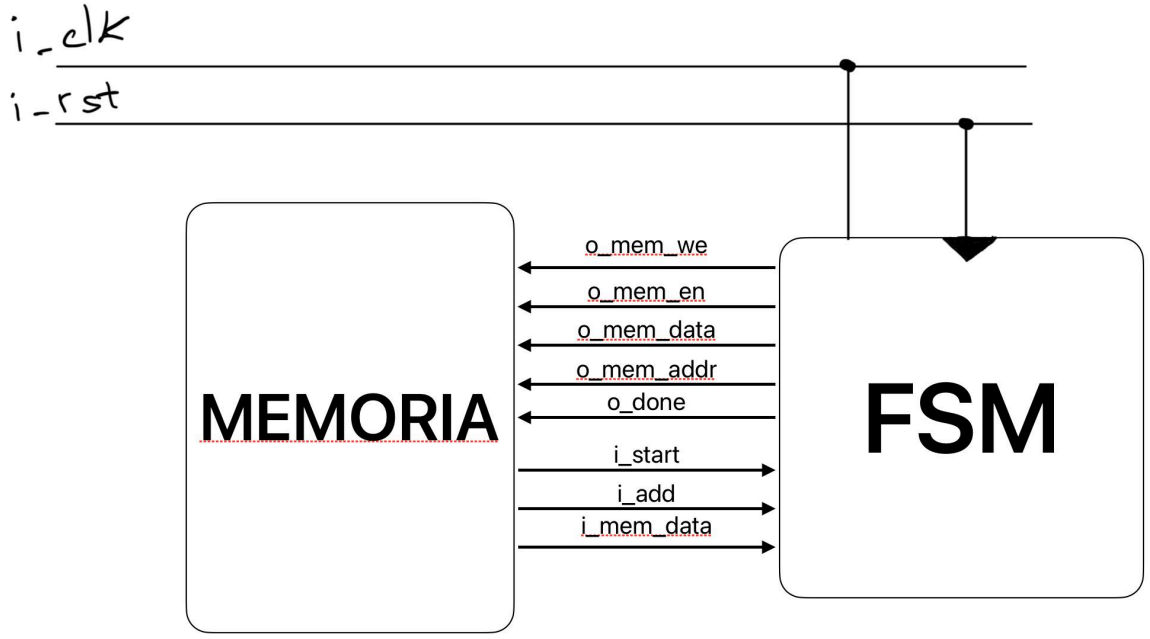
```
architecture Behavioral of project_reti_logiche is

    type state is (s0, s1, s1wait, s2, s3, s4, s5, s6, s7f3, s7f5, s8, s9, s9wait, s9read, s10);
    signal s: state;
    signal k1, k2: unsigned(7 downto 0);
    signal k: integer range 0 to 65535;
    signal f: std_logic;
    signal sum: signed (19 downto 0);
    signal last: integer range 0 to 7;
    type coeff_3 is array(0 to 6) of signed(7 downto 0);
    signal coeff3: coeff_3;
    type coeff_5 is array(0 to 6) of signed(7 downto 0);
    signal coeff5: coeff_5;
    type data is array(0 to 7) of signed(7 downto 0);
    signal W: data;

    signal count3: integer range 0 to 6;
    signal count5: integer range 0 to 6;
    signal kcount: integer range 0 to 65535;
    signal count: integer range 0 to 7;
```

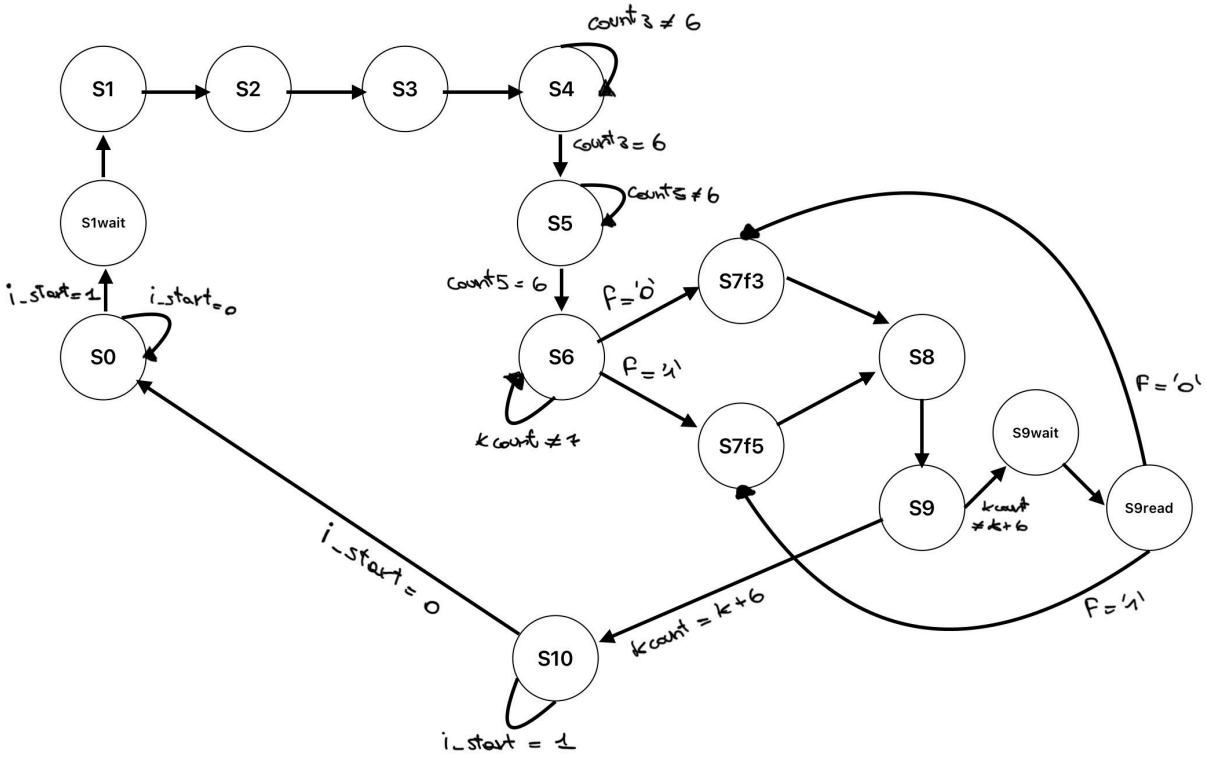
Come descritto in precedenza, abbiamo tutti i vari dati utili al fine di leggere, applicare il filtro e infine scrivere in memoria il risultato finale. Nei signal si possono notare un paio di variabili in più rispetto a quelle descritte e sono sum, last e f (che in realtà e' la s descritta dalla specifica). sum e' semplicemente una variabile intermedia che serve per calcolare la conversione effettuata con il filtro dato che e' diviso in più stati, mentre last serve per regolare l'ultimo calcolo da effettuare dato che ho adottato uno sliding array per caricare i dati, prima se no salvandoli tutti ero al di sopra di gran lunga dei limiti concessi dalla fpga non riuscendo a eseguire la sintesi.

2.Architettura



L’architettura proposta e’ formata da due componenti, la fsm e la memoria data. Come si vede la fsm interagisce con la memoria sia in lettura che in scrittura come richiesto dalla specifica. Legge i dati a partire dall’indirizzo dato in modo incrementale fino a raggiungere i W dati i quali vengono gestiti attraverso lo sliding array occupandosi prima quindi della scrittura del risultato e poi la lettura di un nuovo elemento.

MACCHINA A STATI



Non essendo un vero e proprio stato, quello di reset e' stato omesso dato che viene attivato dal segnale i_rst che passa il controllo durante il clock successivo a s0 il quale si occupa del vero e proprio reset dei vari signals.

STATO	FUNZIONE
S0	Inizializzazione dei signals e lettura primo indirizzo di memoria
S1Wait	Stato transitorio per lettura di k1, serve appunto per accedere al valore giusto della memoria
S1 fino S6	Lettura e scrittura nei signal corrispondenti del valore letto, in più preparano la memoria per la lettura successiva
S7f3 / S7f5	Due stati separati che corrispondono ai due filtri richiesti, ognuno di essi sovrascrive anche lo sliding array con un dato nella posizione corretta

S8	Stato che si occupa della normalizzazione
S9	Stato che si occupa della saturazione
S9Wait	Imposta la memoria sul nuovo dato in base al filtro
S9Read	E' stato aggiunto per evitare problemi di lettura come lo stato S1Wait iniziale
S10	Stato finale il quale rimanda a S0 sotto dovute condizioni

3.Risultati Sperimentali

Site Type	Used	Fixed	Prohibited	Available	Util%
Slice LUTs*	8521	0	0	134600	6.33
LUT as Logic	8521	0	0	134600	6.33
LUT as Memory	0	0	0	46200	0.00
Slice Registers	296	0	0	269200	0.11
Register as Flip Flop	296	0	0	269200	0.11
Register as Latch	0	0	0	269200	0.00
F7 Muxes	15	0	0	67300	0.02
F8 Muxes	0	0	0	33650	0.00

Timing Report

Slack (MET) : 1.437ns (required time - arrival time)

Dal report utilisation possiamo vedere come i latch risultanti siano 0, non creando così comportamenti non deterministici. Inoltre vengono usati 15 f7 muxes, essenziali per i calcoli complessi come i resize utilizzati durante il calcolo nel filtro per avere la massima precisione. Sono consapevole del fatto che le look up table e i flip flop siano in numero significativo, ma considerando comunque la fpga utilizzata, non incidono su di essa negativamente. Inoltre

passa con successo tutti i testbench utilizzati, quello fornito e quelli pubblicati dal professor Palermo, sia in behavioural sia in post synthesis simulation.

Lo slack come mostrato in figura e' preso (MET) e avremmo ancora a disposizione 1.437ns.