Prova Finale - Reti Logiche

Docente: Palermo Gianluca - Anno Accademico: 2023/2024

Pianalto Riccardo (Codice Persona: 10835980)
Pica Mirko (Codice Persona: 10811404)

Indice

1 Introduzione

La specifica della Prova Finale di Reti Logiche 2023/2024 riguarda la progettazione di un modulo hardware che interfacciandosi con una memoria valuti la credibilità dei valori presenti.
Un esempio pratico potrebbe essere un termometro che registra in memoria la temperatura di una stanza e in caso di dati mancanti il modulo li ricostruisce assegnandogli un adeguato valore di credibilità. Nello specifico, se una misurazione viene ritenuta valida per 5 secondi, allo scadere del lasso di tempo se non è stata memorizzata una nuova misurazione, il modulo decrementa la credibilità dell'ultima temperatura valida fino all'inserimento di un nuovo valore da parte del termometro.

1.1 Specifica generale

A livello generale, il sistema legge un messaggio costituito da una sequenza di K parole W il cui valore è compreso tra 0 e 255. La parola "0" all'interno della sequenza sta ad indicare che il valore non è specificato. La sequenza da elaborare è memorizzata a partire da un indirizzo specificato (ADD), ogni 2 byte (i.e. ADD, ADD+2, ADD+4, ..., ADD+2*(K-1)).
Il modulo da progettare ha il compito di completare la sequenza, sostituendo gli zero laddove presenti con l'ultimo valore letto diverso da zero, ed inserendo un valore di "credibilità" C, nel byte mancante, per ogni valore della sequenza. La sostituzione degli zero avviene copiando l'ultimo valore valido (non zero) letto precedente e appartenente alla sequenza. Il valore di credibilità C è pari a 31 ogni volta che il valore W della sequenza è non zero, mentre viene decrementato rispetto al valore precedente ogni volta che si incontra uno zero in W. Il valore C associato ad ogni parola W viene memorizzato in memoria nel byte subito successivo (i.e. ADD+1 per W in ADD, ADD+3 per W in ADD+2,...). Il valore C è sempre maggiore o uguale a 0 ed è reinizializzato a 31 ogni volta che si incontra un valore W diverso da zero. Quando C raggiunge il valore 0 non viene ulteriormente decrementato.

	Ingresso	Uscita
ADD	**51**	**51**
ADD+1	0	31
ADD+2	**0**	**51**
ADD+3	0	30
ADD+4	**57**	**57**
ADD+5	0	31
ADD+6	**24**	**24**
ADD+7	0	31
ADD+8	**0**	**24**
ADD+9	0	30
ADD+10	**0**	**24**
ADD+11	0	29
ADD+12	**126**	**126**
ADD+13	0	31
ADD+14	**0**	**126**
ADD+15	0	30
ADD+16	**192**	**192**
ADD+17	0	31
ADD+18	**0**	**192**
ADD+19	0	30

Figura 1: Valori in decimale (in grassetto i valori W)

1.2 Interfaccia del componente

Il componente descritto possiede la seguente interfaccia:

```
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);
        i_k      : in std_logic_vector(9 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 entity;
```

In particolare:

- i_clk è il segnale di CLOCK in ingresso generato dal Test Bench;
- i_rst è il segnale di RESET 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) che indica l'indirizzo di partenza;
- i_k è il segnale (vettore) che indica il numero di parole da leggere;
- o_done è il segnale di uscita che comunica la fine dell'elaborazione;
- o_mem_addr è il segnale (vettore) che indica l'indirizzo su cui sta lavorando la memoria;
- i_mem_data è il segnale (vettore) di ingresso dalla memoria al nostro componente;
- o_mem_data è il segnale (vettore) di uscita dal componente verso la memoria.
- o_mem_en è il segnale di ENABLE da dover mandare alla memoria per poter comunicare (sia in lettura che in scrittura);
- o_mem_we è il segnale di WRITE ENABLE da dover mandare alla memoria per poterci scrivere.

2 Architettura

Dopo che il segnale i_rst verrà portato ad 1 e poi abbassato, il modulo partirà nell'elaborazione quando il segnale i_start in ingresso verrà portato a 1. Il segnale i_start rimarrà alto fino a che il segnale o_done non verrà portato alto. Al termine dell'elaborazione, il modulo alza il segnale o_done che notifica la fine. Se a questo punto viene rialzato il segnale i_start, il modulo ripartirà con la fase di elaborazione.

2.1 Scelte progettuali

Essendo programmatori abbiamo deciso di dividere tutto in funzioni. Si è scelto, quindi, di costruire il modulo come una macchina a stati finiti (FSM) che gestisce altri componenti.
In particolare, il modulo leggerà dalla memoria la parola W che in caso sia diversa da zero abiliterà il registro che la memorizzerà e inizializzerà la credibilità (a 31). A questo punto un contatore a 10 bit si occuperà dello spostamento dell'indirizzo di memoria al byte successivo così da poter scrivere il valore di credibilità. Nel caso in cui, la parola sia zero non verrà abilitato il registro ma verrà tenuto il valore precedente e tramite un multiplexer verrà prima scritta la parola presente nel registro e poi nel byte successivo la credibilità opportunamente decrementata da un contatore a 5 bit.
Nel momento in cui il contatore avrà raggiunto l'indirizzo finale tramite un comparatore il modulo alzerà il segnale di fine elaborazione.

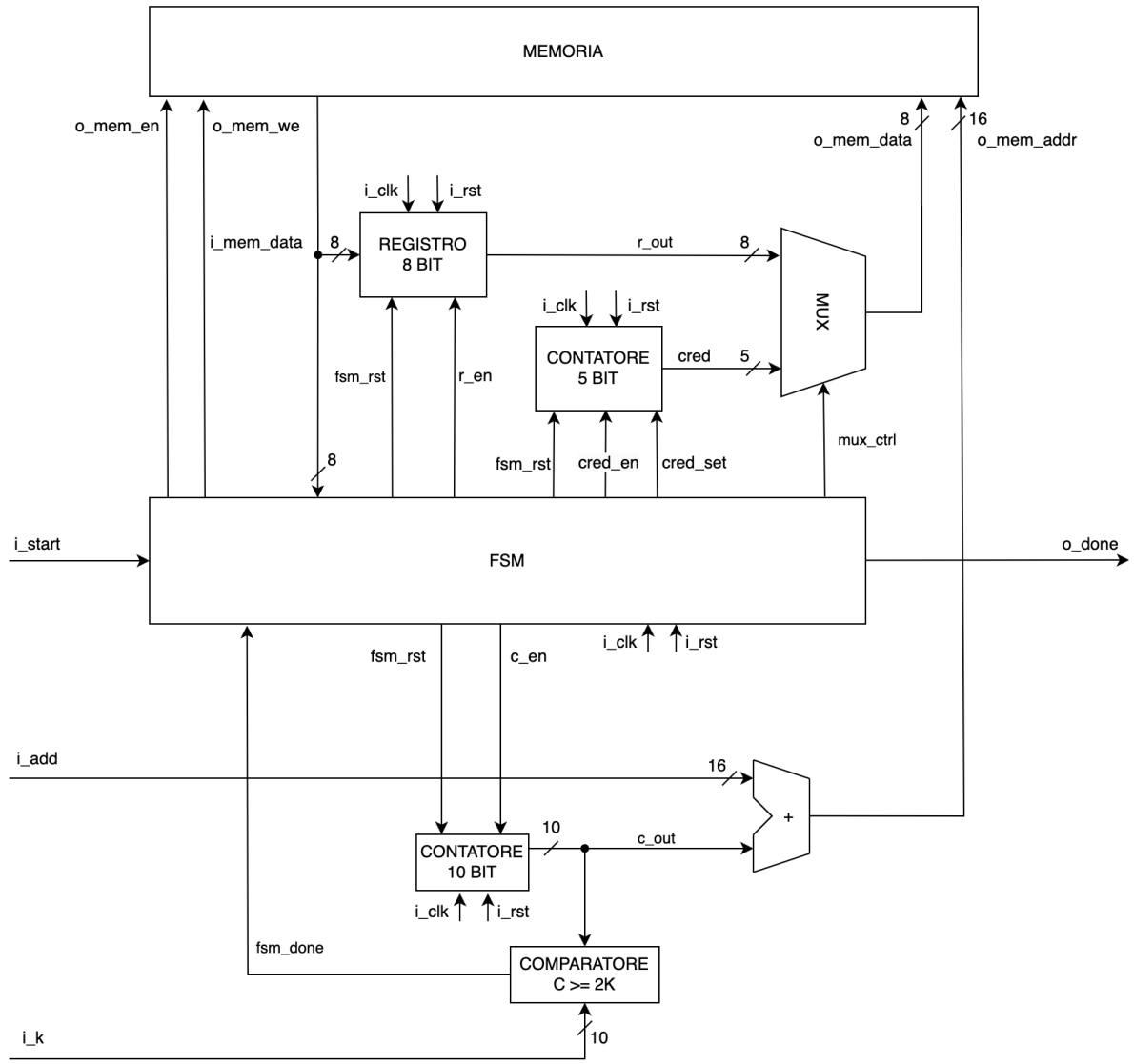

Figura 2: Rappresentazione strutturale del modulo

2.2 Componenti

2.2.1 FSM

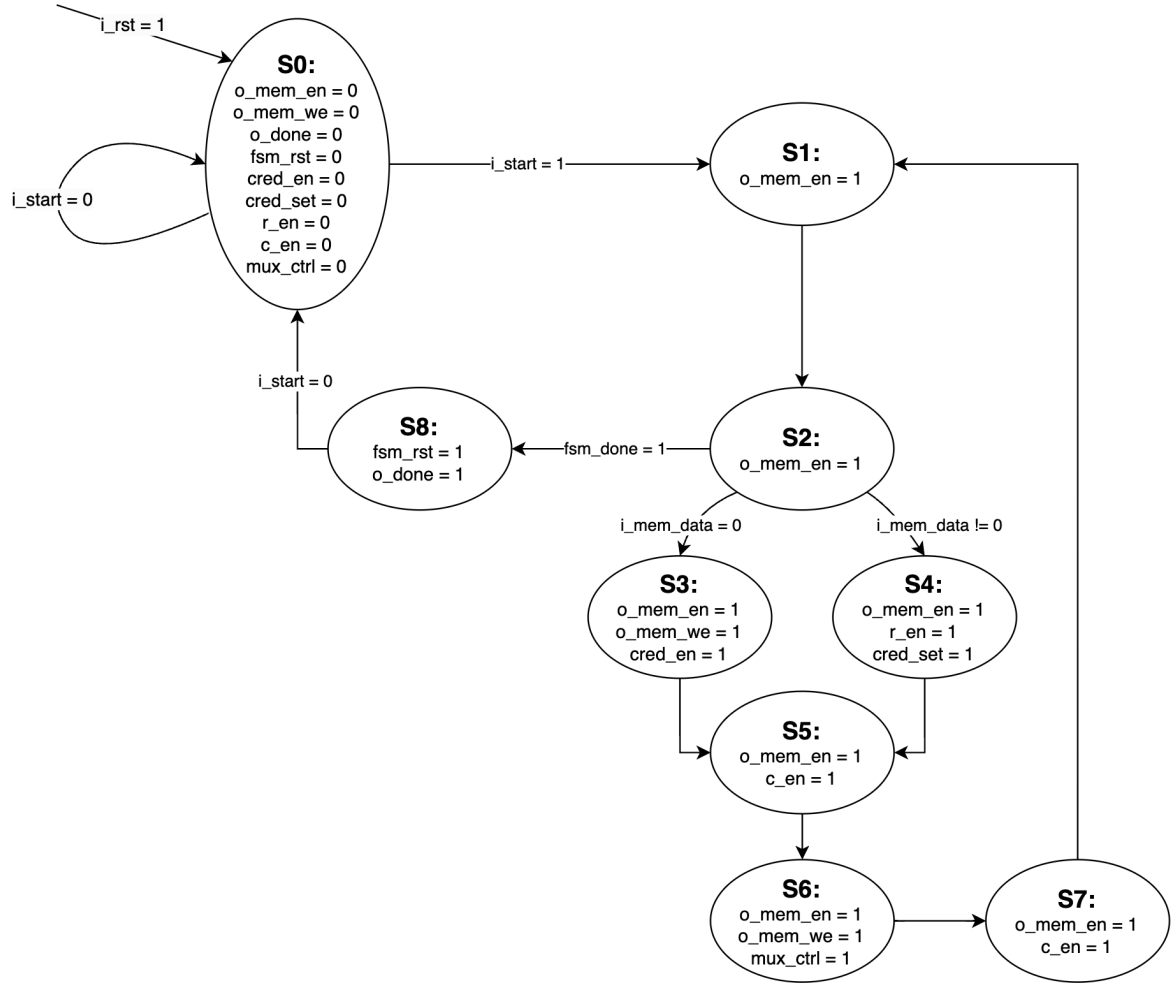

Figura 3: Diagramma degli stati

La FSM utilizzata è un esempio di macchina di Moore ed è quindi stata implementata con un process per il passaggio di stato ed uno per le uscite. Essa (Figura 3) è composta dai seguenti nove stati:

- `S0`:
  Stato iniziale di reset in cui si attende che venga alzato il segnale `i_start`. In caso di segnale `i_rst='1'` si torna in questo stato.

- `S1`:
  Stato in cui viene abilitata la lettura in memoria.

- `S2`:
  Stato in cui la FSM legge il dato e sceglie l'opportuna transizione:
  - i_mem_data = '0' va in S3;
  - i_mem_data != '0' va in S4;
  - fsm_done = '1' va in S8.

- `S3`:
  Stato in cui viene scritta la parola già presente nel registro in memoria e decrementata la credibilità.

- `S4`:
  Stato in cui viene memorizzata nel registro la nuova parola e inizializzata la credibilità.

- `S5`:
  Stato in cui ci si sposta al prossimo indirizzo in cui verrà salvata la credibilità.

- `S6`:
  Stato in cui viene scritta la credibilità

- `S7`:
  Stato in cui ci si sposta al prossimo indirizzo per leggere la prossima parola.

- `S8`:
  Stato in cui si inizializzano i componenti e viene alzato il segnale di o_done.

2.2.2 Contatore Credibilità

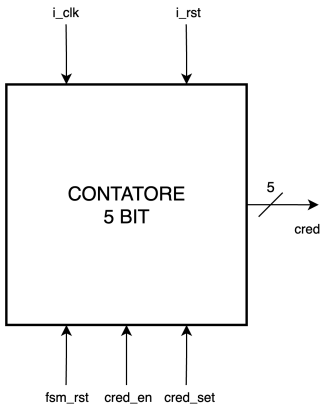

Figura 4: Rappresentazione della credibilità

Si occupa di tener conto della credibilità di una parola. A seguito del segnale di reset (sia esterno che dalla fsm) la credibilità è inizializzata a zero. Quando viene letta una parola valida viene ricevuto il segnale `cred_set` che imposta la credibilità a 31. In caso di valore non specificato viene ricevuto il segnale `cred_en` che permette di decrementare la credibilità.

2.2.3 Multiplexer

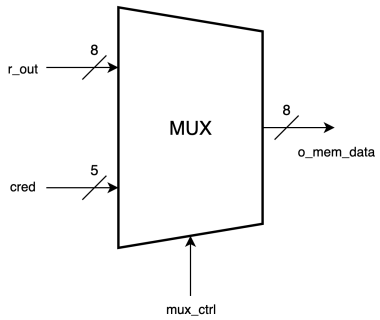

Figura 5: Rappresentazione del multiplexer

Si occupa di scegliere il dato da scrivere in memoria. Quando `mux_ctrl = '0'` manderà la parola presente nel registro, quando `mux_ctrl = '1'` manderà la credibilità estesa ad 8 bit.

2.2.4 Contatore Indirizzi

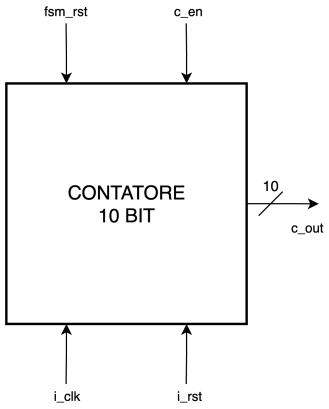

Figura 6: Rappresentazione del contatore degli indirizzi

Si occupa dell'avanzamento degli indirizzi di memoria. A seguito del segnale di reset (sia esterno che dalla fsm) il contatore è inizializzato a zero. Quando il segnale c_en è alto permette al contatore di incrementare.

2.2.5 Sommatore

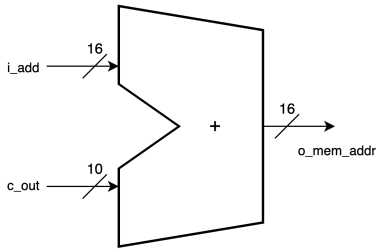

Figura 7: Rappresentazione del sommatore

Si occupa di trovare l'indirizzo di memoria su cui stiamo lavorando. Somma l'indirizzo di partenza e il contatore degli indirizzi.

2.2.6 Comparatore

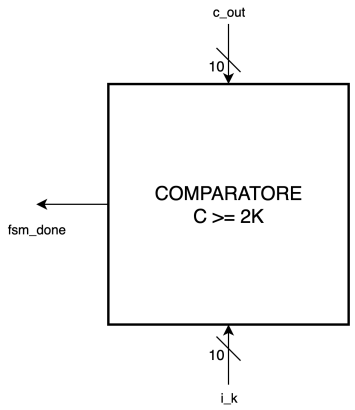

Figura 8: Rappresentazione del comparatore

Si occupa di notificare il termine della sequenza. Quando il contatore degli indirizzi raggiunge il valore 2K, dove K è il numero di parole della sequenza, il comparatore alza il segnale fsm_done.

2.2.7 Registro

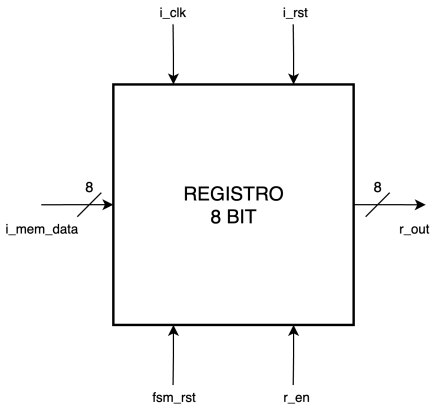

Figura 9: Rappresentazione del registro

Si occupa di salvare la parola valida che dovrà essere scritta in memoria in caso di valore non specificato. A seguito del segnale di reset (sia esterno che dalla fsm) il registro è inizializzato a zero. Quando il segnale di enable è alto il registro salva il valore di i_mem_data.

3 Risultati sperimentali

3.1 Simulazioni

Il funzionamento del componente è stato verificato tramite l'utilizzo di appositi test bench. Sono state eseguite due tipi di simulazioni: *Behavioural* e *Post-Synthesis Functional*.
Oltre al test bench di esempio sono stati creati altri test bench che cercassero di testare i vari scenari e casi limite di esecuzione del modulo. Le situazioni che sono state controllate sono le seguenti:

1. **Caso base**: sequenza con valori vari, tra cui quelli forniti nelle specifiche.
2. **Tutte parole valide**: sequenza in cui tutte le parole sono valide, quindi a tutte è associata la massima credibilità.
3. **Credibilità fino a 0**: sequenza in cui solo la prima parola è valida, quindi si controlla il corretto decremento della credibilità da 31 a 0.
4. **Prima parola 0**: sequenza in cui la prima parola è zero, quindi relativa credibilità zero.
5. **Tutti 0**: sequenza di tutti zero, in cui la credibilità è ovunque nulla.
6. **Reset durante un elaborazione**: il reset interrompe l'esecuzione della prima sequenza, a cui segue l'inizio di una successiva elaborazione.
7. **Doppio start senza reset**: elaborazione di due sequenze senza ricevere il reset.

3.2 Sintesi

Il componente è stato sintetizzato con successo. Riportiamo di seguito i risultati ottenuti dallo strumento di sviluppo Vivado.

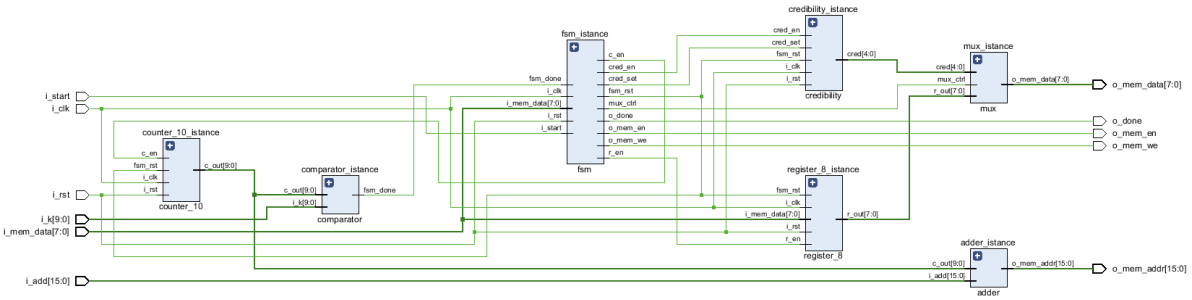

Figura 10: Design elaborato generato da Vivado

3.2.1 Area utilizzata sulla FPGA

Dal report di utilizzo otteniamo la seguente tabella dalla quale possiamo verificare l'assenza di latch indesiderati e la percentuale di utilizzo della FPGA utilizzata.

Site Type	Used	Fixed	Available	Util%
Slice LUTs*	47	0	134600	0.03
LUT as Logic	47	0	134600	0.03
LUT as Memory	0	0	46200	0.00
Slice Registers	32	0	269200	0.01
Register as Flip Flop	32	0	269200	0.01
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

Figura 11: Report di utilizzo

3.2.2 Frequenza di funzionamento

Dal report temporale possiamo verificare che il componente soddisfa i requisiti di tempo. Considerando il periodo di clock T_{clk} di $20ns$, indicato nelle specifiche, si ottiene un *Worst Negative Slack* WNS pari a $16.791ns$. Tramite un calcolo approssimativo è possibile ottenere la massima frequenza a cui può operare il componente.
Il minimo periodo utilizzabile risulta:

$$T_{min} = T_{clk} - WNS = 20ns - 16.791ns = 3.209ns \tag{1}$$

Quindi si ottiene la massima frequenza di funzionamento:

$$f_{max} = \frac{1}{T_{min}} = \frac{1}{3.209ns} \approx 311.62MHz. \tag{2}$$

4 Conclusioni

Il componente sviluppato elabora correttamente sequenze di parole assegnando l'opportuna credibilità ad ognuna.
L'utilizzo della FSM per la gestione dei vari componenti si è rivelata una scelta funzionale, semplificando notevolmente la comunicazione tra i singoli componenti e rendendo gli stessi più compatti e facili da testare.
Ci sembra di aver coperto tutti i casi di test rilevanti e quindi, visti i risultati delle simulazioni e dei report forniti da Vivado, ci sentiamo di aver implementato il modulo correttamente.