Prova finale (Progetto di Reti Logiche)

Prof. William Fornaciari - Anno 2019/2020

Armillotta Domenico (Codice persona: 10582521 Matricola: 888925)
Giovanni Amarù (Codice persona: 10605272 Matricola: 887963)

Indice

1. Introduzione

1.1. Specifiche generali

Quello da noi elaborato è un componente hardware per l'analisi dell'appartenenza di un dato indirizzo codificato come numero a 7 bit, prelevato dalla memoria all'indirizzo 8, a un insieme dato di 8 working zone, codificate come numeri a 7 bit negli indirizzi da 0 a 7 in memoria. L'indirizzo appartiene a una working zone se esso è uguale all'indirizzo della working zone o è maggiore di un offset pari al più a 3. Se si trova una corrispondenza, in output viene mandato un numero da 8 bit, codificato secondo la maniera 1 - XXX - YYYY, dove i bit XXX rappresentano l'indirizzo in memoria in cui è presente la working zone con cui è avvenuto il match, e i bit YYYY rappresentano l'offset in codifica one-hot. Se non si trova un match, l'indirizzo in ingresso viene mandato in output senza modifiche concatenando un bit posto a 0 nella posizione più significativa.

1.2. Interfaccia del componente

Il componente da noi elaborato ha un'interfaccia così definita:

```
entity project_reti_logiche is
   Port ( i_clk : in STD_LOGIC;
        i_start : in STD_LOGIC;
        i_rst : in STD_LOGIC;
        i_data : in STD_LOGIC_VECTOR (7 downto 0);
        o_address : out STD_LOGIC_VECTOR (15 downto 0);
        o_done : out STD_LOGIC;
        o_en : out STD_LOGIC;
        o_we : out STD_LOGIC;
        o_data : out STD_LOGIC_VECTOR (7 downto 0));
end project_reti_logiche;
```

Dove:

- i_clk è il segnale di clock in ingresso, generato dal testbench;
- i_start è il segnale di start generato dal testbench;
- i_rst è il segnale di reset, utilizzato per inizializzare la macchina prima di ricevere il primo segnale di start
- i_data è il segnale (vettore) che arriva dalla memoria in seguito a una richiesta di lettura;
- o_done è il segnale di uscita che comunica la fine dell'elaborazione e il dato di uscita scritto in memoria;

- o_en è il segnale di enable da dover mandare alla memoria per poter comunicare in lettura o scrittura;
- o_we è il segnale di write enable da mandare alla memoria per poter scrivere su essa. Se si vuole leggere da memoria deve essere posto a 0;
- o_data è il segnale di uscita dal componente verso la memoria.

1.3 Dati e descrizione memoria

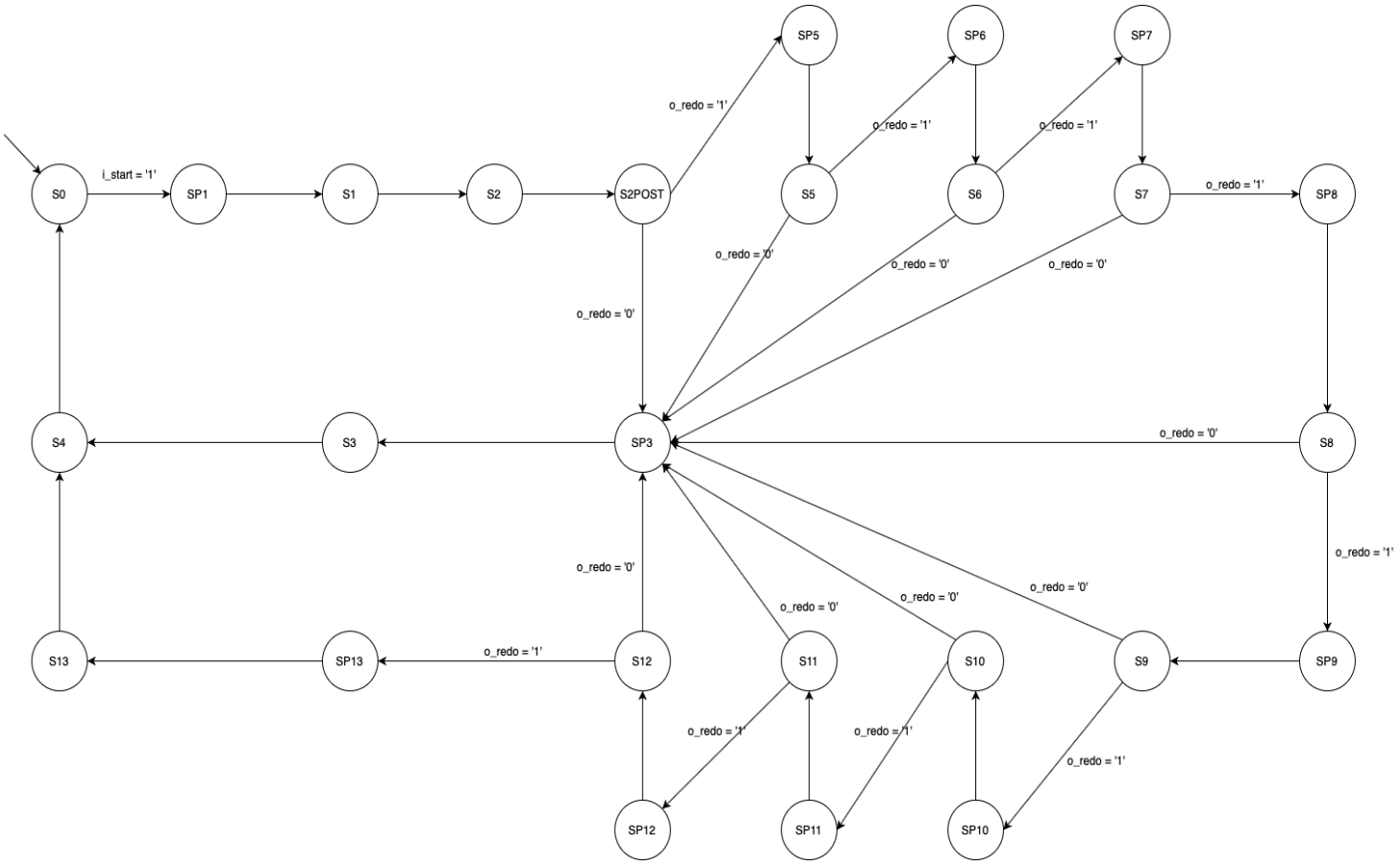
I dati, ciascuno di dimensione 8 bit, sono memorizzati in una memoria con indirizzamento al byte:

- L'indirizzo 8 è usato per memorizzare il dato di ingresso;
- Gli indirizzi dallo 0 al 7 sono usati per memorizzare gli indirizzi base delle working zone;
- L'indirizzo 9 è usato per scrivere il risultato computato in uscita.

Indirizzo working zone 0
Indirizzo working zone 1
Indirizzo working zone 2
Indirizzo working zone 3
Indirizzo working zone 4
Indirizzo working zone 5
Indirizzo working zone 6
Indirizzo working zone 7
Indirizzo di ingresso
Dato di uscita

2. Design

2.1. FSM



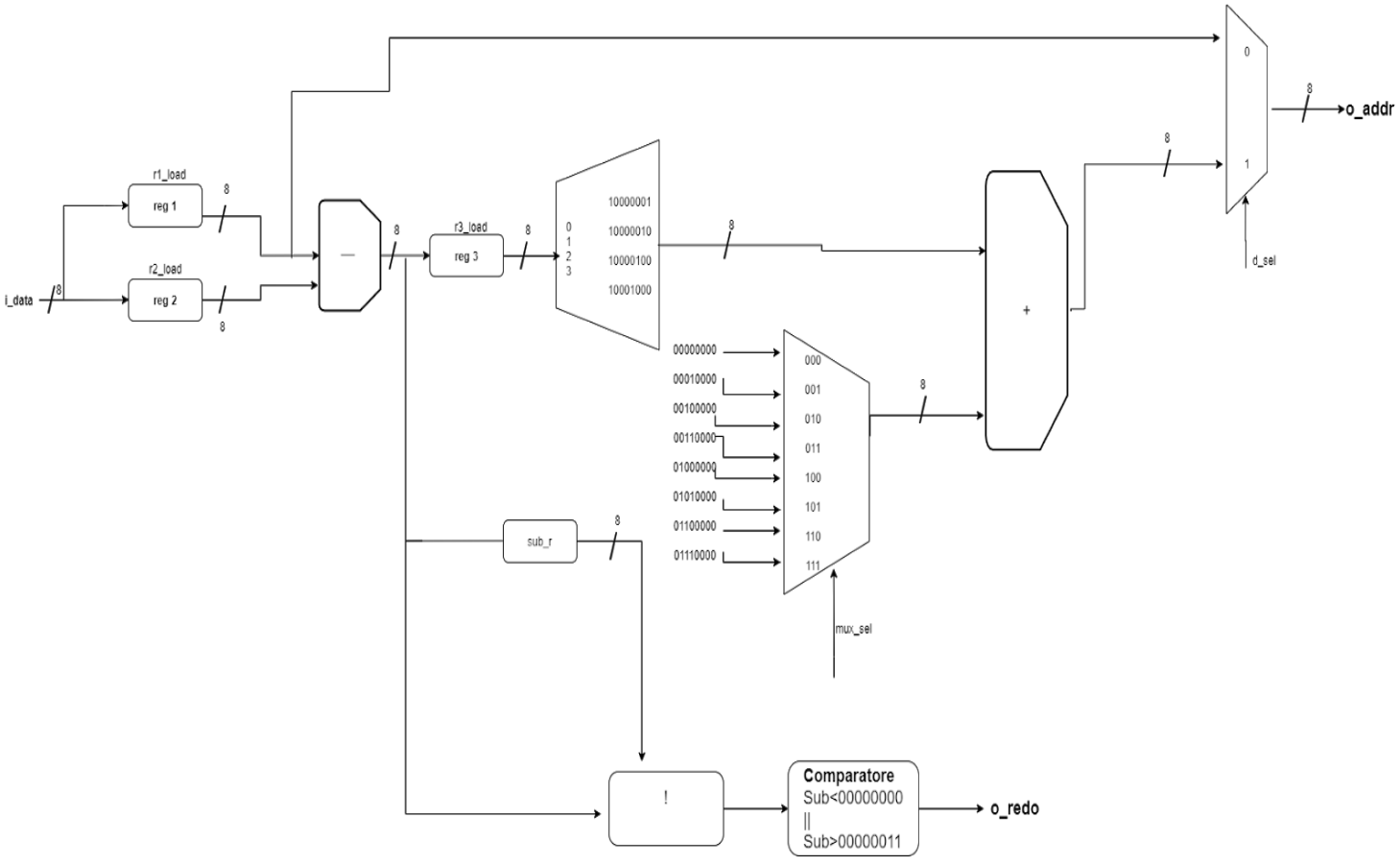
STATI:

- **S0:** È lo stato di idle del componente, nel quale si aspetta che il segnale di start sia posto a 1 per iniziare la computazione.
- **SP1:** Questo stato carica dall'indirizzo 9 di memoria il dato di ingresso e attiva la memoria in lettura.
- **S1:** Questo stato è quello da cui parte la computazione vera e propria, poiché attiva la memoria e il registro 2 per caricare il valore della working-zone presente all'indirizzo 0 in memoria.
- **S2,S2POST:** Questi due stati permettono di caricare i valori computati dal datapath nei registri e di sincronizzarli correttamente per il primo valore caricato. Da questo

stato in poi, sarà possibile proseguire nella computazione o avere già un match con una working-zone.

- **SP5, S5, SP6, S6, SP7, S7, SP8, S8, SP9, S9, SP10, S10, SP11, S11, SP12 ,S12:** Questo insieme di stati compie lo stesso lavoro, ovvero: lo stato SPX si occupa di attivare la memoria in lettura e sceglie da quale indirizzo procurare il dato e in più attiva il registro 3 per poter caricare il valore dell'offset; lo stato SX si occupa solamente di preparare il registro 2 a caricare nuovamente il valore in ingresso nel caso in cui la computazione debba continuare. Se in uno stato SX o_redo = 0, rimanda allo stato SP3;
- **SP3, S3:** Questi due stati corrispondono al caso in cui la computazione termina con un match; dunque in questi due stati si abilita in scrittura la memoria all'indirizzo 9 e si pone d_sel = 1, informazione necessaria al datapath per poter mandare in uscita la codifica elaborata correttamente.
- **SP13, S13:** Questi due stati si comportano allo stesso modo di SP3 ed S3, ma per il caso di computazione terminata senza match, dunque l'unica differenza sta nel fatto che viene posto d_sel = 0, per fare scegliere al datapath il valore del dato in ingresso, il quale era stato precedentemente salvato in un registro.
- **S4:** Viene settato o_done=1 per comunicare la fine della computazione.

2.2. Data path



2.3. Scelte progettuali

La scelta principale che abbiamo attuato è stata quella di implementare il componente mediante un Datapath e una FSM, poiché abbiamo trovato comodo dividere il lavoro del componente in una parte di calcolo puro e una parte di controllo.
Abbiamo inizialmente creato la struttura base del DataPath in modo da avere un riferimento, poi abbiamo iniziato a progettare la FSM partendo dal presupposto di avere minimo uno stato per ogni Working-zone, così da identificarle.

Nel Datapath abbiamo deciso di verificare subito se avviene la corrispondenza tra indirizzo in input e indirizzo della working-zone corrente, caricati in opportuni registri, tramite un sottrattore, il cui risultato viene salvato su un ulteriore registro. In questo modo, possiamo subito controllare se l'operazione è da ripetere oppure la computazione può finire qui, perché a ridosso del sottrattore è posto un comparatore che permette di verificare se l'offset tra ingresso e working zone corrente è un valore plausibile, producendo il segnale o_redo. Nel caso di offset valido, produciamo già parte della codifica di output tramite un decoder, il quale traduce il nostro offset in codifica one-hot, più il bit più significativo posto a 1, mentre i bit dal secondo al quarto sono lasciati a 0, poiché manca ancora l'informazione su quale indirizzo in memoria contenga la working zone corrente. Una volta che tale informazione viene comunicata dalla macchina a stati, un multiplexer manda alla sua uscita il valore di quei tre bit mancanti, il quale viene poi sommato alla codifica parziale prodotta prima. Alla fine di questa catena il secondo multiplexer decide quale dei suoi ingressi mandare in output.

Una volta creata questa struttura abbiamo notato una desincronizzazione tra il caricamento dei registri e le operazioni,quindi per ovviare a questo problema abbiamo aggiunto per ogni working-zone un ulteriore stato chiamato SPX (dove X è un numero da 1 a 13), per poter sincronizzare sul fronte di salita del clock il contenuto dei vari registri ed effettuare correttamente le operazioni che il datapath deve svolgere.
Quando o_redo=0 ci si avvia verso la fine elaborazione dato che il comparatore ha trovato una coincidenza, dunque la FSM raggiunge la parte di stati che corrispondono a fine computazione corretta. Se invece o_redo continua a essere posto a 1, la FSM continua a caricare gli altri indirizzi base delle successive working zone finché non trova una corrispondenza, eventualmente finendo negli stati che corrispondono a fine computazione ma senza aver trovato un match nelle working zone presenti in memoria.

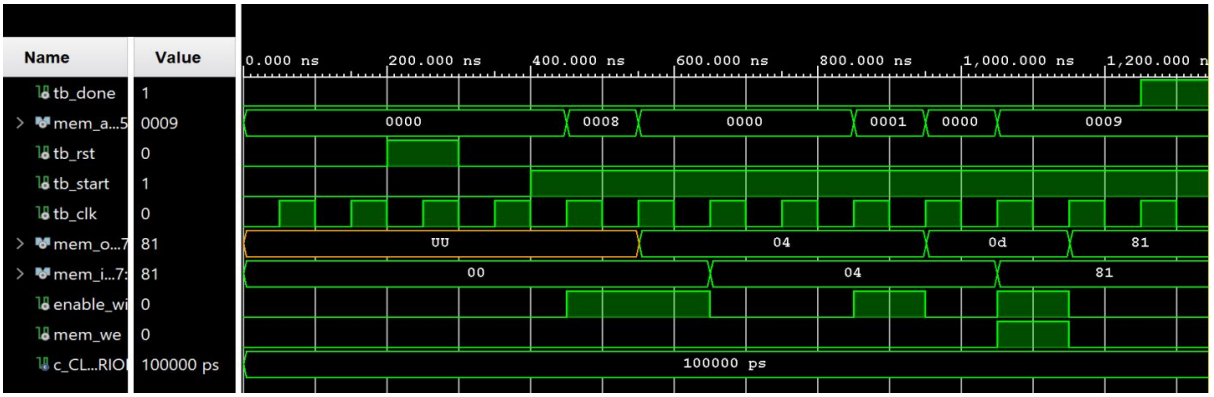
Abbiamo scelto di tradurre questo algoritmo di decisione nella maniera più lineare possibile, sacrificando così una possibile riduzione del numero degli stati per assicurarci un corretto svolgimento delle operazioni del componente.

3. Test effettuati e risultati

Per verificare il corretto funzionamento del componente sintetizzato, dopo averlo testato con il test bench di esempio, abbiamo definito altri test (tra i quali anche quelli che spingono la simulazione verso i corner case) in modo da cercare di massimizzare la copertura di tutti i possibili cammini che la macchina può effettuare durante la computazione.
Abbiamo testato ogni possibile working-zone con tutti e quattro i possibili offset con diversi valori, tra cui i due estremi della working zone e uno centrale.

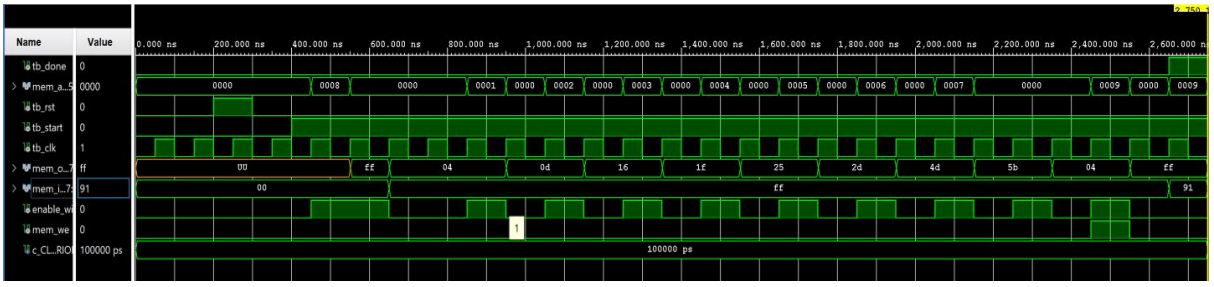
- **TEST BEST CASE**

L'indirizzo che riceve in input è localizzato nella prima working zone e con offset 0, cioè il valore 129 decimale (81 hex) in output.



- **TEST WORST CASE**

Quando l'indirizzo che cerchiamo non è presente in nessuna working zone e come output,dopo aver controllato tutte le possibili working-zone, restituisce il valore iniziale di input.



Oltre a questi due casi limite, abbiamo anche verificato per provare il corretto funzionamento del componente, il caso in cui in ingresso sia dato un indirizzo che abbia un offset negativo rispetto a una working-zone, per verificare che il comparatore nel datapath funzioni correttamente.

4. Risultati sintesi

Dopo aver verificato che la simulazione in behavioral funzionasse correttamente, abbiamo fatto partire la sintesi per verificare che il componente sia effettivamente realizzabile; dalla sintesi risulta che il datapath viene tradotto fedelmente così come la macchina a stati, motivo per cui abbiamo verificato che i test che inizialmente avevamo pensato per la simulazione in behavioral non generassero comportamenti anomali. Fatta questa verifica abbiamo anche verificato i test sopra descritti.