PROVA FINALE (PROGETTO DI RETI LOGICHE)

prof. William Fornaciari, A.A. 2021/2022

Di Lorenzo Aicardi (CP 10675881, Matr 932301) in collaborazione con Giovanni Arriciati (CP 10683631, Matr 938214)

INDICE:

Introduzione

Scopo del progetto:

Il progetto consiste nell'implementare un modulo hardware che si interfacci con la memoria e elabori la sequenza di bit letta dalla memoria secondo la specifica dalla macchina a stati scrivendo poi in memoria l'output della macchina a stati.

Descrizione dell'interfaccia:

Il componente ha quest'interfaccia:

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

Dato e gestione della memoria:

La lettura dei dati in ingresso avviene a partire dall'indirizzo di memoria 0, fino al 255; la scrittura in memoria avviene invece dall'indirizzo 1000 (mille) fino all'indirizzo 1509.

Quando la lettura e la scrittura dei dati terminano, viene inviato il segnale "`o_done`" e la macchina torna nello stato di idle in attesa di un nuovo segnale "`i_start`".

Anche nel caso venga inviato un segnale di reset il nostro componente tornerà nello stato di idle.

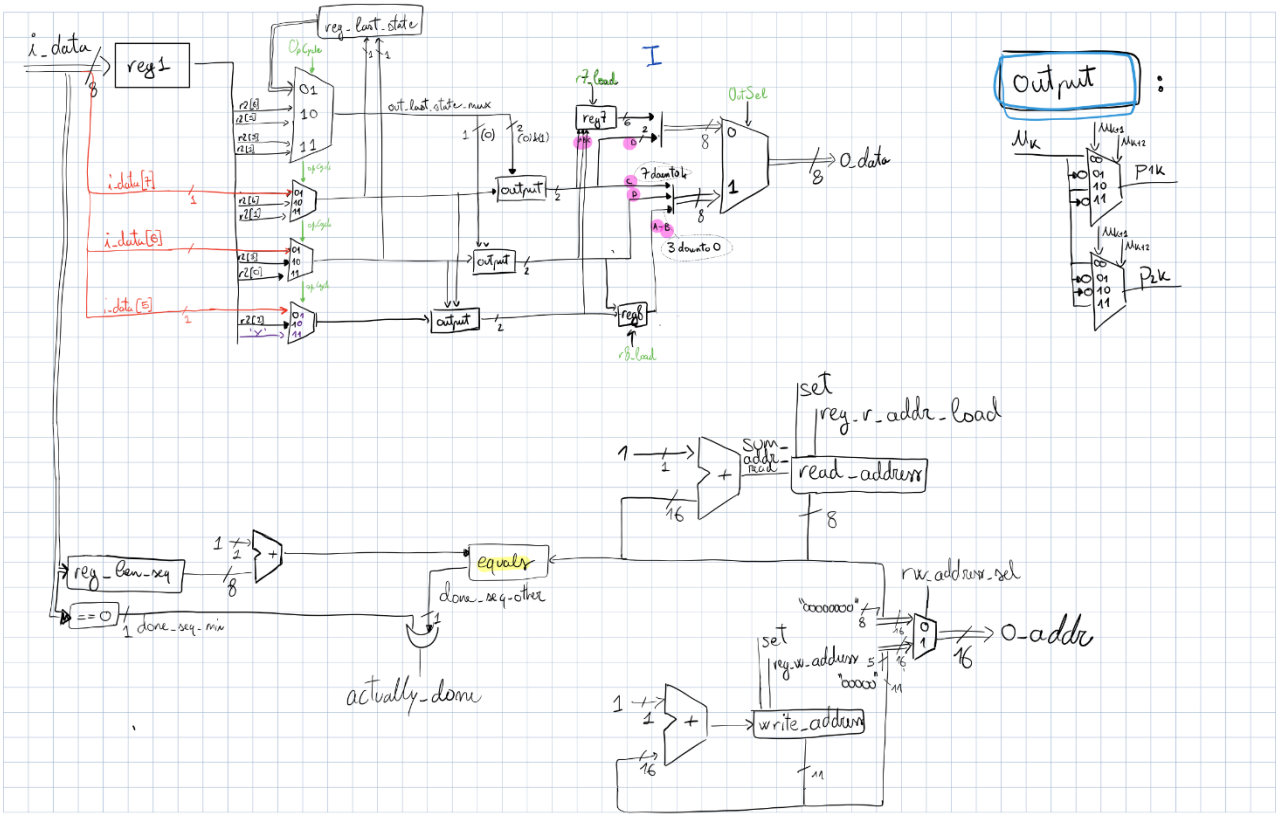
Descrizione del design

Datapath e macchina a stati:

Per semplificare lo svolgimento del progetto ci siamo aiutati con dei disegni fatti inizialmente su carta e poi digitalizzati, sia per fruirne da remoto, sia per consegnare il progetto. Abbiamo creato un datapath in tre moduli, come poi sarà anche visibile nel codice. Una parte è dedita alla scelta dell'indirizzo, una parte al controllo del termine dell'elaborazione e una parte, la più consistente, all'elaborazione delle parole in ingresso e alla loro scrittura in memoria. Per controllare il funzionamento sincrono dei vari componenti, ci siamo aiutati con la macchina a stati.

Sin dalla prima versione del progetto una delle idee portanti è stata quella di limitare più possibile il ritardo dato dai registri per permettere la massima efficienza. Nonostante, a causa di mancanza di esperienza col software, vi si siano presentate difficoltà, l'idea è stata realizzata con successo: l'uso dei registri è stato infatti ottimizzato, permettendo tempi particolarmente brevi di esecuzione, che verranno descritti nella conclusione.

I disegni del datapath e della macchina a stati sono riportati nelle figure. Il primo dato in ingresso da memoria viene inserito in un registro `len_seq_reg` che (+1) viene confrontato con l'indirizzo di scritta negli altri casi e in caso di eguaglianza porti ad alzare il segnale di controllo "`actually_done`".

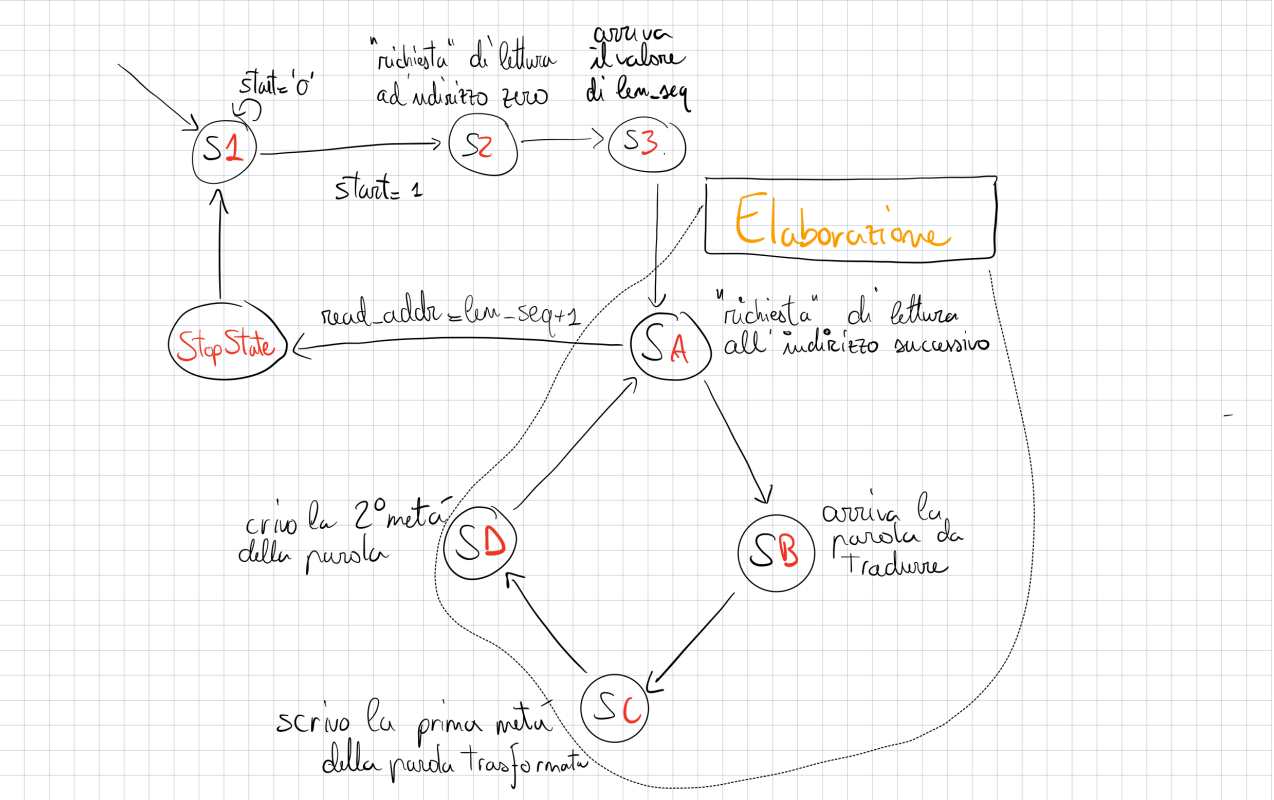


Nella sua versione finale, abbiamo scelto di gestire in maniera indipendente l'output del componente e la scelta dell'indirizzo di memoria, dato che le loro esecuzioni sono svincolate l'una dall'altra. La scelta dell'indirizzo è dettata da due sommatori, uno per l'indirizzo di lettura e uno per quello di scrittura, che vengono incrementati di "`00000001`" per ogni relativo accesso (non accediamo mai, durante un ciclo di

lavoro del componente, due volte allo stesso indirizzo). Poi la scelta dell'indirizzo da scegliere è dettata ciclicamente da un mux pilotato dal segnale "`rw_address_sel`"

Per quanto concerne l'elaborazione del dato, ogni quattro cicli di clock si esegue la lettura di una parola (nei primi due cicli) e si scrivono due parole in memoria (negli ultimi due cicli): l'operazione viene svolta traducendo insiemi di tre bit durante il 1° `op_cycle`, tre durante il 2° e i rimanti due durante il 3°.

La traduzione poi avviene attraverso l'uso di multiplexer, a loro volta pilotati da due segnali in ingresso che rappresentano lo stato e un segnale che analogo all'input della macchina a stati (3 input e 2 output), si veda ill modulo *output* nel datapath. L'output dell'elaborazione viene memorizzato nei registri `reg7` e `reg8` che fungono da buffer prima della scrittura in memoria.



Stati della macchina:

La macchina è composta da 8 stati. In ciascuno di essi avvengono le seguenti operazioni:

S1: Stato di idle, in cui si attende l'arrivo del segnale `i_start`

S2: Stato in cui avviene la prima lettura, accedendo all'indirizzo di memoria 0.

S3: Ricezione del valore contenuto in 0, definito nel datapath come `len_seq`, che contiene il numero di parole da leggere dalla memoria.

SA: Accesso all'indirizzo di memoria di lettura successivo.

SB: Ricezione della parola dall'indirizzo di memoria e prima fase dell'operazione.

SC: Scrittura della prima parola di output e proseguo dell'elaborazione.

SD: Conclusione dell'elaborazione e scrittura della seconda parola in memoria.

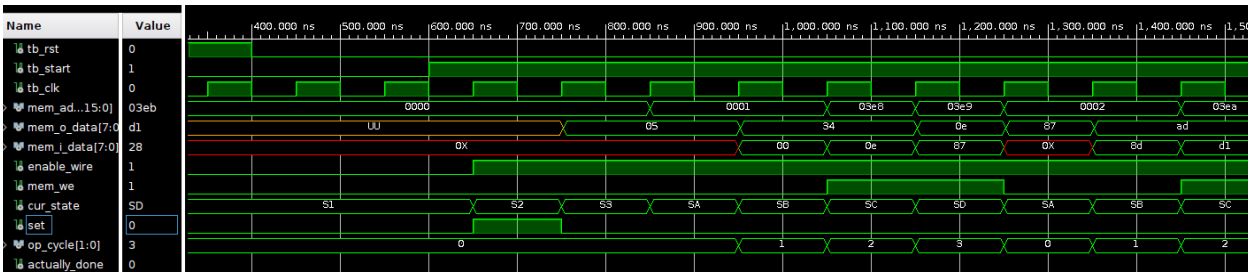
StopState: Stato di terminazione dell'esecuzione: `o_done` è portato a uno.

Conclusioni

Risultati dei test:

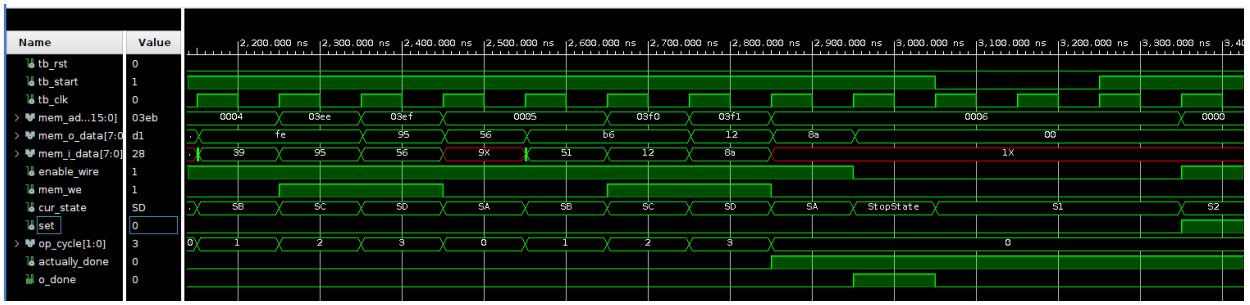
Una volta conclusa la descrizione e la sintesi, il progetto è stato testato con ogni test bench fornito dai docenti, con un comportamento che risponde correttamente ad ognuno dei test semplici. Nel caso limite, ovvero in presenza di 255 valori da leggere e tradurre, il test impiega un tempo pari a `103050 ns` (con il clock dato, il cui periodo è circa 3 volte quanto richiesto dal nostro componente).

Il comportamento della prima fase dell'elaborazione è mostrato in figura: dopo il segnale di start due cicli di clock per il setup e l'elaborazione ha inizio.



Quando il programma termina la scrittura, prima di giungere in stato di IDLE passa SA e per StopState.

Ci siamo curati che la scittura avesse termine prima di inviare il segnale di "`o_done`".



Il progetto è stato provato anche con test specifici per valutare la resistenza a tutti i possibili casi.

A questo scopo, sono stati usati test di casi limite quali:

-Caso con presenza di più segnali di reset e varie scritture;

-Caso con zero input;

-Caso con 255 input;

-Caso con un solo reset ma più segnali di start e quindi diverse elaborazioni.

Ottimizzazioni:

Nelle sue prime versioni il progetto prevedeva una macchina a stati composta da un numero più elevato di stati, pari a 12. Un primo miglioramento è consistito nella drastica riduzione nell'attesa dell'elaborazione del dato, avvenuta attraverso il passaggio dei segnali diretto in elaborazione senza passaggio per registri superflui. Un secondo miglioramento è stato nella rimozione di stati aggiuntivi atti all'esecuzione distinta della prima operazione; questo cambiamento è stato reso possibile dall'inserimento del segnale di set nel registro last_state.

Sintesi

Sia la sintesi che la simulazione post-sintesi sono avvenute con successo.

Il componente da noi sviluppato contiene 0 Latch e dunque possiamo dire che il suo comportamento è completamente prevedibile.