Politecnico di Milano
Relazione Prova Finale Reti Logiche

Ferrarini Andrea (10751746)
andrea4.ferrarini@mail.polimi.it

Consegna in maggio 2023

Contents

1 Introduzione

L'obbiettivo della seguente relazione è quello di illustrare le modalità con cui è avvenuta la progettazione, e susseguente realizzazione in VHDL, della prova finale di reti logiche dell'anno accademico 2022-2023.

In sintesi, il componente hardware specificato ha il compito di fungere da "adattatore" fra una memoria (indirizzata al byte e con indirizzi a 16bit) ed un ingresso di tipo seriale. Parafrasando, dunque, è necessario esso effettui la conversione di un ingresso di tipo seriale in un indirizzo a 16bit ben formato.

A complicare le cose, tuttavia, vi è il fatto che il dato letto da memoria non dovrà semplicemente essere mostrato in uscita, ma dovrà, bensì, essere condotto verso uno di quattro possibili *canali*. Il canale su cui è necessario scrivere viene sempre indicato secondo una codifica binaria naturale in un preambolo dell'ingresso (a cui, d'ora in avanti, ci riferiremo per semplicità con *intestazione*) dalla dimensione di 2 bit.
È importante notare che fra letture successive della memoria i canali d'uscita non dovranno essere azzerati, ma dovranno continuare a mostrare i dati letti precedentemente. Inoltre, tali dati dovranno essere visibili solamente per la durata di un ciclo di clock al termine della lettura da memoria, in tutti gli altri casi i bit di uscita dovranno essere posti a zero.

Infine, un ultimo fatto da tenere in considerazione in fase di progettazione è legato al formato dell' ingresso: a seguito dei due bit di intestazione, dovrà, naturalmente, seguire un indirizzo di memoria. Quest'ultimo, tuttavia, avrà una lunghezza **variabile**, pari a N bit, con $N \in [0, 16]$.
Pertanto l'utente esterno che si interfaccerà con il nostro modulo avrà la possibilità di fornire in ingresso indirizzi di memoria completi ($N = 16$), assenti ($N = 0$), o perfino parziali. In ogni caso, è compito del modulo assicurarsi gli indirizzi vengano forniti alla memoria sempre ben formati, e da specifica ciò dovrà avvenire tramite un'eventuale estensione con zeri della parte più significativa della sequenza fornita.

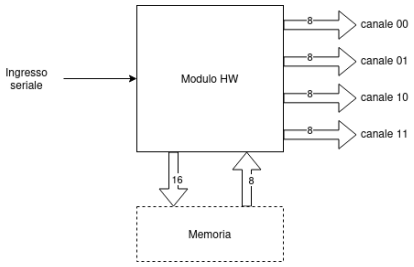

Figure 1: Schema semplificato

2 Architettura

Prima di cominciare a descrivere più in dettaglio il modulo progettato, definiamo formalmente i segnali in ingresso ed uscita a cui faremo riferimento:

- Ingresso seriale (contenente intestazione e indirizzo): **i_w**
- Segnale di controllo (segnala l'inizio della trasmissione di *i_w*): **i_start**
- Segnale di abilitazione alla lettura della memoria: **o_mem_en**
- Indirizzo fornito alla memoria: **o_mem_addr** - bus 16 bit
- Dato letto da memoria: **i_mem_data** - bus 8 bit
- Uscite del modulo (una per ognuno dei quattro canali): **o_z0**, **o_z1**, **o_z2**, **o_z3** - bus 8 bit
- Segnale di terminazione (atto a segnalare la fine della scrittura sulle uscite): **o_done**

Naturalmente, saranno anche presenti un segnale di clock (**i_clk**) e uno di reset del modulo (**i_rst**).

Logicamente, possiamo suddividere il problema in due sotto parti, delegando i diversi compiti del nostro componente a sotto-moduli specializzati.
Otteniamo, quindi, il seguente schema di alto livello:

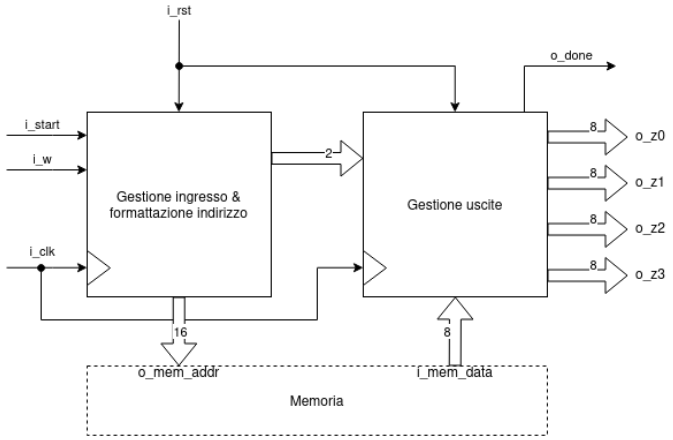

Figure 2: Schema moduli

Analizziamo ora in dettaglio i due moduli sopra illustrati.

2.1 Modulo gestione ingressi e conversione indirizzi

Il primo modulo del nostro componente ha il compito di ricevere in ingresso i bit forniti da *i_w*, convertire i bit di indirizzo in indirizzi a 16 bit ben formati, trasmettere i 2 bit di intestazione ricevuti al modulo di gestione delle uscite e, infine, trasmettere alla memoria l'indirizzo ben formato da cui effettura la lettura.
Otteniamo, in definitiva, il seguente schema circuitale per il primo modulo:

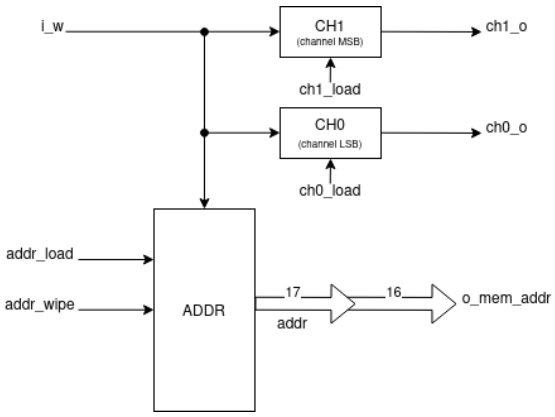

Figure 3: Modulo gestione ingressi e conversione indirizzo (*i_clk* e *i_rst* impliciti)

Analizziamo, a questo punto, la sua struttura:

- Sono presenti due flip-flop, *CH1* e *CH0*, adibiti alla memorizzazione dei bit di intestazione da inoltrare successivamente al modulo di gestione uscite. I due flip-flop sono controllati da due bit di abilitazione alla scrittura indipendenti.
- È presente un registro serie-parallelo *ADDR* da 17 bit. Ad esso sono collegati, oltre al bit in ingresso, due bit di controllo:
  1. **addr_load:** bit di abilitazione alla scrittura del registro. Quando questo bit è alto, il contenuto del registro subisce uno shift logico a sinistra e l'unico bit in ingresso diventa il nuovo LSB del registro.
  2. **addr_wipe:** bit di azzeramento **sincrono** del contenuto del registro.

Da un punto di vista funzionale osserviamo subito che il registro serie-parallelo *ADDR*, se azzerato all'inizio di ogni sequenza in ingresso (*wipe*), converte correttamente gli N bit di indirizzo in indirizzi ben formati. Tuttavia, salta subito all'occhio il fatto che *ADDR* sia un registro a 17 bit, e non a 16 bit. Per comprenderne la ragione, consideriamo il seguente problema: non appena il bit di controllo *i_start* passa da alto a basso, sarà necessario concludere la lettura dei bit in ingresso forniti da *i_w*. Non è possibile a tal fine, però, effettuare semplicemente una transizione di stato che disabiliti la scrittura in *ADDR*. Così

facendo, infatti, la scrittua nel registro serie-parallelo verrebbe effettivamente interrotta solo il ciclo di clock successivo.
Per ovviare a tale problema, è sufficiente dimensionare *ADDR* con un bit aggiuntivo rispetto al numero massimo di 16, e considerare sempre come validi soltanto i 16 bit più significativi memorizzati (in quanto il LSB sarà sempre frutto di una scrittura in eccesso).

2.2 Modulo gestione uscite

Il secondo sotto-modulo del nostro componente HW riceverà in ingresso i 2 bit di intestazione (inoltrati dal primo modulo) ed una parola da 8 bit letta da memoria (*i_mem_data*). Il suo compito è semplicemente quello di redirigere il byte in ingresso su uno dei quattro canali d'uscita. Dovendo il modulo, fra letture da memoria successive, mantenere traccia dei dati letti precedentemente, diventa necessaria l'associazione di ogni canale d'uscita ad un registro parallelo-parallelo dalla dimensione di 8 bit. Si ottiene quindi il seguente schema circuitale:

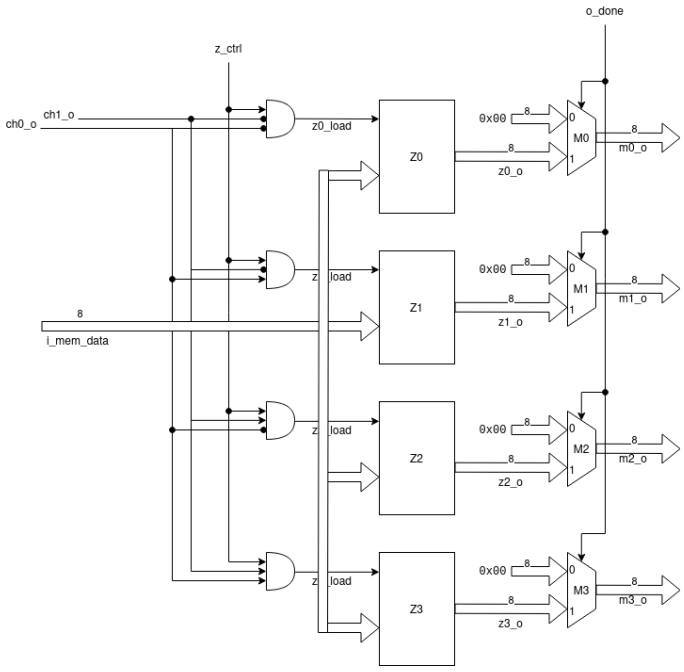

Figure 4: Modulo gestione uscite (*i_clk* e *i_rst* impliciti)

La scrittura di ogni registro, come si può vedere, viene effettuata semplicemente collegando il bit di abilitazione alla scrittura all'uscita di una porta logica AND, la quale prende in ingresso i due bit di intestazione ed un bit di controllo *z_ctrl*. L'uscita di ogni porta AND sarà alta se, e soltanto se, i due bit di intestazione contengono effettivamente la codifica del canale d'uscita cui la porta

AND è associata, ed il bit di controllo è anch'esso alto.
Come nota conclusiva, osserviamo che ciascun registro è collegato ad un multiplexer pilotato dal segnale di uscita *o_done*. Tramite questo sistema siamo in grado di mostrare in uscita i dati letti da memoria solo quando desiderato (ovvero, solamente durante il ciclo di clock in cui *o_done* è alto).

2.3 Schema dettagliato complessivo

Mettendo insieme quanto detto riguardo i due moduli, otteniamo il seguente schema circuitale complessivo, o *datapath*:

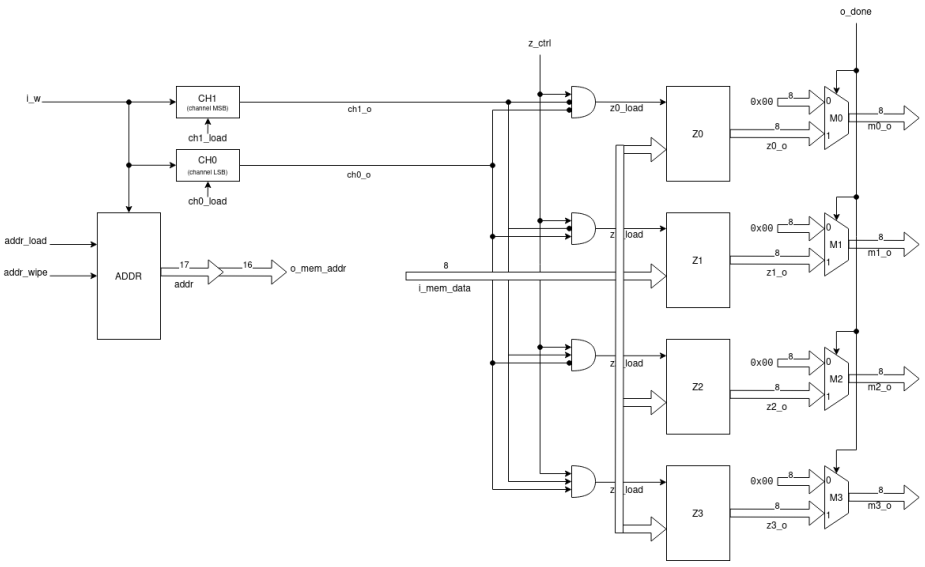

Figure 5: Schema complessivo (*i_clk* e *i_rst* impliciti)

Lo schema mostrato in 5 contiene tutta la circuiteria necessaria per il corretto funzionamento del nostro componente HW, ciononostante, è necessario associarvi anche un qualche *controllore* che piloti appropriatamente i bit di controllo interni, attivando e disattivando i diversi componenti e percorsi logici al momento giusto.
Implementiamo tale controllore con una macchina a stati dotata dei seguenti ingressi:

- I segnali di clock e reset
- Il segnale di controllo in ingresso *i_start*

E dotata delle seguenti uscite:

- Il segnale di azzeramento del registro ADDR *addr_wipe*

- I segnali di abilitazione alla scrittura di tutti i registri, ovvero *ch1_load*, *ch0_load*, *addr_load* e *z_ctrl*

- Il segnale di controllo dei multiplexer M0, M1, M2 ed M3, nonché il segnale di terminazione di scrittura in uscita, *o_done*

Avendo già discusso le specifiche dei due moduli del datapath e avendo già descritto la funzione di ciascun segnale di controllo interno, otteniamo:

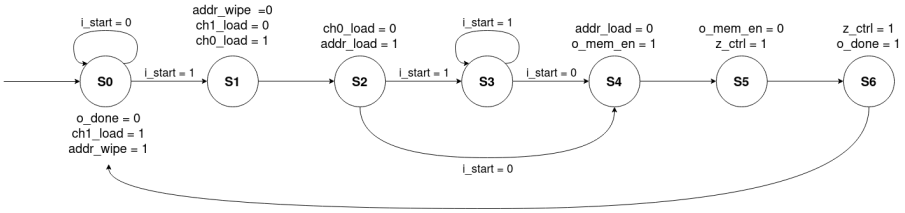

Figure 6: Macchina a stati finiti

Si tratta, chiaramente, di una macchina di Moore, in quanto le uscite non hanno alcuna dipendenza dagli ingressi.
Riassumiamo brevemente di seguito la funzione di ciascuno stato:

- **S0:** è lo stato di reset del circuito. Non appena raggiunto viene immediatamente azzerato il registro serie-parallelo *ADDR* e abilitata la scrittura di *i_w* in *CH1*.

- **S1:** raggiunto da **S0** non appena *i_start* passa da basso ad alto, in questo stato si inibisce la scrittura di *i_w* in *CH1* e la si abilita in *CH0*. Inoltre, il segnale di azzeramento di *ADDR* viene spento.

- **S2:** questo stato viene raggiunto sempre a partire da **S1** (transizione forzata) dopo un ciclo di clock. Il suo scopo è semplicemente quello di inibire la scrittura di *i_w* in *CH0* e di abilitarla in *ADDR*.

- **S3:** a partire da **S2**, se il bit di controllo in ingresso *i_start* risulta ancora alto dopo due cicli di clock, potremmo concludere di non trovarci nel caso limite $N = 0$. Dunque, effettueremo una transizione a questo stato, dove rimarremo fintantoché *i_start* non torni basso.

- **S4:** a partire da **S2** o **S3**, non appena *i_start* passa da alto a basso, si raggiunge questo stato. Finalmente la lettura di *i_w* termina (si spegne il bit di controllo *addr_load*) e si effettua la lettura da memoria (attivando *o_mem_en*).

- **S5:** completata la lettura da memoria, questo stato si assicura il dato letto venga scritto nel registro d'uscita appropriato.

- **S6:** terminata la scrittura nel registro indicato dai due bit di intestazione, si porta da basso ad alto il segnale di terminazione *o_done*, e, trascorso un ciclo di clock, si effettua una transizione forzata allo stato iniziale **S0**, dove il bit di terminazione viene riportato basso.

2.4 Sviluppo in VHDL del modulo

Arrivati a questo punto della progettazione non ci resta che tradurre in codice VHDL il modulo progettato.
Partiamo con alcuni aspetti preliminari:

- Come prima cosa, è stato definito un file *clock.xdc* per la specifica del constraint di clock. Dovendo il progetto funzionare con ciclo di clock non superiore a 100ns, il constraint è stato imposto proprio a 100ns.
- La sintesi è avvenuta su FPGA **Artix-7 FPGA xc7a200tfbg484-1**.
- Il segnale di reset è stato considerato asincrono.
- La memoria considerata è sensibile al fronte di salita del segnale di clock, e le operazioni di lettura hanno una latenza di 1ns *(dato reperibile dal sorgente dei test-bench forniti)*.
- Come richiesto da specifica, tutti gli organi di memoria sono stati implementati sensibili al fronte di salita del segnale di clock.

Da un punto di vista implementativo, si è fatto uso di due *architecture*, una per il datapath ed una per la macchina a stati (la quale crea un *component* del datapath per interagirci).
Ogni organo di memoria è stato realizzato prendendo come riferimento il seguente codice:

```
process (i_clk, i_rst)
begin
    if (i_rst = '1') then
        ch1_o  <= '0';
    elsif (rising_edge(i_clk)) then
        if (ch1_load = '1') then
            ch1_o <= i_w;
        end if;
    end if;
end process;
```

Discorso leggermente diverso va fatto per il registro serie-parallelo: ad essere memorizzato, nel suo caso, è uno `std_logic_vector` di 17 bit costruito nel seguente modo:

```
next_addr <= addr_o(15 downto 0) & i_w;
```

Ovvero concatenando i 16 bit meno signficativi già in memoria con il bit in ingresso. Ciò descrive perfettamente il comportamento di un registro serie-parallelo. Ricordiamo, infine, che ad essere collegati all'uscita *o_mem_addr* saranno solamente i 16 bit più significativi di *ADDR*:

```
o_mem_addr <= addr_o(16 downto 1);
```

3 Risultati sperimentali

La sintesi circuitale del progetto è avvenuta con successo, ed eseguendo i comandi *report_utilization* e *report_timing* nella console Tcl, è stata constatata l'assenza di latch ed un tempo di slack pari a 96.562ns, valore piuttosto vicino al limite desiderato di 100ns. Osserviamo, comunque, che riducendo il valore imposto in *clock.xdc* a 10ns e rieseguendo la sintesi, lo slack effettivamente raggiunge i 6.409ns, per cui quanto a periodo di clock i risultati sono soddisfacenti.

Passiamo, a questo punto, ai casi di test forniti:

- **TB1, TB2 & TB4:** sono semplici test circa il corretto funzionamento del componente, con più di un reset e diverse sequenze in ingresso dalla lunghezza variabile. TB1 e TB4, in particolare, inviano una stessa sequenza di *i_w* due volte, infamezzando la ripetizione con un reset. TB1 e TB2, inoltre, provano a scrivere su tutti i differenti canali di uscita, assicurandosi il corretto funzionamento per ognuno.
- **TB3 & TB5:** con durate attorno ai $\sim$ 600 cicli di clock, decisamente i tb più vasti. Si tratta, pertanto, di stress test del componente a fronte di stimoli numerosi e disparati. La differenza principale fra TB3 e TB5 sta nel fatto che TB3 esegua un unito reset del componente in principio, mentre TB5 ne esegue diversi nell'arco delle prove, verificando anche una corretta re-inizializzazione del componente *in medias res*.
- **TB6 & TB7:** vengono testati i casi limiti di *i_w* di lunghezza massima e minima, e composto da soli bit alti o soli bit bassi.

La simulazione dei test-bench, sia di tipo *behavioral* che *post-synthesis* è avvenuta con successo, indicando un corretto funzionamento del componente.

4 Conclusioni

Lo sviluppo è avvenuto in circa **12 giorni pieni di lavoro**, di cui i primi cinque giorni sono stati dedicati alla definizione dello schematico e della macchina a stati. Nel corso della giornata seguente è avvenuta la scrittura del codice VHDL, cui hanno seguito 2 giorni di perfezionamenti, correzioni, nonché test e sintesi. Infine, i 4 giorni finali sono stati impiegati per la stesura della seguente relazione.

Lo schematico mostrato è **frutto di diverse idee ed intuizioni che si sono susseguite**: una prima versione ha tentato di implementare il tutto facendo uso di un unico registro parallelo-parallelo per la memorizzazione di *i_w*. Ha seguito una versione con separazione dei registri (per canale ed indirizzo) in cui per gli indirizzi si è fatto uso di un registro serie-parallelo da 16 bit. Solo a seguito dello sviluppo e dei test-bench effettuati è emerso il problema di questa architettura, e si è giunti alla versione definitiva con serie-parallelo da 17 bit.

Un **possibile miglioramento futuro** del componente potrebbe consistere nella sostituzione del registro serie-parallelo da 17 bit con uno da 16 bit (dimensione potenzialmente più standard a livello di disponibilità sul mercato), affiancato ad una logica più sofisticata per il caso in cui l'indirizzo in ingresso sia esattamente da 16 bit.

5 Circuiti sintetizzati

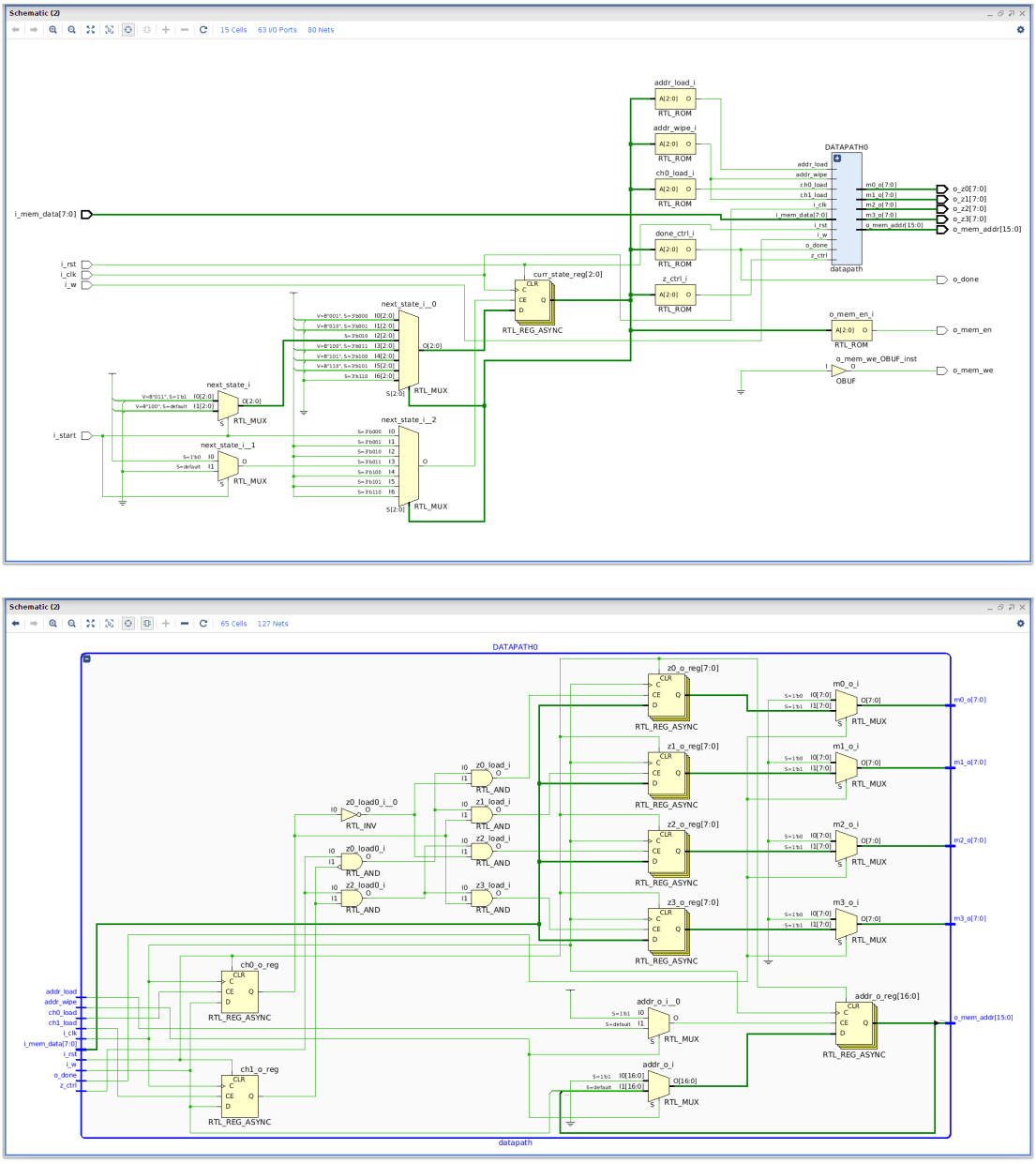

Figure 7: Macchina a stati e datapath sintetizzati