Prova Finale di Reti Logiche

Filippo Balzarini, Matteo Boglioni
10719101, 10766461

March 2023

Contents

1 Introduzione

Il progetto è un' implementazione di un componente hardware in VHDL che, leggendo da un ingresso seriale un indirizzo di memoria e la codifica del canale d'uscita desiderato, interpelli la memoria ed effettivamente restituisca il risultato.

1.1 Funzionamento generale

La rete possiede due ingressi principali entrambi da 1 bit: il primo è chiamato START e indica quando la sequenza ricevuta sul secondo ingresso W è valida. W è invece un ingresso sul quale viene ricevuta una sequenza serializzata di lunghezza variabile fra 2 e 18 bit. I primi due bit ricevuti tramite W indicano su quale dei 4 canali d'uscita del componente (Z0/Z1/Z2/Z3) si desideri ricevere il dato letto da memoria. I successivi bit di W sono l'inidirizzo di memoria di cui si vuole leggere il contenuto. Gli indirizzi di memoria sono da 16 bit, nel caso i bit ricevuti su W (oltre ai 2 iniziali) siano meno si effettua del padding con sequenze di bit 0 fino a raggiungere la lunghezza standard. Ricevuta l'intera sequenza in ingresso il componente interroga la memoria dalla quale riceve il risultato su un canale a 8 bit. Una volta disponibile il componente pone a 1 l'uscita DONE e restituisce sul canale richiesto il dato appena ricevuto. Importante evidenziare che, alla fine di questo procedimento, il componente torna allo stato iniziale ma il valore sui canali d'uscita deve essere mantenuto affinché ogni volta che DONE viene posto a 1 ciascuno dei canali d'uscita assuma l'ultimo valore restituito, ad eccezione ovviamente del canale che in quella specifica situazione ha appena aggiornato il proprio valore.

1.2 Interfaccia

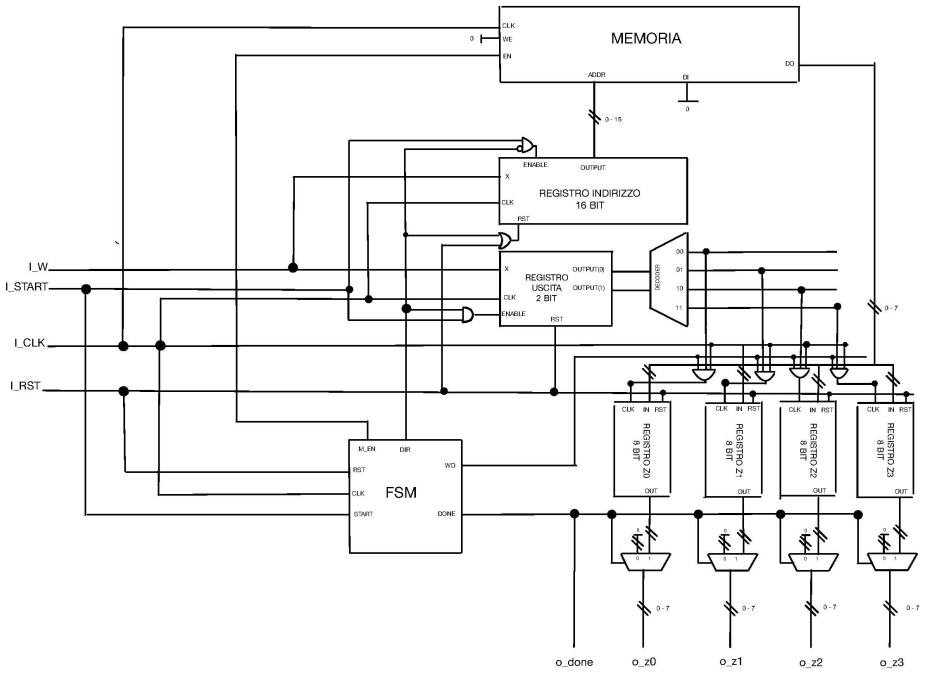
```
entity project_reti_logiche is
    port (
        i_clk : in std_logic;
        i_rst : in std_logic;
        i_start : in std_logic;
        i_w : in std_logic;

        o_z0 : out std_logic_vector(7 downto 0);
        o_z1 : out std_logic_vector(7 downto 0);
        o_z2 : out std_logic_vector(7 downto 0);
        o_z3 : out std_logic_vector(7 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_we : out std_logic;
        o_mem_en : out std_logic
```

```
    );
end project_reti_logiche;
```

Nello specifico:

- i_clk è il segnale di clock che scandisce il funzionamento del componente e la serializzazione dell'ingresso W.
- i_rst è il segnale di reset, quando viene attivato il componente si riporta immediatamente nello stato iniziale con tutti i registri a 0.
- i_start come descritto nel paragrafo precedente questo ingresso indica quando la sequenza in ricezione sull'ingresso i_w sia valida.
- i_w su questo ingresso a 1 bit viene ricevuta la sequenza d'ingresso serializzata descritta sopra.
- o_z0 / o_z1 / o_z2 / o_z3 sono i quattro canali d'uscita sui quali è possibile restituire il dato letto da memoria. Sono linee a 8 bit perchè restituiscono parallelamente tutti i bit del dato.
- o_done è l'uscita che viene posta a 1 quando il componente è pronto per restituire il valore aggiornato delle uscite.
- o_mem_addr è un'uscita a 16 bit sulla quale viene indicato l'indirizzo di memoria.
- i_mem_data è un'ingresso a 8 bit che contiene il risultato di un'operazione di lettura da memoria.
- o_mem_en è il segnale che abilita la memoria per le operazioni di lettura e scrittura.
- o_mem_we è il segnale che è abilita la scrittura sulla memoria di un dato, nel nostro caso questo segnale è sempre a 0 visto che tutte le operazioni che dobbiamo svolgere sono in lettura.

2 Architettura



Per costruire l'architettura del progetto la scelta è ricaduta su un approccio modulare, in cui i vari sottocomponenti sono stati suddivisi in moduli indipendenti tra loro, ognuno con le proprie interfacce di input ed output. I moduli utilizzati sono otto, nel particolare: sei registri di cui due con ingresso seriale ed uscita parallela e quattro con entrambi ingresso ed uscita parallele, un decoder ed una macchina a stati. Nota importante per lo sviluppo è che ciascuno di questi componenti è stato testato stand-alone prima di assemblare l'intera rete per cui si ha garanzia del funzionamento di ogni sottoparte.

2.1 Registri serie - parallelo

La specifica richiede di trattare un ingresso seriale, è quindi stato necessario individuare un modulo semplice capace di immagazzinare la stringa ricevuta. Per questo motivo l'approccio più intuitivo ed efficace è risultato essere una coppia di registri, seguendo la già preannunciata tecnica di modularità, in particolare il primo si occupa di memorizzare l'uscita selezionata, mentre il secondo per memorizzare e fornire l'indirizzo alla memoria.
Una grande attenzione è stata posta nell'utilizzo dei **segnali di enable** di ogni registro: piuttosto che gestire una combinatoria per regolarne l'ingresso, si è preferito un approccio che si è rivelato fondamentale per l'intero progetto: il segnale di ingresso W arriva infatti contemporaneamente a entrambi i registri

ma questi ultimi aggiornano i loro valori solo quando il loro singolo enable viene attivato, in tutti gli altri casi mantengono quanto precedentemente memorizzato. In particolare, i due registri serie-parallelo utilizzati sono:

2.1.1 Registro per la selezione dell'uscita

Un registro a due bit, che regola l'uscita sulla quale memorizzare il dato prelevato dalla memoria. Il suo segnale di enable è caratterizzato da un **and** tra il segnale di start (se questo fosse a 0 logico significherebbe che l'ingresso su W non è significativo) e il segnale interno al componente definito come **director** che attiva in modo esclusivo questo registro quando director = '1' mentre attiva il registro di selezione dell'indirizzo, qui sotto descritto, quando director = '0'. È garantito infatti intrinsecamente dalla serializzazione che i due bit che indicano il canale d'uscita e quelli riguardanti l'indirizzo non siano sovrapposti sull'ingresso W.

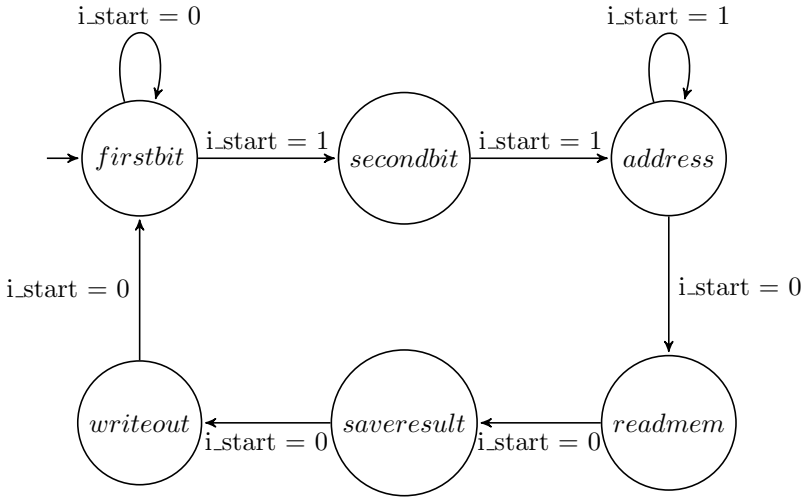
2.1.2 Registro per la selezione dell'indirizzo

Un registro a sedici bit, che permette di memorizzare l'indirizzo del dato desiderato in memoria. Il suo segnale di enable è caratterizzato da un **and** tra il segnale di start e il negato del segnale director prima descritto. Inoltre, il padding di zeri nell'indirizzo, necessario per specifica, è già naturalmente implementato. Per renderlo completamente funzionante è stata utilizzata una porta OR nell'ingresso del RESET, tra il segnale director della FSM e il segnale di RESET in modo che il registro venga resettato se il segnale di reset viene portato a 1 o quando inizia un nuovo ciclo di funzionamento (ovvero nel momento in cui si ricevono i primi due bit sull'ingresso w).

2.2 Registri parallelo - parallelo

I registri parallelo - parallelo, tutti e quattro da 8 bit, sono stati utilizzati per mantenere l'ultimo dato caricato dalla memoria, in modo da renderlo disponibile su ciascuna delle uscite quando necessario. In questi registri si è preferito, rispetto alla creazione di un segnale di enable, mascherare l'ingresso del clock con l'uscita del decoder ed il segnale in uscita dalla fsm di *wo* (spiegato successivamente). In questo modo, nonostante la memoria sia perennemente collegata a tutti e quattro registri, solo il valore di uno di essi può essere aggiornato, mentre gli altri manterranno lo stato precedente.

2.3 Macchina a stati finiti



La macchina a stati è stata concepita come macchina di Moore a singolo ingresso. Lo stato di essa infatti dipende unicamente dal segnale di start. L'interfaccia della macchina a stati è la seguente:

```
entity fsm_controller is
        port(
            clk         : in std_logic;
            reset       : in std_logic;
            start       : in std_logic;
            dir         : out std_logic;
            wo          : out std_logic;
            done        : out std_logic;
            mem_en      : out std_logic
        );
end fsm_controller;
```

Nello specifico, per quanto riguarda i segnali di input l'interpretazione è intuitiva in quanto connessi agli omonimi segnali dell'intera rete, i segnali di output invece svolgono i seguenti ruoli:

- **dir**: è il segnale director sopra descritto, abilita in modo esclusivo l'aggiornamento del registro per la selezione dell'uscita (2 bit) o il registro per la selezione dell'indirizzo di memoria (16 bit).
- **wo**: è il segnale che abilita l'aggiornamento dei registri che memorizzano il dato ricevuto dalla memoria.
- **done**: come intuibile, è il segnale di done della rete.
- **mem_en**: è il segnale utilizzato per attivare la memoria durante le operazioni di lettura.

In seguito sono descritti tutti gli stati con i rispettivi output della fsm.

- **firstbit**: è lo stato iniziale della macchina, ogni qual volta la rete viene resettata la fsm torna in questo stato. Il funzionamento è il seguente: se i_start continua a rimanere a '0' la fsm rimane in questo stato, se invece va a '1' si passa allo stato successivo. Gli output della macchina a stati nello stato di firstbit sono:

Output	Valore Logico	Note
dir	1	Il bit ricevuto deve essere memorizzato nel registro per la selezione dell'uscita
wo	0	Non ci sono dati in arrivo dalla memoria da memorizzare
done	0	Il risultato ovviamente non è già disponibile
mem_en	0	l'indirizzo di lettura in memoria non è ancora stato specificato

- **secondbit**: una volta ricevuto il primo bit dobbiamo necessariamente, da specifica, ricevere un ulteriore bit per completare la selezione del canale d'uscita desiderato. In questo stato gli output rimangono gli stessi dello stato precedente.

- **address**: ricevuti i bit di indirizzo si ricevono i bit che indicano l'indirizzo di memoria da cui si desidera leggere. In questo stato, fintanto che il segnale i_start rimane a '1', si continua a memorizzare l'indirizzo, non appena i_start si abbassa si può passare allo stato successivo.

Output	Valore Logico	Note
dir	0	Il bit ricevuto deve essere memorizzato nel registro per la selezione dell'indirizzo
wo	0	Non ci sono dati in arrivo dalla memoria
done	0	Il risultato ovviamente non è già disponibile
mem_en	0	l'indirizzo di lettura in memoria non è ancora stato completamente specificato

- **readmem**: a questo punto la fsm può avviare la lettura da memoria, è quindi sufficiente attivare il segnale di mem_en.

Output	Valore Logico	Note
dir	0	Il valore del segnale è inifluente, essendo in AND con il segnale di start (garantito dalla specifica essere a '0') i registri non subiranno alcun aggiornamento.
wo	0	non ci sono ancora dati da memorizzare
done	0	Il risultato ovviamente non è già disponibile
mem_en	1	Si abilita l'operazione di lettura in memoria

- **saveresult**: da specifica è noto che al ciclo di clock successivo alla richiesta di lettura dalla memoria potremo trovare tale dato disponibile. Proprio per questo è necessario salvare nel registro d'uscita corrispondente i valori in arrivo da memoria.

Output	Valore Logico	Note
dir	0	Vedi stato precedente
wo	1	Dati in arrivo dalla memoria da memorizzare nel registro d'uscita
done	0	Il risultato ovviamente non è già disponibile
mem_en	1	È necessario mantenere la memoria abilitata per salvare correttamente il dato

- **writeout**: una volta memorizzato il dato, è necessario renderlo disponibile sulle uscite, questo stato si occupa infatti di abilitare le uscite alzando il segnale di done.

Output	Valore Logico	Note
dir	0	Vedi stato precedente
wo	0	Blocca l'aggiornamento dei registri d'uscita
done	1	Il risultato è disponibile sulle uscite
mem_en	0	Memoria non in uso

2.4 Decoder

Il decoder implementato è un decoder binario a 4 bit in uscita. Questo componente permette di selezionare quale dei 4 registri d'uscita si vuole aggiornare. Il vettore da 2 bit in ingresso infatti è la codifica del canale d'uscita desiderato mentre i vettori in uscita sono connessi ciascuno a un registro differente. È evidente che si potrà contemporaneamente aggiornare solo uno dei 4 registri presenti.

2.5 Multiplexer

Le uscite dei registri di memorizzazione del risultato sono ciascuna in ingresso a un multiplexer il cui selettore è il segnale di done. Se done è '1' si manda in uscita il contenuto del registro, nel caso in cui invece il segnale di done sia '0' il multiplexer seleziona un segnale con tutti gli 8 bit a '0'.

2.6 Scelte Progettuali

Di seguito un breve elenco di quelle che sono state le scelte progettuali con i corrispettivi trade-off:

- **Modularità**: nella fase preliminare di progetto della rete si è tentato quanto più possibile di renderla modulare dividendola in sotto-componenti

da sviluppare in modo indipendente. Questo ha consentito non solo di identificare più facilmente le parti funzionali da implementare ma ha anche permesso di avere una maggiore efficienza durante l'unione di queste ultime avendo garanzia che ciascuna fosse funzionante.

- **Semplicità**: altro punto cardinale è stato quello di ideare una combinatoria semplice e asciutta, sia per rendere più agevole un eventuale debugging che per questioni di latenza.

- **Registri con architettura indicata per il loro utilizzo**: identificare quale tipologia di registro fosse la più indicata per ogni componente (serie-parallelo e parallelo-parallelo nel nostro caso) ha permesso di velocizzare di moltissimo il funzionamento del componente.Ad esempio, all'interno del registro per la memorizzazione dell'indirizzo di memoria non è necessario gestire il padding: essendo il registro inizializzato a '0' su ogni bit e riempiendolo serialmente dal bit meno significatico questa funzionalità è garantita senza dover implementare alcuna combinatoria ulteriore.

- **FSM senza stato di attesa**: la fsm è stata pensata priva di uno stato di attesa in cui rimanere prima di ricevere il primo bit di start a '1', la macchina a stati rimane infatti nello stato di lettura del primo bit. Il vantaggio è che non appena il segnale di start va a '1' il registro di memorizzazione dell'uscita desiderata entra in funzione senza alcun ritardo. In presenza di start a '0' se anche il segnale w assumesse valori logici alti il registro comunque non li memorizzerebbe essendo il suo segnale di enable basso (vedi 2.1.1).

- **Segnali di enable**: permettono di tenere sempre direttamente collegati i registri di selezione uscita e di memorizzazione indirizzo all'ingresso w senza dover gestire una complicata combinatoria.

- **Segnale di reset del registro di memorizzazione indirizzo**: l'unico registro che è necessario manualmente resettare ad ogni nuovo ciclo di funzionamento è quello per la memorizzazione dell'indirizzo,per fare questa cosa in modo semplice il suo segnale di reset è l'OR fra il segnale di reset vero e proprio e il segnale director (già esposto in precedenza). director infatti assume valore '1' ad inizio del ciclo di funzionamento del componente quando riceviamo i primi due bit che indicano il canale d'uscita.

3 Risultati Sperimentali

3.1 Sintesi

Il componente sviluppato è stato sintetizzato tramite Vivado con FPGA target Artix-7 xc7a200tfbg484-1. Il report di timing riporta uno Slack (required time - arrival time) di 97.306ns in cui il tempo richiesto è di 100ns da specifica quindi la rete ottiene ottimi risultati in termini di timing.
Il report di utilization post-sintesi riporta:

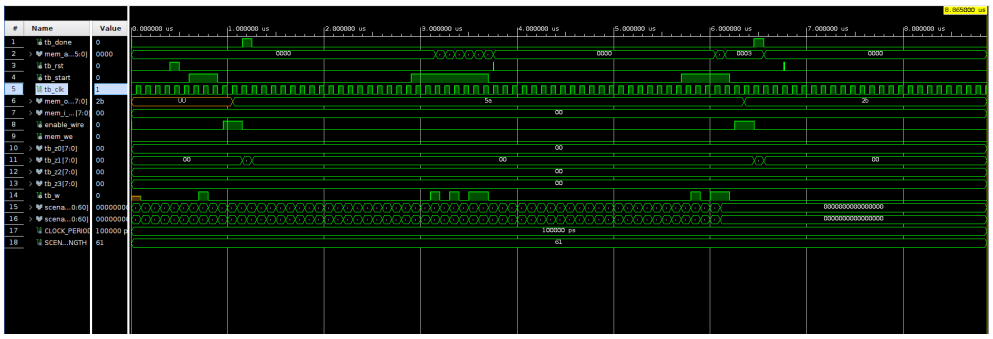
Site Type	Used	Fixed	Prohibited	Available	Util%
Slice Registers	56	0	0	269200	0.02
Register as Flip Flop	56	0	0	269200	0.02
Register as Latch	0	0	0	269200	0.00

La rete non presenta infatti alcun latch, il che permette di evitare comportamenti indesiderati.

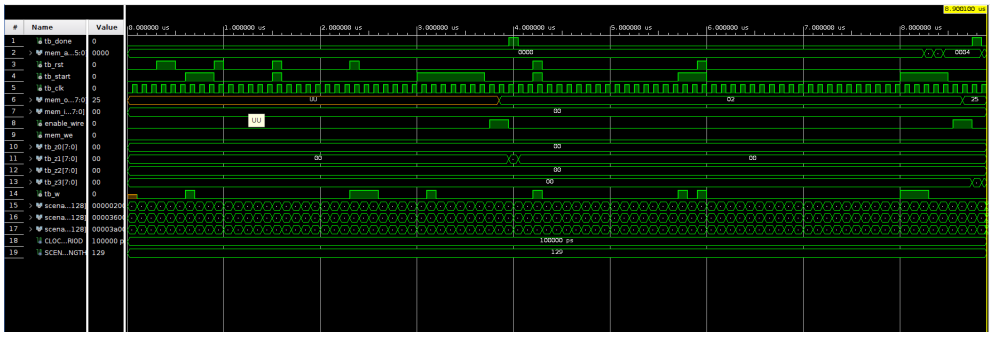
3.2 Valutazione dei casi limite

Al fine di verificare il funzionamento della rete in tutti quanti più casi possibile, oltre al test fornito, sono stati sviluppati alcuni testbench che simulassero particolari condizioni di utilizzo. I test sono stati verificati sia su simulazione Behavioral che sulla rete sintetizzata in entrambi Post-Synthesis Functional che Post-Synthesis Timing in modo da avere vincoli di funzionamento quanto più solidi possibili. I casi valutati (e correttamente superati) sono i seguenti:

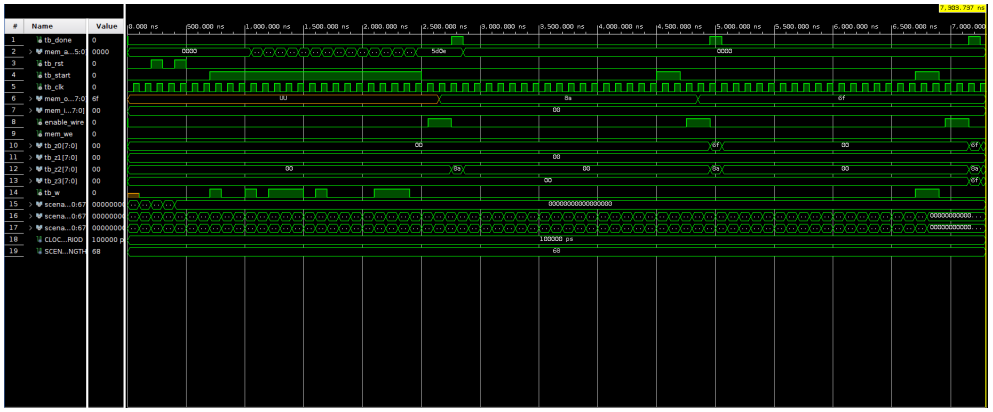
- **Reset Asincrono**: dal momento che nel testbench fornito il segnale di reset era sincrono sul fronte del clock, è stato implementato un testbench che valutasse il funzionamento della rete con segnali di reset asincroni.



- **Reset Multipli e Concorrenti al Segnale di Start alto**: È stato valutato il corretto comportamento della rete in corrispondenza di reset multipli consecutivi, anche in presenza del segnale di start alto.



- **Valori di soglia**: In questo testbench si è verificato che il componente si comporti correttamente in corrispondenza del segnale di start attivo per due soli cicli di clock (con padding indirizzo in memoria richiesto $\underbrace{0\ldots0}_{16bit}$) e con il segnale di start attivo per 18 cicli di clock (2 cicli per selezione uscita e 16 per l'inserimento dell'indirizzo).



- **Sovascrittura dei Registri di Uscita**: al fine di valutare il corretto funzionamento dei registri di uscita abbiamo valutato la sovrascrittura dei valori di uscita. In questo testbench è stato richiesto al componente più volte di riportare il dato ricevuto dalla memoria sul medesimo canale di output.
- **Segnali di Start e W Asincroni**: è stata valutata la risposta della rete a segnali di start e w asincroni, per accertare che la macchina a stati riuscisse a gestirli correttamente.

- **Segnale W alto con Start basso**: è stato verificato che con start a '0' anche se il segnale W assume valore logico alto questo non influenza il componente che, in questo caso, continua ad attendere un segnale di start alto.

4 Conclusioni

A valle del processo di sviluppo e testing eseguito abbiamo perciò buone basi per poter sostenere che il componente sviluppato rispetta i requisiti definiti in fase di specifica e risulta perciò conforme.

4.1 Possibili sviluppi futuri

Una percorribile ottimizzazione in termini di latenza potrebbe essere ottenuta rilasciando il risultato sulle uscite al ciclo direttamente successivo alla richiesta del dato in memoria: ciò prevederebbe l'aggiornamento del registro d'uscita in contemporanea con una combinatoria che porti il risultato all'esterno. Il vantaggio consisterebbe nel permettere al segnale di DONE di salire in anticipo di un ciclo di clock, non è stato implementato in questa versione per due principali ragioni:

1. Semplicità: la combinatoria da aggiungere sarebbe stata una possibile fonte di errori senza portare particolari vantaggi
2. Specifica: il componente si è già rivelato piuttosto rapido nell'esecuzione rispetto alla specifica, non è stata valutata perciò necessaria tale ottimizzazione.