Prova Finale Reti Logiche

Patrick Poggi

15 maggio 2023

Indice

1 Introduzione **3**
1.1 Specifica del componente 3
1.2 Descrizione e funzionamento ad alto livello 3

2 Architettura **5**
2.1 I segnali 5
2.1.1 I segnali esterni 5
2.1.2 I segnali interni 5
2.2 I moduli 6
2.2.1 FSM 6
2.2.2 Demultiplexer 6
2.2.3 Registri 6
2.2.4 Multiplexer 7

3 Funzionamento interno **8**
3.1 FSM 8
3.2 Demultiplexer 9
3.3 Registri 9
3.4 Multiplexer 9

4 Risultati sperimentali **10**
4.1 Risultati della sintesi 10
4.2 Test bench forniti dal docente 11
4.3 Test bench scritti in fase di progettazione 13

5 Conclusioni **15**

1 Introduzione

1.1 Specifica del componente

Il componente progettato si interfaccia con altri componenti che lo utlizzano attraverso alcuni ingressi e alcune uscite. Gli ingressi più importanti sono *START*, *W*, *DATA*. Le uscite più importanti sono 5: *Z0*, *Z1 Z2*, *Z3*, *ADDRESS*. I primi due ingressi sono stream seriali di bit (il componente riceve un bit alla volta) mentre il terzo è uno stream parallelo di 8 bit. Le prime quattro uscite invece sono ciascuna un bus parallelo a 8 bit, mentre la quinta è un bus parallelo a 16 bit.

In particolare il componente dovrà interfacciarsi con una memoria che, interrogata dal componente tramite l'uscita ADDRESS, fornisce a quest'ultimo il dato contenuto all'indirizzo di memoria fornito. Questo dato proveniente dalla memoria sarà trasportato in ingresso al componente dal bus DATA.

I segnali START e W sono controllati da un altro componente che si interfaccia con il nostro, ma del quale non importa ai fini della specifica. Analogamente le uscite Z* vengono lette da altri.

Infine ci sono altri segnali di ingresso e di uscita: *CLOCK*, *RESET* (ingresso) e *DONE*, *MEMEN*, *MEMWE* (uscita). Si noti che il RESET è asincrono.

1.2 Descrizione e funzionamento ad alto livello

L'obiettivo era quello di realizzare un componente che ricevendo uno stream di bit sull'ingresso *W* fosse in grado di dedurne due informazioni: la prima verrà chiamata *intestazione* e la seconda *indirizzo*. Più precisamente il componente progettato deve avere la capacità di leggere un bit dallo stream seriale W quando il segnale sull'ingresso *START* è pari a '1' (ossia: ha un valore logico alto) e memorizzare tutti i bit letti finché il segnale START non torna a un valore logico basso, '0'. È compito del componente dedurre le due informazioni intestazione e indirizzo da questo insieme di bit. La prima sarà composta dai primi due bit che il componente ha ricevuto, in ordine cronologico. La seconda, l'indirizzo, sarà sempre lungo 16 bit ma non necessariamente tutti questi saranno stati determinati da quanto letto dall'ingresso W. Il segnale di START infatti può non rimanere a un valore logico alto per il tempo necessario a far sì che su W vengano trasmessi 16 bit (più i 2 di intestazione che invece sono garantiti). Per questo motivo il componente ottiene un numero x di bit tale che $0 <= x <= 16$ e questi, ancora una volta in ordinde cronologico, verranno a comporre la parte meno significativa

della stringa di bit che rappresenta l'indirizzo. Se x è minore di 16 allora ci saranno $16 - x$ bit pari a 0 nella parte più significativa dell'indirizzo.

Il dato che giunge al componente a seguito dell'interrogazione della memoria dovrà essere posto su una delle quattro uscite Z*, quale di queste verrà indicato dall'intestazione in base al significato dei suoi due bit come numero binario intero positivo. Le uscite devono inoltre avere una memoria dell'ultimo dato che hanno presentato e presentarlo tutte e sole le volte che il segnale di DONE è al valore logico alto.

Tutti ciò, vale a dire il cambiamento di valore dei vari segnali in ingresso e in uscita, viene orchestrato dal segnale di CLOCK in ingresso al componente. Vincolo del sistema è che il segnale di DONE presenti un valore logico alto solo per un ciclo di clock. Altro vincolo del componente è che quando il segnale di RESET ha un valore logico alto tutte le uscite (Z*) resettino la propria memoria dei dati precedentemente presentati.

2 Architettura

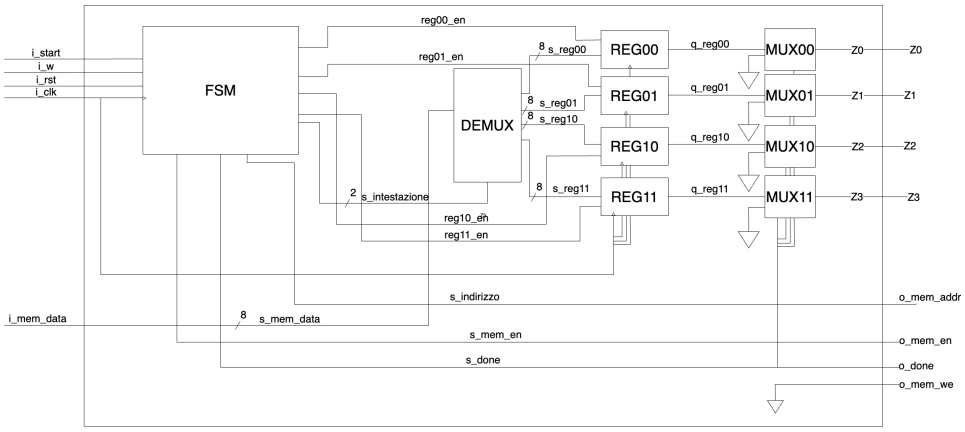


Figura 1: Schema del componente

Quella che si presenta è una descrizione top-down dell'architetura del componente. Dall'esterno si ha quanto detto nella specifica, ovvero una *entity* con i sopraelencati segnali.

2.1 I segnali

2.1.1 I segnali esterni

I nomi finora utilizzati per i segnali in ingresso e in uscita al componente sono quelli che meglio rendono l'idea del loro significato. Tuttavia nel progetto vero e proprio sono stati usati nomi leggermente diversi. START è stato chiamato i_start, W: i_w, CLOCK: i_clk, RESET: i_rst, DATA: i_mem_data, Z*: z* (es.: z0), ADDRESS: o_mem_address, MEMEN: o_mem_en, MEMWE: o_mem_we, DONE: o_done.

2.1.2 I segnali interni

All'interno del componente troviamo invece s_intestazione, s_indirizzo, s_done, s_mem_en e uno di ciascuno dei seguenti per ognuno dei quattro registri e mutex (si veda: I moduli): regxy_en, s_regxy, q_regxy, dove xy rappresenta la stringa di due bit il cui valore come intero binario senza segno rappresenta il numero da 0 a 3 corrispondente a una delle 4 uscite Z*.

2.2 I moduli

All'interno del componente troviamo diversi moduli: FSM, Demultiplexer, Registri, Multiplexer. Essi, dal punto di vista implementativo, sono processi. All'interno del componente, come si vedrà, viene fatto largo uso di segnali interni collegati o agli ingressi o alle uscite.

Più precisamente all'interno del componente troviamo una FSM, un Demultiplexer, quattro Registri (uno pero ogni uscita Z*) e quattro Multiplexer (uno per ogni usicta Z*).

2.2.1 FSM

I segnali in ingresso alla FSM sono START, W, DATA, CLOCK, RESET (cioè tutti i segnali in ingresso al componente). La FSM produce tre uscite che vengono gestite come segnali interni al componente: *s_ intestazione*, *s_ indirizzo* e *s_ done*. Il primo dei due è posto in ingresso al Demultiplexer, così come DATA. Il secondo, cioè s_indirizzo, è collegato all'uscita ADDRESS. Il segnale s_done è collegato sia all'uscita DONE sia ai 4 multiplexer.

2.2.2 Demultiplexer

Il demultiplexer riceve in ingresso il segnale s_intestazione, il quale agirà da selettore, e il segnale *s_ mem_ data*, che altro non è che una replica dell'ingresso DATA. Si tratta di un demultiplexer 1 a 4, dove tutti i canali sono bus paralleli a 8 bit. Le quattro uscite sono una per ogni registro e corripondono ai segnali s_regxy.

2.2.3 Registri

Ciascun registro riceve in ingresso il segnale che esce dal demultiplexer (es.: s_reg10 per il resgistro relativo all'uscita Z2), che sarà quello preso in cosiderazione in fase di aggiornamento dello stato. Inoltre ogni registro prende in ingresso anche un segnale di *enable* sincrono (uno dei quattro regxy_en) che è appunto un segnale di enable che quando è al valore logico alto fa sì che sul fronte di salita del clock il registro aggiorni il suo stato. Ovviamente ciascun registro ha in ingresso anche il segnale di CLOCK (ingresso del componente) e il RESET asincrono. In uscita a ciascun registro si trova un bus parallelo a 8 bit, chiamato q_regxy, diretto verso il corrispondente multiplexer.

2.2.4 Multiplexer

Si tratta di multiplexer 2 a 1 con canali di ingresso da 8 bit e quindi ogni multiplexer ha in ingresso un bus parallelo a 8 bit collegato "a terra" e pertanto sempre pari al valore logico basso, l'uscita del registro corrispondente (q_regxy) e il segnale s_done menzionato precedentemente che funziona da selettore. In uscita si ha solo un bus parallelo a 8 bit connesso all'uscita corrispondente.

3 Funzionamento interno

3.1 FSM

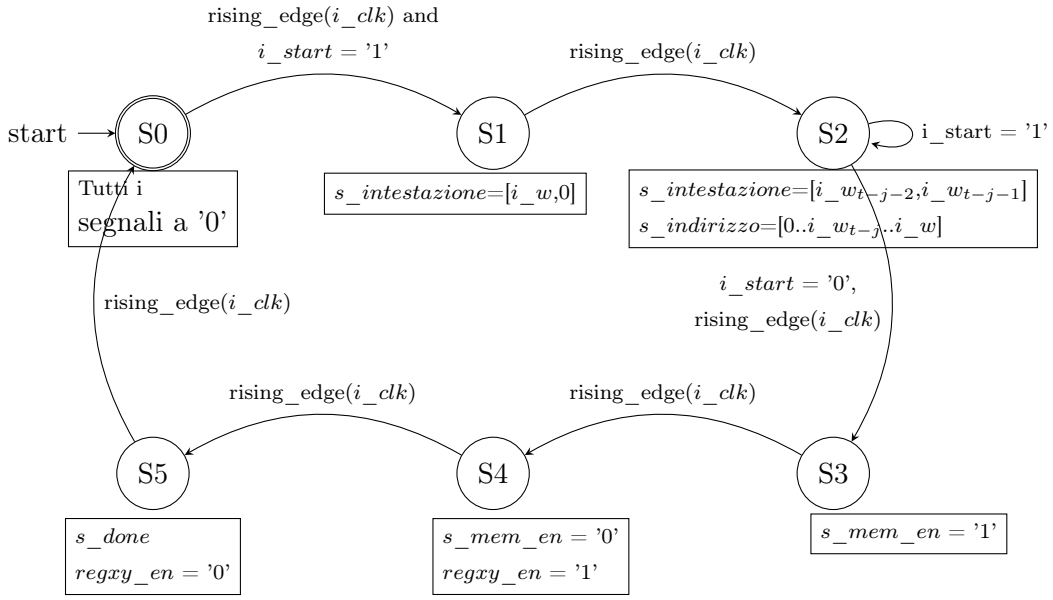


Figura 2: Diagramma degli stati sintetico.

Il primo modulo del componente è quello che legge gli ingressi, rimanendo nello stato iniziale dal momento in cui il componente riceve il reset. Quando START diventa '1' la FSM legge il valore di w e lo porta sull'uscita, in particolare nel bit più significativo del bus (parallelo a due bit) dell'intestazione (segnale s_intestazione). A questo punto la macchina passa nello stato S1, nel quale legge il valore di W per scriverlo come bit meno significativo dei due bit dell'intestazione. Si ricorda che il segnale di START è garantito rimanere alto per almeno due cicli di clock. Il prossimo stato è S2, stato durante cui la FSM legge tutti gli eventuali bit di indirizzo (infatti potrebbero anche essere zero) e aggiorna, ad ogni ciclo di clock in cui START è '1', il valore dell'uscita s_indirizzo. Al ciclo di clock successivo alla transizione da alto a basso di START la FSM passa allo stato S3, durante il quale abilita la memoria in lettura ponendo MEMEN a '1' e la sollecita aggiornando l'uscita ADDRESS con il valore determinato allo stato S2. Al prossimo ciclo di clock, siccome è garantito un tempo di risposta della memoria non superiore

a un ciclo di clock, la FSM passa allo stato S4. Nello stato S4 il dato che arriva dalla memoria al componente tramite il segnale DATA viene inoltrato al Demultiplexer, che oltre a questo riceve un secondo ingresso: l'intestazione (determinata dalla FSM nello stato S1). Contestualmente a ciò vengono valutate espressioni booleane che permettono di impostare al valore logico alto il segnale di enable del solo registro corrispondende all'ucita Z corretta così da abilitarne l'aggiornamento dello stato e re-impostato a '0' il valore dell'uscita MEMEN. Al prossimo fronte di salita del clock la FSM passa nello stato S5: in questo stato viene impostato a '1' il segnale s_done che, come detto, è collegato all'uscita DONE e ai quattro multiplexer. Il prossimo stato della FSM è di nuovo lo stato di reset: S0.

3.2 Demultiplexer

Il demultiplexer riceve in ingresso un bus parallelo a 8 bit sul quale viene trasportato il segnale DATA ricevuto dalla memoria e il segnale s_intestazione (parallelo a 2 bit). Il suo funzionamento prevede che in modo asincrono al cambiamento dell'intestazione cambi il canale, dei quattro in uscita, che vedrà il proprio valore passare da "00000000" a quello sull'ingresso del demultiplexer. Tutti i tre segnali non corrispondenti all'uscita indicata da s_intestazione vengono posti al valore logico basso ("00000000").

3.3 Registri

Ciascuno dei quattro registri è costituito da 8 flip-flop di tipo D con reset asincrono e enable sincrono. Pertanto in caso dell'arrivo di un eventuale segnale di reset (lo stesso in ingreso al componente, cioè RESET) lo stato del registro passa forzatamente a "00000000". D'altro canto, qualsiasi cambiamento avvenga sull'ingresso principale del registro non avrà effetto sullo stato finché non si presenterà un fronte di salita del clock (CLOCK) in un momento in cui il segnale di enable (regxy_en) di quel registro è al valore logico alto ('1'). L'uscita q_regxy del registro è connessa, come anticipato, a un multiplexer.

3.4 Multiplexer

Ciascuno dei quattro multiplexer agisce da filtro sull'uscita corrispondente: si occupa cioè di trasmettere la costante "00000000" oppure lo stato corrente del registro associato a seconda che, rispettivamente, il selettore (s_done) sia pari a '0' o a '1'.

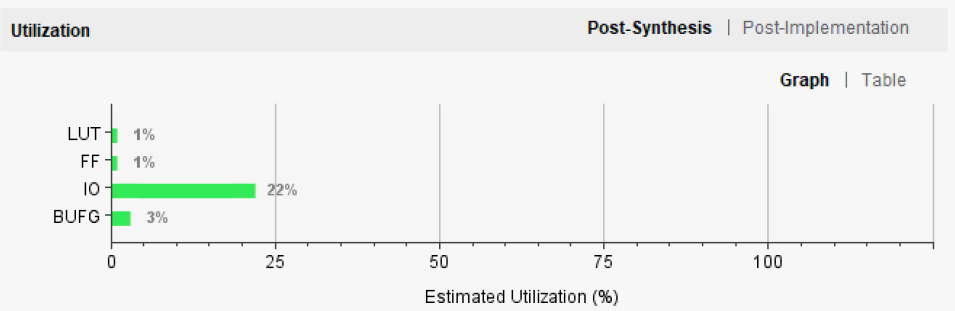
4 Risultati sperimentali

4.1 Risultati della sintesi

Di seguito il report di sintesi di Xilinx Vivado e la stima sull'utilizzo percentuale del componente. La descrizione hardware è stata implementata su una scheda Artix-7 xc7a200tfbg484-1.

Report	Report Type	Options	Size
Synthesis			
Synth Design (synth_design)			
synth_1_synth_report_utilization_0	Report on utilization of resources on the targeted device (report_utilization)		7.0 KB
synth_1_synth_synthesis_report_0	Vivado Synthesis Report		21.8 KB

(a) Report post sintesi



(b) Stima dell'utilizzo post sintesi (grafico)

Utilization — Post-Synthesis | Post-Implementation

Graph | Table

Resource	Estimation	Available	Utilization %
LUT	48	134600	0.04
FF	59	269200	0.02
IO	63	285	22.11
BUFG	1	32	3.13

(c) Stima dell'utilizzo post sintesi (tabella)

Figura 3: Report e Utilizzo post sintesi

Dal grafico della stima post sintesi sull'utilizzo percentuale dei componenti della scheda il nostro progetto occuperà l'1% delle Look Up Table(s) (LUT), ma osservando la tabella vediamo che in realtà ne userà solo lo 0.04%, pertanto si deduce che probabilmente il grafico mostra le percentuali con un

limite inferiore all'1% probabilmente per motivi estetici. Un discorso analogo vale per i Flip Flop (FF) dove la percentuale effettivamente stimata è dello 0.02%. Per le interfacce di Input e Output (IO nella tabella) si ha invece un utilizzo stimato pari al 22% circa, mentre per gli elementi ad alto fanout utilizzati per segnali come clock e reset (BUFG) si ha un utilizzo del 3%.

4.2 Test bench forniti dal docente

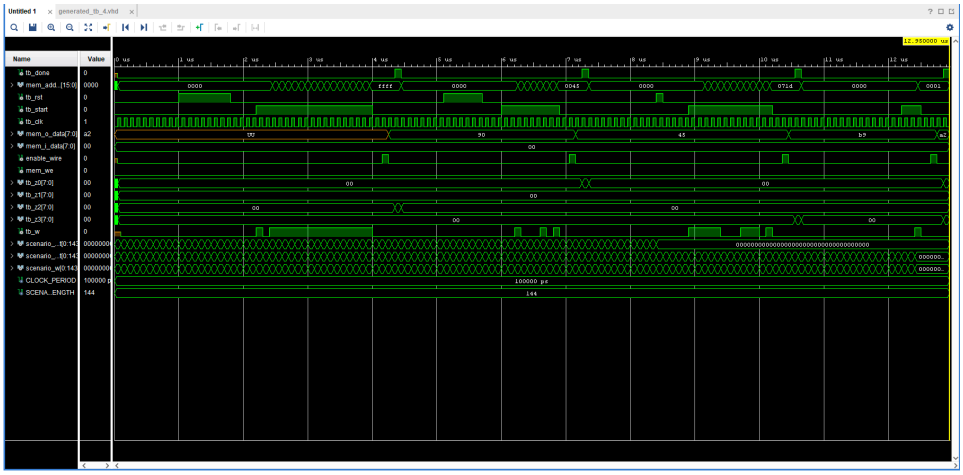


Figura 4: Un esempio di *waveform* di test bench

Di seguito, per ognuno dei sette test bench forniti dal docente, si riportano i soli casi testati da quello specifico test bench e non dai precedenti (nell'elenco), ma è ovviamente possibile che lo stesso caso sia testato da pià test bench.

1. **Test bench 1:** Questo test bench verifica il corretto funzionamento dell'unità testata, cioè il nostro progetto, nei seguenti casi:

    - Il segnale di START rimane al valore logico alto solo per due cicli di clock, pertanto ci si aspetta che l'indirizzo rispetto al quale la memoria sarà interrogata sia "0000000000000000". [caso limite]
    - Il dato letto dalla memoria deve essere messo almeno una volta su ognuna delle uscite Z*, pertanto in questo caso si sta testando il corretto funzionamento del componente su tutte le uscite e che esso non abbia un comportamento disomogeneo. [caso limite]

- Il segnale di START rimane al valore logico alto per più di due cicli di clock. Si tratta di testare il corretto funzionamento del componente quando riceve input "normali". [affidabilità]
- La stessa uscita viene usata più di una volta per presentare il dato letto da memoria, pertanto ci si aspetta che essa presenti il dato corretto sia la prima volta sia, successivamente alla lettura del dato, tutte le volte che DONE è pari a '1'. In altre parole, facendo riferimento alla soluzione qui adottata, si testa il corretto funzionamento dei registri nell'aggiornare il loro stato tutte e sole le volte che è necessario. [affidabilità]
- Tutte le quattro uscite Z* sono poste al valore logico basso quando il segnale di DONE è pari a '0'. [affidabilità]
- Tutte le uscite conservano memoria dei dati precedentemente presentati (a livello pratico questo controllo è incluso nei precedenti). [affidabilità]
- Il segnale di RESET passa dal valore logico basso a quello alto più volte durante il test, pertanto ci si aspetta che dopo ogni volta che dal valore logico alto è ritornato a quello basso i valori sulle uscite siano tutti "00..0" e le quattro uscite Z* abbiano "perso" la memoria. [affidabilità]
- Pssano esattamente 20 cicli di clock da quando il segnale di START è tornato al valore logico basso a quando torna a quello alto: siccome il componente da specifica deve rispondere entro 20 cicli di clock ci si aspetta che questo non sia un problema. [caso limite]
- Indirizzi di memoria interrogati: 0, 3, 6, 985, 3059.

2. **Test bench 2:** Questo test bench verifica il corretto funzionamento dell'unità testata, cioè il nostro progetto, nei seguenti casi:

   - Il segnale di RESET rimane al valore logico alto solo per un ciclo di clock. Ci si aspetta che questo non influisca sul corretto funzionamento del componente. [caso limite]
   - Il segnale di RESET viene posto al valore logico alto solamente prima dell'utilizzo del componente, in corrispondenza di quella che potrebbe essere pensata come la "accensione" del componente. Ci si aspetta quindi che le quattro uscite Z* mantengano sempre memoria dei dati precedentemente presentati. [caso limite]
   - Il segnale di START rimane alto per 18 cicli di clock. In altre parole si sta testando il corretto funzionamento del componente

nel caso in cui l'indirizzo a cui richiedere il dato sia lungo 16 bit, cioè il massimo possibile. [caso limite]

- L'indirizzo rispetto al quale interrogare la memoria è l'ultimo indirizzo disponibile: "1111111111111111". [caso limite]
- Indirizzi di memoria interrogati: 721, 65535, 1821, 1312, 69.

3. **Test bench 3:** Questo test bench verifica il corretto funzionamento dell'unità testata, cioè il nostro progetto, nei seguenti casi:
   - Indirizzi di memoria interrogati: 1, 57, 214.
4. **Test bench 4:** Questo test bench verifica il corretto funzionamento dell'unità testata, cioè il nostro progetto, nei seguenti casi:
   - Indirizzi di memoria interrogati: 420, 7765.
5. **Test bench 5:** Questo test bench verifica il corretto funzionamento dell'unità testata, cioè il nostro progetto, nei seguenti casi:
   - Lo scenario di questo test è particolarmente lungo.
   - Si vedano Test bench precedenti.
6. **Test bench 6:** Questo test bench verifica il corretto funzionamento dell'unità testata, cioè il nostro progetto, nei seguenti casi:
   - Si vedano Test bench precedenti.
7. **Test bench 7:** Questo test bench verifica il corretto funzionamento dell'unità testata, cioè il nostro progetto, nei seguenti casi:
   - Si vedano Test bench precedenti.

4.3 Test bench scritti in fase di progettazione

Si può notare come i test forniti dal docente non mettano alla prova il componente sotto alcuni fronti, in particolar modo quello della robustezza. Di seguito vengono elencati quindi gli aspetti che io stesso ho testao, scrivendo test bench ad-hoc.

1. **Test bench 1:** I segnali di START e RESET passano dal livello logico basso a quello alto nello stesso momento (comportamento inatteso). Ci si aspetta che prevalga il reset, forzando il ripristino del componente. [robustezza]

2. **Test becnh 2:** Segnale W che continua a oscillare (cambiare valore '0' <–> '1') anche dopo che il segnale di START è ritornato a '0'. Ci si aspetta che i valori di W dopo lo START siano ignorati [robustezza]

3. **Test bench 3:** Segnale W che inizia a oscillare prima che il segnale di START passi al valore logico alto. Ci si aspetta che i valori di W prima dello START siano ignorati. [robustezza]

4. **Test bench 4:** Segnale di START che passa al valore logico alto mentre il segnale di RESET transisce dal valore logico alto a quello basso. Ci si aspetta che non cambi nulla siccome il componente deve essere reattivo all'innalzamento dell'onda del segnale di START. [robustezza]

5. **Test bench 5:** Il segnale DONE deve rimanere al valore logico alto per esattamente un ciclo di clock. [affidabilità]

5 Conclusioni

In conclusione si può dire che è altamente probabile che il componente progettato rispetti tutti i requisiti di fnuzionamento e che sia anche robusto di fronte a un uso improprio o impreciso. Bisogna osservare, tuttavia, che testare la totale affidabilità del progetto nel suo dominio di input richiederebbe di testare un numero infinito di valori. Ciò non è dovuto alla finitezza della dimensione dell'alfabeto $\Sigma = \{0, 1\}$ da cui estrarre i valori con cui comporre gli input, tantomeno dalla finitezza della dimensione della memoria; bensì dalla non finitezza del limite superiore alla dimensione delle sequenze di input al componente. Si ritiene comunque di aver testato il componente nei casi limite in modo da far sì che qualsiasi altro funzionamento possa essere assunto corretto di conseguenza. Si vuole inoltre osservare che non è stao eseguito nessun test d'unità sui vari *component* interni in quanto abbastanza elementari e strettamente coinvolti nel funzionamento del componente principale, tanto da essere testati in modo indiretto attraverso il testing di quest'ultimo.