(Progetto di Reti Logiche)

Prova Finale

Prof Palermo Gianluca

Anno 2022-2023

Ram Jaskaran (matricola 959645 – C.P. 10735884)
Reitani Lorenzo (matricola 958460 – C.P. 10734888)

Indice

1 Introduzione al progetto

1.1 Scopo dell'hardware

Il progetto da noi sostenuto consiste nel descrivere un componente hardware, in stile behavioural in VHDL attraverso il software Vivado, in grado di leggere dati da un dispositivo di memoria esterno e mostrare in uscita, quando richiesto, il valore letto su una porta specifica tra le 4 disponibili, mentre le altre 3 porte continueranno a mostrare in uscita il loro ultimo valore trasmesso. La procedura di mettere in output dei valori dal nostro componente viene scandita con l'assunzione del valore 1 dell'uscita o_done. L'indirizzo di memoria dove prendere le informazioni e la rispettiva porta su cui trasmettere l'output vengono stabiliti da un segnale seriale ricevuto in input (i_w), il quale verrà processato e interpretato secondo le specifiche all'interno del componente ciclo per ciclo.

1.2 Interfaccia del componente

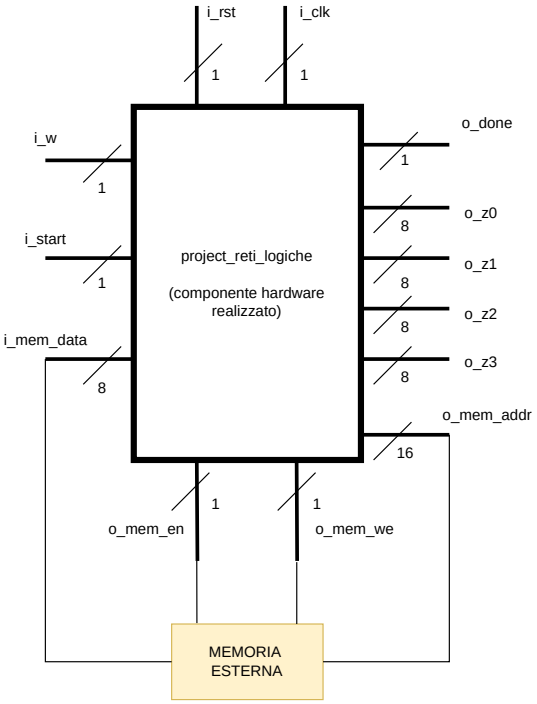
Il componente da descrivere deve avere la seguente interfaccia:

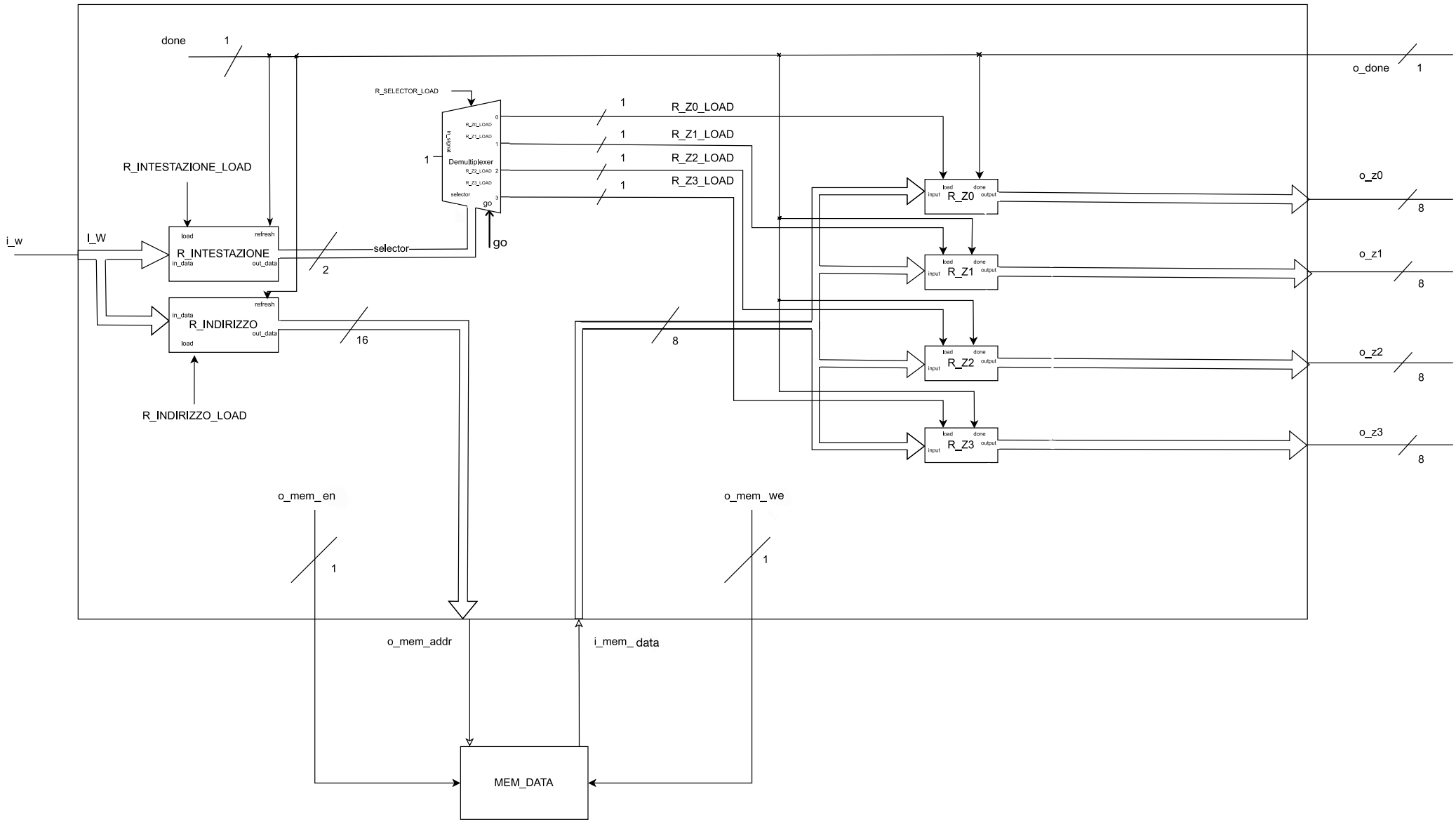
```
entity project_reti_logiche is
    port (
      i_clk   : in std_logic;
      i_rst   : in std_logic;
      i_start : in std_logic;
      i_w     : in std_logic;

      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;
```




done
1
o_done
1
R_SELECTOR_LOAD
Demultiplexer
in_signal
selector
go
0
1
2
3
R_Z0_LOAD
R_Z1_LOAD
R_Z2_LOAD
R_Z3_LOAD
1
1
1
1
R_INTESTAZIONE_LOAD
i_w
I_W
load
refresh
in_data
out_data
R_INTESTAZIONE
selector
2
go
R_INDIRIZZO
16
8
R_INDIRIZZO_LOAD
R_Z0
R_Z1
R_Z2
R_Z3
input
done
output
o_z0
o_z1
o_z2
o_z3
8
o_mem_en
o_mem_ we
1
1
o_mem_addr
i_mem_ data
MEM_DATA

2 Architettura

Ogni componente all'interno del nostro circuito prende in ingresso i segnali di clock e di reset per far si che lavorino tutti sincronizzati sul fronte di salita del clock mentre in caso di reset alto tutti i segnali e i registri interni vengono inizializzati a '0' a livello hardware, a livello logico invece la macchina a stati torna allo stato S0. Questi due segnali, assieme al segnale 'i_start', nell'immagine precedente non appaiono per rendere il circuito più leggibile; combinati tra loro, scandiscono le transizioni che la macchina a stati del componente esegue dallo stato attuale allo stato successivo.

2.1 Register

L'entity Register è un componente che è in grado di leggere e memorizzare un segnale di 8 bit e mostrare in output ciò che ha salvato se il segnale di done è '1' altrimenti mostra '00000000' (8 bit in parallelo).

Il circuito ne implementa 4: r_z0, r_z1, r_z2, r_z3, ognuno associato ad una singola porta d'uscita, rispettivamente o_z0, o_z1, o_z2, o_z3. Ricevono in input il valore dalla memoria, ma lo salvano solo quando il rispettivo segnale di load è '1'. Il segnale di load viene attivato tramite un apposito demultiplexer che permette di selezionare il registro d'interesse.

Sono necessari perché quando done è alto il registro che ha salvato il nuovo valore mostrerà quest'ultimo, mentre gli altri mostreranno l'ultimo valore trasmesso da ciascuno.

2.2 ShiftRegister

L'entity ShiftRegister è un componente con attributo variabile N che sta ad indicare la dimensione della memoria interna. E' in grado di leggere un segnale seriale da 1 bit e un ciclo per volta, li salva tutti per poi dare in uscita i primi N-1 bit, ovvero l'ultimo bit letto non viene mostrato in uscita.

Quando il segnale di load è '1' legge il segnale in ingresso (1 bit), shifta i bit salvati in memoria di una posizione verso sinistra (il bit con peso maggiore viene perso, in posizione N-1) e viene aggiunto da destra (in posizione 0) il bit letto dall'ingresso.

Questo componente legge in ingresso anche il segnale di 'refresh' che quando è alto resetta il contenuto del registro a tutti '0'. Abbiamo aggiunto questa specifica perché ci risultava necessario re-inizializzare il contenuto del registro ogni qualvolta finissimo una sequenza di lettura (che inizia con start e termina con done alto), ma che al tempo stesso fosse distinto dal segnale di reset, che invece scaturisce un reset di tutti i componenti.

Nel circuito ce ne sono implementati 2: r_intestazione e r_indirizzo.

- Il primo con N=3 si salva nel registro e mette in uscita un 'intestazione' di 2 bit (codifica binaria) che indica quale delle 4 uscite deve essere utilizzata per mostrare l'output.
- Il secondo con N=17 salva e dà in uscita l'indirizzo di memoria di 16 bit da dare in input alla memoria esterna per poi ottenere il contenuto di quell'indirizzo.

Abbiamo optato di realizzare un solo componente che svolgesse la funzione di prendere in input e shiftare un segnale seriale da 1 bit per questione di riusabilità del componente. La motivazione per cui il registro è dotato di un bit in più rispetto all'uscita è perché il nostro componente deve leggere l'indirizzo di memoria di 16 bit (il quale necessita di essere sempre di questa di dimensione, in caso contrario deve essere 'paddato' con valori '0'), ma siccome non conosciamo a priori le cifre di indirizzo che riceveremo in input, quando il segnale di lettura si interrompe il registro legge lo stesso un ulteriore bit, il quale semplicemente viene scartato e verranno dati in uscita gli altri 16 bit senza alcuna perdita di valori dovuti allo shift dei bit.

2.3 Selector

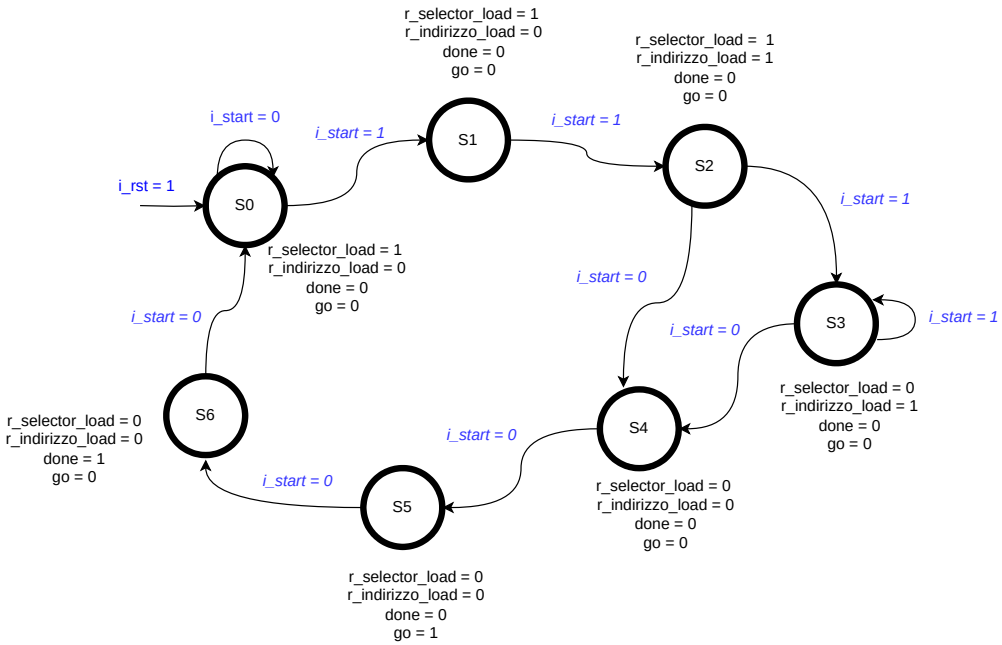
L'entity Selector è un demultiplexer che riceve in ingresso il segnale in_signal, sempre ad '1', e il segnale input da 2 bit che seleziona su quale uscita far uscire in_signal. Il segnale di ingresso 'go' quando alto abilita l'uscita selezionata.

Nel nostro circuito ne abbiamo implementato 1 (demultiplexer), che serve appunto ad attivare il load di uno dei 4 registri (r_z0,r_z1,r_z2,r_z3) associato all'uscita richiesta.

Il segnale 'go' è necessario per evitare che dopo la re-inizializzazione di r_selector, ossia quando riceve in ingresso '00' ma non corrisponde alla vera intestazione richiesta (r_selector), venga abilitata erroneamente l'uscita r_z0_load (mappata sulla sequenza '00).
'go' va ad 1 nello stato di macchina in cui c'è la certezza di avere l'intestazione corretta.

2.4 Macchina a stati

La macchina a stati descrive il procedimento che segue il circuito per il completo e corretto funzionamento. In tutto lo svolgimento o_done <= done (era necessario un segnale interno che assumesse lo stesso valore dell'uscita o_done), o_mem_we <= '0' (non è mai necessario scrivere in memoria ma solo leggere) e o_mem_en <= '1' (viene posto ad 1 per comodità perché tanto tutti gli altri componenti sono controllati affinché leggano e scrivano negli stati di macchina corretti) ;
i_start è il segnale in input che definisce le transizioni della macchina a stati. i_rst agisce in maniera asincrona.



Nei 7 stati accade:

-S0:

Il circuito rimane in questo stato fino a quando start diventa '1'. Il load dello shift register abilita la lettura solo al ciclo di clock successivo all'accensione e la disabilita al ciclo successivo allo spegnimento. Pertanto, r_selector_load rimane a '1' in modo da essere pronto per il passaggio allo stato successivo. Anche se leggesse dei valori durante questo periodo, non ci sarebbero problemi poiché verrebbero automaticamente shiftati e scartati. r_indirizzo_load, done e go sono tutti impostati su '0'. Quando start diventa '1', si passa allo stato S1. Il segnale reset asincrono, quando è alto, porta sempre la macchina in questo stato.

-S1:

Arriva in ingresso da i_w il primo bit da leggere e salvare in r_intestazione. Tutti i segnali rimangono come nello stato precedente. Al successivo ciclo di clock è assicurato che start sia ‘1’ e si passa allo stato S2.

-S2:

Arriva in ingresso da i_w il secondo bit da leggere e salvare in r_intestazione. r_indirizzo_load viene impostato su '1' per fare in modo che, nello stato successivo, r_indirizzo sia pronto a leggere. Se start è '1' nel ciclo di clock successivo, si passa allo stato S3; altrimenti, allo stato S4.

-S3:

Quando si arriva in questo stato per la prima volta, viene spento r_selector_load ma viene comunque letto un ultimo bit che fa shiftare i due bit di intestazione precedentemente letti nelle posizioni 2 e 1 che sono proprio quelli che vengono inviati al demultiplexer per attivare il registro associato all'uscita corretta. r_indirizzo_load rimane a '1' per continuare a leggere da i_w e salvare l'indirizzo di memoria corretto. Se start è '1' nel ciclo di clock successivo, si rimane in questo stato; altrimenti, si passa allo stato S4

-S4:

In questo stato, per lo stesso ragionamento dello stato precedente r_indirizzo_load viene impostato a ‘0’ ma in output vengono comunque inviati i bit corretti. Se si arriva da S2 e non da S3, avviene qui lo shift aggiuntivo del r_selector.
Il segnale go viene messo ad ‘1’ per attivare il demultiplexer e attivare il load del registro giusto per far si che riceva in ingresso i dati presi dalla memoria. Attivandolo solo ora, si evita che vengano sovrascritti gli altri registri per errore. Al successivo ciclo di clock è garantito che start sia ‘0’ e si passo allo stato S5.

-S5:

Questo stato ha tutti i segnali uguali al precedente e serve a prelevare dalla memoria i dati e salvarli nel registro giusto. Al successivo ciclo di clock è assicurato che start sia ‘0’ e si passo allo stato S6.

-S6

Il segnale done viene messo ad ‘1’ per mandare in uscita ciò che è salvato in memoria all’interno dei registri. Il done ad 1 fa il refresh dei shift-register azzerandoli.
Il segnale go passa a ‘0’ per evitare ti trascrivere erroneamente la memoria del di r_Z0.
Al successivo ciclo di clock è assicurato che start sia ‘0’ e si passo allo stato S0.

3 Risultati sperimentali

3.1 Sintesi

Il nostro componente è sintetizzabile e correttamente simulabile in post-synthesis.
Dal report della sintesi mostra che non ci sono latch a conferma del fatto che la gestione della memoria è correttamente gestita solo e soltanto dai componenti che abbiamo descritto.

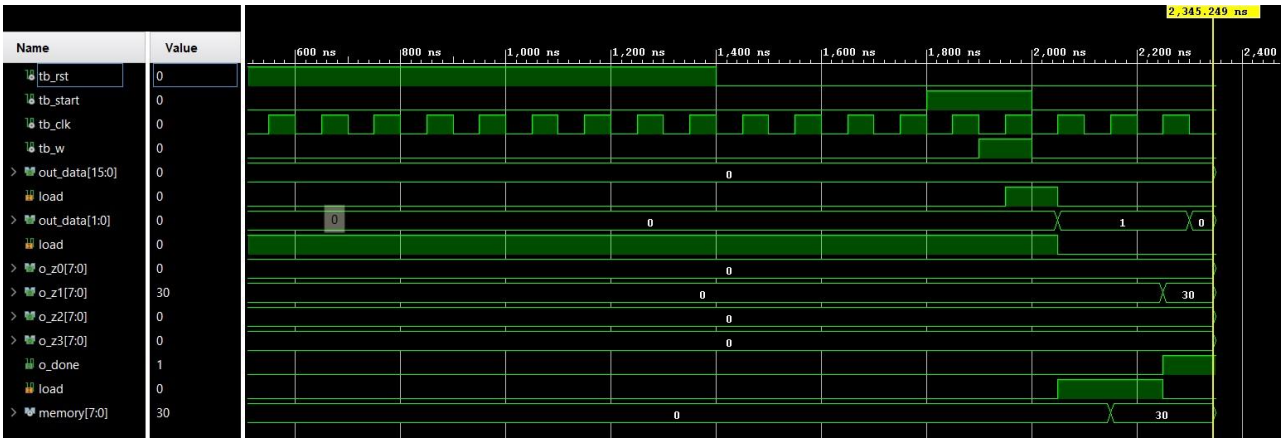
Site Type	Used	Fixed	Available	Util%
Slice LUTs*	23	0	134600	0.02
LUT as Logic	23	0	134600	0.02
LUT as Memory	0	0	46200	0.00
Slice Registers	55	0	269200	0.02
Register as Flip Flop	55	0	269200	0.02
Register as Latch	0	0	269200	0.00
F7 Muxes	0	0	67300	0.00
F8 Muxes	0	0	33650	0.00

3.2 Simulazioni

Per verificarne il corretto funzionamento il circuito è stato sottoposto a varie simulazioni, sia behavioral che post-synthesis. Inizialmente abbiamo sfruttato i test bench forniti come esempio per verificare i casi più comuni e frequenti. In seguito ne abbiamo costruiti alcuni ad hoc per verificare dei casi limiti nel nostro circuito. Di seguito discuteremo alcuni dei casi più interessanti testati.

Indirizzo di memoria composto da soli 0

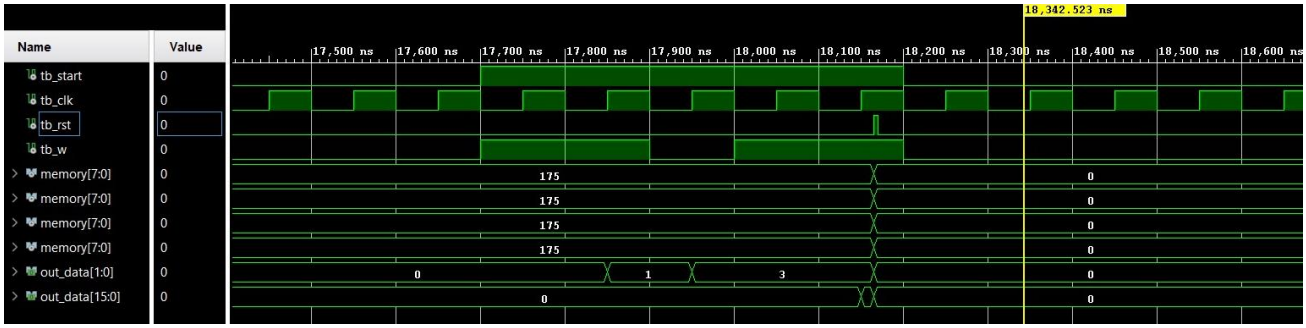
Nel primo test bench abbiamo voluto testare il caso limite in cui l'indirizzo sia composto da tutti '0' e quindi che lo start vale '1' solo per 2 cicli di clock. Questo è l'unico caso in cui la nostra macchina a stati passa dallo stato S2 allo stato S4 senza passare per lo stato S3.



Come possiamo vedere dall'immagine il circuito riesce a leggere correttamente l'intestazione, in questo caso 1, per poi trasmettere i dati letti dalla memoria sulla porta o_z1. Il primo load che appare è quello che attiva la lettura dell'indirizzo, è messo ad 1 per un ciclo di clock perché non sappiamo a prescindere quanti bit è lungo l'indirizzo di memoria ma deve essere comunque pronto a leggere. In questo caso è 0, quindi viene messo a '0' nel ciclo successivo leggendo comunque un valore ma l'output dell'indirizzo di memoria (out_data[15:0]) non cambia per come abbiamo costruito lo shift register.

Reset asincrono

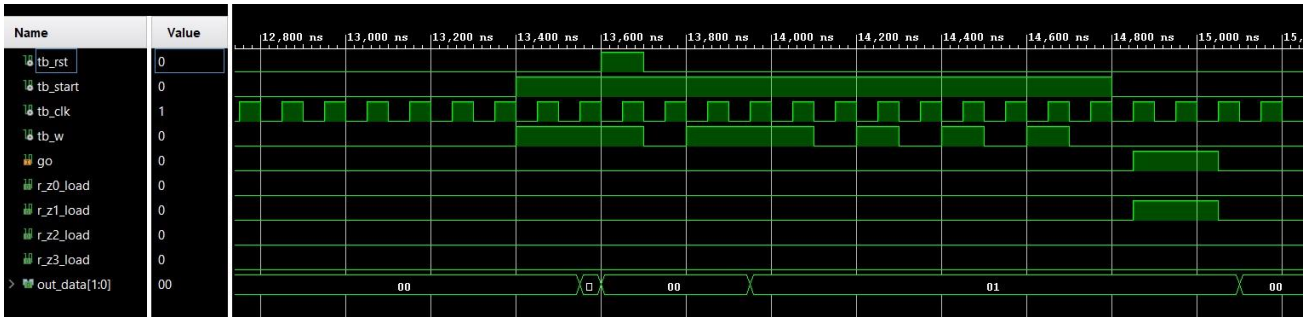
Il secondo test verifica la risposta del circuito a segnali asincroni, nello specifico ad un reset.



Prima del segnale di reset tutte le memorie dei registri contengono dei dati. Dopo l'arrivo del segnale di reset asincrono in concomitanza con il segnale start ad '1', tutti i componenti del circuito vengono re-inizializzati con tutti '0' confermando il corretto funzionamento del circuito.

Segnale go

In questo test bench abbiamo verificato il funzionamento del segnale go che attiva le uscite del demultiplexer per far si che i load dei 4 registri collegati alle uscite si attivino solo quando il circuito ha già salvato l'intestazione e indirizzo corretti per evitare che vengano modificati registri che non dovrebbero attivarsi con dati errati.



Si può notare che quando il segnale go viene messo ad '1' contemporaneamente anche r_z1_load diventa '1' per scrivere in memoria nel suo registro. Questo avviene solo dopo che il segnale di start diventa '0' per essere certi che tutte le informazioni corrette siano salvate nel circuito.

4 Conclusioni

In generale, siamo soddisfatti del nostro componente poiché siamo stati in grado di realizzarlo con un numero ridotto di stati di macchina e di cicli di clock per produrre il risultato richiesto dalla specifica. Tuttavia, desideriamo approfondire le possibilità di ottimizzazione del componente per renderlo ancora più veloce, pur essendo consapevoli che ciò potrebbe comportare un aumento della complessità logica e circuitali.

1. In particolare, una possibile ottimizzazione riguarda la realizzazione di uno Shift-Register a doc per l'intestazione, differenziandolo da quello utilizzato per l'indirizzo di memoria. Entrambi eseguono lo shift dei bit in ingresso per salvarli e successivamente darli in uscita, ma il registro di intestazione non presenta i problemi dell'indirizzo di memoria, che richiede il "padding" con '0' fino a raggiungere i 16 bit e la re-inizializzazione a tutti '0' dopo la terminazione di una sequenza di lettura. Questo perché sappiamo esattamente quanti cicli impieghiamo per la sua lettura (uno stato di macchina per ogni bit letto) e ciò permette di non dover esser inizializzato e tanto meno di dover fare una lettura extra di bit e successivamente scartarlo.
   È stato deciso di riciclare il componente per questioni di pulizia del codice e di scalabilità a livello hardware.

2. Un'altra possibile ottimizzazione riguarda la creazione di un componente unico con 4 registri interni e rispettivi segnali di caricamento che abilitano l'aggiornamento dei registri. Il valore letto dalla memoria viene inserito in ingresso, mentre le uscite sono regolate da un multiplexer che prende in ingresso l'output dei registri e l'output diretto della memoria. Il comportamento del componente prevede che, quando il segnale o_done va a 1, cioè il valore in memoria è stato letto, tutti i registri danno in uscita il valore che contengono, ad eccezione del registro relativo alla porta di interesse in quella sequenza di lettura. In questo caso, il registro mette direttamente in uscita l'output della memoria, piuttosto che il suo contenuto ancora in aggiornamento, grazie al segnale di caricamento che manipola sia il multiplexer che la scrittura del registro con il nuovo valore. Questa ottimizzazione consente di risparmiare un ciclo di clock sovrascrivendo il nuovo output nel registro di "backup" mentre viene mandato contemporaneamente in uscita.

3. L'ultima ottimizzazione possibile riguarda la gestione dei segnali load che abilitano la lettura e il salvataggio dell'input. Si potrebbe considerare la modifica di tali segnali in base al segnale asincrono *i_start* al fine di evitare la lettura di bit in eccesso, che sarebbero successivamente scartati. Questo comporterebbe l'abilitazione di tutti i load in anticipo, in modo che la memoria venga aggiornata solo al fronte di salita del clock. Tuttavia, in questa situazione è necessario gestire il caso particolare in cui il segnale start scende contemporaneamente al fronte di salita del clock. Nella nostra implementazione, questo caso viene sempre gestito in modo controllato e affidabile.