Politecnico di Torino

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA ELETTRONICA

SISTEMI DIGITALI INTEGRATI

Laboratorio 2

Fabio Busignani 197883
Giovanni Campanella 197206
Tommaso Clemente 195555

January 22, 2013

Indice

1 Introduzione

L' obbiettivo di questa seconda esercitazione di laboratorio è quello di progettare un' interfaccia che permetta ad una periferica master di effettuare operazioni di lettura e scrittura su di un bus al quale sono connesse 3 ulteriori periferiche:

- 16 switch
- 16 LED
- 1 memoria SRAM

Il Master dovrà leggere gli switch, scrivere il dato appena letto dagli switch in una locazione della SRAM, leggere il dato appena scritto e scriverlo sulla periferica LED.
Abbiamo realizzato 3 macchine a stati, una per ogni periferica, in grado di gestire i segnali di controllo e interfacciarsi col master. Lo standard di interfaccia che abbiamo utilizzato è il WISHBONE.
Per accedere alle periferiche ci si avvale del metodo *interface memory mapped register based* dove ciascun blocco del sistema, apparte il processore, deve essere visto come una sequenza di locazioni di memoria tipicamente adiacenti. Inoltre, tipicamente è buona norma far si che il blocco sia allineato alla sua stessa dimensione, ossia l'indirizzo di partenza è un multiplo della dimensione del blocco stesso. Indicando con *Base_Add* l'indirizzo di partenza del blocco e con *Block_Size* la sua dimensione, si ha che se vale l'alliniamento:

$$Base\_Add = N \cdot Block\_Size.$$

Considetando che il bus indirizzi è composto da 16 bit, ciò vuol dire che abbiamo a disposizione 2^{16} indirizzi per le nostre periferiche. Consideriamo inosltre che la quantità di informzione associata ad un singolo indirizzo sia anchessa di 16 bit. Allora sotto queste ipotesi le periferiche LED e SWITCH richiedono un solo indirizzo di locazione, mentre la memoria SRAM richiede di ben $256K$ indirizzi (dove K è da intendersi come 2^{10}). Visto che questo numero di indirizzi è superiore a quelli a nostra disposizione, dedichiamo dall'indirizzo 0 a $2^{16} - 3$ alla SRAM, l'indirizzo $2^{16} - 2$ alla periferica composta dagli array di LED e $2^{16} - 1$ a quella realizzata con gli switch.

2 Periferica array di switch

2.1 Datapath

Sulla development board DE2 sono presenti 16 switch, coonnessi come mostrato in (Fig. 1).

Se l'interruttore è aperto sul filo ci sarà un 1 logico, se l'interruttore è chiuso sul filo ci sarà uno 0 logico. Il datapath potrà essere fatto da un semplice buffer collegato alla rete di comunicazione. Poichè l'interruttore è un ingresso asincrono potenzialmente con un buffer potrei violare i tempi di setup e di hold e il sistema

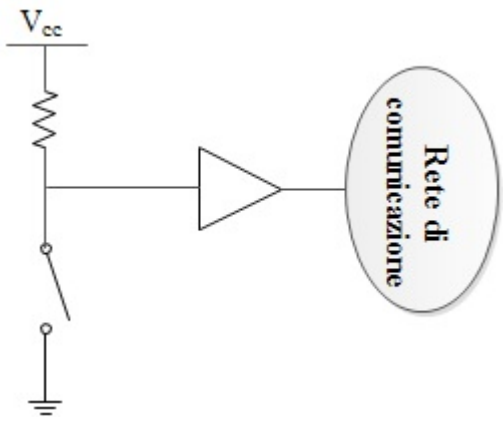

Figura 1: Connesione circuitale sulla scheda DE2

potrebbe entrare in uno stato metastabile. Di solito se ci sono ingressi asincroni invece di un semplice buffer si preferisce mettere un sincronizzatore, un flip flop, che risincronizza il segnale di ingresso col clock del sistema. In realtà anche il flip flop può entrare in meta stabilità dando in uscita delle commutazioni spurie. Ma se campiono il segnale con un flip flop perche il sistema vada in meta stabilità devono succedere due eventi congiunti. Il primo è che il flip flop vada in meta stabilità e il secondo è che questo tempo è cosi lungo da mandare in meta stabilità anche il sistema nel suo complesso. Poichè la probabilità è un numero minore di 1, la probabilità di due eventi congiunti è minore della probabilità del singolo evento. Quindi ho ridotto la probabilità di entrare in metastabilità. In realtà se non mi bastasse un flip flop ne potrei aggiungere degli altri, ma nel nostro caso la probabilità di entrare in uno stato metastabile è molto bassa quindi abbiamo deciso di mettere un solo flip flop. Quindi il datapath di questa prima periferica è quello mostrato in (Fig. 2).

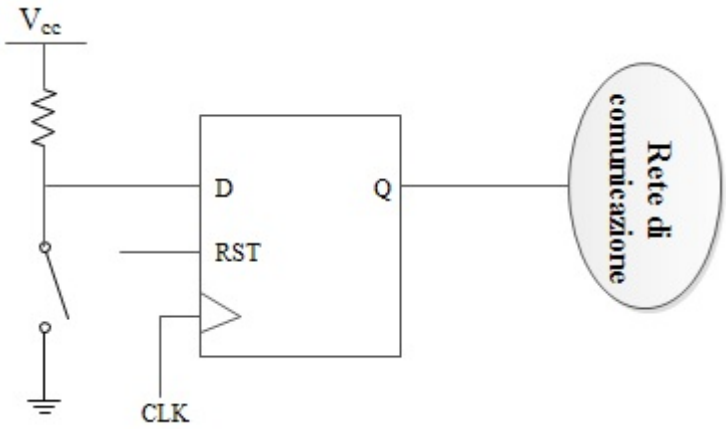

Figura 2: Datapath della periferica switch

2.2 Timing Diagram

In questa prima fase del progetto ci siamo preoccupati di realizzare una macchina a stati che si occupi, su richiesta del MASTER, di fornire il dato presente sugli switch e scriverlo sul data bus nell' istante di tempo opportuno.
Come prima cosa abbiamo realizzato il timing diagram, vedi (Fig. 3).

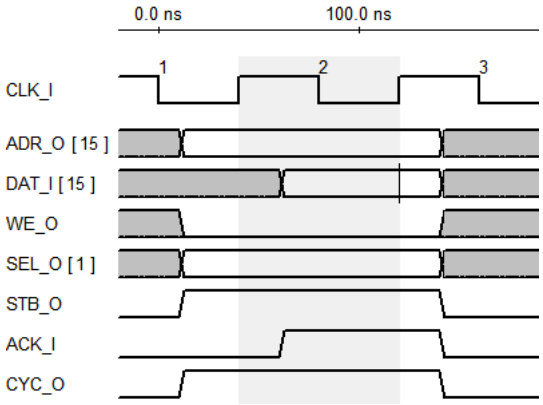

Figura 3: Timing diagram degli switch

2.3 Macchina a stati

Dal timing diagram descritto pocanzi si è ricavato il grafo degli stati rappresentato in (Fig. 4).

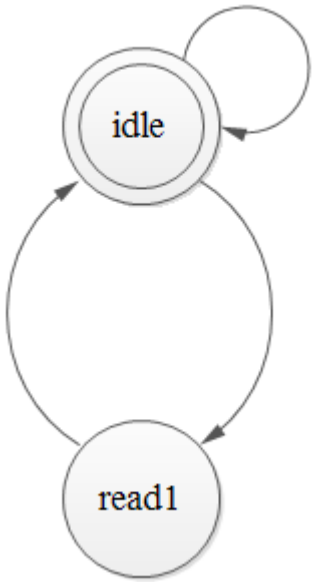

Figura 4: Diagramma degli stati degli switch

La nostra macchina possiede un reset asincrono collegato allo switch17, che è quello che viene collegato ai registri che compongono i datapath[1], e un reset sincrono

[1] Vedi (Fig. 2) e (Fig. 6)

gestito dal Master. Fintanto che il reset sincrono è uguale a 1 la macchina rimane nello stato di IDLE. Quando il Master vuole leggere il dato dagli switch presenta l'indirizzo degli switch su ADR_O,asserisce STB_O e CYC_O e nega WE_O. Se l'indirizzo è valido il prossimo stato sarà READ, lo Slave presenta i dati su DAT_I e asserisce ACK_I per indicare la validità dei dati. Al colpo di clock successivo il Master campionerà i dati, negherà STB_O e CYC_O mentre lo Slave negherà il segnale di ACK_I. Il segnale SEL_O mi dice se voglio leggere gli 8 bit più significativi del dato o gli 8 meno significativi. Se SEL_O= 11 sto leggendo i 16 bit.

2.4 Testbench

Una volta fatte tutte queste considerazione abbiamo scritto il codice VHDL e simulato questa prima parte del progetto utilizzando il software *ModelSim*. I risultati sono visibili in (Fig. 5).

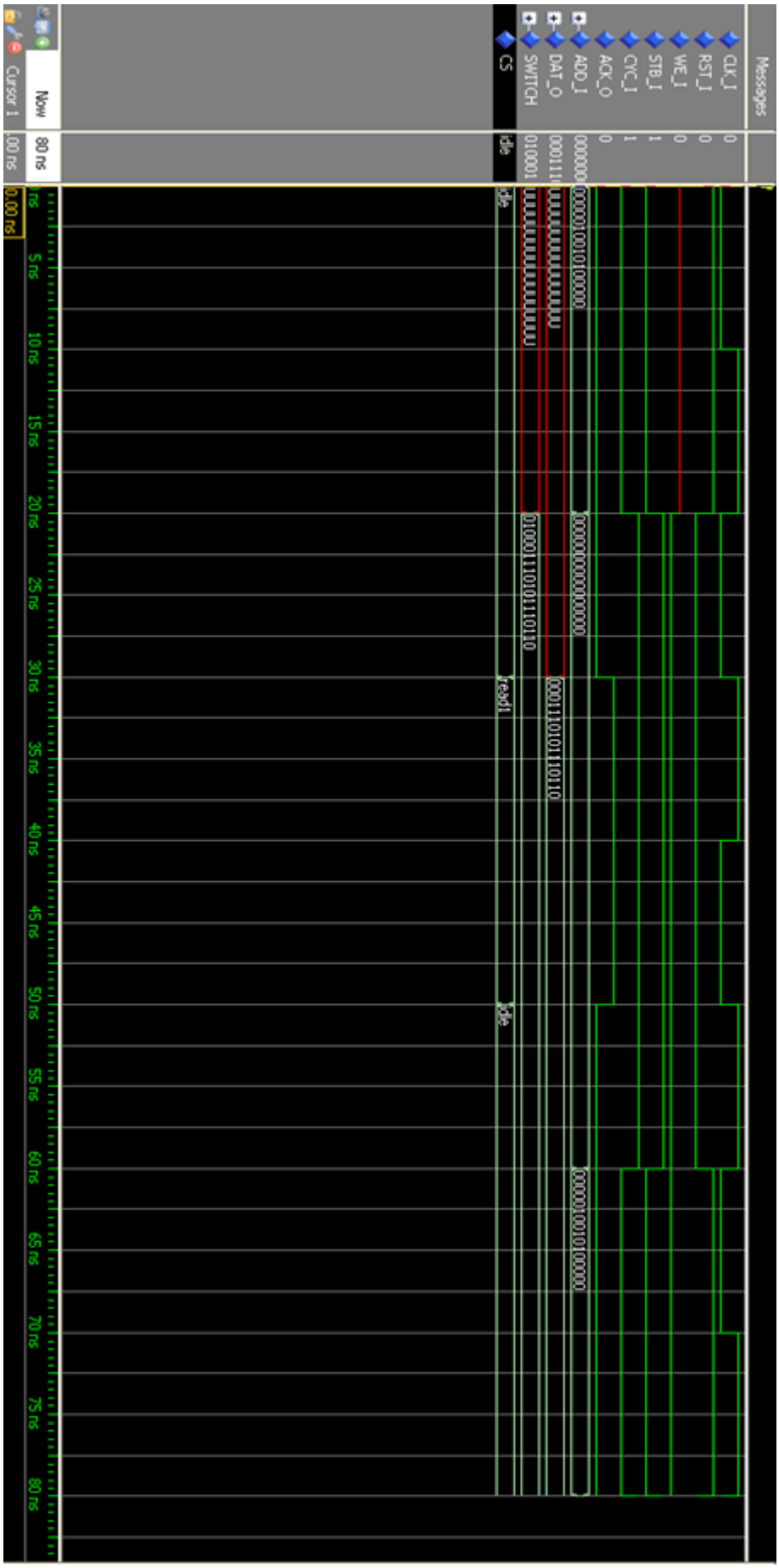

Figura 5: Simulazione con ModelSim degli switch

3 Periferica array di Led

3.1 Datapath

La macchina a stati dovrà essere progettata in modo tale che su richiesta del master possa accendere opportunamente i LED sulla base del dato ricevuto. Sulla DE2 ci sono 16 LED il cui schema elettrico è mostrato in (Fig. 6).

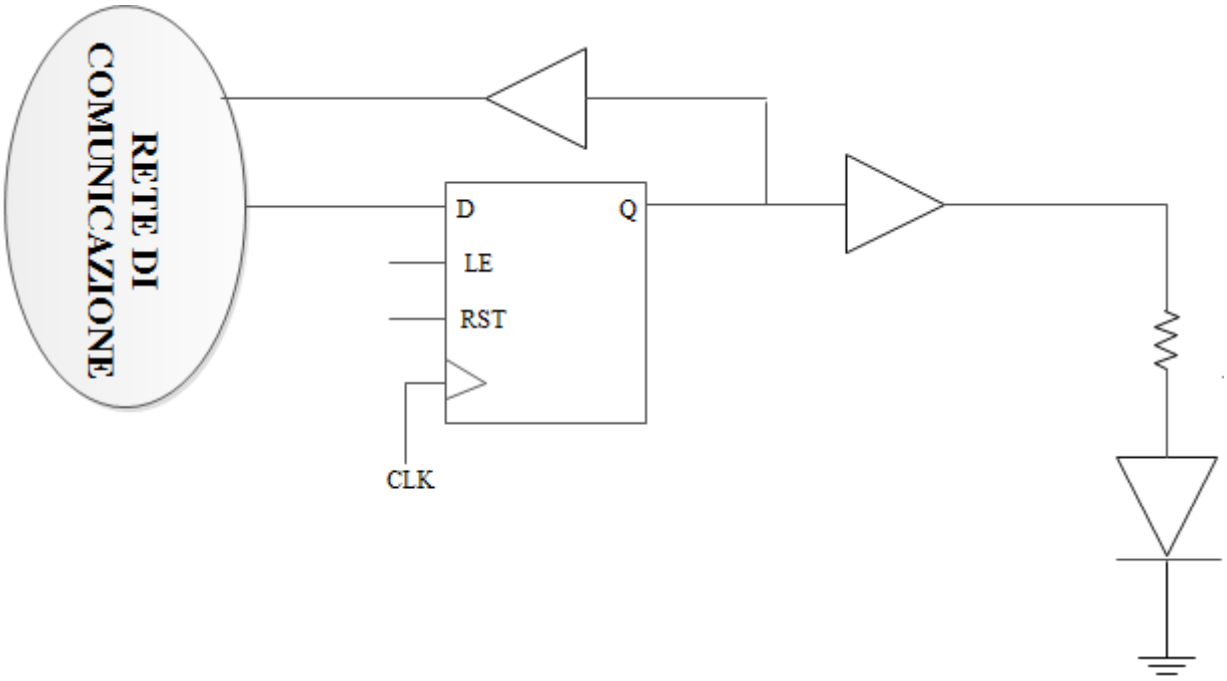

Figura 6: Datapath della periferica LED

I buffer si trovano sui piedini dell' FPGA. Nel datapath Inseriamo un registro in quanto quando finisco un ciclo di scrittura devo mantenere l'informazione precedente affinché il LED rimanga acceso. Lo stato del registro dovrà cambiare se e solo se il processore sta facendo una scrittura del registro, quindi il registro avrà un ingresso di Load Enable che dirà se caricare o meno il dato. Questo registro viene implementato con un MUX comandato dal segnale di LE che quando è a 0 fa passare l'uscita Q del flip flop a valle e quando è a 1 il nuovo dato da scrivere nel LED. Inoltre è stato previsto un ciclo di lettura dei LED qualora il master voglia verificare lo stato di quest'ultimi.

3.2 Timing Diagram

Come già avevamo fatto per gli switch anche per i LED abbiamo tracciato il timing diagram (Fig. 7) e da questo abbiamo ricavato il grafo degli stati.

3.3 Macchina a Stati

Anche per i LED valgono le stesse considerazioni sul Reset fatte per gli switch. Nel caso di un ciclo di scrittura il Master nel momento in cui vuole scrivere sui LED

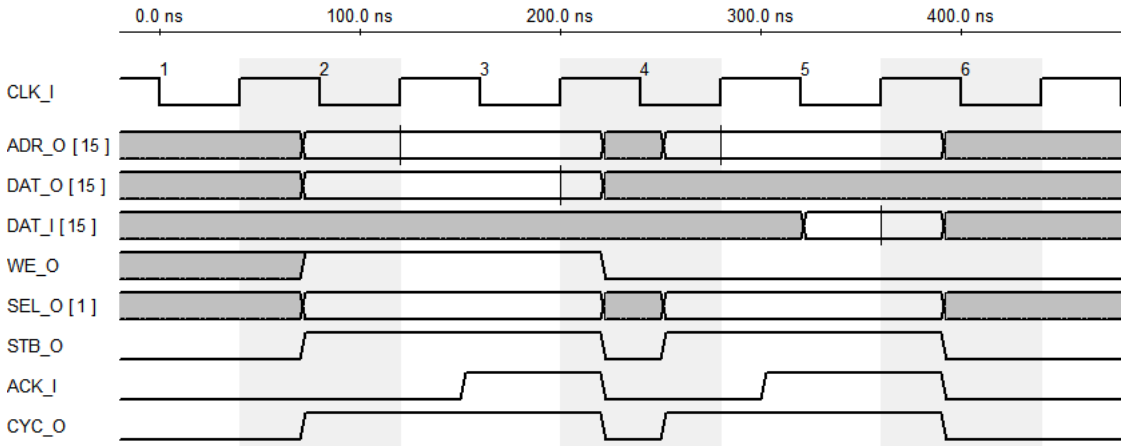

Figura 7: Timing diagram della periferica LED

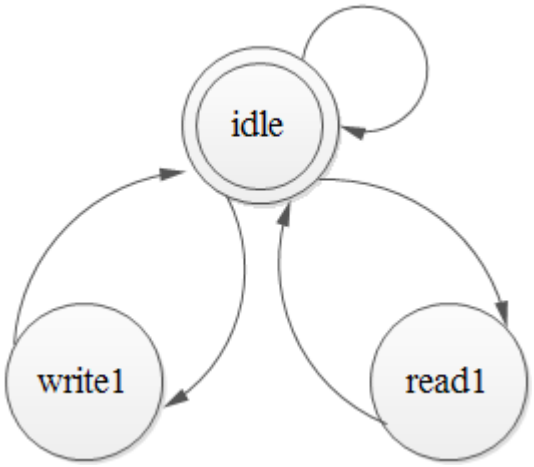

Figura 8: Diagramma degli stati della periferica LED

presenta un indirizzo valido,il dato che vuole scrivere e asserisce STB_O ,CYC_O e WE_O. Nello prossimo stato LOAD1, lo slave asserisce ACK_I e al prossimo fronte di salita del clock campiona il dato che deve essere scritto. Quindi dopo che il dato è stato campionato il master nega STB_O ,CYC_O e WE_O mentre lo slave nega ACK_I. Per quanto riguardo la gestione del segnale di selezione è stata creata una rete combinatoria con due porte logiche AND con ingressi i LE dei flip flop e i segnali di selezione SEL_O. La lettura dei LED è indipendente e avviene nello stesso modo della lettura degli switch.

3.4 Testbench

Dopo aver scritto il codice VHDL abbiamo simulato anche questa parte del progetto ottenendo i risultati illustrati in (Fig. 9).

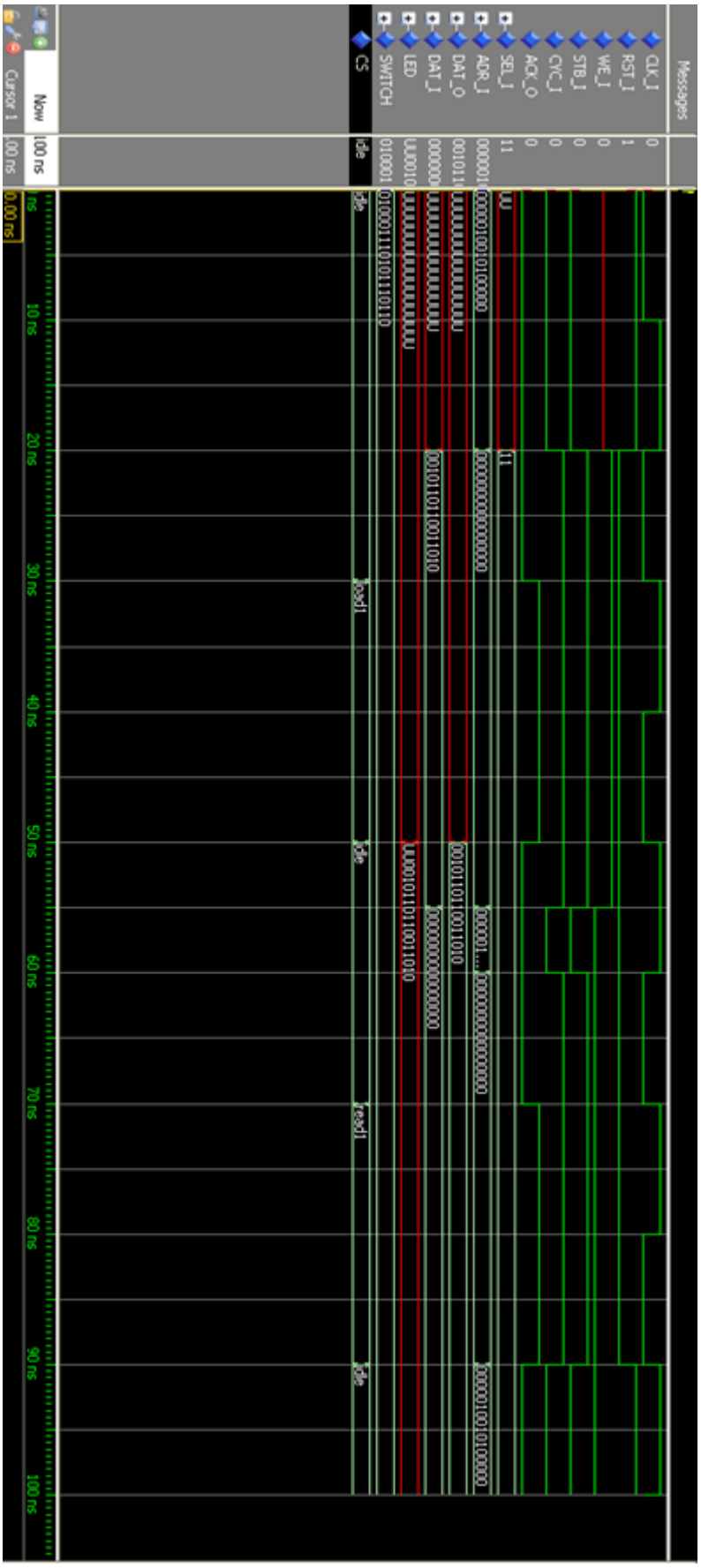

Figura 9: Simulazione con ModelSim dei LED

4 Periferica SRAM

Il terzo slave da interfacciare è la SRAM. Il sistema che viene descritto in questo punto è rappresentato in (Fig. 10).

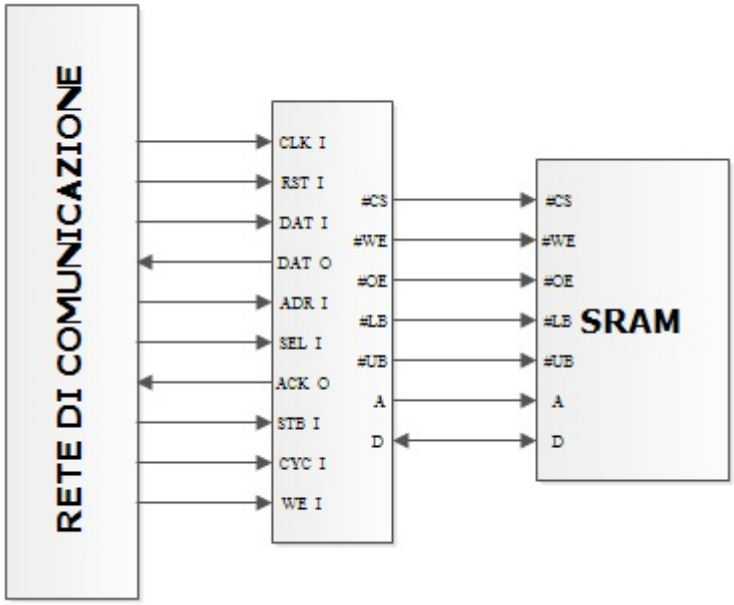

Figura 10: Schema a blocchi dell'interfaccia slace della SRAM

Quindi, quello che abbiamo dovuto realizzare in questa parte è il blocco posto tra la *SRAM* e la *rete di comunicazione* e prende il nome di **wrapper** o **adapter**. Questo *wrapper* svolge un'importante funzione, quella di convertire il protocollo di interfaccia. Ossia adatta il protocollo della periferica con quello dell'infrastruttura di comunicazione e viceversa. Quindi, deve obbligatoriamente saper interfacciarsi con entrambe.
Come è stato già detto, l'infrastruttura di comunicazione adotta come protocollo di interfaccia il *wishbone* che è un protocollo sincrono, mentre la SRAM utilizzata in questa esperienza (la *IS61LV25616AL*) è di tipo asincrona. Ciò significa che il wrapper da noi realizzato dovrà prevedere anche una desicronizzazione e una sincronizzazione (in funzione della direzione del flusso di dati).

4.1 Datapath

Come primo passo per la realizzazione del nostro wrapper si è dovuto disegnare il datapath, ossia l'insieme delle unità logiche necessarie a rispettare le specifiche imposte.

Come si può notare dalla (Fig. 11) è stato necessario utilizzare un buffer tristate per poter interfacciare i due canali dati utilizzati dal protocollo di interfaccia wishbone con il solo canale bidirezionale che va verso la RAM.
Il wishbone utilizza due bus differenti in funzione della direzione del flusso di dati e

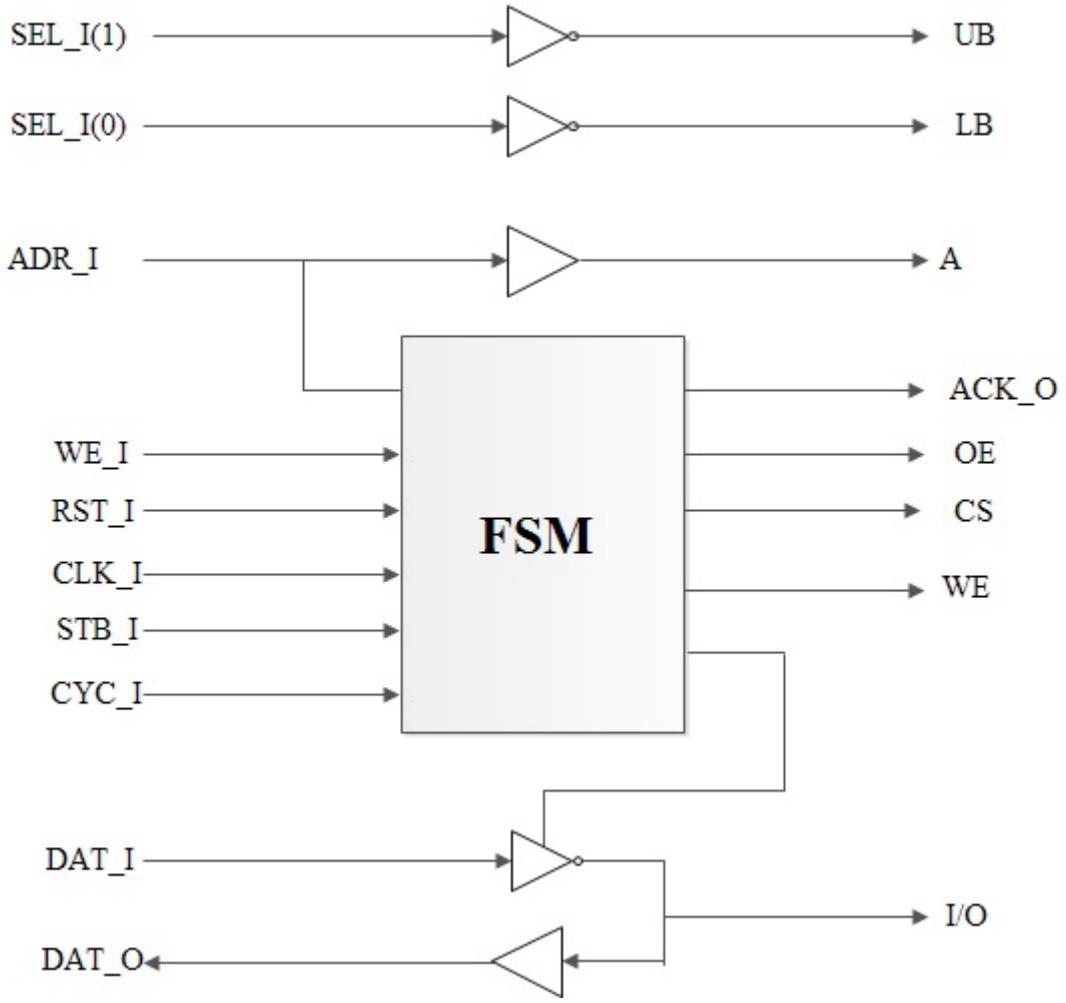

Figura 11: Datapath del wrapper per la SRAM

non un unico canale bidirezionale, perchè nelle sue specifiche non vengono previsti segnali ad alta impedenza. Questa scelta è dovuta al fatto che il wishbone nasce come protocollo di interfaccia tra diversi *Intellectual Property* (IP) contenuti in un *System on a Chip* (SoC) ed internamente ad un circuito integrato avere un filo ad alta impedenza è un potenziale problema, il suo valore è infatti sconosciuto (nessun driver infatti li sta pilotando) e può portare ad una caduta di tensione non digitale che causa un consumo di potenza inutile.
Nella (Fig. 11) si nota pure la presenza della macchina a stati, il suo diagramma degli stati viene descritto passando prima attraverso il timing diagram.

4.2 Timing Diagram

Per disegnare il timing diagram occorre fare un'analisi *worst case* delle tempistiche dei segnali interessati nella comunicazione per capire se un ciclo di scrittura o un ciclo di lettura può essere eseguito in un unico colpo di clock.
Iniziamo col considerare il ciclo di lettura, il caso peggiore in cui possiamo trovarci è quello in cui la richiesta di lettura da parte del master arriva un istante di

tempo prima del fronte di salita del clock, in tal caso infatti occorre considerare il tempo impiegato dai segnali di indirizzo per uscire dal dispositivo FPGA ed essere stabili in ingresso alla SRAM, tale intervallo temporale che viene indicato con $t_{co_{ADR}}$ è stato ricavato dal datasheet del FPGA contenuto nella development board DE2, ossia Cyclone II EP2C35F672C6 dell'Altera, ed ha un valore massimo di 6.053 ns. Una volta avviata la lettura della SRAM e offerto in ingresso un indirizzo valido, il componente impiega un determinato tempo per fornire in uscita il dato corrispondente. Dai datasheet del dispositivo di memoria si scopre che questo tempo d'accesso alla memoria (t_{acc}) vale 12 ns. Una terza componente temporale da sommare alle due precedenti è rappresentata dall'intervallo necessario affinché il dato in ingresso all'FPGA venga portato al suo interno. Quest'ultimo intervallo, che indicheremo con $t_{pi_{DQ}}$, vale 1.385 ns ed è anchesso riportato nel datasheet del FPGA.
Come detto, questi tre valori temporali concorrono alla lettura della SRAM e, sommati tra loro si ottiene:

$$t_{read} = t_{co_{ADR}} + t_{acc} + t_{pi_{DQ}} = 6.053 + 12 + 1.385 = 19.438\ ns.$$

Le specifiche ci impongono una frequenza di clock pari a $f_{CK} = 50\ MHz$ e quindi si ha che il ciclo di lettura riesce ad essere garantito in un unico colpo di clock ($t_{read} < T_{CK} = 20\ ns$).
Il timing diagram per quanto riguarda il ciclo di lettura della SRAM viene dunque rappresentato in (Fig. 12).

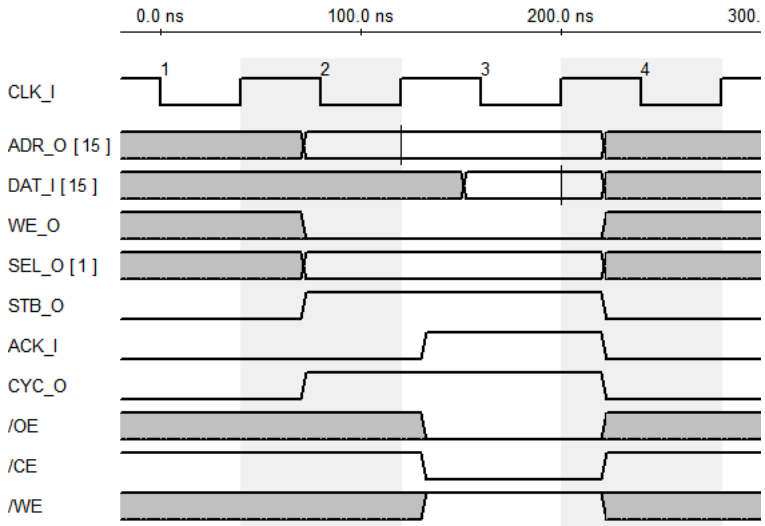

Figura 12: Timing Diagram del ciclo di lettura della SRAM

Le stesse considerazioni sono state fatte poi per il ciclo di scrittura. Così come accadeva per la lettura anche adesso occorre considerare la possibilità che il segnale di indirizzo venga fornito un istante prima del fronte valido del clock e quindi si ha che il primo intervallo di tempo da tenere in considerazione è dato da $t_{co_{ADR}}$. Successivamente occorre considerare il tempo dopo il quale il dato d'uscita risulta essere essere stabile sull'uscita del FPGA, questo lasso temporale, che indicheremo con $t_{co_{DQ}}$, ha come valore massimo lo stesso valore assunto da $t_{co_{ADR}}$, in qunato

entrambe fanno riferimento al tempo necessario affinché un dato contenuto in un registro interno al FPGA risulti valido al pin d'uscita del FPGA stesso. Infine, anche in questo caso dobbiamo considerare il tempo d'accesso della memoria, che si è considerato uguale al caso precedente e, quindi, abbiamo che:

$$t_{write} = t_{co_{ADR}} + t_{co_{DQ}} + t_{acc} = 6.053 + 6.053 + 12 = 24.106\ ns.$$

Quindi in questo caso sarà richiesto un ciclo di wait nel quale la scrittura della SRAM sarà iniziata ma il segnale di acknowledge non sarà asserito. Il timing diagram del ciclo di scrittura viene mostrato in (Fig. 13).

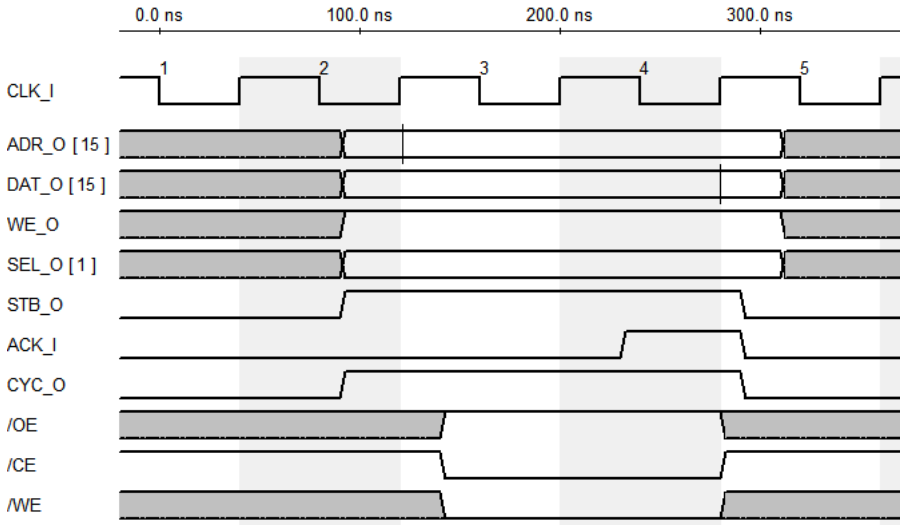

Figura 13: Timing Diagram del ciclo di scrittura della SRAM

Dai timing diagram è facile ottenere il diagramma degli stati che dovrà avere la macchina a stati presente nel wrapper e che quindi avrà il compito di pilotare i segnali occupati nella comunicazione.

4.3 Macchina a stati

Quindi, come conseguenza dei discorsi fatti in precedenza si ottiene il seguente diagramma degli stati che rappresenta la FSM di (Fig. 14).

Così come in tutte le interfacce slave realizzate in questa prova di laboratorio, anche quella riguardante la SRAM possiede due differenti segnali di reset: quello sincrono, tipico dell'interfaccia wishbone e fornito dal *System Controller*, che ha il compito di bloccare la macchina a stati, Mentre quello asincrono resetta tutti i registri utili per la comunicazione.

4.4 Testbench

Dopo aver scritto il codice VHDL lo abbiamo simulato utilizzando il software ModelSim, i risultati di questa simulazione sono illustrati in (Fig. 15) dove per ottenere delle risposte dal componente SRAM, è stato necessario scrivere in VHDL il suo comportamento behavioural.

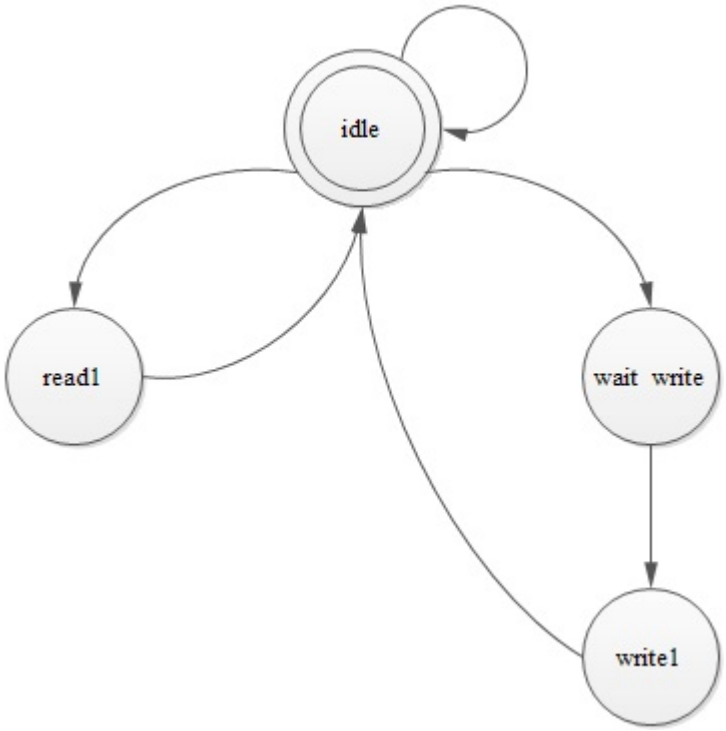

Figura 14: Diagramma degli stati dell'interfaccia della SRAM

5 Unione dei blocchi

Una volta che ci è stato fornito il listato del *master*, abbiamo unito i vari blocchi precedentemente descritti e, dopo che la simulazione con ModelSim ha dato ancora una volta esiti positivi, vedi (Fig. 16)[2], abbiamo caricato la nostra descrizione hardware sulla development board DE2 dove abbiamo potuto verificare anche fisicamente la buona riuscita di questa esperienza di laboratorio.

[2]Dove questa volta abbiamo però fatto la simulazione senza il behaviural della SRAM

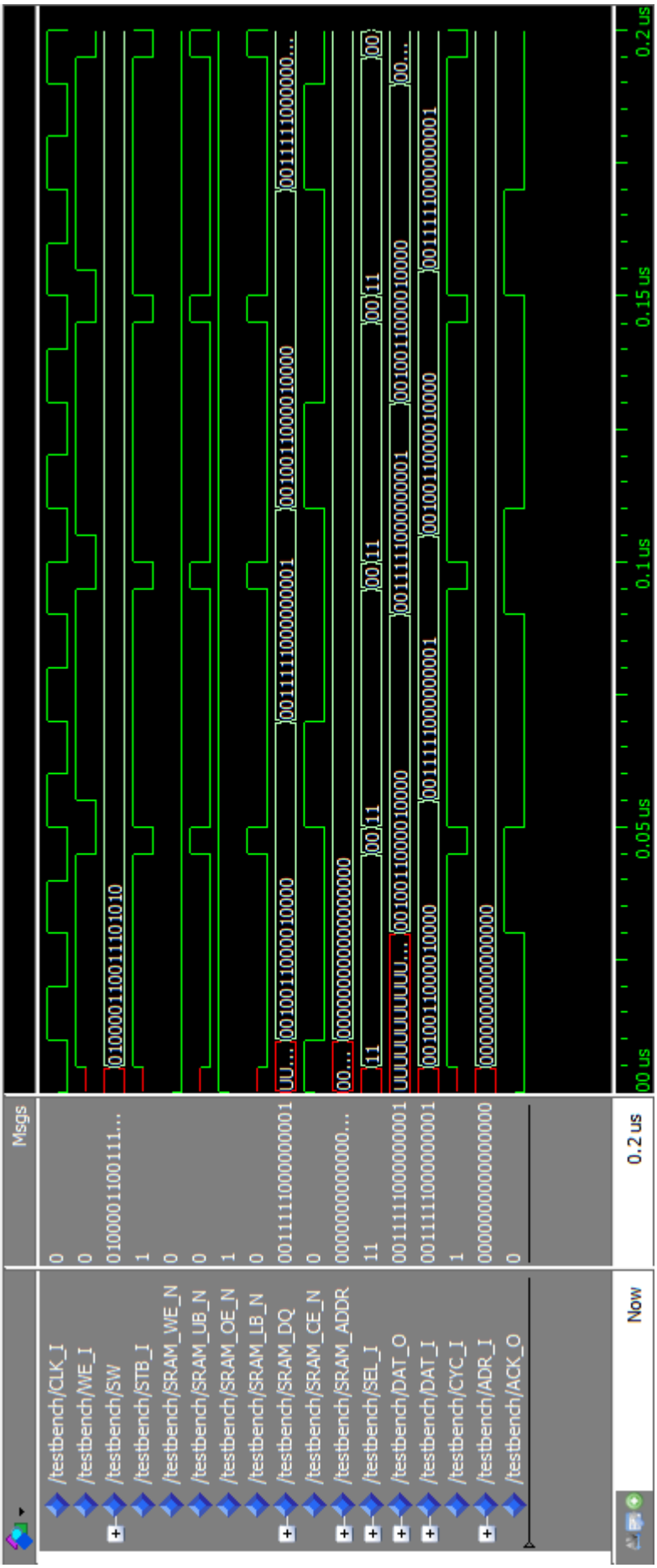

Figura 15: Simulazione con ModelSim della SRAM

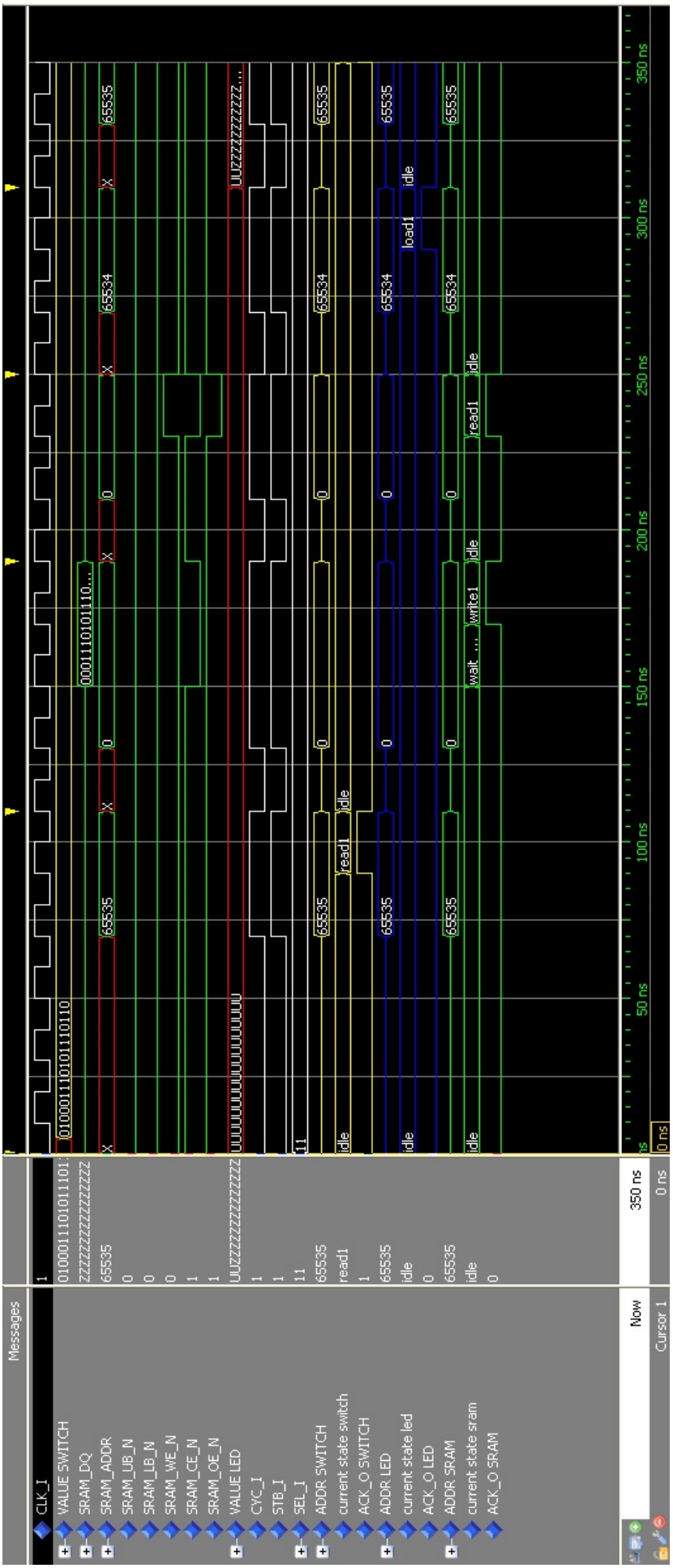

Figura 16: Simulazione con ModelSim del sistema