Implementazione in VHDL di un

Modulo per la gestione della memoria SDRAM

Su scheda Xess XSA-50

Christian Gregorutti

1 Introduzione

Questo progetto si propone di realizzare un modulo per la gestione della memoria SDRAM presente sulla scheda XSA-50. Questo lavoro si appoggia in parte a quello svolto dalla Xess Corporation (*www.xess.com*) [1] la quale implementa un modulo per far sembrare la SDRAM una semplice RAM statica.

La scheda XSA-50 è equipaggiata di una memoria Hynix HY57V641620HG, ovvero una *67,108,864-bit CMOS Synchronous DRAM* organizzata in 4 banchi di memoria da 1M x 16 bit (in totale 8 Mbyte).

Prima di vedere come questo modulo è stato implementato introduciamo brevemente le principali caratteristiche di una memoria SDRAM.

2 Concetti base delle memorie SDRAM

2.1 Il termine

La SDRAM (Syncronous Dynamic RAM) è una memoria ad accesso casuale dinamica e sincrona: dinamica in quanto, per come è costruita, ha bisogno di essere costantemente "rinfrescata" (si veda la sezione 2.2), sincrona perché utilizza un segnale di clock esterno per la sincronizzazione delle operazioni di I/O; questo permette un incremento delle prestazioni e una maggiore efficienza. La natura sincrona del chip fa sì che si possano implementare complessi modi operativi, pipeline interne e trasferimenti di dati a blocchi (*burst).*

2.2 Le celle di memoria

Come mostra la figura 1 l'informazione binaria è immagazzinata in unitá che consistono in un transistor e in un piccolissimo condensatore del valore di circa 20-40 fF (Femtofarad, 0.020-0.040 pF) per ogni cella. Un condensatore carico ha valore logico 1, uno scarico ha valore logico 0.

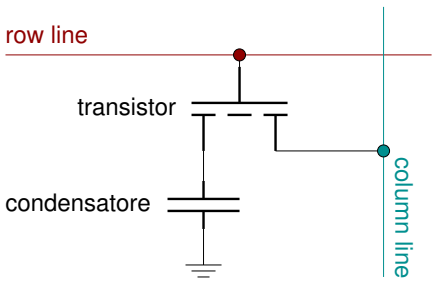

Figura 1: Cella elementare

Sfruttando l'effetto capacitivo dei CMOS si riesce a memorizzare un bit di informazione utilizzando pochissimi transistor. Il prezzo da pagare per questo risparmio in complessità realizzativa é che il valore inserito in una cella di memoria resta "memorizzato" solo per una quantità di tempo relativamente breve (frazioni di secondo), dopo di che viene "dimenticato" irrimediabilmente. L'idea che consente di realizzare dei moduli RAM dinamici consiste nel "rinfrescare la memoria" al dispositivo un attimo prima che il valore memorizzato vada perso. L'operazione di refresh di una memoria dinamica avviene leggendo il contenuto e riscrivendo immediatamente lo stesso valore appena letto, in modo che questo possa essere mantenuto per un'ulteriore frazione di secondo. L'operazione di refresh deve essere ovviamente ripetuta periodicamente per tutti i bit memorizzati in un modulo SDRAM, ed il ciclo di refresh di tutte le celle deve completarsi in un tempo inferiore rispetto a quello necessario ad una cella per "dimenticare" il proprio contenuto.

2.3 L'accesso in memoria

Essendo la SDRAM una memoria di grande capacità, i progettisti per risparmiare linee hanno deciso di spezzare la trasmissione dell'indirizzo in due parti, dette *Column Address* e *Row Address*. Queste rappresentano la riga e la colonna dove si trova il bit nelle matrici interne di memoria. Tale soluzione comporta vantaggi e svantaggi: se da un parte ha l'enorme pregio di far risparmiare molto spazio, dall'altra rende piú complicato l'interfacciamento con questo tipo di memorie.

Per accedere ad una locazione di memoria è quindi necessario prima specificare il Row Address (che viene memorizzato dal chip in un buffer interno) a cui segue la trasmissione del Column Address usando le stesse linee. Se gli accessi successivi sono nella stessa "pagina" (ossia hanno lo stesso Row Address) non è necessario specificare di nuovo il Row Addrerss ma solo il Column Address. Vediamo brevemente lo schema di accesso alla memoria:

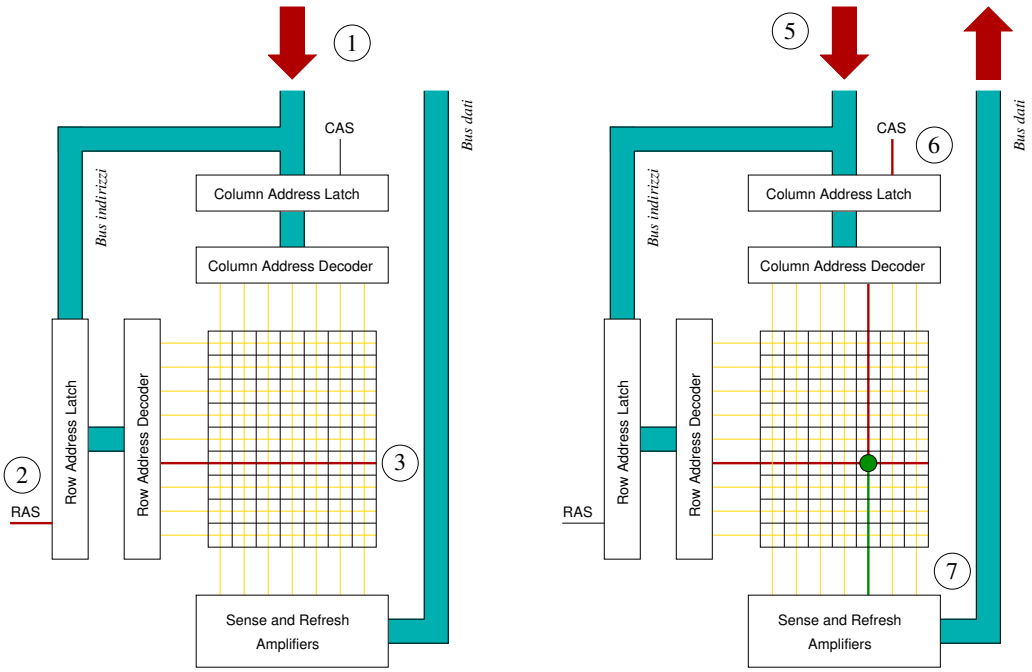

Figura 2: Schema di accesso alla memoria

1. Si immette sul bus indirizzi il Row Address;
2. Si attiva il segnale RAS (Row Address Strobe) che agendo su un apposito Latch memorizza il Row Address ad uso interno;
3. Il valore memorizzato nel Latch viene decodificato ed identifica una specifica riga (Row) nella matrice di memoria;
4. I segnali ed il bus vendono disasseriti;
5. Si immette sul bus indirizzi il Column Address;
6. Si attiva il segnale CAS (Column Address Strobe) che agendo su un apposito Latch memorizza il Column Address;
7. Il valore immagazzinato permette di individuare la colonna in cui si trova il dato. L'incrocio tra colonna e riga individua univocamente la locazione di memoria e in caso di lettura il suo contenuto viene inviato sul bus dati, in caso di scrittura il contenuto del bus dati viene scritto nella locazione.

2.4 Struttura e interfacciamento con l'esterno

In figura 3 possiamo vedere il diagramma funzionale a blocchi della nostra memoria (Hynix HY57V641620HG). In particolare sono immediatamente visibili i quattro banchi da 1M x 16bit, i bus degli indirizzi (in azzurro il bus condiviso, in verde l'indirizzamento delle righe e in arancio quello delle colonne) e il bus dati (in giallo).

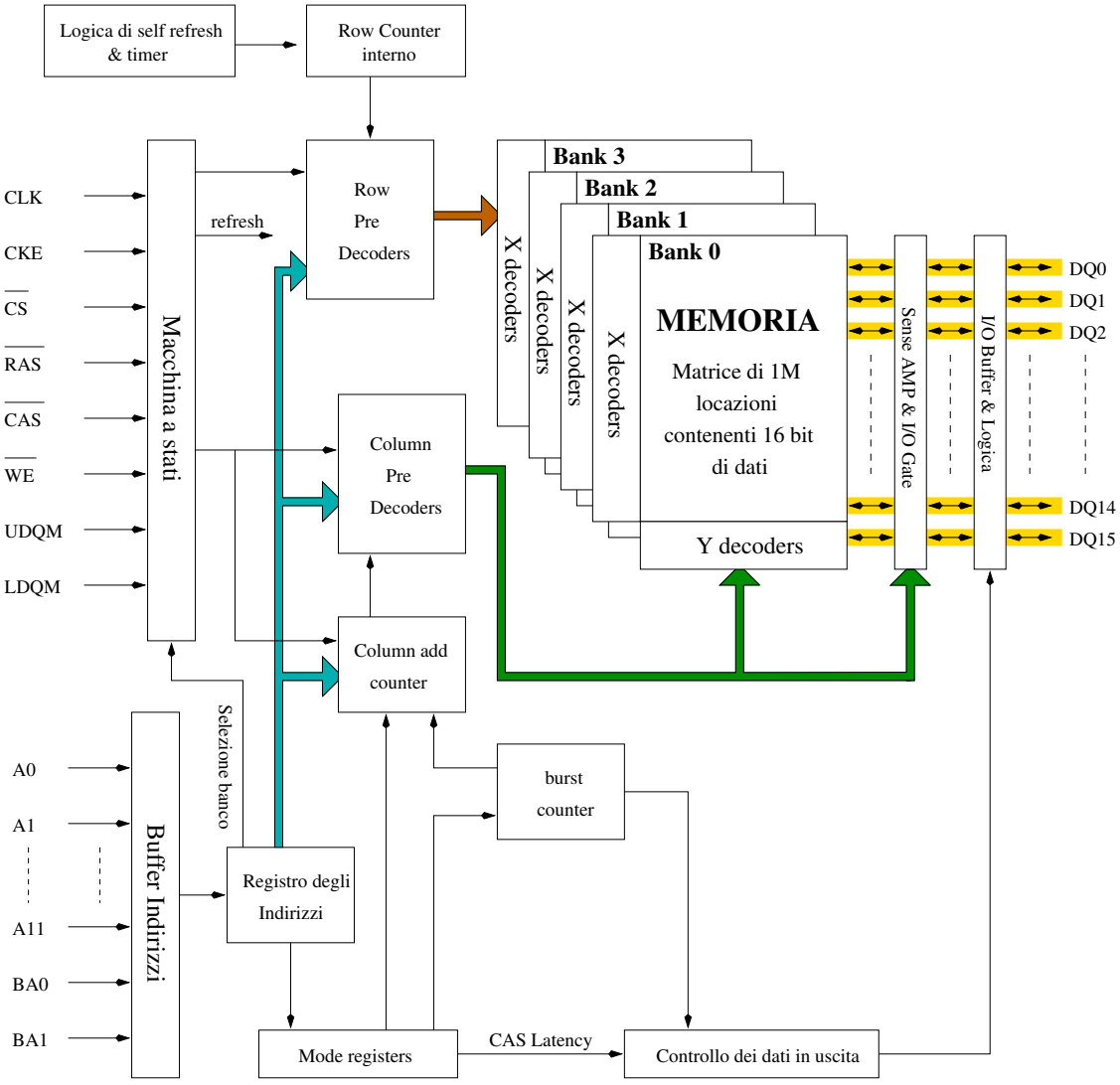

Figura 3: Diagramma funzionale a blocchi

Per quanto riguarda invece l'interfacciamento con l'esterno, `DQ0-DQ15` è il bus bidirezionale che trasporta i dati da e verso la memoria, `A0-A11` è il bus degli indirizzi, mentre `BA0` e `BA1` servono a selezionare il banco. Gli altri otto ingressi sono rispettivamente:

`CLK`: il master clock della memoria (**S**DRAM, memoria sincrona);

`CKE`: il clock enable, serve per abilitare il clock;

`CS`: chip select, attiva o disattiva il chip (queste memorie spesso sono organizzate in banchi - si pensi alla classica RAM dei PC - per cui CS serve a indicare per quale chip sono destinati i comandi che viaggiano sul bus condiviso);

`RAS`: Row Address Strobe, prende i dati presenti nel registro degli indirizzi e li fa interpretare al chip come indirizzi di riga;

`CAS`: Column Address Strobe, prende i dati presenti nel registro degli indirizzi e li fa interpretare al chip come indirizzi di colonna;

`WE`: Write Enable, indica se l'operazione da fare in memoria è di lettura o scrittura;

`LDQM/UDQM`: Data Input/Output Mask: i DQM controllano i buffer tristate presenti sui pin dei dati; LDQM controlla le linee dati dal DQ0 al DQ7, viceversa UDQM controlla le linee dati da DQ8 a DQ15: i due byte vengono gestiti in modo indipendente.

Per quanto concerne gli indirizzi, 1M locazioni a banco significa uno spazio di indirizzamento di 20 bit (2^{20}=1M): avendo un bus indirizzi di 12 bit (`A0-A11`) ogni locazione è identificata usando 12 bit per la riga e 8 bit per la colonna; a questo si aggiungono 2 bit per identificare il banco.

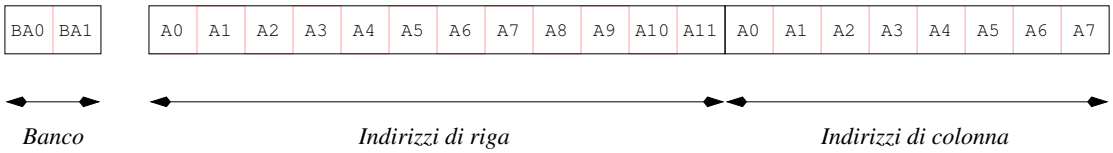

Figura 4: Indirizzamento

Qui si ferma la nostra breve panoramica sulle memorie SDRAM; tutti i dettagli su come funzionano questi componenti si possono trovare in [2] e [3], mentre per i dettagli inerenti alla memoria in dotazione sulla XSA-50 si rimanda ai data sheet forniti dalla Hynix [4], [5] e [6].

3 Funzionamento

Il funzionamento di questo modulo è molto elementare; come mostra la figura 5 gli ingressi e le uscite del modulo consistono in: un DIP switch (XSA-50), un display a sette segmenti (XSA-50) e tre pulsanti (XStend Board 2.0).

Tramite il primo switch si seleziona uno dei due indirizzi di memoria riservati (indirizzo 0 e 1); nella locazione individuata da tale indirizzo è possibile:

- leggere il dato in essa contenuto e visualizzarlo sul display a sette segmenti;
- scrivere il dato generato dagli altri tre switch (questi ultimi determinano il contenuto dei tre bit meno significativi di un dato a 16 bit);

queste operazioni sono comandate dai due pulsanti (*read*) e (*write*).

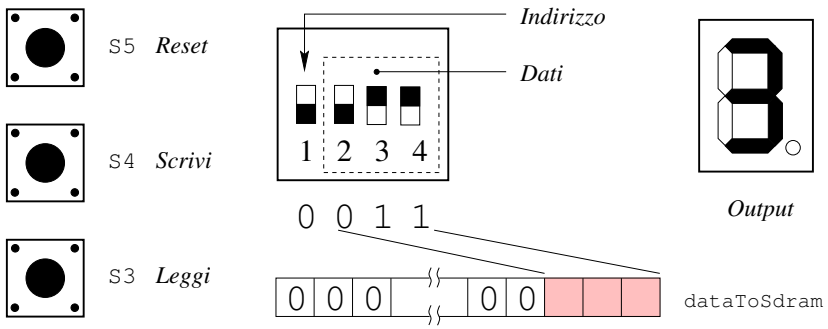

Figura 5: Ingressi e uscite del modulo

4 L'implementazione

Diamo ora uno sguardo all'implementazione: in figura 6 abbiamo uno schema che, in modo un po' semplificato, permette di vedere le entity presenti e le loro interconnessioni; la gestione del protocollo è affidata al modulo `sdramcntl` (implementato dalla Xess [1]), mentre i moduli `write` e `read` si occupano di gestire gli opportuni segnali di controllo frapponendosi tra l'utente e il modulo `sdramcntl`.

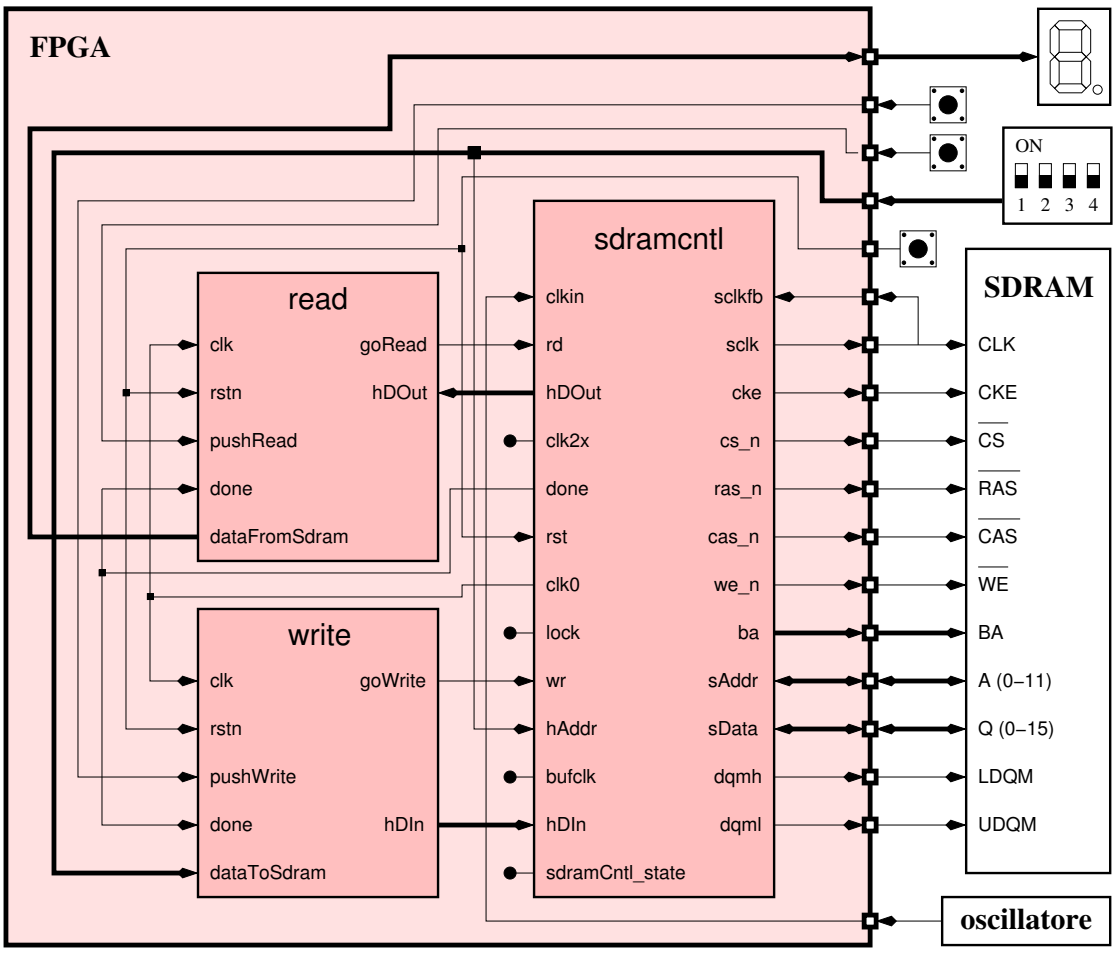

Figura 6: Schema a blocchi del modulo

4.1 `sdramcntl`

Questo è il controller della memoria SDRAM; acetta da parte della logica esterna semplici richieste di lettura e scrittura (come se si trattasse di una memoria statica) e si occupa di generare tutti i (complessi) segnali per adattare queste richieste ad una memoria dinamica quale la SDRAM. Per avere un'idea della complessità di queste memorie si tenga presente che il controllore deve: inizializzare la memoria, programmarne la circuiteria interna in modo da settare alcuni parametri che possono modificare radicalmente il suo comportamento, gestire il refresh, gestire l'indirizzamento in termini di righe/colonne, e molte altre cose (per i dettagli si rimanda a [2], [3] e [5]) oltre alle "normali" operazioni di lettura e scrittura.

Di seguito vengono presi in esame i segnali che questo modulo scambia con il resto della logica su FPGA; per i segnali *SDRAM-side* si rimanda alla sezione 2.4.

clkin *in*: è l'ingresso del master clock, preso direttamente dall'oscillatore programmabile DS1075. Particolare attenzione bisogna porre al fatto che la frequenza di clock dev'essere maggiore o uguale a 25 MHz (frequenze inferiori non riescono a garantire una velocitá di refresh adeguata).

rst *in*: reset sincrono per la logica interna al controller della SDRAM. Quando viene asserito il controller inizializza la SDRAM per l'uso.

done *out*: questo segnale di uscita sincrono passa al valore logico 1 per indicare il compimento della corrente operazione di lettura o scrittura.

rd *in*: quando questo segnale passa allo stato logico 1 viene iniziata la lettura di una singola parola dalla SDRAM. Viene campionato sul fronte di salita del clock e deve essere tenuto alto per tutta la durata dell'operazione di lettura; deve inoltre tornare basso non appena il segnale `done` va a 1 e prima del successivo fronte di salita del clock, altrimenti viene iniziata una nuova lettura.

wr *in*: quando questo segnale passa allo stato logico 1 viene iniziata la scrittura di una singola parola dalla SDRAM. Viene campionato sul fronte di salita del clock e deve essere tenuto alto per tutta la durata dell'operazione di scrittura; deve inoltre tornare basso non appena il segnale `done` va a 1 e prima del successivo fronte di salita del clock, altrimenti viene iniziata una nuova scrittura.

hAddr *in*: l'indirizzo della parola che deve essere letta dalla (scritta nella) SDRAM viene passato attraverso questo bus d'ingresso. Il valore dell'indirizzo deve essere tenuto stabile per tutto il tempo dell'operazione di lettura (scrittura).

hDIn *in*: i dati che devono essere scritti nella SDRAM passano attraverso questo bus d'ingresso. Il valore dei dati deve essere antenuto stabile per tutta la durata dell'operazione di scrittura.

hDOut *out*: i dati letti dalla SDRAM escono da questo bus. Questi dati devono essere presi sul fronte di salita del clock dopo che il segnale done sia passato al livello logico 1.

Per capire a cosa corrispondono gli ultimi due pin (**clk0** e **sclkfb**) dobbiamo prima introdurre un concetto molto interessante: un particolare dell'implementazione che rappresenta uno dei punti di forza delle Spartan II.

4.1.1 Il Delay-Locked Loop

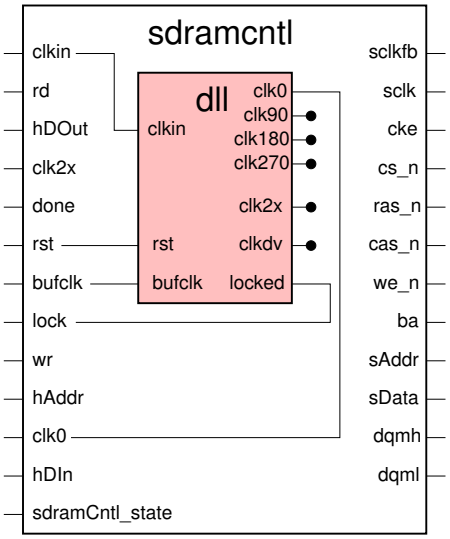

Figura 7 Il modulo dll.

Al fine di dover trasferire grosse quantitá di dati in pochissimo tempo è richiesta una tecnologia avanzata per il trattamento del clock: in particolar modo il ritardo sul clock e il *clock skew*, inteso come differenza del tempo d'arrivo del clock in due moduli differenti, sono punti chiave per le prestazioni.
A tal proposito tuttle le FPGA della famiglia delle Spartan II posseggono integrati quattro chip DLL (Delay-Locked Loop) che si occupano di mantenere una propagazione nulla del ritardo sul clock e un bassissimo *clock skew*.
In aggiunta, al fine di evitare ritardi di propagazione in clock inseriti dall'utente, i circuiti DLL possono comportarsi come duplicatori o come divisori (fino a 16x) di clock.
Per i dettagli si rimanda a [7].

sclkbf *in*: questo ingresso è una copia del segnale di clock della SDRAM con aggiunto il ritardo dovuto al passaggio FPGA → SDRAM → FPGA. Il controller della memoria usa appunto uno dei circuiti DDL per compensare questo ritardo.

clk0 *out*: questo clock è mantenuto dal DLL sincronizzato con il clock sclk usato per pilotare la SDRAM. Questo clock si usa per il resto della circuiteria programmata sull'FPGA.

4.2 `write` e `read`

L'implementazione di questi due moduli è banale: per quanto riguarda il modulo `read` si tratta semplicemente di una macchina a stati che pilota la linea `rd` in modo tale che essa rimanga alta da quando viene premuto il tasto READ a quando la linea `done` passa allo stato logico '1', bufferizzando nel frattempo i dati inviati dalla memoria. La macchina a stati in questione è mostrata in figura 8.

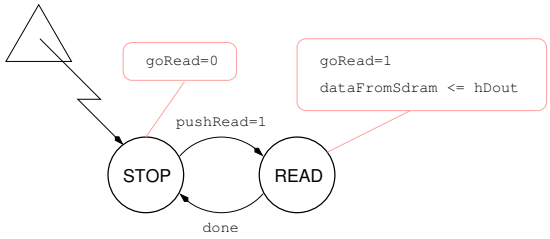

Figura 8: Macchina a stati del modulo read

In maniera perfettamente duale è stato sviluppato il modulo `write` (con la piccola differenza che qui a essere bufferizzati sono i dati che devono essere spediti alla tastiera, e questo avviene *prima* che venga premuto il tasto WRITE).

Per quanto riguarda i pin:

rstn *in*: il master reset;

clk *in*: il clock che esce dal circuito DLL pilota questi moduli;

pushWrite *in*: collegato al pulsante "WRITE";

pushRead *in*: collegato al pulsante "READ";

done *in*: segnale che il controllore della memoria invia alla logica su FPGA per indicare che la corrente operazione di lettura/scrittura è terminata;

hDIn *out*: dati bufferizzati nei registri interni del modulo `write` che devono essere scritti nella memoria SDRAM;

hDOut *in*: i dati letti dalla SDRAM da bufferizzare nei registri interni del modulo `read`;

dataToSdram *in*: i dati da inviare alla memoria;

dataFromSdram *out*: i dati da letti dalla SDRAM e bufferizzati: questo bus a tre linee viene poi rimappato in uno da sette per pilotare il display;

goWrite *out*: quando questo segnale passa allo stato logico 1 viene iniziata la lettura di una singola parola dalla SDRAM;

goRead *out*: quando questo segnale passa allo stato logico 1 viene iniziata la scrittura di una singola parola dalla SDRAM;

Bibliografia

[**1**] Xess Corporation: SDRAM controller module
http://www.xess.com/ho03000.html#Examples

[**2**] G. Baccolini, C. Offelli: "Microelaboratori: note di Hardware"
edizioni Clup, Milano 1983

[**3**] Jon Stokes: "Asynchronous and Synchronous DRAM"
http://www.arstechnica.com/paedia/r/ram_guide/ram_guide.part2-1.html

[**4**] Hynix HY57V641620HG - Data sheet
http://www.hynix.com/datasheet/pdf/dram/HY57V641620HG[L]T.pdf

[**5**] Hynix HY57V641620HG - Device Operation
http://www.hynix.com/eng/products/dram/down/SDRAM_operation(Rev1.1).pdf

[**6**] Hynix HY57V641620HG - Timing Diagram
http://www.hynix.com/eng/products/dram/down/SDRAM_timing.PDF

[**7**] Using Delay-Locked Loops in Spartan-II FPGAs
http://webservices.polito.it/matdid/3ing_eln_L1740_TO_0/ETLCIVR/matappnote/Xapp174DLL.pdf

[**8**] VHDL Reference Manual, Synario:
http://cslab.snu.ac.kr/course/cad99/vhdl_ref.pdf

[**9**] XSA-50 Board V1.2 Manual
http://www.xess.com/manuals/xsa-manual-v1_2.pdf