Controller di memoria DDR3 per bus AMBA

Laureando: Fabrizio Ballarin

Relatore: Dott. Daniele Vogrig

Corso di laurea specialistica in Ingegneria Elettronica

UNIVERSITÀ
DEGLI STUDI
DI PADOVA

a.a. 2010/2011

Indice

1. Introduzione

1.1. Tecnologie e progettazione

L'aumento delle capacità computazionali riscontrato negli ultimi anni in tutti i dispositivi elettronici è legato a doppio filo con la miniaturizzazione dei componenti e quindi con l'integrazione di molteplici strutture sullo stesso chip.

Inglobare, ad esempio, sullo stesso *die* (la piastrina di materiale semiconduttore che racchiusa all'interno del package costituisce il vero e proprio chip) della CPU le periferiche di comunicazione con l'esterno (coprocessori, controller ethernet, controller USB, controller di memoria…) consente allo stesso tempo di ridurre dimensioni e consumi ma anche di incrementare le prestazioni generali del sistema complessivo aumentando le velocità con cui i dati possono essere trasferiti tra le diverse strutture integrate.

Nonostante questa fusione di moduli eterogenei, il sistema risultante, denominato *SoC (System on Chip)* continua a rispettare l'architettura classica per un elaboratore formalizzata dal matematico John Von Neumann e contraddistinta da un'unità di elaborazione (a sua volta integrante al proprio interno un'unità di controllo), un bus di sistema, la memoria di sistema e le unità di Input/Output più adatte in base alla destinazione d'utilizzo del SoC.

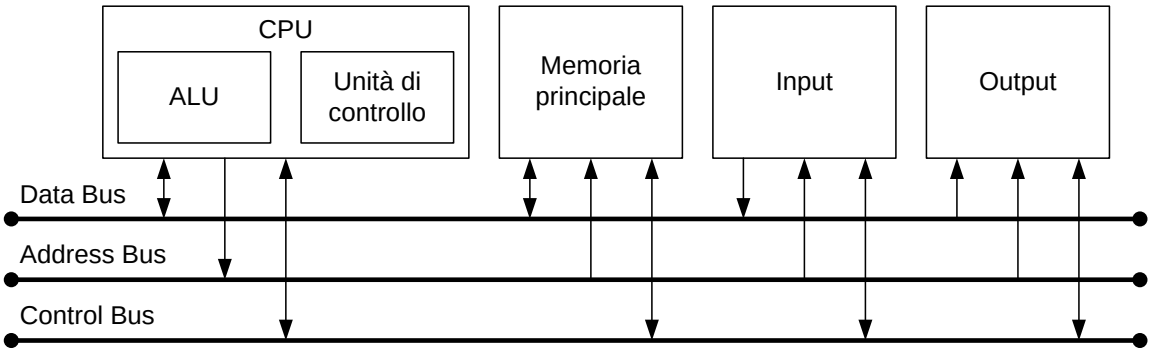


Figura 1: Architettura di una macchina di Von Neumann

Per poter realizzare questo tipo di SoC, la tecnologia attuale offre agli sviluppatori la possibilità di scegliere tra l'utilizzo di *FPGA (Field Programmable Gate Array)* o circuiti integrati *ASIC (Application Specific Integrated Circuit)*, con questi secondi che si fanno preferire ai primi per il conseguimento di migliori prestazioni ed un miglior sfruttamento delle risorse, area in primis, mentre gli FPGA riscuotono maggior successo grazie alla flessibilità offerta dalla programmazione degli stessi direttamente via software che li rende altamente competitivi, anche in termini di costi, nel caso di produzioni su piccola scala o nelle fasi di prototipaggio, a fronte di prestazioni leggermente inferiori.

1.2. Programmazione e realizzazione

Qualunque sia la tecnologia utilizzata per la realizzazione del SoC desiderato, un punto in comune tra gli FPGA e gli ASIC è rappresentato dai linguaggi utilizzati in fase di progettazione ed interpretabili dagli strumenti software dedicati [8.7], per la descrizione del comportamento e delle caratteristiche del prodotto che si vuol realizzare.

Tra i vari linguaggi, detti *HDL (Hardware Description Language)* per il loro ambito applicativo, se ne è scelto quello denominato *VHDL (VHSIC Hardware Description Language)*. Questo risulta avere alcuni tratti in comune con i classici linguaggi di programmazione, come ad esempio i costrutti *if-then-else* e le operazioni aritmetico logiche, ma a differenza di questi permette di descrivere alcune funzioni concorrenziali (eseguite contemporaneamente) in maniera molto più naturale rispetto agli altri linguaggi che al più possono appoggiarsi alla gestione dei *thread*.

Questa differenza è dovuta e permessa al linguaggio VHDL in quanto con questo si va a descrivere direttamente l'*hardware dedicato* allo svolgimento di una particolare operazione, mentre con i più classici linguaggi di programmazione (C, C++, Java) il

programmatore descrive ad alto livello le operazioni che dovranno essere successivamente adattate per poter essere realizzate da un qualche microprocessore o sistema generico che, non essendo necessariamente ottimizzato per tali compiti, potrebbe non disporre delle necessarie risorse hardware richieste per parallelizzare completamente le operazioni.

Il processo di realizzazione di un circuito integrato prevede una serie di passaggi che conducono da un livello di astrazione superiore verso un livello di astrazione inferiore. L'insieme di tali passaggi costituisce il flusso di progettazione *VLSI (Very Large Scale Integration)* composto delle seguenti fasi principali:

Descrizione funzionale: permette di redigere le specifiche funzionali e gli eventuali limiti richiesti (area – prestazioni – consumi);

Descrizione algoritmica: specifica la sequenza ordinata di operazioni che portano all'ottenimento della funzione desiderata;

Descrizione RTL (Register Transfer Level): a questo livello si introducono le risorse fisiche necessarie (operazioni aritmetico logiche, registri, mux-demux…) ed il legame temporale tra tali strutture rappresentato dal clock;

Descrizione logica: il circuito viene rappresentato utilizzando le porte logiche combinatorie e sequenziali;

Descrizione fisica: vengono definite geometricamente le maschere e le interconnessioni utilizzate per la realizzazione del circuito integrato.

Il passaggio da un livello di descrizione al successivo denominato sintesi (algoritmica – logica – RTL – fisica) può essere automatico o, in alcuni casi, richiedere al progettista la definizione di vincoli e la scelta tra diverse soluzioni possibili.

Ad ogni sintesi effettuata deve necessariamente seguire una fase di verifica e simulazione per garantire che la traduzione del circuito da un livello di astrazione elevato ad uno più basso, risulti ancora conforme a quanto richiesto inizialmente al sistema.

Per fare ciò viene quasi sempre utilizzato un secondo circuito, o modello software, che funge da banco di prova (*testbench*) incaricato di fornire tutti i segnali necessari per stimolare il componente sotto test. Gli ingressi e le risposte offerte dal circuito sotto

esame vengono poi analizzate sfruttando degli strumenti di simulazione, perlopiù software, attraverso i quali sarà possibile verificare la corrispondenza delle risposte ottenute con quelle attese.

1.3. Obiettivi

L'obiettivo del progetto prevede la realizzazione di un memory controller DDR3 da poter integrare in un sistema a microprocessore modulare basato sul processore LEON3 [8.6].

La scelta del microprocessore è ricaduta sul LEON3 in quanto basato su un'architettura diffusa ed efficiente in molti campi d'utilizzo, ed al tempo stesso il modello VHDL dello stesso viene fornito in licenza gratuita per scopi di ricerca o comunque a prezzi concorrenziali per scopi commerciali.

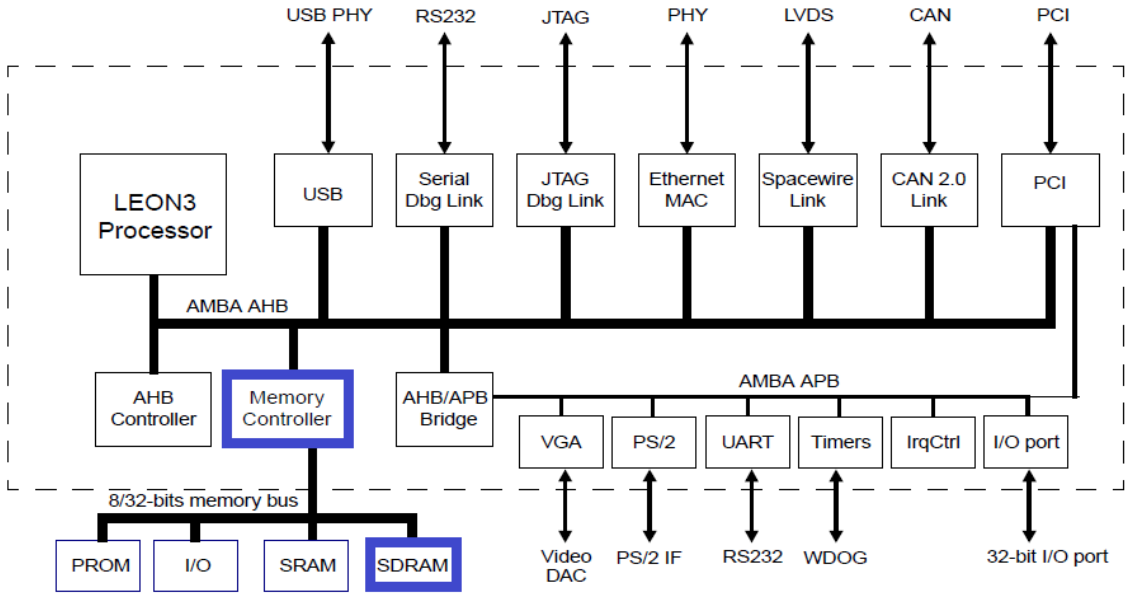


Figura 2: Schema a blocchi di un sistema a microprocessore

L'aggiunta di un memory controller compatibile con le SDRAM-DDR3 permette quindi di dare maggior flessibilità ed aggiornare un sistema duttile e modulare utilizzabile in diverse applicazioni.

La Figura 2 illustra una possibile composizione di un sistema completo, al quale possono essere aggiunti, rimossi o sostituiti alcuni moduli con degli altri necessariamente compatibili con le interfacce di comunicazioni caratterizzanti il sistema, ma che possono avere prestazioni differenti; molti di questi moduli sono già stati sviluppati in passato e le

loro realizzazioni sono raccolte all'interno della libreria *GRLIB IP Libray (Gaisler-Research).*

1.4. Organizzazione della tesi

Nei capitoli 2 e 3 verranno introdotti gli standard su cui il controller di memoria dovrà poggiarsi, rispettivamente per quanto riguarda il bus di sistema (AMBA) e la tipologia di memorie compatibili (DDR3).

Nel capitolo 4 verrà invece introdotto il controller di memoria stesso e ne verranno descritte le funzioni, la struttura e le particolarità.

Questa sezione sarà, a differenza della precedente, molto più vicina al mondo della progettazione, e verranno trattate in particolar modo le problematiche tecniche a cui il progettista deve far fronte

Il capitolo 5 illustra le fasi di simulazione e contiene analisi e considerazioni sui risultati ottenuti

Infine nel capitolo 6, oltre ad alcune considerazioni finali vengono introdotti alcuni dei possibili futuri sviluppi in grado di perfezionare e completare il memory controller oggetto di questo documento.

2. Standard AMBA

Questo capitolo tratterà lo standard *AMBA (Advanced Microcontroller Bus Architecture)* con il quale ARM [8.1] ha introdotto un completo set di bus di sistema adatti a diversi ambiti di utilizzo perlopiù orientati alle soluzioni di tipo embedded.

Più in particolare verrà descritto il bus *AHB (Amba High performance Bus)* sul quale poggerà il controller di memoria DDR3 oggetto di questo documento

Le questioni più strettamente legate all'implementazione verranno più approfonditamente trattate nel capitolo 4, mentre qui di seguito verranno descritte le funzionalità offerte da questo tipo di bus.

2.1. Ambiti di utilizzo

Lo standard AMBA è stato concepito, e negli anni aggiornato da ARM, azienda leader nella realizzazione di processori e microcontrollori per sistemi di tipo embedded che trovano molti impieghi nei dispositivi elettronici di largo consumo (riproduttori multimediali, telefoni cellulari …), ed in particolar modo in molti dei sistemi a microprocessore di tipo *SoC (System on a Chip)*.

L'avere alle proprie spalle un'azienda come ARM riconosciuta a livello mondiale e leader nel detenere e fornire *IP (Intellectual Property)* ha portato lo standard AMBA a diventare uno *standard de facto* per il settore.

Oltre a tali librerie fornite dalla stessa ARM o dalla comunità di sviluppatori indipendenti, un'ulteriore spinta allo standard AMBA è giunta dall'assenza di *royalties* che nel caso di altri standard ed interfacce gravano sui costi di sviluppo.

Ricordando che AMBA definisce delle specifiche per l'interconnessione di differenti periferiche andiamo rapidamente a valutare come alcuni parametri possano direttamente influire sulle prestazioni dell'intero sistema.

Ampiezza del bus dati e del bus indirizzi sono tra i primi fattori discriminanti per la scelta o per la definizione di un apparato d'interconnessione, ma non sono certo gli unici parametri che ne definiscono le prestazioni, l'efficienza o la complessità generale.

È sufficiente aggiungere alcune variabili quali la frequenza del clock usato per la sincronizzazione delle trasmissioni, oppure il grado di parallelismo ottenibile differenziando i bus di lettura/scrittura ed ancora le procedure previste per il trasferimento delle informazioni e per il controllo del bus di comunicazione tra due dispositivi collegati allo stesso bus, per ottenere un'enorme varietà di strutture che possono essere tutte raggruppate sotto la generica definizione di bus d'interconnessione, ma che eseguono lo stesso compito con metodi ed operazioni molto diversi l'uno dall'altro.

Non bisogna inoltre sottovalutare il peso assunto dal bus d'interconnessione in merito alla complessità realizzativa dello stesso (che dovrà estendersi e raggiungere buona parte dell'area del SoC) e di tutte le periferiche che andranno connesse ad esso; la maggior parte delle scelte progettuali che possono essere intraprese per migliorare le prestazioni del bus di sistema comportano un incremento del costo realizzativo del bus e spesso anche delle periferiche compatibili che si troveranno a dover implementare particolari controlli sulle diverse procedure che costituiscono un trasferimento dati sul bus.

2.2. Strutture ammesse

Le specifiche AMBA prevedono la distinzione tra due categorie di dispositivi connessi sullo stesso BUS, le unità operanti in qualità di "Master" e quelle invece operanti come "Slave".

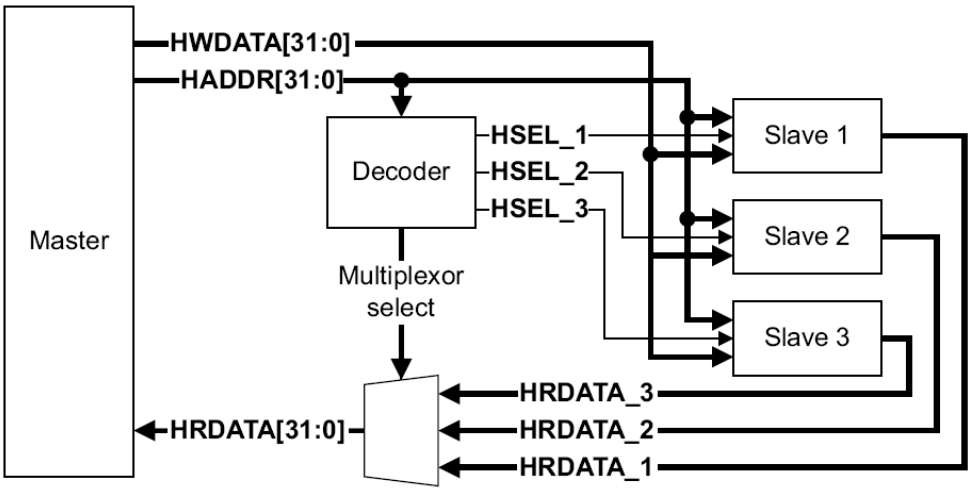


Figura 3: Esempio di sistema con un dispositivo Master ed una pluralità di Slave

Al Master spetta il compito di iniziare la comunicazione e successivamente di gestirne lo stato (terminarla, mantenerla attiva, sospenderla) mentre lo slave, al contrario, rimane in ascolto dei segnali provenienti dal bus e risponde quando interpellato.

L'unità Master per eccellenza è rappresentata da un microprocessore, mentre gli slave possono essere di natura molto varia, e con scopi altrettanto diversi, come ad esempio co-processori, interfacce per comunicazioni con l'esterno (eg. controller ethernet-USB-RS232…) e controller di memoria.

In questo documento non verranno trattate problematiche strettamente riguardanti il comportamento del Master e delle diverse configurazioni disponibili (eg. sistemi multi-Master, Master arbiter…) essendo il controller di memoria da realizzare una struttura di natura Slave.

2.3. Tipologie e varianti

Gli aggiornamenti ai protocolli definiti nel corso degli anni, accumunati sotto la stessa famiglia AMBA, hanno portato in alcuni casi a rinnovamenti della medesima struttura, in altri invece all'inserimento di nuovi protocolli definenti strutture specifiche che andassero a ricoprire particolari utilizzi.

Vediamo quali sono le strutture definite nelle diverse revisioni degli standard AMBA tuttora supportate:

AMBA2 ha introdotto per la prima volta i protocolli per le interfacce *AHB (Amba High performance Bus)* e *APB (Amba Peripheral Bus).*

AMBA3 ha previsto un aggiornamento delle strutture AHB ed APB precedentemente definite introducendo in particolar modo dei controlli relativi al comportamento dei dispositivi Master quando questi sono sul bus ve ne sono presenti una pluralità, ed ha introdotto un'interfaccia *ATB (Amba Trace Bus)* con funzionalità di debug ed un'interfaccia AXI (Advanced eXtensible Interface) per sistemi ad alta efficienza che richiedono particolari accortezze tra le quali frequenze variabili per mantenere sotto controllo il consumo energetico, trasferimenti d'informazioni full duplex e molto altro.

AMBA4 ha aggiunto tre ulteriori tipi d'interfacce *AXI4*, *AXI4-Lite* ed *AXI4-Stream* che rispettivamente aggiornano, semplificano ed estendono lo standard AXI definito nelle specifiche AMBA3.

Tra tutte queste interfacce il lavoro svolto si è concentrato sul protocollo **AHB** che permette di ottenere risultati soddisfacenti senza per questo esasperare la complessità del bus stesso.

2.4. Modalità e fasi di funzionamento

Come già detto le comunicazioni in un bus di tipo AHB iniziano per mano del Master, che in una prima fase dovrà trasmettere sul bus l'indirizzo del dispositivo a cui vuol accedere e le informazioni relative al tipo d'accesso, mentre nella fase successiva ottiene l'informazione dal dispositivo interrogato (o completa il trasferimento dei dati verso il dispositivo indirizzato precedentemente, se l'operazione da compiere è di scrittura piuttosto che di lettura).

Questo funzionamento caratterizzato dalle due distinte fasi di indirizzamento (*Address Phase*) e trasmissione delle informazioni (*Data Phase*) permette di implementare un sistema di comunicazione che prevede la parallelizzazione delle operazioni su di una pipeline.

Permettendo al Master di inoltrare una nuova richiesta di informazioni (anche verso uno slave diverso) mentre lo stesso Master rimane in attesa dell'informazione richiesta nel ciclo di clock precedente, a regime e nelle migliori condizioni operative, si riesce a raddoppiare il flusso di informazioni circolanti attraverso il bus.

Le due fasi caratterizzanti il bus AHB sono sincrone al fronte di salita dell'unico clock presente nel bus AHB come riportato nel diagramma temporale di Figura 4:

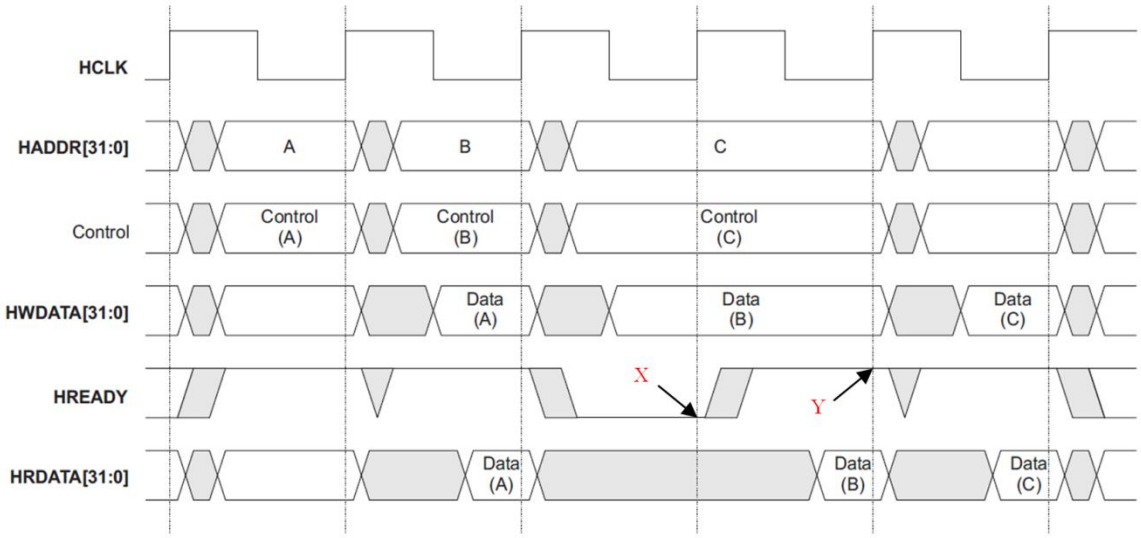


Figura 4: Trasferimento dati su AHB bus

Sul fronte di salita del clock lo slave campiona il bus degli indirizzi ed i vari segnali di controllo, mentre sul successivo fronte di salita il Master ottiene i dati richiesti attraverso il bus di lettura (oppure trasmette attraverso il bus dati di scrittura le informazioni da salvare sullo slave qualora l'operazione da compiere fosse di tipo "write") ed al tempo stesso è pronto ad interrogare una nuova periferica caricando un nuovo indirizzo sul bus indirizzi.

Qualora la periferica slave interrogata non riuscisse a fornire per tempo l'informazione richiesta, lo slave stesso dovrà darne comunicazione al Master abbassando il segnale *HREADY* (come evidenziato nel punto X del diagramma temporale), il Master, preso atto di ciò ripeterà la medesima interrogazione alla periferica successiva mantenendo rigorosamente l'ordine all'interno della pipeline, fino a che non otterrà una risposta affermativa dal primo slave (Y).

In particolare, qualora come riportato nell'esempio di qui sopra il Master intendesse accedere alle celle di memoria A, B, C e così via, e la periferica relativa alla cella B dovesse richiedere un arbitrario numero di cicli di clock prima di riuscire a fornire un risultato valido, le operazioni riguardanti la cella C verrebbero fatte slittare di altrettanti cicli di clock evitando così che il risultato relativo alla cella C possa pervenire prima di quello relativo alla cella B.

Lo standard AMBA predilige quindi un concetto di pipeline "*in-order*", in cui la sequenza di operazioni che si susseguono viene determinata a priori e verrà conclusa seguendo il medesimo ordine, qualunque siano i tempi di esecuzione per le diverse operazioni.

Un sistema di tipo "*out-of-order*", che preveda la possibilità di cambiare l'ordine di risposta delle periferiche nel caso una di queste richiedesse un maggior tempo per elaborare le informazioni, favorendo quelle successive che invece si trovano con dei risultati già disponibili, si dimostrerebbe sicuramente più efficace ma altresì più complicato da implementare e gestire.

2.5. Data Bus AHB

Il bus AHB prevede l'utilizzo di due diversi bus dati, uno dedicato alle informazioni in transito dal Master verso gli slave (scrittura) ed uno per le informazioni transitanti nel verso opposto (lettura).

L'utilizzo di due BUS separati evita che vengano usati dei buffer tri-state per la gestione bidirezionale dello stesso BUS semplificando la realizzazione fisica di tutte le porte d'interfaccia dei vari dispositivi connessi al bus.

Entrambi i bus hanno ovviamente la stessa larghezza pari a 32 bit, e vengono denominati dalle specifiche AMBA come:

- *HRDATA*[31:0]: BUS dati di lettura dagli slave
- *HWDATA*[31:0]: BUS dati di scrittura verso gli slave

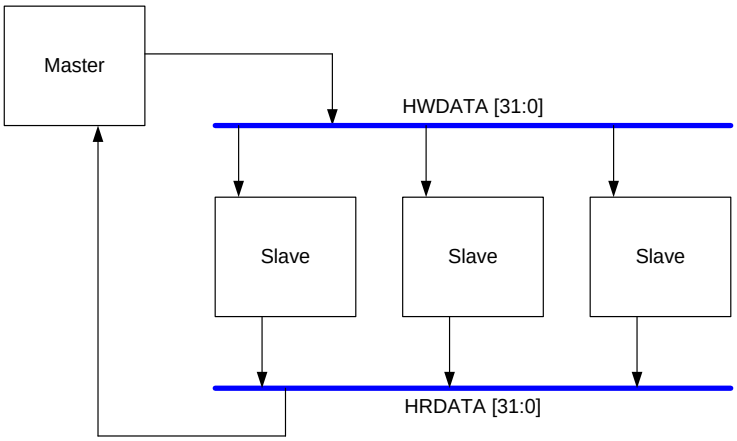


Figura 5: Flusso delle informazioni attraverso i bus dati in un bus AHB

2.6. Address Bus AHB

Anche il bus indirizzi, come i due bus dati, prevede una larghezza di 32 bit e viene utilizzato unidirezionalmente, ovvero è sempre il Master a scrivere su di esso l'indirizzo del dispositivo che dovrà ricevere o fornire l'informazione richiesta, mentre gli slave rimangono passivamente in ascolto sulle stesse linee.

In mancanza di altre informazioni o vincoli per il proseguimento della progettazione si è fatto riferimento ad un sistema LEON3 anch'esso basato su bus AMBA per partizionare i 2^{32} indirizzi disponibili.

Core	Address range	Bus Index
MCTRL	0x00000000 - 0x20000000 : PROM area 0x20000000 - 0x40000000 : I/O area 0x40000000 - 0x80000000 : SRAM/SDRAM area	0
APBCTRL	0x80000000 - 0x81000000 : APB bridge	1
DSU3	0x90000000 - 0xA0000000 : Registers	2
ETH_OC	0xFFFB0000 - 0xFFFB1000 : Registers	5
CAN_MC	0xFFFC0000 - 0xFFFC1000 : Registers	4
AHB plug&play	0xFFFFF000 - 0xFFFFFFFF : Registers	-

Figura 6: Organizzazione e suddivisione della memoria disponibile nel sistema LEON3

Volendo dedicare alla memoria di sistema ¼ dell'intero spazio indirizzabile, risultano utilizzabili 2^{30} celle ognuna della quali contenente 4 byte (32bit dettati dalla larghezza del bus dati); il tutto porta ad una memoria di sistema complessiva massima pari a 4 GByte.

3. Memorie DDR3

In questo capitolo verranno introdotte le memorie DDR3 per le quali il memory controller sarà sviluppato, ponendo l'attenzione sulle caratteristiche tecniche e sulle differenti versioni descritte e ratificate dagli organi di standardizzazione competenti.

3.1. Memorie dinamiche

In un sistema *general purpose* la memoria disponibile per i programmi in esecuzione è quasi sempre di tipo dinamico per via del compromesso costi / prestazioni favorevole a quest'ultima rispetto alla tipologia statica.

Questo miglior rapporto costo/prestazioni deriva direttamente dalla maggior densità ottenibile considerato che la cella fondamentale costituente una memoria dinamica sfrutta poco più di un transistor ed un capacitore per immagazzinare la carica relativa ad un singolo bit, mentre tipicamente nelle memorie statiche sono presenti sei transistor per adempiere allo stesso compito.

La maggior semplificazione nella struttura di un chip di memoria RAM dinamica, comporta, d'altra parte, una complicazione a livello del memory controller, ovvero del dispositivo che coadiuva il funzionamento dei moduli di memoria e che si pone tra la memoria stessa ed il resto del sistema (bus di sistema o cpu) con lo scopo di favorire le

comunicazioni e sollevando il microprocessore dal compito di rispettare tutti i vincoli imposti dallo standard relativo alle memorie dinamiche.

Le ultime specifiche, redatte dall'ente di standardizzazione dei semiconduttori JEDEC [8.2], relative alle memorie *SDRAM (Synchronous Dynamic Random Access Memory)*, risalenti a novembre 2008, hanno dato vita allo standard *DDR3 (Double Data Rate)*.

Questo ampio set di norme coprono tutti gli aspetti relativi alle memorie DDR3, toccando punti di natura

- *Fisica*, package utilizzabili e la piedinatura dei diversi moduli di memoria;
- *Logica*, come le diverse combinazioni di segnali necessarie per il corretto funzionamento;
- *Temporale*, prevedendo frequenze, tensioni e ritardi ammissibili nelle diverse condizioni operative;

Allo standard che definisce i singoli chip di memoria dinamica, si aggiunge quello relativo alle configurazioni implementabili sfruttando più chip DDR3 e che di fatto definisce le caratteristiche dei moduli *DIMM (Dual In-line Memory Module)*, quelle piccole schede che ospitano molteplici chip di memoria DDR appositamente interconnessi e che ad esempio trovano posto all'interno dei normali computer desktop o laptop sottoforma di schedine facilmente installabili e removibili di dimensioni contenute.

Per scopi specifici il produttore potrebbe optare per la realizzazione di strutture di memoria personalizzate con l'obiettivo di sfruttare al meglio gli spazi a disposizione oppure per creare una struttura maggiormente adatta ad immagazzinare le informazioni del caso.

3.2. Struttura di un modulo DIMM

Analizziamo di seguito l'organizzazione di un tipico modulo DIMM del tutto compatibile con quelli utilizzati all'interno dei normali personal computer.

Il diagramma funzionale di Figura 7 ben evidenzia i 16 chip di memoria SDRAM organizzati in modo da formare parole da 64 bit [DQ0: DQ63] ciascuna per ogni indirizzo disponibile.

Ci si trova quindi di fronte ad un collegamento di moduli in parallelo connessi in modo da espandere i bit per ogni parola indirizzata

- nello schema i chip D0-D1-D2-…-D7 formano una parola da 64 bit fornendo ognuno una parola di 8 bit, quando questi vengono richiamati dallo stesso indirizzo

ed altri collegati in serie

- il gruppo di chip D0-D7 è collegato in serie al gruppo D8-D15 per poter raddoppiare il numero di indirizzi disponibili, un bit tra quelli che identificano l'indirizzo discriminerà quale dei due blocchi viene interpellato

Il modulo DIMM risultante, per raggiungere la capacità totale di 2 Gbyte in parole da 64 bit ciascuna è perciò suddiviso in

- 2 RANK ognuno dei quali formato da:
    - 8 chip di SDRAM ciascuno ospitante,
        - 128 Mbyte suddivisi in parole da,
            - 8bit

Con il termine *RANK* solitamente vengono indicate le facce del modulo DIMM e quindi in questo caso il modulo monta chip di memoria su entrambi i lati del *PCB (Printed Circuit Board).*

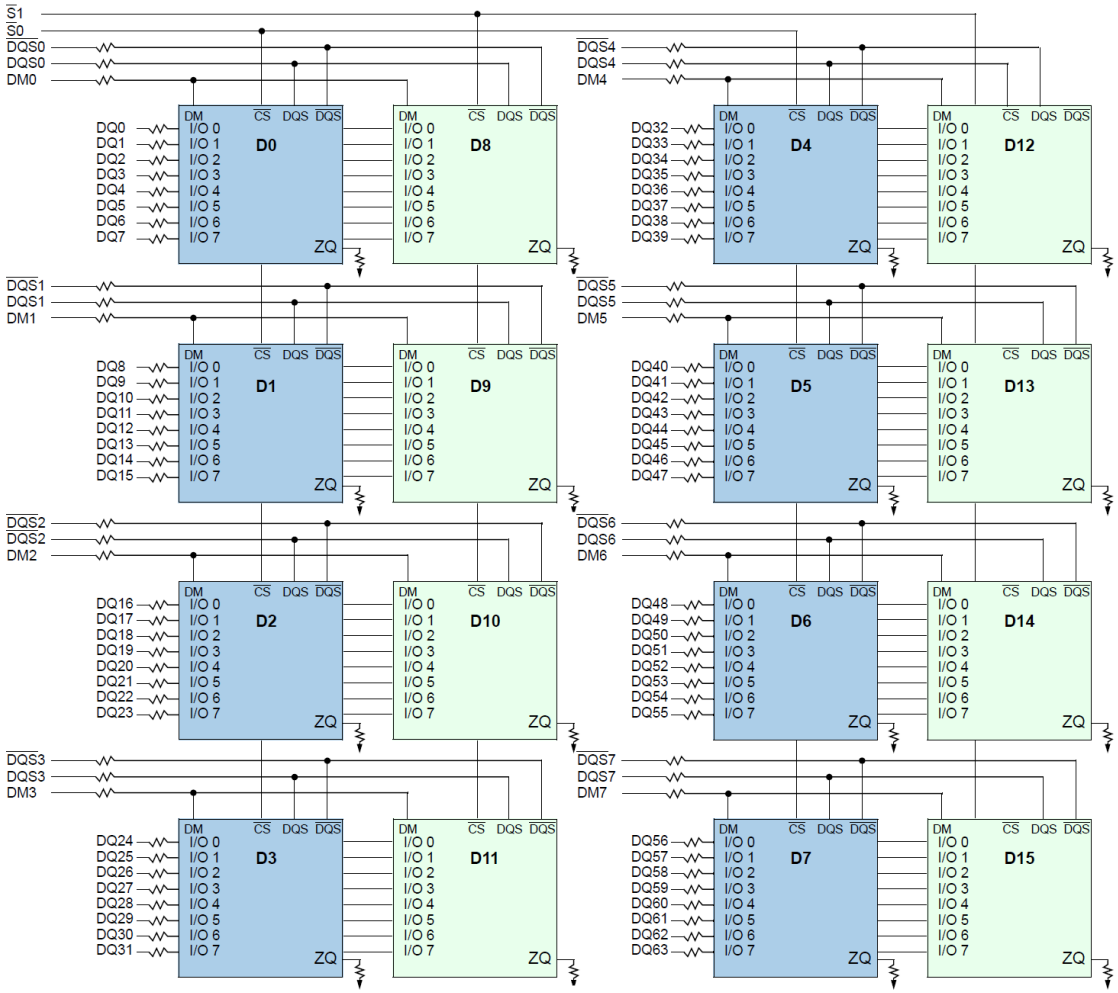


Figura 7: Diagramma funzionale di un modulo DIMM DDR3 da 2 Gbyte

3.3. Struttura di un chip DDR3

I 16 chip DDR3, che abbiamo visto comporre un modulo DIMM, a loro volta prevedono una suddivisione degli indirizzi al loro interno in una struttura diversa dal semplice array con un indice che spazia dalla prima all'ultima delle parole disponibili.

Ogni singolo chip [8.3] è infatti suddiviso in banchi, righe e colonne come sintetizzato dalla Figura 8.

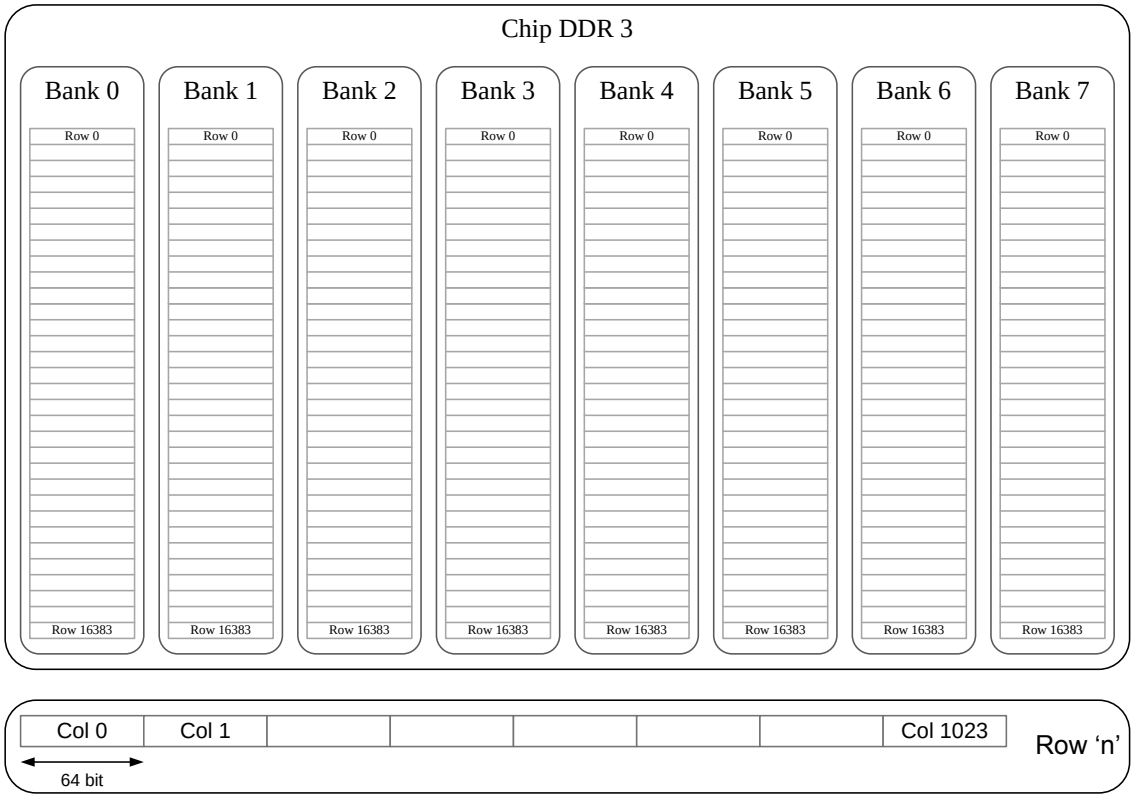


Figura 8: Organizzazione interna di un chip di memoria DDR3

Per indirizzare una particolare cella di memoria sarà necessario specificare gerarchicamente ed ordinatamente l'indirizzo della cella stessa. L'ordine viene richiesto dal fatto che le informazioni per l'indirizzo completo non vengono fornite contemporaneamente, ma in due fasi separate, dapprima si specificano simultaneamente il banco e la riga e successivamente (con un certo tempo di latenza) viene indicata la colonna.

3.4. Operazioni su memorie DDR

Come è stato detto nel paragrafo 3.1 le memorie di tipo dinamico sono dotate di una struttura estremamente semplice a favorire la densità di memoria totale.

Basando però la memorizzazione delle informazioni sulla presenza o meno di carica elettrica all'interno di un condensatore, che per ragioni di occupazione d'area dovrà assumere dimensioni il più possibile ridotte, questo tipo di memorie soffrono della naturale ed inesorabile perdita della carica immagazzinata durante la fase di scrittura, a causa delle correnti parassite che scaturiscono dal non perfetto isolamento dell'elemento atto a preservare la carica.

Per prevenire che la perdita di carica porti a degradare il contenuto della memoria è necessario che il memory controller esegua regolarmente delle operazioni di *refresh* con le quali le celle di memoria che hanno parzialmente perso della carica vengono ricaricate completamente, mentre quelle altre che teoricamente vuote hanno acquisito una ridotta quantità di carica elettrica possano essere nuovamente svuotate.

Oltre a questa complicazione dovuta alla struttura di una memoria dinamica, il memory controller dedicato dovrà farsi carico anche delle operazioni di ***activate*** e ***precharge*** legate all'organizzazione interna dei chip DDR3.

Ricordando che ognuno degli 8 banchi di memoria ospitano al loro interno una matrice di parole organizzate in righe e colonne, va aggiunto che le operazioni su una singola parola, definita dalla coppia (riga; colonna) richiedono che la riga che la contiene venga attivata mediante un comando *activate* con il quale l'intera riga viene predisposta ad essere letta o scritta.

Una volta conclusa l'operazione sulla parola specificata è possibile mantenere attiva la riga precedentemente utilizzata oppure disattivarla mediante un'operazione di *precharge*.

In particolar modo, il de-asservimento della riga attivata risulta necessario prima di attivare un'altra riga nello stesso banco di memoria o comunque prima di iniziare le operazioni di refresh dell'intera memoria.

3.5. Latenze

Le memorie DDR in generale, e per questo anche le versioni DDR3, richiedono un elevato tasso di sincronizzazione per tutte le operazioni, siano esse le classiche operazioni di lettura e scrittura, ma anche quelle di inizializzazione e quelle più specifiche descritte nel paragrafo 3.4.

Per fare ciò viene fatto largo uso di cicli di clock di attesa che determinano le latenze relative alle diverse operazioni; ad esempio tra l'operazione di attivazione di una riga e la successiva richiesta di lettura/scrittura possono dover intercorrere 7 cicli di clock, tra un'operazione di precharge ed un'attivazione possono dover intercorrere 9 cicli di clock.

I valori di tali latenze, che sono dichiarati sui rispettivi datasheet dei chip DDR3, determinano, a parità di altre caratteristiche, la bontà delle memorie, chip di memoria con latenze inferiori garantiranno infatti un accesso ai dati più rapido in lettura e scrittura.

3.6. Double Data Rate

Nel definire le memorie dinamiche ed in particolare le DDR, non è stato posto l'accento sul significato dell'acronimo Double Data Rate.

Con questo termine si identificano delle memorie capaci di gestire un flusso di informazioni ad una frequenza doppia di quella operativa.

Per meglio spiegare il loro funzionamento prendiamo le memorie di tipo DDR3-1066, esse funzionano ad una frequenza di clock pari a 533 MHz e tutti i cicli di latenza vengono definiti in base al clock di 533 MHz.

La denominazione 1066 vuol d'altronde evidenziare che questi chip di memoria, nelle operazioni di lettura, trasmettono il loro contenuto ad una frequenza doppia rispetto i 533MHz reali, e quindi proprio a 1066 MHz.

Per ottenere questo comportamento si usano due clock in quadratura entrambi a 533Mhz ed i dati, quando pronti, saranno trasmessi in corrispondenza dei fronti di salita di ciascuno dei due clock, ovvero in corrispondenza sia dei fronti di salita, sia dei fronti di discesa del clock principale.

4. Controller di memoria realizzato

Lo scopo del memory controller è di fare da ponte tra il bus di sistema, e quindi il microprocessore, ed i moduli di memoria, in modo da sgravare la CPU dal controllo di tutte quelle operazioni definite dagli standard delle memorie e che vengono ripetute frequentemente. Tali operazioni se fossero direttamente gestite dalla cpu, comporterebbero per la stessa un'eccessiva occupazione di tempo e conseguente riduzione delle capacità di calcolo.

Tra queste operazioni in particolare si evidenziano quelle d'inizializzazione delle memoria, tutti i controlli delle latenze relative a letture e scritture e le operazioni di refresh necessarie per evitare il degradarsi delle informazioni contenute nella memoria di tipo dinamico.

In questo capitolo verranno quindi descritti i compiti e le operazioni svolte dal controller di memoria e si tratteranno anche le problematiche strettamente legate al memory controller da realizzare per il nostro progetto.

4.1. Tipo e quantità di memoria DDR3

Tra le diverse scelte offerte dal mercato delle memoria dinamiche, per questo progetto di studio si è cercato di fare una scelta il più possibile generica e conservativa. Sono stati scelti dei moduli di memoria DIMM [8.4] della capacità di 2 GByte ciascuno e frequenza operativa di 533 Mhz (DDR3-1066).

In questo modo accoppiando due schede di memoria è possibile raggiungere i 4 GByte di memoria che, organizzati in parole da 32 bit richiedono un totale di 30 bit per essere indirizzati, esattamente quanti sono quelli dedicati alla memoria di tipo SDRAM in un sistema LEON3 (vedi Figura 6)

Tuttavia, la scelta di utilizzare gli stessi moduli di memoria utilizzati dai normali computer desktop comporta un'immediata complicazione legata proprio all'organizzazione di tali moduli DIMM.

Mentre il bus dati AHB prevede un'ampiezza pari a 32 bit, le parole all'interno dei moduli di memoria sono organizzate in modo da poter discriminare 64 bit per volta.

Per risolvere questa difficoltà verrà aggiunto al memory controller una memoria cache capace di essere indirizzata e modificata sia su parole da 32 bit, sia su quelle da 64 bit utilizzate dai moduli DIMM-DDR3.

Una più approfondita descrizione del problema, e del funzionamento di tale memoria cache saranno oggetto del paragrafo 4.2.3.

4.1.1. Indirizzamento della memoria

Abbiamo visto nei paragrafi 3.2 e 3.3 che la memoria DDR3 è suddivisa ed organizzata in diverse strutture; vediamo quindi nel caso specifico del nostro memory controller quanta memoria sarà disponibile e come questa dovrà essere organizzata.

Già nel paragrafo 2.6 si era anticipato che il memory controller avrebbe supportato 4 GByte di memoria DDR3, ma per ottenere una tale capacità ci sono innumerevoli combinazioni. Nel progetto realizzato sono stati utilizzati:

- due moduli DIMM-DDR3 ciascuno di capacità pari a 2 GByte

Entrambi i moduli sono di tipo:

- DUAL RANK, ovvero montano i chip di memoria su entrambe le facce del PCB

Queste caratteristiche comportano la presenza di alcune linee dedicate per l'abilitazione rispettivamente del modulo DIMM e di una delle facce dello stesso.

I segnali sono di tipo *CS (Chip Select)* e quindi saranno necessarie un totale di 6 segnali dedicati:

- Abilitazione DIMM 0
- Abilitazione RANK 0 del DIMM 0
- Abilitazione RANK 1 del DIMM 0
- Abilitazione DIMM 1
- Abilitazione RANK 0 del DIMM 1
- Abilitazione RANK 1 del DIMM 1

Questi segnali saranno ottenuti attraverso la decodifica dei 2 bit più significativi dell'indirizzo DDR3 secondo la tabella di conversione di Figura 9.

MSB indirizzo	CS DIMM 0	CS DIMM 1	CS RANK 0 DIMM 0	CS RANK 1 DIMM 0	CS RANK 0 DIMM 1	CS RANK 1 DIMM 1
00	0	1	0	1	X	X
01	0	1	1	0	X	X
10	1	0	X	X	0	1
11	1	0	X	X	1	0

Figura 9: Segnali di selezione di DIMM e RANK

Si nota quindi che i segnali (tutti attivi a livello basso) consentono l'attivazione, qualsiasi sia l'indirizzo richiesto, di uno solo del totale di quattro RANK disponibili.

Considerando anche la suddivisione della memoria in banchi righe e colonne, l'intero indirizzo può essere suddiviso come in Figura 10

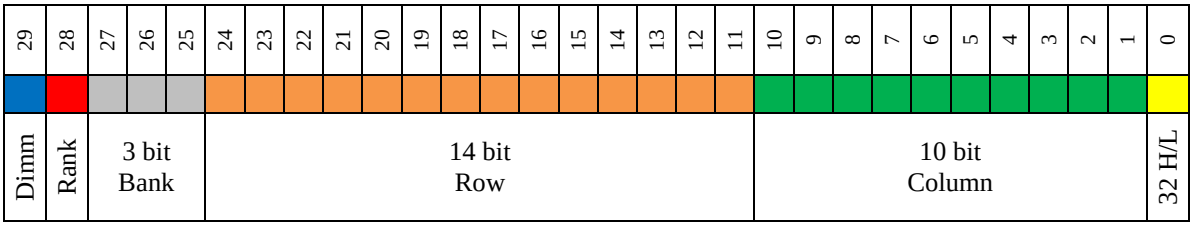


Figura 10: Suddivisione dell'indirizzo DDR3

Gli altri due bit che permettono di raggiungere il totale dei 32 bit usati dal bus AHB sono, ai fini del memory controller, utilizzati solo per verificare che la richiesta proveniente dal bus AHB sia diretta alla memoria DDR3 e secondo la tabella di Figura 6 sono costanti e di valore 00.

4.2. Sezioni del memory controller

Oltre a decodificare ed inoltrare le richieste di lettura e scrittura sulla memoria DDR3 provenienti dal bus AMBA, il memory controller che si sta sviluppando dovrà occuparsi anche della fase di inizializzazione dei moduli di memoria, e della gestione delle periodiche operazioni di refresh.

Per fare tutto questo la struttura stessa del memory controller è stata pensata e sviluppata in blocchi separati e dedicati alle varie operazioni:

reset controller (`reset_ctrl`)

si occupa dell'inizializzazione dei moduli di memoria, in particolare delle operazioni di attivazione dei diversi segnali di clock ed abilitazione che richiedono attenzioni particolari subito dopo la fase di accensione o comunque dopo ogni segnale di reset

refresh controller (`refresh_ctrl`)

tiene traccia del tempo intercorso dall'ultima operazione di refresh eseguita e qualora necessario provvede a richiedere ed eseguirne un'ulteriore

cache

è la memoria intermedia utilizzata sia per facilitare la conversione tra parole a 32 e 64 bit sia per migliorare le prestazioni generali del sistema

read controller (`read_ctrl`)

si occupa della lettura della memoria DDR3 ed il conseguente trasferimento di informazioni sulla memoria cache

sync controller (`sync_ctrl`)

è il modulo complementare al read controller, in quanto trasferisce il contenuto della cache all'interno della memoria RAM

top controller (`top_ctrl`)

il suo compito è quello di comunicare direttamente con il bus AHB e con tutti gli altri moduli, di decodificare le richieste e quindi di attivare le sottosezioni del memory controller necessarie per il completamento dell'operazione.

4.2.1. Reset Controller

La fase di attivazione dei moduli DIMM, viene di per se ben descritta dal diagramma temporale di Figura 11, nel quale possono identificarsi molteplici operazioni ed intervalli di attesa tra le stesse.

In una prima fase (che nel diagramma temporale si conclude in corrispondenza di T_d) viene curata la stabilità dei segnali destinati ai chip DDR3 e proprio in questa fase si possono evidenziare delle latenze molto elevate se rapportate al clock DDR3; i 500 µs richiesti tra la disattivazione del segnale di reset e l'abilitazione del clock corrispondono infatti ad oltre 267000 cicli di clock (il periodo del clock DDR corrisponde a 1,87 ns).

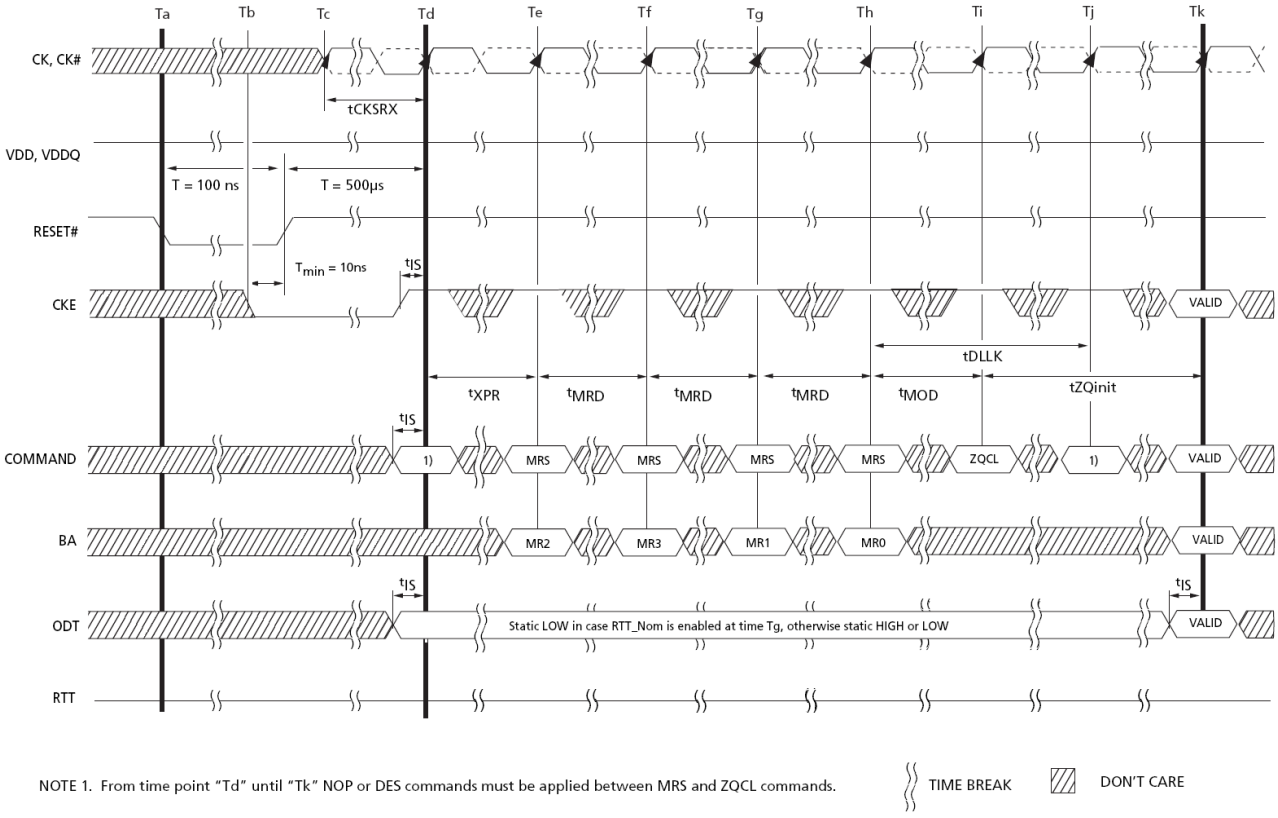


Figura 11: Operazioni di inizializzazione della memoria DDR3

Una volta attivato il segnale di clock ed abilitato lo stesso mediante `CKE (Clock Enable)`, vengono settati i parametri di funzionamento nei quattro registri delle memorie mediante la successione dei comandi `MRS (Memory Registry Set)`. Un ultimo comando (`ZQCL`) atto a calibrare i valori di impedenza e le terminazioni dei vari chip di memoria DDR3 particolarmente sensibili alla temperatura di funzionamento si

rivela necessario prima di poter operare le consuete operazioni di lettura e scrittura della memoria.

Per ottenere questa successione ordinata, il modulo `reset_ctrl` prevede un segnale che codifica e che tiene traccia dei vari stati di una macchina (a stati finiti) ed almeno un contatore utilizzato per ottenere i tempi d'attesa tra le varie operazioni.

Un'immediata considerazione può essere fatta riguardo ai tempi di latenza richiesti nelle operazioni di inizializzazione; nonostante questi siano molto elevati non inficeranno le prestazioni generali, in quanto si presenteranno solo in casi di riavvii del sistema, ovvero quando viene richiesto dall'esterno e dal microprocessore un reset.

Tuttavia gestire un contatore ad almeno 19 bit (per codificare il valore 267000) può rappresentare una difficoltà o comunque una scelta non ottimale, un clock dedicato o comunque poter sfruttare un clock disponibile nel sistema con un periodo maggiore potrebbe essere più efficace per tenere traccia di intervalli temporali di simili entità.

4.2.2. Read Controller

Il modulo `read_ctrl` si occupa della lettura di una porzione delle informazioni dalla memoria DDR3, e contestualmente della copia delle stesse direttamente sulla cache.

Per fare ciò il `read_ctrl` deve aver accesso a tutti i segnali di controllo della memoria RAM, quindi in particolare al bus di lettura scrittura dei dati, al bus indirizzi ed ai segnali di controllo che codificano i comandi da impartire.

Il diagramma temporale di Figura 12 descrive l'operazione di lettura focalizzando l'attenzione sulle operazioni svolte dal lato memoria DDR.

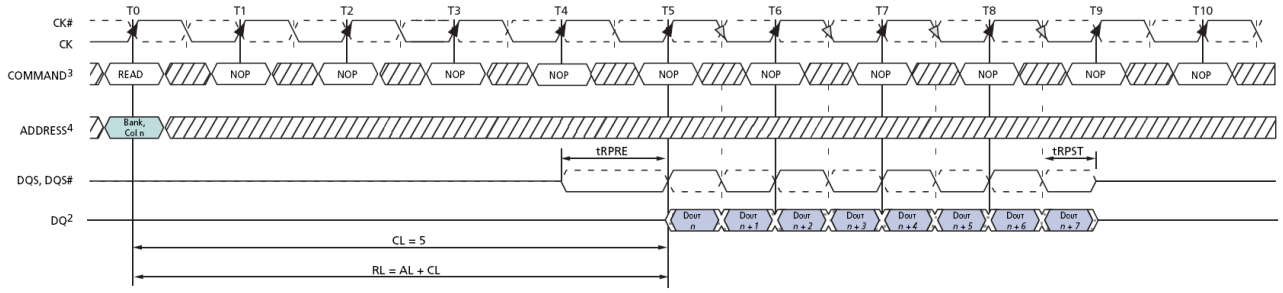


Figura 12: Diagramma temporale di un'operazione di lettura su memoria DDR3

Dapprima è necessario attivare la riga di memoria (del particolare banco) interessata dall'operazione di lettura, quindi, dopo un tempo di latenza t_{RCD} *(Row Column Delay)*, viene fornito il comando di lettura delle 8 celle contigue specificando al tempo stesso il banco e la colonna contenente la prima parola interessata dalla lettura.

Dopo un tempo di latenza t_{CL} *(column latency)* la memoria DDR genera il flusso delle parole da leggere, segnalando il passaggio da una parola all'altra mediante la variazione dei segnali *DQS (Data Strobe)* e *DQS#* (la versione negata di DQS).

È in questa circostanza che viene evidenziato il "double data rate", ovvero la fruizione di una quantità d'informazioni doppia nel singolo periodo di clock.

L'ordine, e le convenzioni adottate dallo standard DDR, con le quali queste parole vengono fornite in uscita dalla memoria verranno meglio descritte nel sottoparagrafo 4.2.2.2.

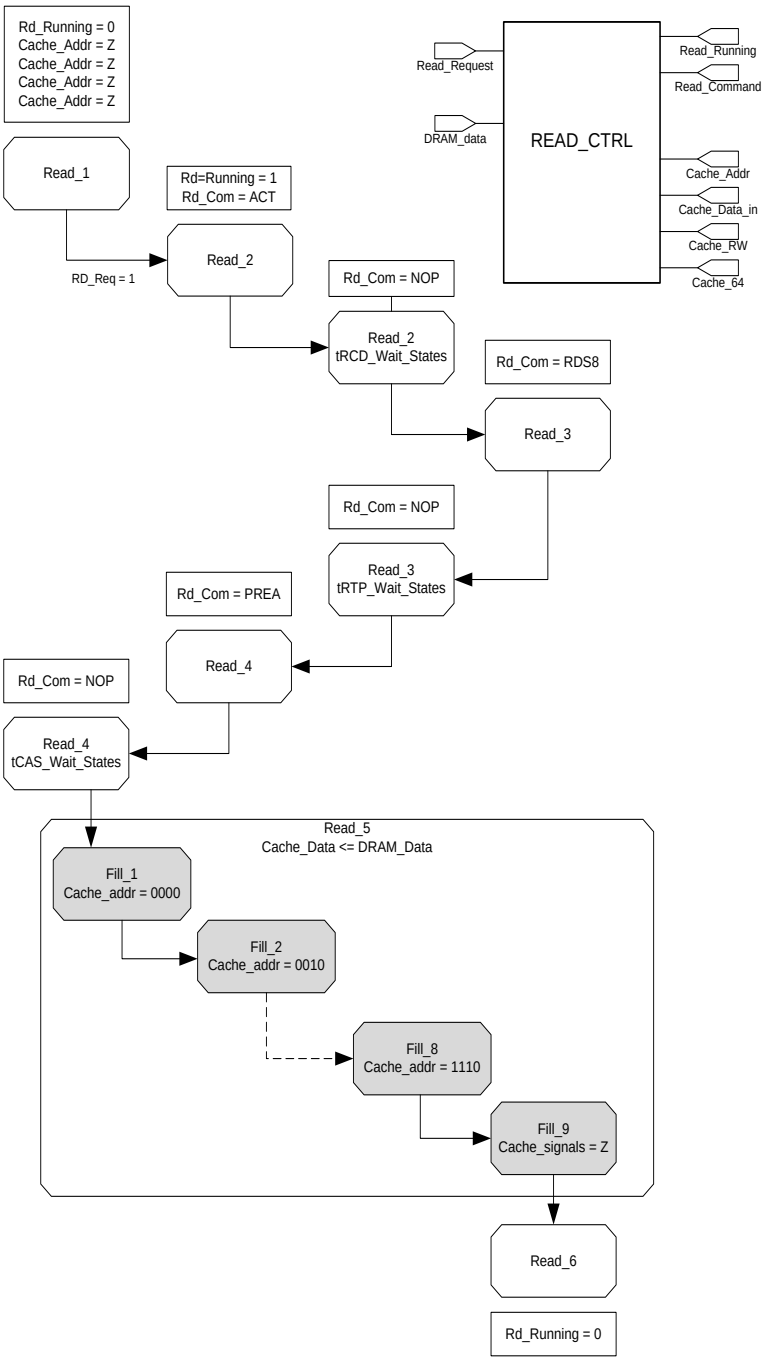


Figura 13: Struttura della macchina a stati finiti del `read_ctrl`

La Figura 13 rappresenta gli stati che attraversa la procedura di lettura dalla memoria DDR3. Ad ogni stato è associato il comando che il memory controller trasmette ai moduli di memoria (*NOP*, *PREA*, *RDS8*…) ed eventuali altre variazioni significative dei segnali di controllo (`rd_request, rd_running, cache_addr`).

La fase denominata "*Read5*" racchiude la ricezione dei dati dalla memoria DDR3 ed il loro inoltro verso la cache del memory controller (da cui segue la variazione del segnale `cache_addr` in ognuno degli step della categoria Fill)

4.2.2.1. Ottimizzazione del Precharge

Eventuali operazioni di lettura/scrittura sulla stessa riga precedentemente attivata non richiederebbero la riattivazione della stessa qualora queste richieste non avvengano oltre un tempo limite t_{RAS_MAX}.

Tener traccia di questo periodo, oltre che della riga attivata per ogni banco di memoria disponibile, comporterebbe l'aggiunta di registri, contatori e controlli i quali finirebbero solo per aumentare la complessità del controller di memoria senza per questo garantire un sostanziale e certo (inteso come sufficientemente probabile) miglioramento delle prestazioni.

Infatti, se è vero che risulterebbe possibile evitare l'operazione di attivazione ed il relativo intervallo prima del comando di lettura, è altrettanto vero che qualora la successiva lettura fosse richiesta su una riga diversa da quella attivata, allora l'operazione di attivazione andrebbe effettuata ugualmente facendola tuttavia precedere dal comando di *precharge* atto a disattivare l'ultima riga attivata di quello stesso banco di memoria.

Ecco quindi che predisporre un'operazione di precharge preventivo, che tra l'altro viene consentito tra la richiesta di lettura e la fruizione dei dati in uscita, diventa una scelta che al tempo stesso permette di semplificare la logica del controller e di migliorare le prestazioni dello stesso laddove le operazioni di lettura e scrittura non presentassero un elevato tasso di sequenzialità e si riferissero di volta in volta a righe diverse.

Il diagramma temporale di Figura 14 evidenzia la sequenza di operazioni che è stata ritenuta più opportuna seguire per la logica del controller di memoria.

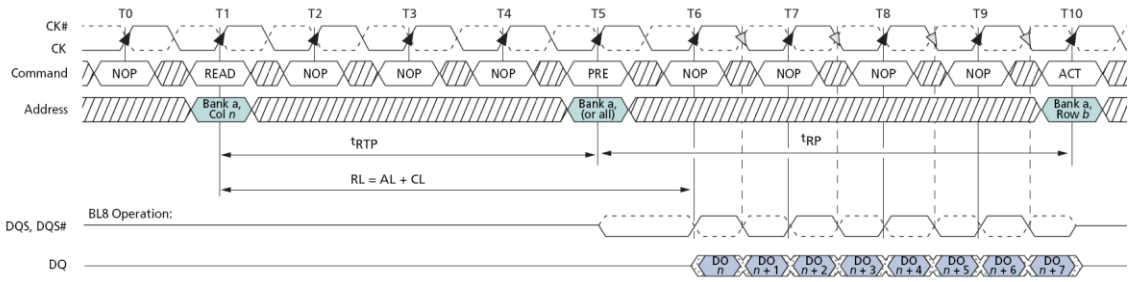


Figura 14: Diagramma temporale di un'operazione di lettura su memoria DDR3 con precharge

4.2.2.2. Sequenza delle celle di memoria lette e loro ordine

Le operazioni di lettura dalla memoria DDR3 vengono effettuate specificando un solo indirizzo (banco + rigo + colonna) ma, come rappresentato nei diagrammi temporali di Figura 12 e Figura 14 l'operazione di lettura si conclude dopo che vengono trasmessi i contenuti di ulteriori 7 celle di memoria oltre a quella specificata nella richiesta di lettura.

Per capire quali celle vengono lette oltre a quella specificata è necessario suddividere ogni riga della memoria DDR3 in gruppi di 8 colonne (parole da 64 bit), a questo punto, richiedendo la lettura di una cella, la memoria DDR3 fornirà all'esterno in successione il contenuto di tutte le celle del gruppo a cui appartiene l'indirizzo originariamente richiesto.

	Indirizzi Memoria							
GruppoA	*0*	*1*	*2*	*3*	*4*	*5*	*6*	*7*
GruppoB	*8*	*9*	*10*	*11*	*12*	*13*	*14*	*15*
GruppoC	*16*	*17*	*18*	*19*	*20*	*21*	*22*	*23*
…								

Figura 15: Ulteriore suddivisione delle righe di una memoria DDR3

Tuttavia, l'ordine con cui queste 8 celle vengono trasmesse all'esterno non è necessariamente quello crescente a cui si potrebbe pensare, ma bensì cambia a seconda della posizione relativa all'interno del gruppo che assume la cella originariamente richiesta. Tale posizione è in pratica determinata dai 3 bit meno significativi identificanti la colonna e viene ben riassunta dalla Figura 16.

Burst Length	READ/ WRITE	Starting Column ADDRESS (A2,A1,A0)	burst type = Sequential (decimal) A3 = 0	burst type = Interleaved (decimal) A3 = 1
8	READ	0 0 0	0,1,2,3,4,5,6,7	0,1,2,3,4,5,6,7
		0 0 1	1,2,3,0,5,6,7,4	1,0,3,2,5,4,7,6
		0 1 0	2,3,0,1,6,7,4,5	2,3,0,1,6,7,4,5
		0 1 1	3,0,1,2,7,4,5,6	3,2,1,0,7,6,5,4
		1 0 0	4,5,6,7,0,1,2,3	4,5,6,7,0,1,2,3
		1 0 1	5,6,7,4,1,2,3,0	5,4,7,6,1,0,3,2
		1 1 0	6,7,4,5,2,3,0,1	6,7,4,5,2,3,0,1
		1 1 1	7,4,5,6,3,0,1,2	7,6,5,4,3,2,1,0
	WRITE	V,V,V	0,1,2,3,4,5,6,7	0,1,2,3,4,5,6,7

Figura 16: Sequenze possibili per la trasmissione delle celle di memoria

Non riscontrando particolari benefici ricavabili dall'utilizzo di una sequenza non convenzionale nell'implementazione del memory controller così strutturato (che in particolar modo prevede una memoria cache), e vista la differenza di comportamento tra la lettura e la scrittura (che a differenza di ciò che accade nella lettura non ammette sequenze diverse da quella strettamente crescente) ecco che, sia le operazioni di lettura, sia quelle di scrittura vengono effettuate mascherando i tre bit meno significativi per garantire uniformità di funzionamento in tutte le situazioni ed ottenere quindi il contenuto di celle crescenti nell'indirizzo

Nei paragrafi 4.2.3 e 4.2.6 dedicati alla memoria cache ed al top controller verrà sottolineato il motivo per cui ottenere l'informazione richiesta come prima o ultima di una successione di 8 celle non influenza minimamente le prestazioni del memory controller sviluppato.

4.2.3. Cache

La motivazione principale per l'introduzione di questo modulo è da ricercare nella necessità di poter indirizzare parole formate da 32 bit usando moduli di memoria strutturati per ospitare ed indirizzare parole di lunghezza doppia.

Effettuare una lettura di 32 bit su una parola da 64, non comporta altro che disporre di un bit ulteriore per definire l'indirizzo e, mediante questo, una volta in possesso dell'intera parola da 64 bit, selezionare la metà della parola di interesse scartando l'altra metà.

Non così a buon mercato si presenta invece la soluzione per la scrittura della metà di una parola a 64 bit; per fare ciò infatti risulterebbe necessario leggerne l'intero contenuto, modificarne la metà che ospita la parola da 32 bit da aggiornare e quindi riscrivere l'intera parola a 64 bit con metà della stessa che di fatto è rimasta inalterata.

La naturale evoluzione della soluzione appena proposta (che in termini fisici prevede una memoria tampone da 64 bit) non è altro quella di ampliare tale memoria temporanea in modo da non dover scartare le altre 7 parole fornite in fase di lettura della DDR (vedi Paragrafo 4.2.2.2) prevedendo una memoria cache di dimensioni multiple di 512 bit (512 = 8*64bit).

Oltre a questa necessità tuttavia, l'aggiunta di una memoria cache, viste anche tutte le latenze presenti nelle operazioni che coinvolgono la memoria DDR3 porta anche notevoli benefici prestazionali quando si verificano degli accessi successivi a celle di memoria con un certo grado di località spaziale (indirizzi vicini). In questo caso infatti sarà possibile operare temporaneamente sulla memoria cache e solo successivamente sincronizzarne il contenuto con la memoria DDR3.

È immediato comprendere come una memoria cache di dimensioni maggiori possa far diminuire la probabilità che si verifichi la necessità di sincronizzarne il contenuto con la memoria di sistema, tuttavia, considerato anche il fatto che aumentare le dimensioni della cache comporterà altresì un aumento dell'area occupata da questo modulo ed una maggior difficoltà a tener traccia delle sezioni di memoria replicate su di essa, inizialmente è stato scelto di dimensionare la memoria cache con un valore minimo che ben si adatta alle operazioni di lettura e scrittura della DDR.

Il modulo cache realizzato prevede quindi la presenza di 16 celle da 32 bit leggibili e modificabili singolarmente oppure a gruppi di due per risultare compatibili tanto al bus AHB, quando ai moduli DIMM-DDR3.

Per distinguere una delle 8 parole da 64 bit saranno necessari 3 soli bit [A_3-A_2-A_1], mentre un quarto bit [A_0] discriminerà la metà più significativa da quella meno significativa di una stessa parola da 64 bit; la Figura 17 riassume l'organizzazione degli indirizzi

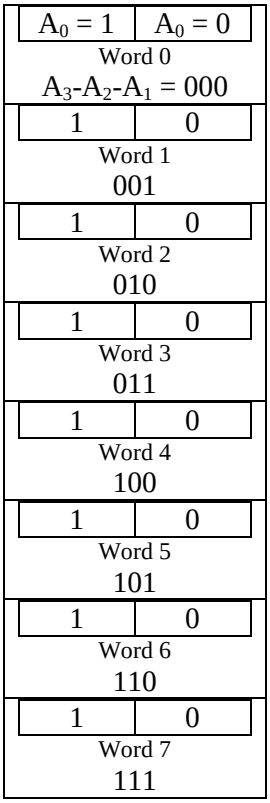


Figura 17: Organizzazione degli indirizzi nella memoria cache

La struttura e l'organizzazione di queste 16 celle devono permettere che le coppie costituenti una parola da 64 bit vengano lette o scritte contemporaneamente e con una frequenza doppia rispetto a quelle del clock DDR3 (visto che i dati vengono inviati dalla memoria al memory controller su entrambi i fronti del clock stesso).

Le due metà di una parola a 64 bit dovranno perciò far parte di due diversi array di celle da 32 bit cosicché possano essere lette e modificate contemporaneamente e non con due accessi diversi qualora condividessero l'array stesso.

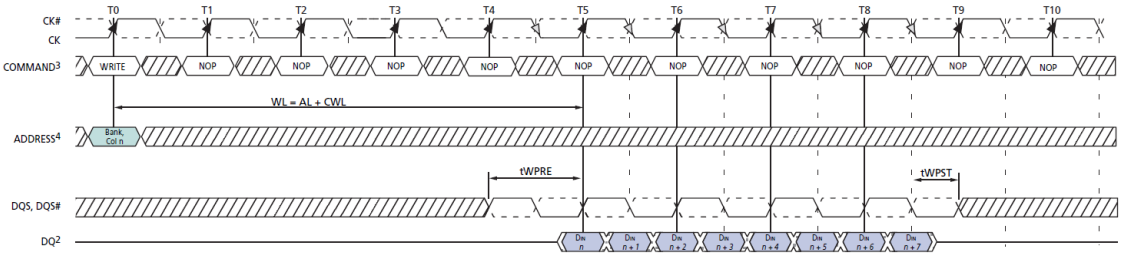
Word 0 H	Word 0 L
Word 1 H	Word 1 L
Word 2 H	Word 2 L
Word 3 H	Word 3 L
Word 4 H	Word 4 L
Word 5 H	Word 5 L
Word 6 H	Word 6 L
Word 7 H	Word 7 L

Figura 18: Organizzazione della memoria cache

4.2.4. Sync controller

Il `sync_ctrl` si occupa dell'operazione di trasferimento della cache verso la memoria DDR, in pratica gestisce l'operazione di scrittura delle 16 celle della memoria cache nelle equivalenti 8 celle della memoria DDR.

La logica di funzionamento e tutte le considerazioni fatte per ciò che riguarda il `read_ctrl` possono essere direttamente trasposte anche a questo modulo e qui di seguito verranno evidenziate le sole differenze.



Comandi e segnali DDR per la scrittura di un burst di 8 celle

Nella fase di scrittura della memoria DDR i segnali *DQS* e *DQS#* dovranno essere pilotati dal memory controller (si ricorda che nella fase di lettura, e quindi nel `read_ctrl`, le variazioni dei segnali DQS vengono utilizzate dal memory controller come indicatori dell'arrivo di una nuova parola di memoria e quindi le variazioni venivano rilevate e non generate).

Ovviamente anche le operazioni sulla memoria cache saranno complementari, in quanto durante la sincronizzazione la cache subirà due letture contemporanee di celle da 32 bit in ogni semiperiodo del segnale di clock DDR ed il loro contenuto correttamente ri-assemblato verrà indirizzato verso il data bus DDR a 64 bit.

4.2.5. Refresh controller

A differenza dei moduli di lettura e sincronizzazione, il modulo adibito al refresh presenta una costante attività e non richiede che sia il top controller ad abilitarlo, infatti per mantenere traccia del tempo restante prima che una nuova operazione di refresh debba avere luogo, un contatore viene decrementato ad ogni ciclo di clock della memoria DDR.

Visto che la richiesta di operare un refresh partirà questa volta dal modulo dedicato stesso, e vista l'importanza di tale operazione di mantenimento delle informazioni contenute nella memoria DDR, ecco che il controller generale (`top controller`) che sta sopra ai diversi moduli dovrà essere in grado di riconoscere la richiesta/necessità di operare un refresh e trattare la stessa con la dovuta priorità rispetto a qualsiasi altra operazione.

Una considerazione ulteriore va fatta in merito allo stato del memory controller durante le operazioni di refresh; è ben chiaro che le operazioni che richiedono l'accesso diretto alla memoria DDR dovranno rimanere in uno stato di attesa fino a quando l'operazione di refresh non abbia portato a termine i suoi compiti, ancor meglio, è bene tener conto che lo scadere del tempo massimo tra due refresh potrebbe avvenire proprio durante un'operazione di accesso alla memoria, quindi, onde evitare di interrompere un'operazione di lettura/scrittura, risulta conveniente mantenere un certo margine sul timer che gestisce l'operazione di refresh così da poter comunque concludere l'operazione iniziata prima di iniziare la procedura di refresh.

Assunto ciò, durante le operazioni di refresh, che per via dei soliti tempi di latenza non saranno immediate, gli altri moduli non avranno accesso diretto alla memoria DDR, ma le operazioni che coinvolgono la sola memoria cache potrebbero comunque avere luogo.

Tuttavia, questo genere di accortezze e migliorie verranno demandate ad una seconda fase di progettazione, dando priorità piuttosto all'ottenimento di un sistema completo e funzionante seppur migliorabile nelle prestazioni.

4.2.6. Top Controller

Oltre a decodificare le richieste provenienti dal bus AHB indirizzate alla memoria DDR3 e rispondere alle stesse non appena il dato richiesto è disponibile, il top controller si occupa anche di modificare lo stato di attività ed inattività di tutti gli altri moduli del memory controller evitando possibili conflitti sulla gestione di segnali condivisi tra più moduli.

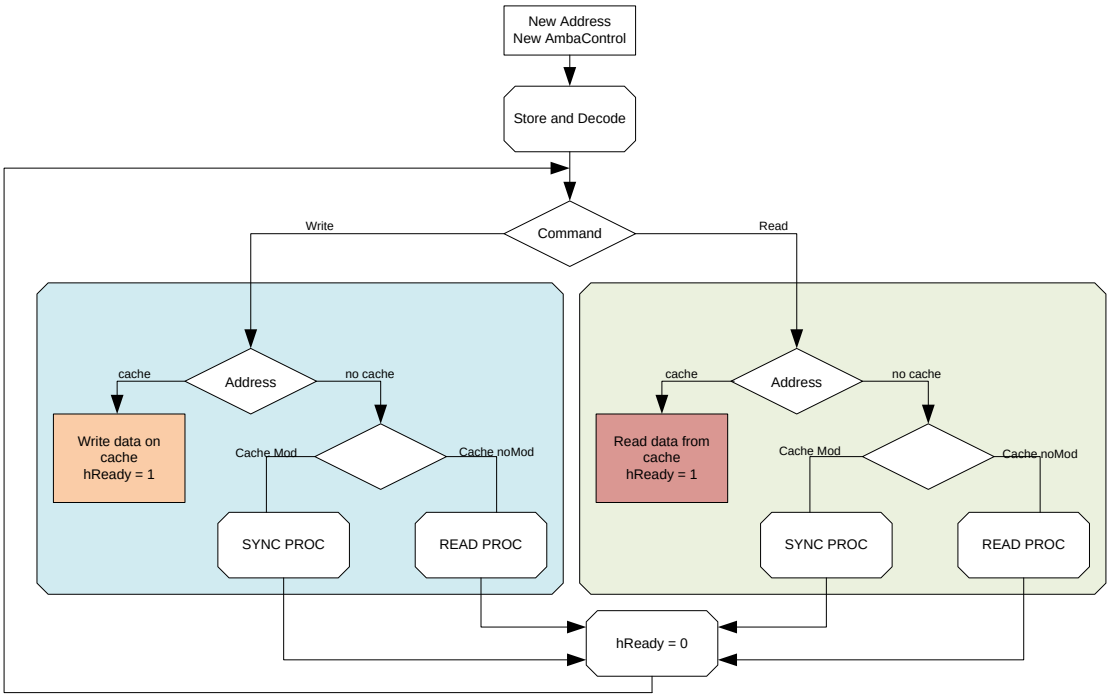


Figura 19: Diagramma di flusso per la decodifica delle richieste AMBA

Come rappresentato dal diagramma di Figura 19 si fa notare che le operazioni di lettura e di scrittura richieste dal bus AMBA trovano risposta solo quando la cella indirizzata è replicata nella memoria cache e quindi qualora venga dapprima richiesta la lettura di una porzione della memoria DDR, la risposta al microprocessore verrà fornita con almeno un ciclo di clock AMBA di ritardo, in quanto alla prima richiesta il memory controller non sarà in grado di dare una risposta immediata.

Uno dei possibili conflitti che possono verificarsi è quello relativo ai comandi che i moduli che gestiscono il reset, il refresh, la lettura e la sincronizzazione della memoria DDR devono fornire ai chip di memoria stessi.

Questi comandi passeranno attraverso un multiplexer comandato proprio dal top controller che si occuperà inoltre di ricodificare il comando dalla convenzione usata

internamente al memory controller agli standard legati alle memorie DDR3 (standard riassunti nella tabella di Figura 20.

Function	Abbreviation	CKE		CS#	RAS#	CAS#	WE#	BA0-BA2	A13-A15	A12-BC#	A10-AP	A0-A9, A11	Notes
		Previous Cycle	Current Cycle										
Mode Register Set	MRS	H	H	L	L	L	L	BA	OP Code				
Refresh	REF	H	H	L	L	L	H	V	V	V	V	V	
Self Refresh Entry	SRE	H	L	L	L	L	H	V	V	V	V	V	7,9,12
Self Refresh Exit	SRX	L	H	H	X	X	X	X	X	X	X	X	7,8,9, 12
				L	H	H	H	V	V	V	V	V	
Single Bank Precharge	PRE	H	H	L	L	H	L	BA	V	V	L	V	
Precharge all Banks	PREA	H	H	L	L	H	L	V	V	V	H	V	
Bank Activate	ACT	H	H	L	L	H	H	BA	Row Address (RA)				
Write (Fixed BL8 or BC4)	WR	H	H	L	H	L	L	BA	RFU	V	L	CA	
Write (BC4, on the Fly)	WRS4	H	H	L	H	L	L	BA	RFU	L	L	CA	
Write (BL8, on the Fly)	WRS8	H	H	L	H	L	L	BA	RFU	H	L	CA	
Write with Auto Precharge (Fixed BL8 or BC4)	WRA	H	H	L	H	L	L	BA	RFU	V	H	CA	
Write with Auto Precharge (BC4, on the Fly)	WRAS4	H	H	L	H	L	L	BA	RFU	L	H	CA	
Write with Auto Precharge (BL8, on the Fly)	WRAS8	H	H	L	H	L	L	BA	RFU	H	H	CA	
Read (Fixed BL8 or BC4)	RD	H	H	L	H	L	H	BA	RFU	V	L	CA	
Read (BC4, on the Fly	RDS4	H	H	L	H	L	H	BA	RFU	L	L	CA	
Read (BL8, on the Fly)	RDS8	H	H	L	H	L	H	BA	RFU	H	L	CA	
Read with Auto Precharge (Fixed BL8 or BC4)	RDA	H	H	L	H	L	H	BA	RFU	V	H	CA	
Read with Auto Precharge (BC4, on the Fly)	RDAS4	H	H	L	H	L	H	BA	RFU	L	H	CA	
Read with Auto Precharge (BL8, on the Fly)	RDAS8	H	H	L	H	L	H	BA	RFU	H	H	CA	
No Operation	NOP	H	H	L	H	H	H	V	V	V	V	V	10
Device Deselected	DES	H	H	H	X	X	X	X	X	X	X	X	11
Power Down Entry	PDE	H	L	L	H	H	H	V	V	V	V	V	6,12
				H	X	X	X	X	X	X	X	X	
Power Down Exit	PDX	L	H	L	H	H	H	V	V	V	V	V	6,12
				H	X	X	X	X	X	X	X	X	
ZQ Calibration Long	ZQCL	H	H	L	H	H	L	X	X	X	H	X	
ZQ Calibration Short	ZQCS	H	H	L	H	H	L	X	X	X	L	X	

Figura 20: Tavola di verità per la codifica dei comandi DDR

In realtà alcuni comandi che non vengono implementati in questa particolare realizzazione di memory controller possono essere omessi dalla codifica per facilitare la sintesi finale; ad esempio è possibile limitare le operazioni di lettura e di scrittura a quelle burst BL8 con auto-precharge e non prevedere le altre risparmiando la codifica di 10 delle 12 modalità rese disponibili dallo standard DDR3.

Per gestire l'attivazione degli altri moduli è stato previsto per ognuno un segnale `request` ed un segnale `running` il primo usato dal top controller per richiedere l'attivazione del modulo, il secondo invece pilotato dal sotto modulo per comunicare lo stato di attività dello stesso.

Per prevenire la possibilità che un qualsiasi modulo possa mantenere occupato l'intero memory controller per un tempo superiore ad un limite prestabilito e dimensionato per garantire l'esecuzione normale di tutti i processi.

Qualora questo watchdog raggiungesse il limite consentito, verrebbe riattivata la procedura d'inizializzazione.

4.3. Visione d'insieme

Lo schema a blocchi riportato in Figura 21 permette di sintetizzare la struttura e le connessioni presenti tra i vari moduli costituenti il memory controller.

Vengono evidenziati i segnali di controllo ed i vari bus generati mentre per non appesantire oltremodo la rappresentazione, i segnali dedicati esclusivamente alla gestione della memoria DDR o delle comunicazioni col bus AMBA vengono genericamente indicati come *Control Signals*.

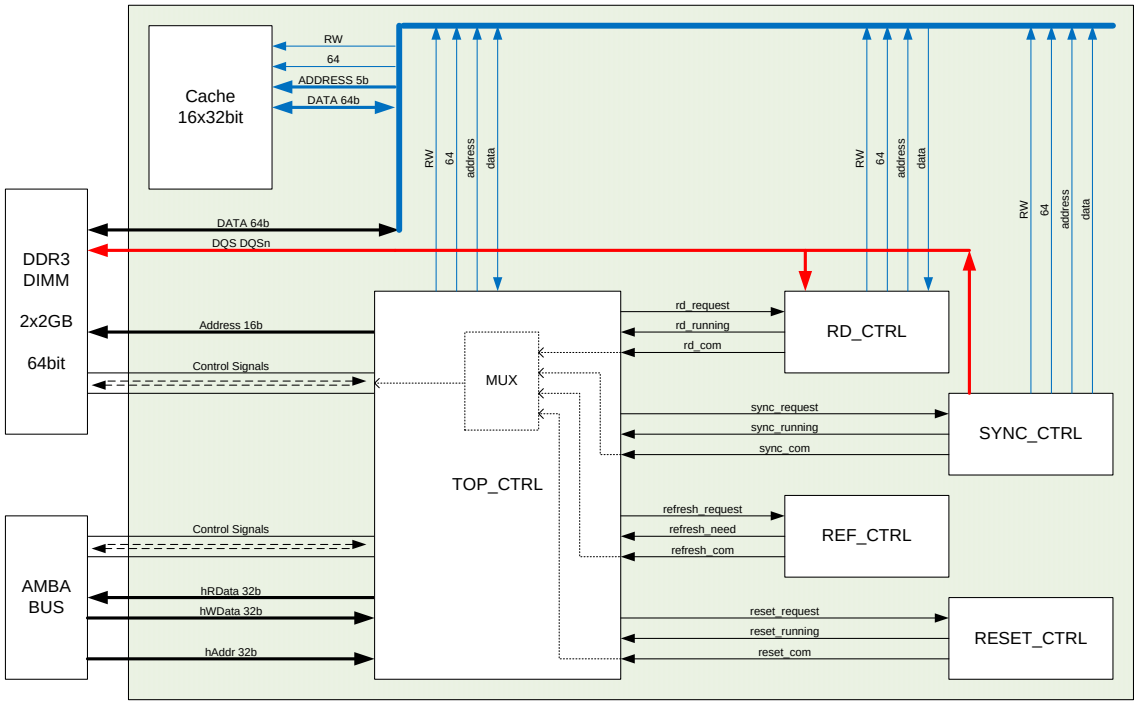


Figura 21:Rappresentazione dei moduli e delle principali interconnessioni

5. Analisi dei risultati ottenuti

In questo capitolo verranno analizzati i comportamenti del memory controller progettato ottenuti nelle diverse fasi di simulazione e verifica.

Per verificare il funzionamento del memory controller è stato utilizzato un testbench che ricoprisse tutte le funzioni eseguibili dal dispositivo sotto test.

5.1. Testbench utilizzato

Il testbench realizzato per le verifiche e le simulazioni si occupa di generare i vari segnali di clock (sia AMBA, sia DDR) ed i vari segnali di richiesta normalmente generati dal dispositivo Master AMBA.

I segnali sono stati programmati affinché si procedesse dapprima ad un reset completo del memory controller, necessario sia per l'inizializzazione di tutti i registri del memory controller, sia per verificare la procedura di reset dello stesso memory controller, una volta conclusa questa fase, con una successione di scritture sequenziali su celle di memoria adiacenti, il testbench simula la scrittura e modifica dell'intera cache, dopodiché le stesse celle di memoria vengono lette così da eseguire le operazioni di lettura della cache ed al tempo stesso verificare che i dati scritti precedentemente non abbiano subito alterazioni.

Con una richiesta di scrittura ulteriore, questa volta su un indirizzo non replicato sulla cache, si testa la procedura di sincronizzazione della cache con la memoria DDR ed allo stesso tempo la lettura di una nuova porzione della stessa.

In generale, i test che coinvolgono i dati contenuti nella memoria DDR si sono rivelati particolarmente difficoltosi da verificare. Realizzare dei segnali di risposta dal testbench non avrebbe molto senso, mentre l'integrazione di un componente aggiuntivo (un modello comportamentale di un modulo di memoria fornito direttamente dal produttore) capace di simulare i vari segnali della memoria DDR nel testbench stesso risulta una procedura tanto efficace quanto complessa, che può essere attuata in uno stadio della progettazione più vicino alla realizzazione del primo prototipo.

5.2. Pre Sintesi

Le simulazioni pre-sintesi consentono di verificare il codice affinché il comportamento descritto e redatto sia coerente con quanto fissato a priori.

Dopo una prima fase di correzione degli errori pacchiani, si sono ripercorsi molteplici cicli di basati su

- Simulazione
- Analisi
- Affinazione del codice

Le simulazioni hanno coinvolto il sistema nella sua interezza, ma laddove le criticità si sono rivelate più ardue da risolvere, si sono sfruttati alcuni testbench specifici per avere maggior visibilità sui segnali di alcuni moduli e velocizzare i tempi di simulazione evitando focalizzando l'attenzione solo su alcuni segnali ed eliminando dalla totalità delle procedure simulate normalmente quelle che non interessano la particolare sezione sotto test.

In questa fase, le verifiche dei risultati si sono concentrate principalmente sul comportamento dei segnali e sull'attivazione dei moduli, interni al controller di memoria, coerentemente all'azione richiesta.

Affinamenti successivi hanno consentito di descrivere il memory controller in modo che tutte le combinazioni di richieste, anche in rapida successione provenienti dal bus AHB potessero essere elaborate correttamente.

Si sono quindi verificate e confrontate diverse soluzioni per descrivere al meglio alcune operazioni, in modo da rispettare i vincoli richiesti, pur mantenendo un certo ordine nel codice atto a favorirne la comprensione.

Nei paragrafi 5.2.1e 5.2.2 viene descritta l'evoluzione che hanno subito i processi di lettura dalla memoria RAM, e di, attivazione e gestione per mezzo del `top_ctrl` dei moduli sussidiari, che per l'appunto hanno richiesto alcune revisioni prima di determinare la soluzione finale.

Un ulteriore affinamento del codice ha permesso di eliminare alcune strutture ridondanti, ed un certo insieme di segnali e registri utilizzati per i soli scopi di simulazione e monitoraggio delle attività del memory controller.

Altri aspetti, quali la verifica dei cicli di latenza, hanno richiesto meno sacrifici ed aggiustamenti in quanto questo genere di interventi sul codice, non andando ad influenzare il comportamento dei segnali di interfacciamento tra i vari moduli, risultavano del tutto trasparenti a tutti gli altri moduli e non provocando modifiche a cascata sugli altri moduli.

5.2.1. Lettura dalla memoria RAM

Descriviamo due diverse soluzioni che sono state intraprese in fase di progettazione al fine di realizzare una delle principali operazioni richieste al memory controller, la lettura di un indirizzo di memoria.

Una volta appurato che l'indirizzo richiesto non è replicato all'interno della memoria cache, e che quindi sarà richiesta una lettura diretta dalla memoria RAM, il `top_ctrl` dovrà fornire tutte le variabili necessarie al `read_ctrl` ed attivarlo.

Quest'ultimo, una volta in possesso del contenuto della memoria, potrebbe essere realizzato in modo da fornire l'informazione richiesta direttamente sul bus di sistema e successivamente comunicare al `top_ctrl` la cessazione delle sue attività.

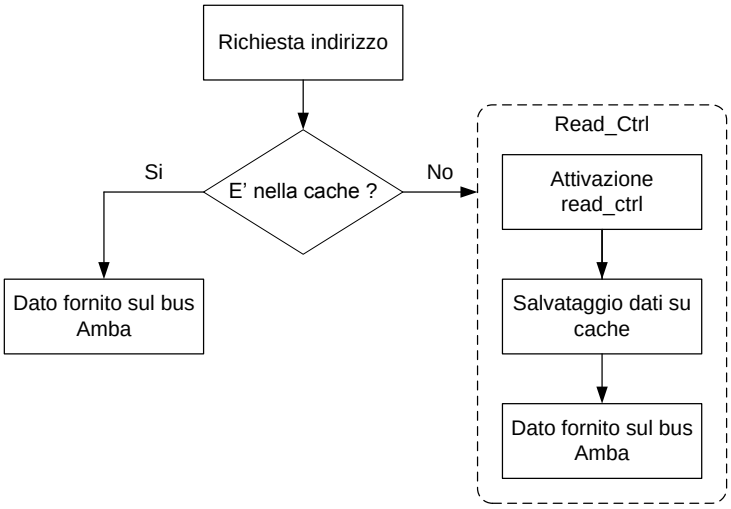


Figura 22: Opzione 1 per la lettura di un nuovo indirizzo dalla memoria RAM

Una tale soluzione tuttavia richiederebbe al `read_ctrl` di lavorare con tre diverse sincronizzazioni per poter trasmettere il dato sul bus AHB (1), per poter acquisire i dati provenienti dalle memoria DDR3-1066 (2) e per comunicare con la memoria cache (3) su cui scrivere il contenuto delle restanti celle lette con il medesimo comando di lettura.

Limitando le comunicazioni con il bus AHB al solo `top_ctrl` ed inibendole a tutti gli altri moduli, si riesce al tempo stesso a:

- ridurre il pericolo di generare conflitti sul bus, a semplificare il funzionamento del `read_ctrl` che non dovrà più preoccuparsi della sincronizzazione col bus di sistema ma solo con cache e memoria DDR3
- rendere più semplici e somiglianti tra loro `read_ctrl` e `sync_ctrl`.

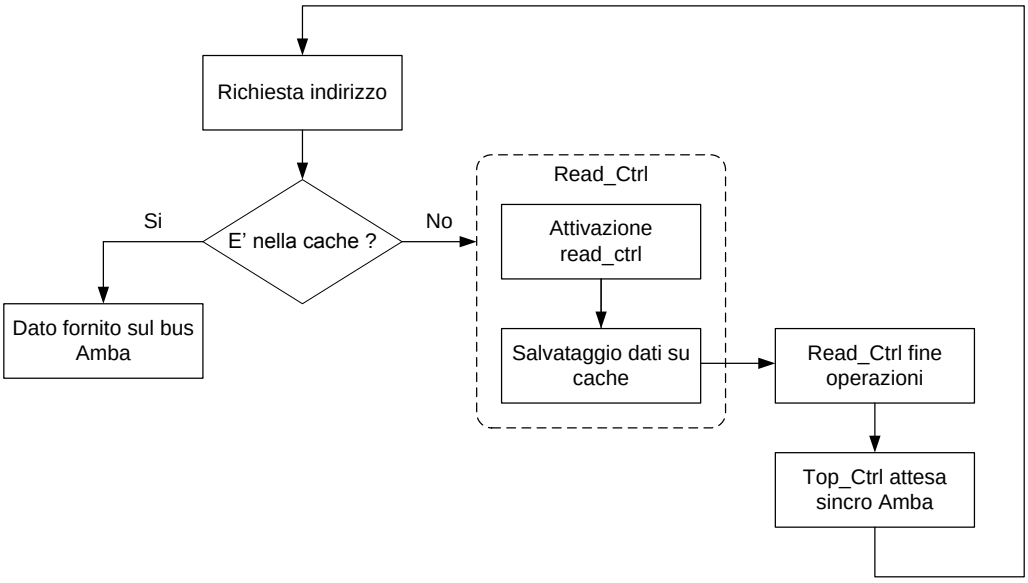


Figura 23: Opzione 2 per la lettura di un nuovo indirizzo dalla memoria RAM

Il passaggio dalla prima versione alla seconda, come si può ben comprendere, ha comportato modifiche a tutti i blocchi che prevedevano interfacce verso il bus AHB.

5.2.2. Comportamento dei segnali di running

Il `top_ctrl` che si occupa di coordinare tutti gli altri moduli è interfacciato con ognuno di essi mediante un segnale `request`, per attivare il modulo associato, ed un segnale `running`, attraverso il quale viene informato dello stato operativo del modulo stesso.

Questi segnali non sono però sufficienti da soli a discriminare delle situazioni diverse che si vengono a creare quando tutti i segnali `running` sono a livello basso.

I segnali `running` possono essere infatti tutti disattivi quando:

- La situazione di inattività perdura nel tempo ed è costante (idle)
- Il modulo attivo (ricordiamo che solo un modulo per volta può essere attivo onde evitare conflitti) conclude le sue operazioni (rientro da un'operazione conclusa)
- Un modulo ha ricevuto la richiesta di attivazione, ma in attesa della sincronizzazione con un diverso fronte di clock non ha ancora provveduto ad attivare il segnale `running`.

Particolare attenzione deve essere posta proprio all'ultimo caso, infatti con un'errata interpretazione si rischia di vedere bypassata l'esecuzione del processo richiesto qualora il livello basso simultaneo di tutti i segnali `running` venga immediatamente inteso dal `top_ctrl` come una prematura conclusione della procedura richiesta.

Per evitare ciò è stato aggiunto un flag (`start_proc_lat`) ed un contatore che inibiscono la decodifica di tutti i segnali `running` a livello basso come il rientro da una procedura conclusasi, ed un ulteriore registro che tenendo traccia dell'operazione attivata permette di combinare azioni ulteriori in fase di conclusione della stessa come descritto nella porzione di codice riportata in appendice [7.1]

5.3. Post Sintesi

Una volta raggiunto uno stadio dello sviluppo che garantisse le risposte ed i comportamenti previsti e ricercati agli ingressi forniti in fase di pre-sintesi, si è iniziato lo step successivo di test composto da ripetizioni cicliche delle seguenti fasi:

- Compilazione e sintesi del circuito
- Simulazione del circuito sintetizzato
- Analisi dei risultati
- Modifica delle opzioni del compilatore
- Modifiche ai sorgenti
    - Simulazione pre-sintesi
    - Analisi del comportamento pre-sintesi

Modifiche consistenti ai file sorgenti in questa fase, apportate per correggere alcuni aspetti emersi dai test post sintesi, richiedono una ripetizione anche dei test comportamentali precedentemente svolti, risultando particolarmente dispendiose in termini di tempi di progettazione e validazione dei risultati.

Giunti a questo punto della progettazione bisogna inoltre tener conto che le operazioni di compilazione e simulazione post-sintesi comportano notevoli tempi di attesa non bypassabili volendo garantire la veridicità dei risultati; un esempio di questa problematica sono i tempi di attesa nella fase di reset incredibilmente elevati paragonati al periodo di clock che non possono essere artificialmente ridotti onde evitare che il compilatore sintetizzi il contatore come un sommatore ad 8 bit piuttosto che a 30 bit come richiesto; lasciare che il simulatore analizzi per intero gli oltre 500µs caratterizzanti la procedura di reset comporta un’attesa di parecchie decine di minuti per ogni ciclo di verifica del sistema con gli strumenti a nostra disposizione.

5.4. Comportamento del tool di sintesi

Dalle esperienze passate avute con gli stessi strumenti software e progettazione di altri circuiti integrati si era appreso che nella sintesi degli stessi, si ottengono risultati migliori dopo che sul circuito nella sua completezza veniva eseguita un'operazione di esplosione delle sottosezioni in cui lo stesso era suddiviso. Tuttavia tale opzione, denominata *ungroup*, nel progetto in questione non solo non porta vantaggi rilevanti, ma al contrario, quando selezionata non consente al software di portare a termine correttamente la sintesi.

In generale l'operazione di ungroup permette al tool di sintesi una maggior libertà di azione in quanto il rispetto dei vincoli temporali complessivi può essere meglio partizionato sulla totalità del circuito piuttosto che suddividendolo uniformemente tra le varie sotto sezioni, il tutto generalmente a favorire il rapporto area / prestazioni.

In una prima fase di stesura del codice, il memory controller è stato descritto mediante una sola entity, ovvero una singola macchina a stati finiti con decine e decine di stati e molte variabili che determinavano il passaggio tra questi. Una tale soluzione non ha permesso al tool di sintesi di trovare una configurazione in grado di rispettare i vincoli temporali richiesti.

Successivamente, descrivendo il memory controller come l'unione di più macchine a stati finiti molto più indipendenti l'una dall'altra (la soluzione descritta nei capitoli precedenti), il software di sintesi è riuscito a trovare una soluzione tale da rispettare tutti i vincoli imposti. Effettuando quindi un'operazione di ungroup per lasciare ancora una volta al compilatore la facoltà di unire moduli separati, l'operazione di sintesi forniva nuovamente risultati non accettabili.

Questo secondo comportamento risulta coerente con quanto successo nella prima versione del memory controller, infatti in entrambi i casi il sistema non presenta nessun sottosistema e la differenza tra i due è solo nel modo in cui si è giunti alla soluzione, direttamente da una descrizione globale ed unificata, oppure mediante l'operazione di ungroup sul sistema descritto in sottosezioni.

5.5. Area occupata

Il modello del memory controller sviluppato dal compilatore vede un'occupazione d'area pari a:

```
****************************************
Report : area
Design : top
****************************************
Library(s) Used: fsc0h_d_generic_core_tt1p2v25c
(File: /nfsd/iclib/farstd13/GENERIC_CORE/FrontEnd/synopsys/fsc0h_d_generic_core_tt1p2v25c.db)

Number of ports:          238
Number of nets:           380
Number of cells:          9
Number of references:     9

Combinational area:       20696.319913 µm^2   (area occupata da circuiti combinatori)
Noncombinational area:    41382.399018 µm^2   (area occupata da circuiti non combinatori)
Net Interconnect area:    undefined  (Wire load has zero net area)

Total cell area:          62078.718750 µm^2
Total area:               undefined
```

Tra questi, analizzando singolarmente i vari moduli per scopi diagnostici, è stato appurato (ma era facilmente prevedibile) che il modulo cache è quello che incide maggiormente (ben oltre la metà) sull'area complessiva.

Non è un caso infatti che i maggiori problemi scaturiscano proprio da quei segnali che devono propagarsi all'interno di questo modulo.

Proprio di queste considerazioni bisogna tener conto qualora si decidesse di realizzare un memory controller dotato di una cache maggiore.

5.6. Problemi ed anomalie emersi nel Post Sintesi

Dalle simulazioni post sintesi sono emerse alcune incongruenze, che verranno elencate di seguito, rispetto al comportamento riscontrato dallo stesso modello in fase di pre-sintesi.

5.6.1. Cache_Addr MSB e situazione di alta impedenza

Il bit più significativo tra i 4 costituenti l'indirizzo di lettura/scrittura della memoria cache non riesce, a differenza degli altri tre segnali aventi la stessa funzione, ad assumere lo stato di alta impedenza, esso viene correttamente pilotato a livello alto o basso, ma quando tutti i moduli (`sync_ctrl, read_ctrl, top_ctrl`) si trovano a portare il segnale stesso a livello 'Z' sulla porta del modulo cache viene riportato un segnale di tipo "*undefined*".

Tale situazione viene a manifestarsi immediatamente dopo l'operazione d'inizializzazione ed in tutte le fasi di idle, ovvero quando non si effettuano particolari operazioni sulla memoria cache, la Figura 25 mostra come il problema sparisca non appena uno dei moduli collegati al bus relativo all'indirizzo cache imponga il proprio valore.

Dalla stessa Figura 25 si possono individuare tutte le porte connesse al bus (`cache_addr_int`) e notare che queste sono effettivamente in uno stato di alta impedenza eccezion fatta per l'interfaccia relativa al modulo cache.

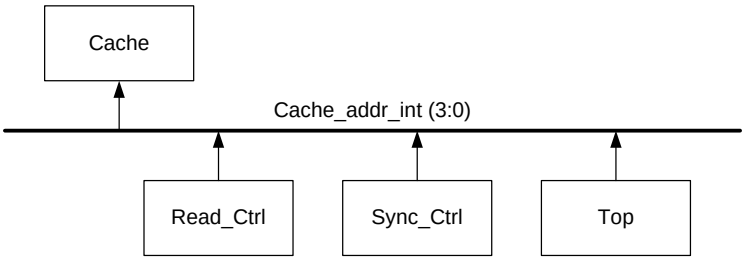


Figura 24: Struttura del bus relativo al segnale cache_addr

Ovviamente la descrizione del circuito controllata in più fasi non presenta particolari anomalie nella definizione del segnale `cache_addr(3:0)` tali da giustificare un diverso comportamento del bit più significativo tra i quattro costituenti l'indirizzo della cache.

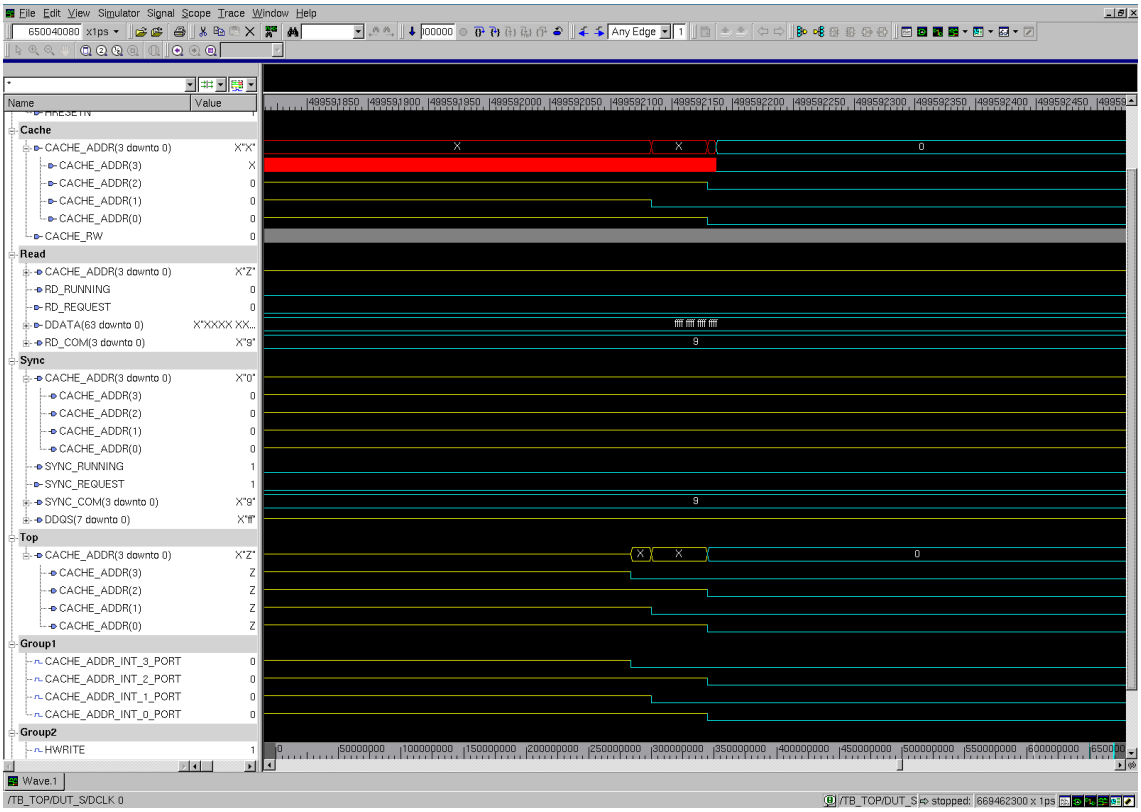


Figura 25: MSB cache address non riesce ad entrare in alta impedenza

5.6.2. Cache addr MSB e Glitch

Sempre relativamente al bit più significativo dell'indirizzo della memoria cache è possibile notare, dallo screenshot del programma di simulazione riportato in Figura 26, come nelle variazioni d'indirizzo che coinvolgono una modifica anche dello specifico segnale `cache_addr(3)` questa permutazione avvenga anticipatamente rispetto a quella degli altri tre segnali simili `cache_addr(2:0)` salvo poi risultare in ritardo rispetto alle altre una volta che il gruppo di segnali si presenta alla porta del modulo cache.

L'anticipo o il ritardo sono singolarmente causa di un glitch (si noti che l'indirizzo `cache_addr` per passare dal valore 7 al valore 8 propone un glitch al valore 0).

Per valutarne le cause è necessario analizzare nel dettaglio il percorso seguito dal segnale `cache_addr(3)` e verificare se questo è distribuito congiuntamente agli altri due `cache_addr(2:0)` o se deve attraversare porte logiche diverse che ne causano una propagazione ritardata dal modulo `top_ctrl` verso il modulo cache.

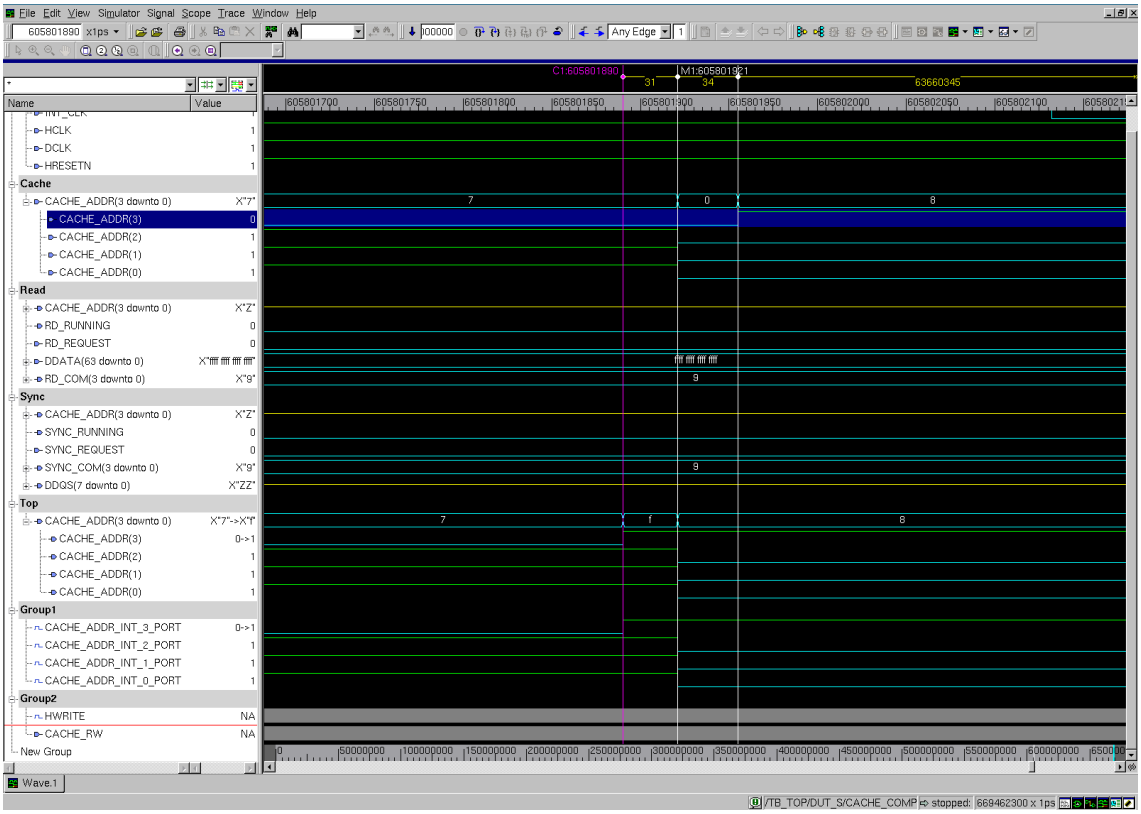


Figura 26: Glitch nel MSB di cache addr

5.6.3. Comportamento generale del sistema

Nonostante i problemi che affliggono in particolare il segnale `cache_addr(3)` il funzionamento complessivo non subisce eccessivamente i difetti generati dallo stesso.

In Figura 27 viene mostrato il comportamento del memory controller durante il riempimento e la successiva lettura della cache che come si può notare non presenta particolari problemi.

Nella fase di lettura vengono correttamente riportati i valori precedentemente scritti nelle rispettive celle di memoria, ed anche l'interazione del modulo di controllo ed esecuzione del refresh non turba il funzionamento generale del memory controller.

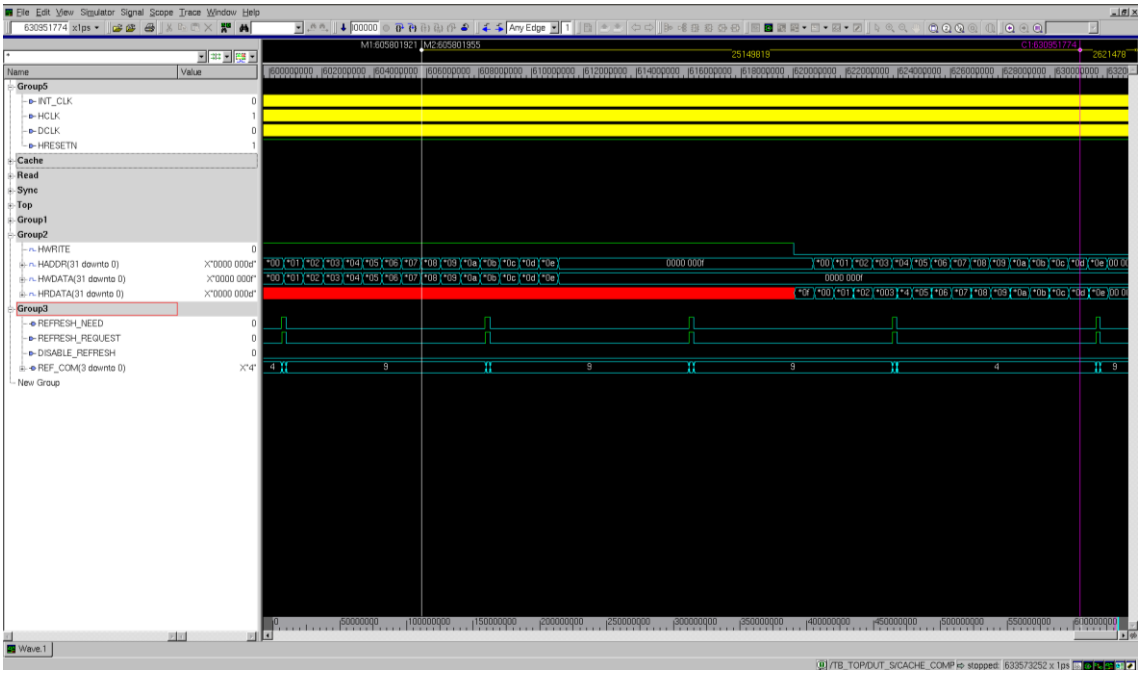


Figura 27: Panoramica del testbench completo

Qualche errore in più interviene nelle simulazioni che portano il segnale `cache_addr(3)` a livello di alta impedenza, mentre i comportamenti di tutti gli altri segnali non risultano significativamente afflitti da errori.

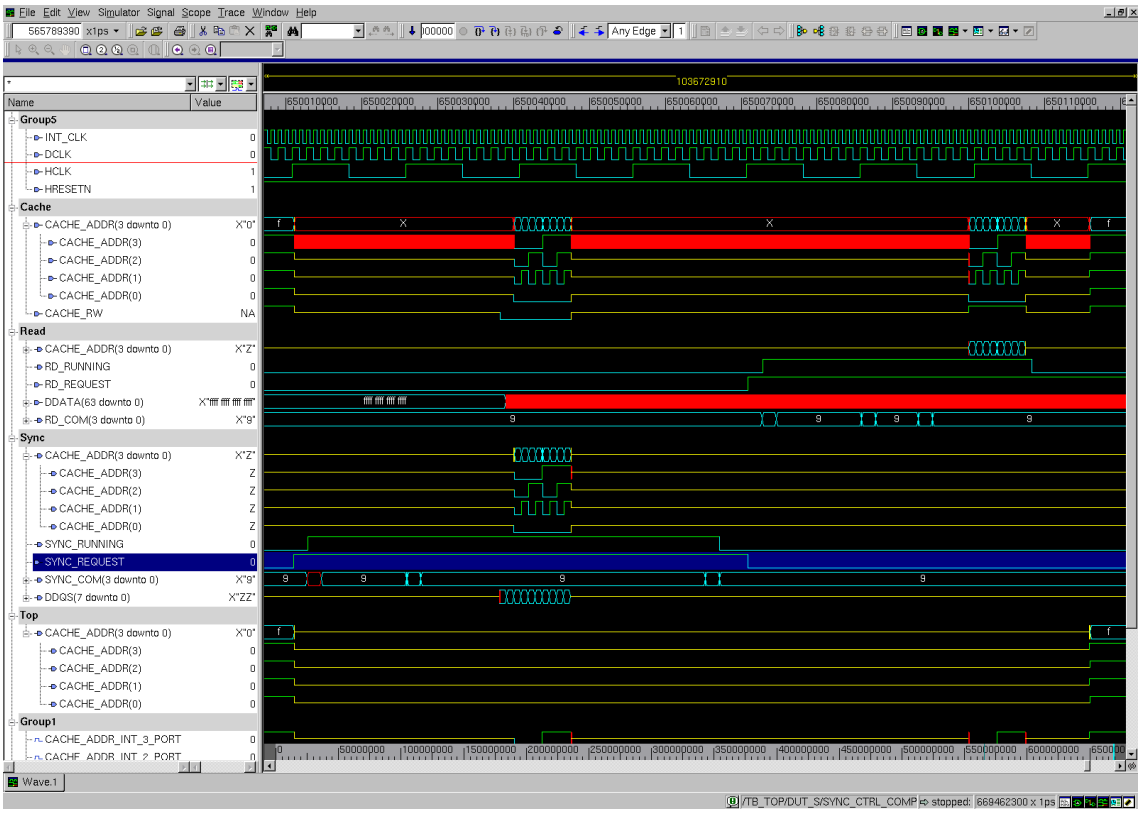


Figura 28: Dettaglio dei problemi provocati da cache_addr(3)

5.6.4. Analisi singola del modulo cache

Nonostante non possa avere un pieno riscontro nella realtà (il compilatore non dovrà considerare anche tutte le altre entity), sono stati effettuati alcuni test singolarmente sui diversi moduli componenti il memory controller per verificare più nel dettaglio alcuni comportamenti dei segnali che li riguardavano direttamente.

Nonostante ciò questo tipo di analisi ha evidenziato alcuni comportamenti che con buona probabilità si riproducono anche nella situazione in cui tutti i moduli vengono sintetizzati e simulati congiuntamente:

- Il tempo di propagazione tra il campionamento del comando e la fruizione in uscita del dato letto pari a 134ps

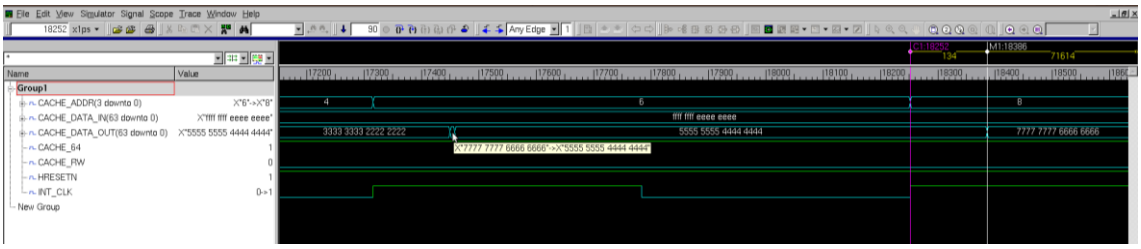


Figura 29: Tempo di scrittura in cache

- Un glitch che si presenta esclusivamente quando l'indirizzo contenuto nel segnale `cache_addr` viene modificato in almeno due dei suoi quattro bit

  0010 -> 0100 cambiano i due bit centrali

  0110 -> 1000 cambiano tutti e tre i bit più significativi

  1010 -> 1100 cambiano i due bit centrali

Mentre non si presenta quando la variazione riguarda solo un bit dell'indirizzo.

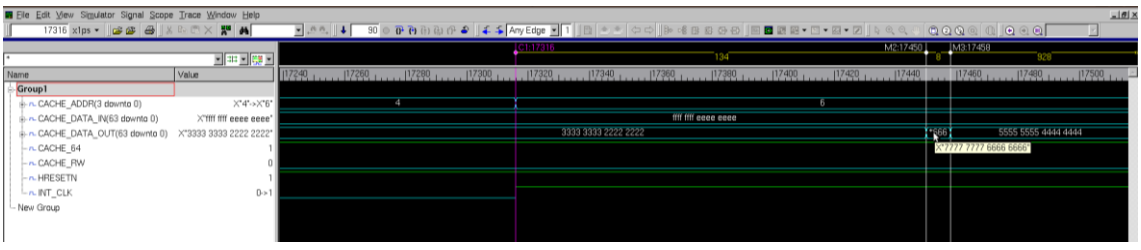


Figura 30: Glitch durante il cambiamento di più bit contemporaneamente

Ipotizzando che il glitch fosse dovuto ad un campionamento difficoltoso dell'indirizzo, è stato verificato il comportamento del modulo facendo variare l'indirizzo sul fronte di

discesa, in modo da lasciare mezzo periodo di tempo al sistema per campionare dei valori stabili.

Tuttavia la risposta del sistema non è stata per niente soddisfacente, infatti il glitch si presentava comunque, ed inoltre i dati in uscita non presentavano dati validi con continuità.

Figura 31: Dati non validi cambiando la sincronizzazione

6. Conclusioni

I problemi incontrati nella fase di simulazione post-sintesi non hanno permesso il proseguimento con i successivi passaggi dello sviluppo del memory controller ed in particolare con la sintesi fisica del modello.

Per poter risolvere tali inconvenienti alcune strade sono state percorse ma non hanno condotto a soluzioni efficaci mentre altre sono rimaste inesplorate.

Una soluzione potrebbe venire ovviamente dall'utilizzo di librerie diverse ed aggiornate ad una nuova tecnologia.

Quelle utilizzate in questo progetto sono basate su transistor con lunghezza di canale di 0,13 µm mentre la tecnologia più recente in ambito di microprocessori ha già raggiunto i 32 nm passando per gli step intermedi rappresentati dai 90 nm, 65 nm e 45 nm.

La possibilità di utilizzare tecnologie più avanzate consentirà agli strumenti software di sintetizzare le diverse strutture in modo molto più efficace e quindi con possibilità di ottenere maggiori margini tanto sui vincoli temporali quanto su quelli legati all'area occupata ed ai consumi, questi ultimi aspetti non ricoprono un ruolo fondamentale nella prima realizzazione ma ovviamente finiscono per migliorare la qualità del prodotto finale.

Inoltre ottenere dei circuiti di dimensioni inferiori si ripercuote positivamente anche nella distribuzione e nella propagazione dei segnali di clock che subiranno meno gli effetti legati al *clock skew* (asincronia del segnale di clock tra diverse sezioni del circuito dovuta principalmente ai ritardi generati attraverso i percorsi attraversati dal segnale di clock).

Oltre all'utilizzo di una tecnologia più avanzata, l'analisi del lavoro per mano di un progettista con maggior esperienza sul campo potrebbe essere fondamentale per aggirare alcune difficoltà che sembrano insormontabili ad una persona con la sola esperienza accademica.

Una maggior familiarità con gli strumenti software e con le opzioni che essi offrono al progettista può infatti rivelarsi fondamentale sia per diagnosticare il problema riscontrato, sia per risolverlo.

Nonostante queste carenze il lavoro di studio eseguito e lo sviluppo che ne è seguito ha portato lo sviluppo di un memory controller DDR3 su bus AHB AMBA ad uno stadio molto avanzato e prossimo alla prototipazione.

6.1. Possibili evoluzioni

Si coglie inoltre l'occasione per introdurre, a coloro i quali dovessero proseguire il lavoro svolto e qui descritto, alcune funzionalità che sicuramente risulterebbero utili ed interessanti per una maggior duttilità del controller:

- Supporto a diversi tipi di memoria DDR3

Per poter utilizzare moduli di memoria con frequenze di funzionamento superiori o tempi di latenza inferiori, ma anche con capienze diverse dai 2Gbyte per modulo per cui l'attuale controller è stato sviluppato.

La configurazione utilizzata può essere impostata dal Master (CPU) del sistema e memorizzata in alcune locazioni di memoria dedicate allo scopo, oppure predisporre delle funzioni di

- Riconoscimento automatico del tipo di memoria installata

L'aggiunta di un processo in fase d'inizializzazione capace di riconoscere autonomamente quali e quanti moduli di memoria siano effettivamente inseriti negli slot a disposizione permetterebbe di sgravare da questo compito il programma caricato sulla cpu ed aumentando ancor più la facilità d'utilizzo del memory controller in diversi contesti.

- Implementazione delle funzionalità di risparmio energetico

Anche le memorie come la maggior parte dei processori in commercio prevedono la possibilità di entrare in uno stato d'inattività che consente loro di ridurre notevolmente i consumi. Tuttavia l'attivazione di tale modalità di funzionamento, ed il ritorno ad un funzionamento standard prevedono procedure aggiuntive la cui gestione finisce per gravare sulle attività del controller di memoria.

- Aumento delle dimensioni della cache

Come detto in precedenza l'aumento delle dimensioni della memoria cache porta sicuramente ad un miglioramento delle prestazioni, quindi aumentarne la capacità di un fattore 8 o 16 potrebbe essere apprezzabile se ciò non aggiungesse ulteriori problematiche di sintesi.

7. Appendice

Sezione dedicata a porzioni di codice che possono rivelarsi utili per una maggior comprensione di alcuni funzionamenti descritti nei paragrafi di pertinenza relativamente ad alcune particolari procedure.

7.1. Gestione avvio e conclusione delle procedure

```
if((start_proc_lat = '1') and (counter /= 0)) then
      counter := counter -1;

else
      counter := "11";
      start_proc_lat := '0';

      if (sync_running = '0' and rd_running = '0' and refresh_running =
      '0' and reset_running = '0') then

            case stateComeBack is
                  when RST_PROC     =>    reset_request <= '0';
                                          stateComeBack <= NORM_PROC;

                  when REF_PROC     =>    refresh_request <= '0';
                                          stateComeBack <= NORM_PROC;
                                          hReady <= '1';

                  when SYNC_PROC    =>    sync_request      <= '0';
                                          cacheModified     := false;
                                          stateComeBack <= NORM_PROC;

                  when RD_PROC      =>    rd_request <= '0';
                                          stateComeBack <= NORM_PROC;
                                          hReady <= '1';

                  when NORM_PROC    =>    stateComeBack <= NORM_PROC;
                                          rd_request <= '0';
                                          sync_request      <= '0';
                                          refresh_request <= '0';
                                          reset_request     <= '0';

            end case;
      end if;
end if;
```

7.2. Cache

```
case cache_rw is

when '1' =>

      case cache_64 is

      when '1' =>
            cache_memory_H (conv_Integer (unsigned (cache_addr(CacheBit-
            1 downto 1) ) ) ) <= cache_data_in (63 downto 32);
            cache_memory_L (conv_Integer (unsigned (cache_addr(CacheBit-
            1 downto 1) ) ) ) <= cache_data_in (31 downto 0);
      when '0' =>
            if (cache_addr(0) = '0') then
            cache_memory_L (conv_Integer (unsigned (cache_addr(CacheBit-
            1 downto 1) ) ) ) <= cache_data_in (31 downto 0);
      else
            cache_memory_H (conv_Integer (unsigned (cache_addr(CacheBit-
            1 downto 1) ) ) ) <= cache_data_in (31 downto 0);
            end if;

      end case;


when '0' =>

      case cache_64 is

      when '1' =>
            cache_data_out (63 downto 32) <= cache_memory_H
            (conv_Integer (unsigned (cache_addr(CacheBit-1 downto 1) ) )
            );

            cache_data_out (31 downto 0) <= cache_memory_L (conv_Integer
            (unsigned (cache_addr(CacheBit-1 downto 1) ) ) );
      when '0' =>
            if (cache_addr(0) = '0') then
                  cache_data_out (31 downto 0) <= cache_memory_L
                  (conv_Integer (unsigned (cache_addr(CacheBit-1 downto
                  1) ) ) ) ;
            else
                  cache_data_out (31 downto 0) <= cache_memory_H
                  (conv_Integer (unsigned (cache_addr(CacheBit-1 downto
                  1) ) ) ) ;
            end if;

      end case;

end case;
```

7.3. Read Controller

```
if (dClk'event and dClk = '1') then

      case read_state is

            when read_1 =>
                  if rd_request = '1' then
                        rd_running  <= '1';
                        rd_com            <= "1000";
                        Timer_Latency     := conv_unsigned(tRCD-
                        1,Timer_Latency'length);

                        read_state  := read_2;
                  end if;

            when read_2 =>
                  if (Timer_Latency /= 0) then
                        Timer_Latency     := Timer_Latency -1;
                        rd_com            <= "1001";
                  else
                        rd_com      <= "0111";
                        Timer_Latency     := conv_unsigned(tRTP-1
                        ,Timer_Latency'length);

                        read_state  := read_3;
                  end if;

            when read_3 =>
                  if (Timer_Latency /= 0) then
                        Timer_Latency     := Timer_Latency -1;
                        rd_com            <= "1001";
                  else
                        rd_com            <= "0101";
                        Timer_Latency     := conv_unsigned(tCAS-tRTP-
                        1,Timer_Latency'length);

                        read_state  := read_4;
                  end if;


            when read_4 =>
                  if (Timer_Latency /= 0) then
                        rd_com            <= "1001";
                        Timer_Latency     := Timer_Latency -1;
                  else
                        Timer_Latency     :=
                        conv_unsigned(4,Timer_Latency'length);

                        read_state  := read_5;
                        fill_request      <= '1';
                  end if;

            when read_5 =>
                  if (Timer_Latency /= 0) then
                        Timer_Latency     := Timer_Latency -1;
                  else
                        fill_request      <= '0';
                        read_state  := read_6;
                        rd_running  <= '0';
```

```
                        cache_rw    <= 'Z';
                        cache_64    <= 'Z';
                        cache_addr  <= (others => 'Z');
                        cache_data_in     <= (others => 'Z');
                  end if;

            when read_6 =>
                  if rd_request = '0' then
                        read_state  := read_1;
                  end if;

      end case;

end if;
```

7.4. Fill Sequence

```
if (int_clk'event and int_clk = '1') then

if (fill_request = '1') then

      cache_rw    <= '1';
      cache_64    <= '1';

      case fill_state is
            when fill_1 =>
                  if fill_request = '1' then
                        cache_addr  <= "0000";
                        cache_data_in     <= dData;
                        fill_state  := fill_2;
                  end if;
            when fill_2 =>
                  cache_addr  <= "0010";        --
                  cache_data_in     <= dData;
                  fill_state  := fill_3;
            when fill_3 =>
                  cache_addr  <= "0100";        --
                  cache_data_in     <= dData;
                  fill_state  := fill_4;
            when fill_4 =>
                  cache_addr  <= "0110";        --
                  cache_data_in     <= dData;
                  fill_state  := fill_5;
            when fill_5 =>
                  cache_addr  <= "1000";        --
                  cache_data_in     <= dData;
                  fill_state  := fill_6;
            when fill_6 =>
                  cache_addr  <= "1010";        --
                  cache_data_in     <= dData;
                  fill_state  := fill_7;
            when fill_7 =>
                  cache_addr  <= "1100";        --
                  cache_data_in     <= dData;
                  fill_state  := fill_8;
            when fill_8 =>
                  cache_addr  <= "1110";
                  cache_data_in     <= dData;
                  fill_state  := fill_9;
            when fill_9 =>
                  cache_rw    <= 'Z';
                  cache_64    <= 'Z';
                  cache_addr  <= (others => 'Z');
                  cache_data_in     <= (others => 'Z');
                  fill_state  := fill_10;
            when fill_10 =>
                  cache_rw    <= 'Z';
                  cache_64    <= 'Z';
                  cache_addr  <= (others => 'Z');
                  cache_data_in     <= (others => 'Z');
                  fill_state  := fill_1;
      end case;

end if;
end if;
```

8. Bibliografia

8.1. AMBA

http://www.arm.com/

http://www.arm.com/products/system-ip/amba/amba-open-specifications.php

https://silver.arm.com/download/download.tm?pv=1085658

8.2. DDR3 SDRAM STANDARD

http://www.jedec.org/

http://www.jedec.org/standards-documents/docs/jesd-79-3d

8.3. SAMSUNG DDR3

http://www.samsung.com/global/business/semiconductor/products/dram/Products_ComputingDRAM.html

http://www.samsung.com/global/business/semiconductor/productList.do?fmly_id=691&xFmly_id=690

8.4. Descrizione DIMM

http://www.samsung.com/global/business/semiconductor/productInfo.do?fmly_id=693&partnum=M378B5673EH1&xFmly_id=690

8.5. DataSheet DIMM Samsung

http://www.samsung.com/global/system/business/semiconductor/product/2009/7/17/192768ds_ddr3_1gb_e-die_based_udimm_rev103.pdf

8.6. LEON3

http://www.gaisler.com/cms/index.php?option=com_content&task=view&id=13&Itemid=53

http://www.gaisler.com/doc/leon3_product_sheet.pdf

8.7. SYNOPSYS - Strumenti di sviluppo

http://www.synopsys.com/home.aspx

http://www.synopsys.com/tools/verification/functionalverification/pages/vcs.aspx