ALMA MATER STUDIORUM
UNIVERSITÀ DEGLI STUDI DI BOLOGNA

Facoltà di Ingegneria

Corso di Laurea in INGEGNERIA INFORMATICA

Progetto di CALCOLATORI ELETTRONICI M

Progetto di una memoria cache per il processore DLX

Componenti Gruppo:

Andrea Grandi

Filippo Malaguti

Massimiliano Mattetti

Gabriele Morlini

Thomas Ricci

Anno Accademico 2009/2010

Indice

Introduzione **5**
1 Obiettivi del progetto 6

1 Caratteristiche della memoria cache **7**
1 Politica di rimpiazzamento 8
2 Struttura e interfacce 9

2 Realizzazione e collaudo **15**
1 Strutture dati 15
2 Implementazione 17
2.1 cache_read 19
2.2 cache_replace_line 19
2.3 cache_hit_on 20
2.4 cache_inv_on 20
2.5 cache_write 20
2.6 get_way 21
2.7 ram_write 21
2.8 snoop 21
3 Diagrammi temporali 22
4 Problematiche principali affrontate 22

3 Integrazione con DLX **23**

4 Block RAM **25**
1 Caratteristiche e segnali della Block Ram 25
2 Configurazione della Block Ram 28
3 Operazioni della Block Ram 33

4 Conflitti d'accesso in Block Ram Dual-Port 34
5 Possibili utilizzi della Block Ram in un progetto su FPGA 37
6 Realizzazione di un progetto d'esempio d'uso 38

Conclusioni **41**

Bibliografia **43**

Introduzione

La necessità di introdurre cache per un processore deriva dal noto problema del collo di bottiglia rappresentato dall'accesso a dispositivi di memoria. Un processore durante il suo funzionamento tende ad accedere in scrittura o a reperire dati in lettura provenienti dalla memoria a valle e tale operazione richiede tipicamente diversi cicli di clock che costringono il processore (più veloce della memoria) ad attendere il dato. Ciò comporta l'introduzione di cicli di wait che ovviamente causano un peggioramento delle performace del processore, il quale attende che la memoria gli presenti il dato richiesto segnalato dal segnale di ready.

Per superare tale problema si utilizzano pertanto delle memorie cache, vicine al processore, di piccole dimensioni e molto veloci (tanto che possono avere tempi d'accesso simili a quelli dei registri interni al processore) da cui vengono reperiti i dati necessari all'esecuzione, consentendo in caso di HIT, ovvero nel caso in cui il dato si trovi in cache, di recuperarlo quasi senza ritardo.

Le cache si posizionano nella gerarchia delle memorie (insieme ai registri) tra i livelli più prossimi al processore e ciò comporta da un lato la rapidità nell'accesso e dall'altro le dimensioni limitate che fanno sì che una cache contenga un subset delle linee di memoria del dispositivo a valle (memoria o un livello superiore di cache se presente).

Pertanto quando si accede a una cache possono capitare due casi:

1. il dato si trova nella cache (HIT);

2. il dato non è presente e deve essere recuperato da un dispositivo a valle (MISS).

Ovviamente in caso di MISS si deve pagare un costo temporale per il reperimento del dato assente, detto miss penalty, dato dalla somma del tempo d'accesso al dispositivo a valle e dal tempo di trasferimento della linea col dato cercato.
Ciononnostante, è dimostrato che l'hit rate e quindi l'efficienza delle cache è tipicamente molto alta (oltre il 95%) grazie alla validità del Principio di Località spaziale e temporale, per il quale un programma in esecuzione tende ad eseguire temporalmente istruzioni eseguite di recente e ad accedere a dati acceduti di recente. Quindi sulla base di tali considerazioni, l'uso di cache contenenti le linee di memoria più recentemente accedute (working set) consente di migliorare notevolmente il tempo di reperimento dei dati necessari all'esecuzione, evitando quindi i ritardi che si avrebbero per ogni accesso diretto in memoria.
Abbiamo scelto questo progetto per approfondire le tematiche e le problematiche legate alla progettazione di un componente cache da affiancare al processore DLX visto a lezione.

1 Obiettivi del progetto

L'attività di progetto svolta si prefigge i seguenti obiettivi

1. **Realizzazione memoria cache:** progetto di un component VHDL che realizza il funzionamento di una memoria cache generica.
2. **Testbench del component:** progetto di una suite di test per il component.
3. **Integrazione con DLX:** modifica del progetto DLX per consentire l'integrazione del component realizzato con il processore

Capitolo 1

Caratteristiche della memoria cache

Si è scelto di progettare una cache di tipo set-associative, la cui schematizzazione è mostrata in Fig. 3. Questa tipologia di cache rappresenta un buon compromesso tra flessibilità e costo in termini di silicio.

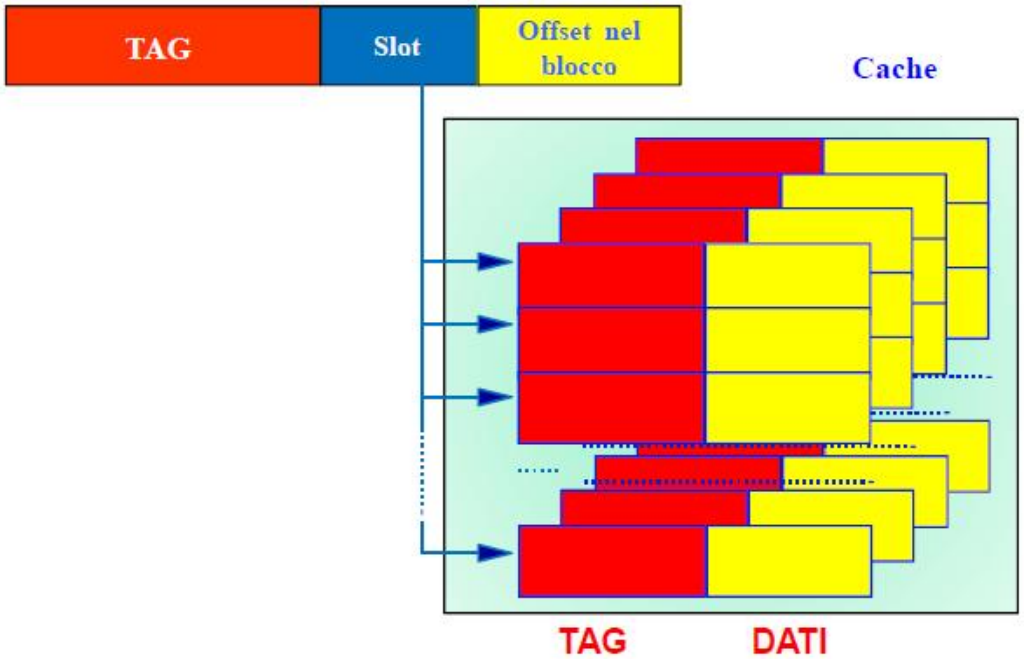


Figura 1.1: Schematizzazione di una cache set-associative

L'indirizzo di partenza del blocco è diviso in TAG (parte alta), INDEX e OFFSET (parte bassa). Il TAG consente di identificare univocamente una linea all'interno di un un sottoinsieme di linee, detto SET. L'INDEX individua immediatamente il SET all'interno del quale è possibile recupeare la linea corrente tramite il confronto del TAG. In questo modo si limita il numero di confronti tra TAG accettando il fatto che ogni linea possa appartenere ad un singolo set. La parte meno significativa dell'indirizzo rappresenta l'OFFSET che consente di individuare il dato all'interno di una linea.
Per garantire maggiore flessibilità si è scelto di parametrizzare alcune delle caratteristiche statiche della cache, quali ad esempio:

- la dimensione dei blocchi
- il numero di vie
- il numero di linee

1 Politica di rimpiazzamento

Nel caso in cui si debba caricare una nuova linea e tutte le vie siano occupate è necessario determinare quale linea rimpiazzare. Un buon algoritmo di rimpiazzamento dovrebbe cercare di individuare la linea vittima che meno probabilmente verrà riutilizzata in seguito.
Il criterio scelto per effettuare il rimpiazzamento è basato su contatori, che implementa una politica LRU (Least Recently Used). Tale politica è tipicamente implementata poichè statisticamente si verifica principio di località. È quindi presente un contatore per ogni via di ogni set tramite il quale si tiene traccia di quanto recentemente si è acceduti a ciascuna linea: un valore basso del contatore indica un accesso recente mentre un valore alto indica un accesso*vetusto*. Evidentemente la linea candidata al rimpiazzamento risulta essere quella alla quale è associato il contatore di valore più elevato.
Nel caso di HIT su una linea, sono incrementati i valori di contatori

più basso rispetto al valore di quello della linea HIT mentre quest'ultimo viene resettato. Nel caso di MISS si procede con un rimpiazzamento e poi si agisce come nel caso di HIT sulla nuova linea. Infine, in caso di invalidazione di una linea, si porta al valore massimo il contatore della linea invalidata e si decrementano di 1 tutti i contatori con valore più elevato di quello della linea invalidata.

2 Struttura e interfacce

La memoria cache si interfaccia con i dispositivi esterni attraverso 4 tipi di interfacce, come mostrato in Fig. 1.2.

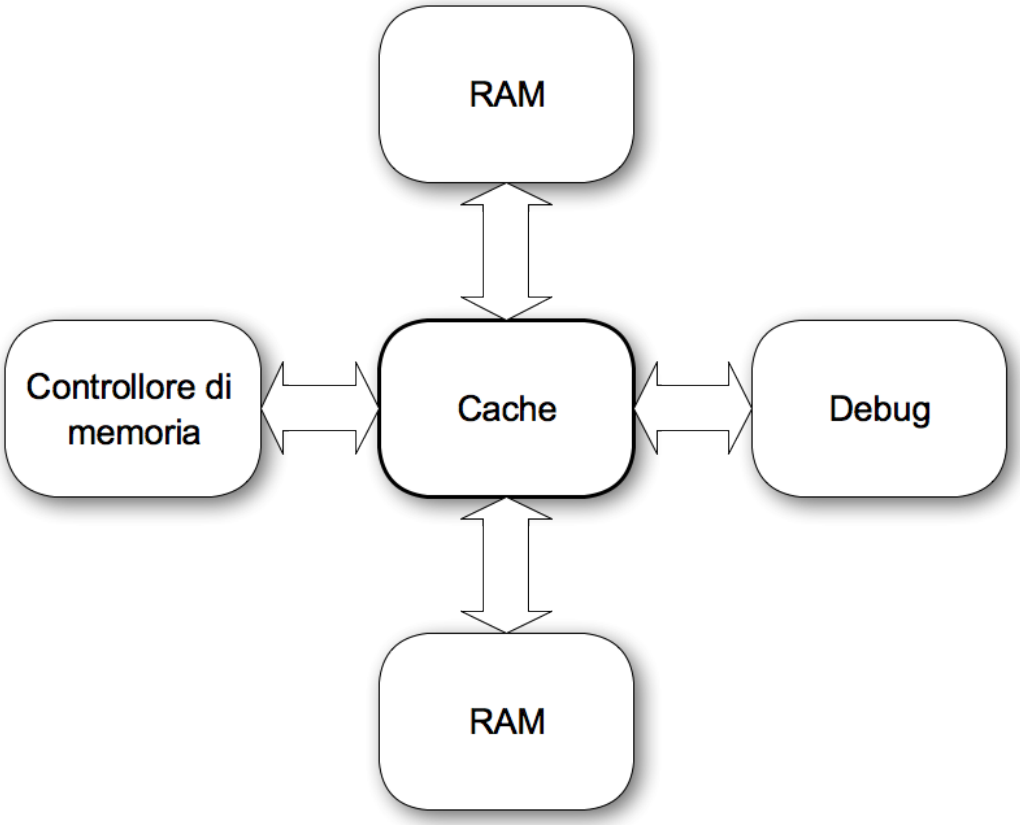


Figura 1.2: Interfacce della memoria cache

L'interfaccia verso il microprocessore, mostrata in Fig. 1.3, consente a quest'ultimo di accedere ai dati memorizzati all'interno della cache.

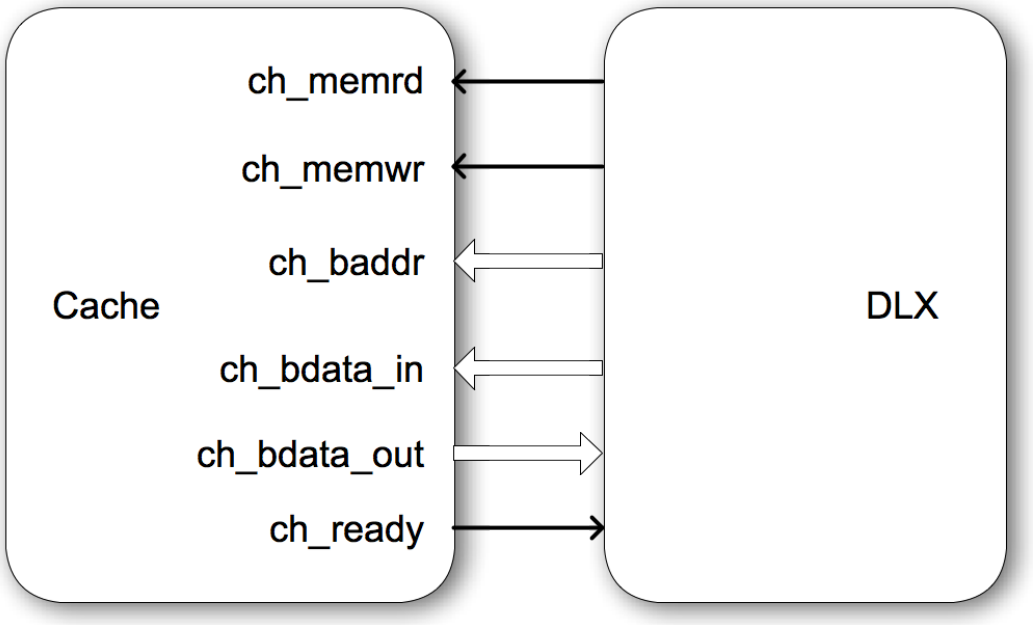


Figura 1.3: Interfaccia della memoria cache verso il processore DLX

In particolare sono presenti i seguenti segnali:

- **ch_baddr(31-2)**: indirizzi a 32 bit emessi dal microprocessore
- **ch_bdata(32-0)**: bus dati con parallelismo 32bit
- **ch_memwr**: segnale per il comando di scrittura in cache
- **ch_memrd**: segnale per il comando di lettura da cache
- **ch_ready**: segnale che indica il termine dell'operazione di lettura/scrittura corrente

Per quanto riguarda gli scambi di dati tra processore e memoria cache, si ipotizza che siano sempre lette e scritte parole di lunghezza fissa a 32 bit.

Anche se la memoria cache progettata non verrà impiegata in sistemi multimaster, si è comunque deciso di affrontare alcune delle problematiche inerenti alla presenza di un controllore di memoria. Tramite l'opportuna interfaccia è ad esempio possibile effettuare l'invalidazione delle linee e lo snooping dei dati presenti in cache.

L'interfaccia verso il controllore di memoria, mostrata in Fig. 1.4, consente di testare e modificare lo stato delle linee.

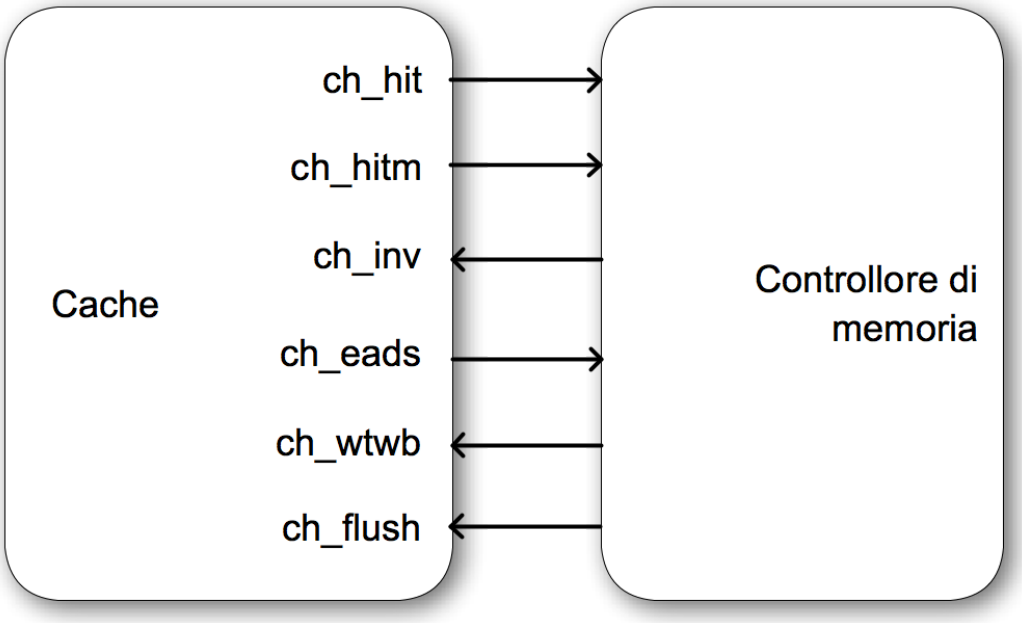


Figura 1.4: Interfaccia della memoria cache verso il controllore di memoria

In particolare sono presenti i seguenti segnali:

- **ch_eads**: inizia il ciclo di snoop
- **ch_inv**: richiede l'invalidazione della linea
- **ch_hit**: indica che la linea richiesta è presente in memoria

- **ch_hitm**: indica che la linea richiesta è presente in memoria in stato modified
- **ch_flush**: scarica il contenuto della cache sulla RAM (tutto o solo un blocco?, funziona?)

L'interfaccia verso la RAM è mostrata in Fig. 1.5 e consente alla cache di recuperare i blocchi dal livello sottostante.

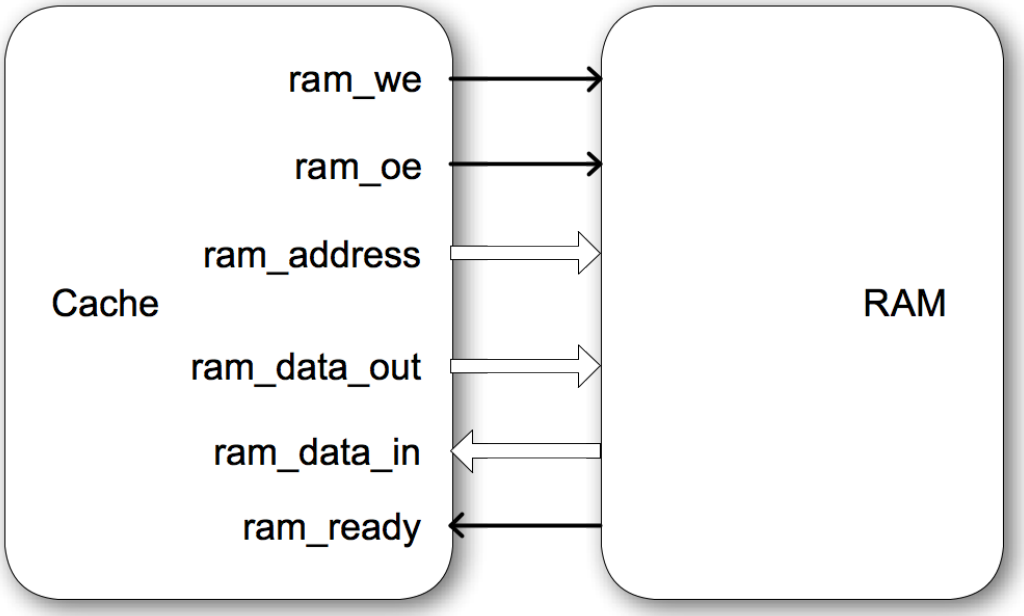


Figura 1.5: Interfaccia della memoria cache verso la RAM

In particolare sono presenti i seguenti segnali:

- **ram_address(31-2)**: indirizzi a 32 bit emessi dalla cache
- **ram_data_out(32-0)**: bus dati di uscita con parallelismo pari alla dimensione di una linea
- **ram_data_in(32-0)**: bus dati di ingresso con parallelismo pari alla dimensione di una linea
- **ram_we**: segnale per il comando di scrittura in RAM

- **ram_oe**: segnale per il comando di lettura dalla RAM
- **ram_ready**: segnale che indica il termine dell'operazione di lettura/scrittura corrente

Si noti che la cache non è a conoscenza del componente posto al livello superiore. Vista la simmetria delle due interfacce è quindi teoricamente possibile sostituire la RAM con un ulteriore livello di cache, inserendo più livelli di cache all'interno del processore.

È presente infine una quarta interfaccia verso l'esterno, utilizzata per monitorare lo stato interno della cache e poter quindi eseguire il debug.
Tale interfaccia, mostrata in Fig. 1.6, non è indispensabile per il corretto funzionamento del dispositivo.

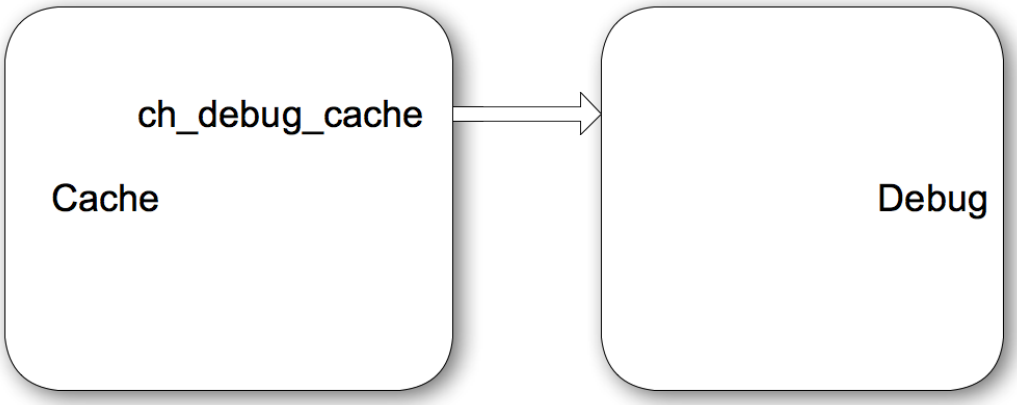


Figura 1.6: Interfaccia utilizzata per il debug della memoria cache

Capitolo 2

Realizzazione e collaudo

La cache è stata realizzata come componente indipendente, detto `Cache_cmp`.
In questo capitolo saranno mostrate le caratteristiche principali di tale componente.

1 Strutture dati

Le strutture dati impiegate nel componente sono definite nel file `Cache_lib.vhd`.

Listing 2.1: Costanti e tipi di dato definiti nel file `Cache_lib.vhd`

```
CONSTANT OFFSET_BIT : natural := 5;
CONSTANT INDEX_BIT : natural := 2;
CONSTANT TAG_BIT : natural := PARALLELISM - INDEX_BIT -
    OFFSET_BIT;
CONSTANT NWAY : natural := 2;

CONSTANT MESI_M : natural := 3;
CONSTANT MESI_E : natural := 2;
CONSTANT MESI_S : natural := 1;
CONSTANT MESI_I : natural := 0;
```

```
TYPE data_line IS ARRAY (0 to 2**OFFSET_BIT - 1) of
    STD_LOGIC_VECTOR (7 downto 0);

TYPE cache_line IS
    RECORD
        data : data_line;
        status : natural;
        tag : STD_LOGIC_VECTOR (TAG_BIT-1 downto 0);
        lru_counter : natural;
    END RECORD;

TYPE set_ways IS ARRAY (0 to NWAY - 1) of cache_line;

TYPE cache_type IS ARRAY (natural range <>) of set_ways;
```

Il numero di bit di offset, indice e tag è stato parametrizzato per rendere più flessibile l'utilizzo del componente.
Sono stati inoltre definiti i seguenti tipi di dati:

- `data_line`: contiene i dati per una linea della cache, la cui dimensione è calcolata in base al numero di bit di offset;
- `cache_line`: record contenente le informazioni su dati e stato di una linea;
- `set_ways`: array di `NWAY` linee che compongono una via;
- `cache_type`: array di vie, costituisce l'intera cache ??? (non so come scrivere... :S).

Per ogni `cache_line` si tiene inoltre traccia di:

- `data`: `data_line` relativa alla linea corrente;
- `status`: indica lo stato MESI della linea;
- `tag`: bit dell'indirizzo che rappresentano il tag della linea;
- `lru_counter`: contatore usato dalla politica di rimpiazzamento.

In Fig. 2.1 è mostrata una schematizzazione delle strutture dati utilizzate all'interno del componente.

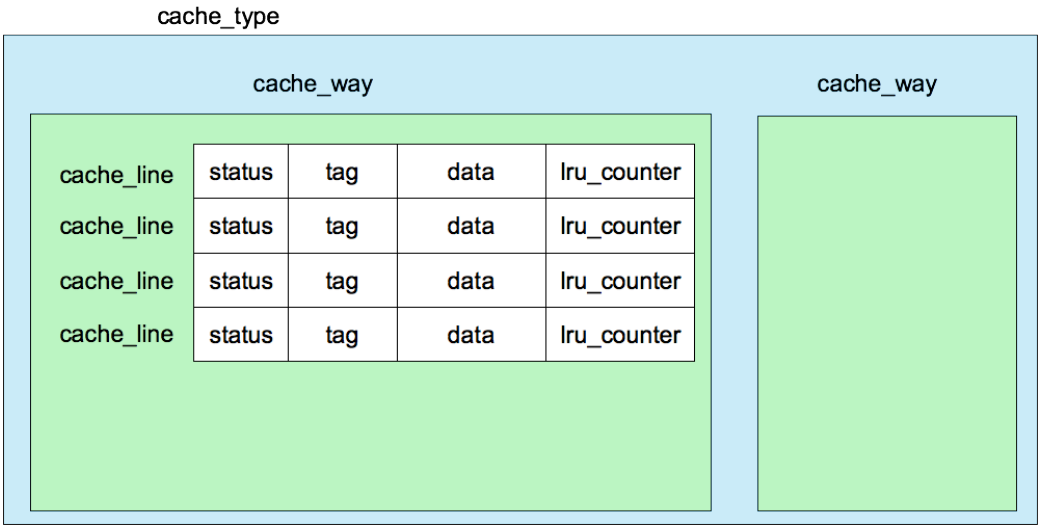


Figura 2.1: Schematizzazione delle strutture dati della cache

2 Implementazione

All'interno del componente è presente un unico process detto `cache_process`, il quale risponde alle variazioni dei segnali di comando `ch_reset`, `ch_memrd`, `ch_memwr`, `ch_eads`.

All'interno del process vengono invocate le opportune procedure, tramite le quali si realizzano tutti i meccanismi per l'accesso e la modifica dei dati contenuti nella cache.

Listing 2.2: Codice VHDL del process `cache_process`

```
cache_process: process (ch_reset, ch_memrd, ch_memwr,
    ch_eads) is
    variable word: STD_LOGIC_VECTOR (31 downto 0):= (others
        => '0');
    variable hit: STD_LOGIC;
    variable hit_m: STD_LOGIC;
begin
    if (ch_reset = '1') then -- reset
        ch_ready <= '0';
        cache_reset;
```

```
        -- Inizializzazione cache e ram per il debug
        for i in 0 to 1023 loop
            RAM(i):= conv_std_logic_vector(i mod 256, 8);
        end loop;
    else
        if(ch_memrd = '1' and ch_memwr = '0') then -- memrd
            cache_read(word);
            ch_bdata_out <= word;
        elsif(ch_memrd = '0' and not ch_memwr'event and not
            ch_reset'event) then -- fine memrd
            ch_bdata_out <= (others => 'Z');
        end if;

        if(ch_memwr = '1' and ch_memrd = '0') then -- memwr
            word:= ch_bdata_in;
            cache_write(word);
        end if;

        if(ch_eads = '1') then -- snoop
            cache_snoop(hit, hit_m);
            ch_hit <= hit;
            ch_hitm <= hit_m;
        else
            ch_hit <= '0';
            ch_hitm <= '0';
        end if;

        ch_ready <= ch_memrd or ch_memwr;

    end if;

    ch_debug_cache <= cache;
end process cache_process;
```

Di seguito saranno brevemente descritte le procedure invocate all'interno del process.

2.1 cache_read

Parametri di output:

- `word`: dato letto

Descrizione:

1. Legge l'indirizzo dal bus separando index, tag e offset
2. Verifica se c'è un hit attraverso `get_way()`
3. In caso di MISS applica la politica di rimpiazzamento richiamando `cache_replace_line()`
4. Legge il dato dalla cache
5. Aggiorna i contatori attraverso `cache_hit_on()`
6. Pone il dato letto in `word` e lo restituisce

2.2 cache_replace_line

Parametri di output:

- selected_way: via sulla quale è stato caricato il dato rimpiazzato

Descrizione:

1. Individua la linea da rimpiazzare, cioè quella con `lru_counter` massimo
2. Controlla se la linea ha stato MESI_M e in tal caso ne fa il write-back invocando `ram_write()`
3. Carica il nuovo blocco nella cache sovrascrivendo il vecchio
4. Modifica il bit di stato in base al valore di WT_WB
5. Restituisce il numero della via sulla quale è presente il dato appena caricato

2.3 cache_hit_on

Parametri di input:

1. `hit_index`: indice al quale si è verificato l'hit
2. `hit_way`: via nella quale si è verificato l'hit

Descrizione:
Applica la politica di invecchiamento aggiornando i contatori, in particolare:

1. incrementa i contatori di valore più basso della via corrente specificata da `hit_way`
2. resetta il contatore della via corrente

2.4 cache_inv_on

Parametri di input:

- `inv_index`: indice da invalidare
- `inv_way`: via da invalidare

Descrizione:
Applica la politica di invecchiamento aggiornando i contatori, in particolare:

1. decrementa i contatori di valore più alto della via corrente specificata da `inv_way`
2. porta al valore massimo il contatore della via corrente

2.5 cache_write

Parametri di input:

- `word`: parola ad scrivere nella cache

Descrizione:

1. Legge l'indirizzo dal bus separando index, tag e offset.

2. Verifica se c'è un hit attraverso `get_way()`
3. In caso di MISS applica la politica di rimpiazzamento richiamando `cache_replace_line()`
4. Scrive il nuovo dato sulla cache
5. Aggiorna i contatori attraverso `cache_hit_on()`
6. Aggiorna il bit di stato ed esegue eventualmente il write-through.

2.6 get_way

Parametri di input:

1. `index`: indice
2. `tag`: tag da controllare

Parametri di output:

- `way`: via nella quale è presente il dato

Descrizione:

1. Verifica se il dato è in cache, cioè se esiste una linea con tag uguale a quello specificato il cui stato è diverso da `MESI_I`
2. Se il dato non è presente restituisce way = -1
3. Se il dato è presente restituisce il numero della via

2.7 ram_write

Parametri di input:

- `tag`: tag della linea da scrivere
- `index`: index della linea da scrivere
- `way`: numero di via in cui si trova la linea da scrivere

Descrizione: 1. Costruisce l'indirizzo del blocco a partire da `tag` e `index` 2. Scrive i dati contenuti nel blocco sulla RAM

2.8 snoop

Da dettagliare in seguito ????

3 Diagrammi temporali

4 Problematiche principali affrontate

(metteri anche tutti i problemi relativi al bus bidirezionale)

Capitolo 3

Integrazione con DLX

Schemi a blocchi
Diagrammi temporali

Capitolo 4

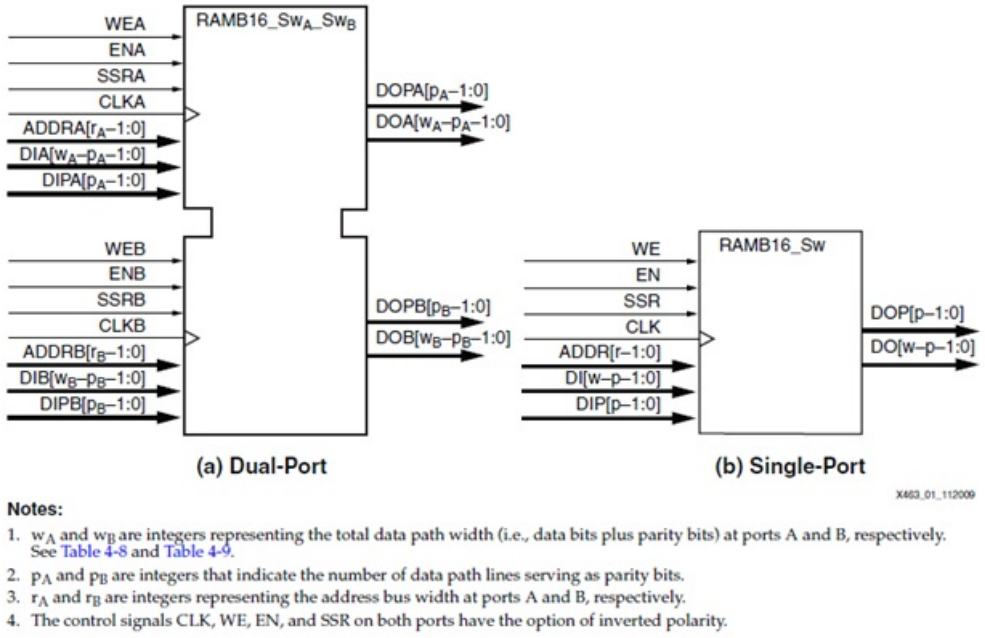
Block RAM

Mentre nel nostro progetto per semplicità abbiamo considerato nulli i tempi d'accesso alla cache e alla memoria principale, ovviamente ciò non accade nella realtà dove la struttura gerarchica delle memorie impone vincoli di dimensione e tempi d'accesso per i vari livelli di memoria. Per tale motivo abbiamo voluto approfondire le problematiche relative alle temporizzazioni per gli accessi in memoria che un progetto tradizionale impone. Per far ciò abbiamo considerato ciò che una tipica FPGA dà a disposizione ad un progettista per implementare una memoria ram e gestirne gli accessi in lettura e scrittura.
Nel nostro caso abbiamo analizzato le caratteristiche dell'FPGA della famiglia Spartan-3E di Xilinx, che per gestire la memorizzazione di dati utilizza le Block Ram.

1 Caratteristiche e segnali della Block Ram

La memoria RAM presente su una FPGA Spartan-3 viene implementata tramite una serie di Block Ram ripartite in colonne il cui numero e capacità dipende dalle caratteristiche stesse della scheda utilizzata. Dal punto di vista implementativo le Block Ram sono realizzate tramite 18,432 celle di memoria SRAM che consentono pertanto di memorizzare 18 Kbits di cui 16 Kbits di dato e 2 Kbits utilizzati tipica-

mente per memorizzare i bit di parità relativi ai dati memorizzati o in alternativa come spazio di memorizzazione aggiuntivo.
L'accesso alla block ram può avvenire o in modalità Single-Port utilizzando una sola porta dati (A o B) oppure in Dual-Port tramite 2 porte indipendenti A e B che consentono di effettuare operazioni di lettura e scrittura su zone diverse del dispositivo.



Notes:

1. w_A and w_B are integers representing the total data path width (i.e., data bits plus parity bits) at ports A and B, respectively. See Table 4-8 and Table 4-9.
2. p_A and p_B are integers that indicate the number of data path lines serving as parity bits.
3. r_A and r_B are integers representing the address bus width at ports A and B, respectively.
4. The control signals CLK, WE, EN, and SSR on both ports have the option of inverted polarity.

Figura 4.1: Pinout Block Ram Single-Port e Dual-Port

Ogni porta della block ram si interfaccia con due bus dati (distinti per l'input e per l'output), con il bus degli indirizzi e dispone di una serie di segnali di comando atti ad abilitare il dispositivo (EN in Single-Port) e a gestire operazioni di lettura (EN) o scrittura (WE). La seguente tabella racchiude i principali segnali illustrati nella figura precedente sia in Single-Port che in Dual-Port.
Segnali di comando:

- `EN` = Enable consente di abilitare il dispositivo e qualora non siano asseriti WE(write enable) o SSR (reset), il segnale comanda a default la lettura della cella di memoria all'indirizzo specificato sul bus degli indirizzi ADDR sul fronte positivo del clock.

Signal Description	Single Port	Dual Port		Direction
		Port A	Port B	
Data Input Bus	DI	DIA	DIB	Input
Parity Data Input Bus (available only for byte-wide and wider organizations)	DIP	DIPA	DIPB	Input
Data Output Bus	DO	DOA	DOB	Output
Parity Data Output (available only for byte-wide and wider organizations)	DOP	DOPA	DOPB	Output
Address Bus	ADDR	ADDRA	ADDRB	Input
Write Enable	WE	WEA	WEB	Input
Clock Enable	EN	ENA	ENB	Input
Synchronous Set/Reset	SSR	SSRA	SSRB	Input
Clock	CLK	CLKA	CLKB	Input
Synchronous/Asynchronous Set/Reset (Spartan-3A DSP FPGA only)	N/A	RSTA	RSTB	Input
Output Register (Spartan-3A DSP FPGA only)	N/A	REGCEA	REGCEB	Input

Figura 4.2: Segnali della Block Ram Single-Port e Dual-Port

- `WE` = Write Enable consente di comandare un ciclo di scrittura in memoria all'indirizzo specificato sul bus degli indirizzi ADDR (con EN asserito), tale operazione in base al valore settato nell'attributo WRITE_MODE può essere affiancata da una lettura contemporanea del dato alla stessa locazione di memoria che viene portato nel buffer di output sul bus DO (della stessa porta).

- `SSR` = Syncronous Set/Reset consente di settare '1' o resettare '0' i registri di ouput sul bus dati in accordo col valore dell'attributo SRVAL.

- `REGCE` = Output Register Enable consente in fase di lettura da ram di salvare il dato letto in un output register.

- `CLK` = è il clock e si può configurare se la memoria debba essere sensibile ai fronti di salita o di discesa.

- GSR = Global Set/Reset segnale di sistema utilizzato per in fase di inizializzazione del sistema (non disponibile all'esterno su un pin).

C'é inoltre la possibilità di configurare le polarità di ogni segnale di comando se da considerarsi asserito alto o basso.
Interfacciamento ai bus:

- ADDR = bus degli indirizzi la cui larghezza (#:0) dipende dalla configurazione della block ram.
- DI = Data Input Bus (#:0) (l'ampiezza del dato da trasferire dipende dalla configurazione della block ram).
- DO = Data Output Bus
- DIP = Data Input Parity Bus (nei bit più significative del Bus Dati di Input)
- DOP = Data Output Parity Bus (nei bit più significative del Bus Dati di Output)

Possibili configurazioni e organizzazioni della Block Ram:

Nel nostro caso, dal momento che il DLX è un processore a 32 bit, la configurazione necessaria per la block ram è la 512x36. Tale configurazione da la possibilitá di accedere fino a 36 bit di dato contemporaneamente, di cui 32 bit di dato veri e 4 di parità posti sui bit più significativi del bus dati. Con tale configurazione la block ram (di 18 Kbit) conterrà 512 entry (memory-depth) da 36 bit (infatti 512x36 bit = 18 Kbits).

2 Configurazione della Block Ram

La configurazione della Block Ram avviene tramite una serie di attributi propri dei componenti ram disponibili nelle librerie di sistema

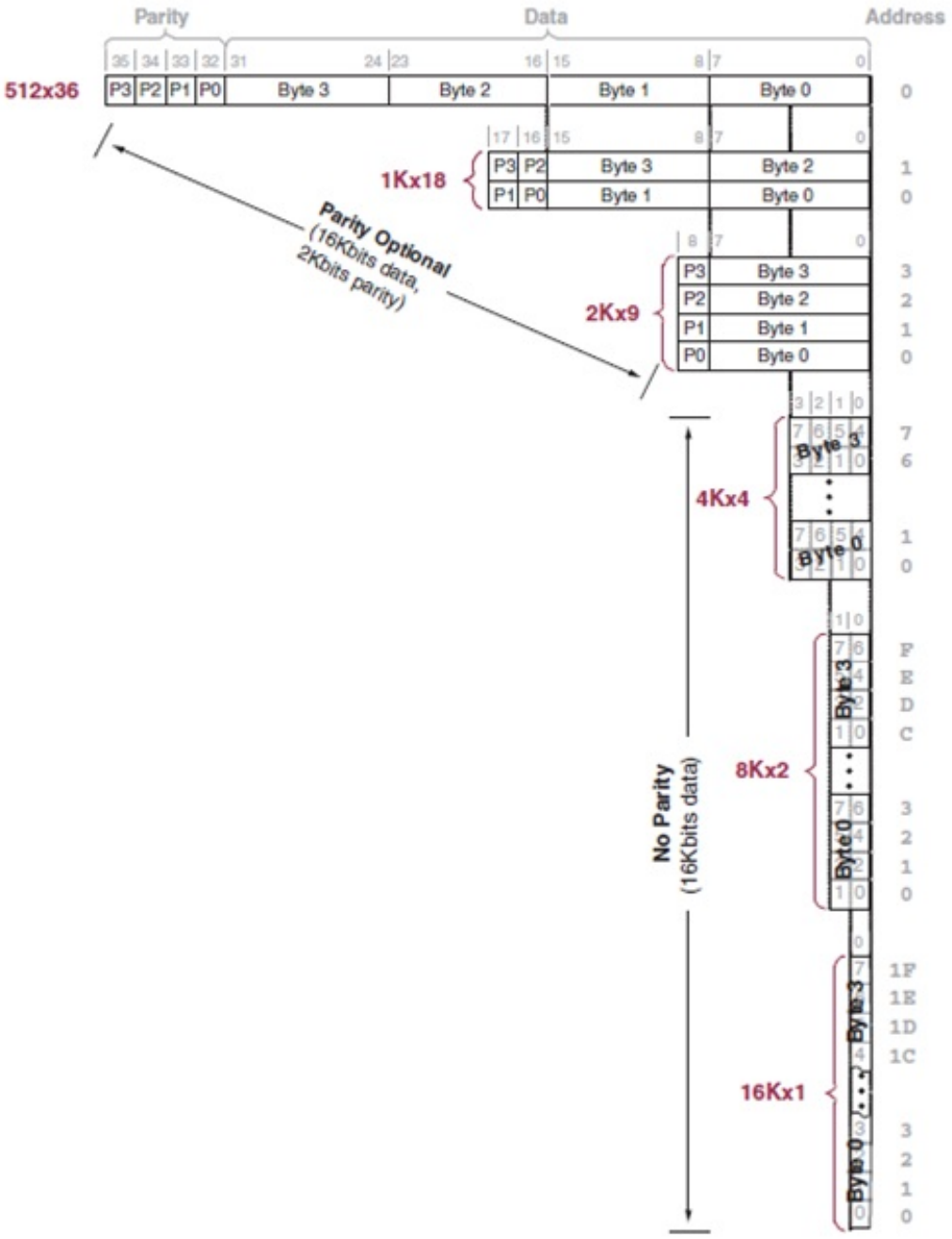


Figura 4.3: Possibili organizzazioni interne della Block Ram

tramite i quali si può settare in base alle specifiche di progetto l'organizzazione interna, la dimensione e diverse altre modalitÃ di funzionamento che la Block Ram offre all'utente.

Generalmente il numero di porte della ram e la sua organizzazione interna possono essere specificati utilizzando Xilinx Core Generator che consente di configurare tramite un wizard la Block Ram ottenendo direttamente il codice VHDL del componente ram desiderato oppure si possono utilizzare i tipi VHDL già associati alla Block Ram RAMB16_Sn dove n corrisponde all'ampiezza del dato + parità.

Organization	Memory Depth	Data Width	Parity Width	DI/DO	DIP/DOP	ADDR	Single-Port Primitive	Total RAM Kbits
512x36	512	32	4	(31:0)	(3:0)	(8:0)	RAMB16_S36	18K
1Kx18	1024	16	2	(15:0)	(1:0)	(9:0)	RAMB16_S18	18K
2Kx9	2048	8	1	(7:0)	(0:0)	(10:0)	RAMB16_S9	18K
4Kx4	4096	4	-	(3:0)	-	(11:0)	RAMB16_S4	16K
8Kx2	8192	2	-	(1:0)	-	(12:0)	RAMB16_S2	16K
16Kx1	16384	1	-	(0:0)	-	(13:0)	RAMB16_S1	16K

Figura 4.4: La tabella mostra le diverse tipologie di RAMB_Sn ottenibili dalla Block Ram in base all'organizzazione interna desiderata

- `INIT_xx – INITP_xx` A default la block ram è inizializzata a tutti 0, ma è possibile in inizializzarne il contenuto in diversi modi o direttamente tramite Core Generator al momento della configurazione del componente oppure tramite opportuni attributi VHDL come INIT_xx e INITP_xx (per inizializzare i bit di parità). Nel primo caso si passa direttamente un file di coefficienti (.coe) che definisce in primo luogo la base numerica dei dati da inserire e in seguito l'elenco dei dati elencati a partire dalla parte bassa della memoria fino agli indirizzi alti. Un esempio della struttura di tale file è il seguente:

  memory_inizialization_radix=16;
  memory_inizialization_vector=80, 0F, 00, 0B, ..., 82;

  Altrimenti si utilizzano direttamente 64 attributi VHDL INIT_xx (da INIT_00 a INIT_3F) che consentono di inizializzare le 64 zone da 256bit con cui è ripartita la memoria. Gli indirizzi del blocco di memoria da inizializzare identificati da xx sono calcolabili nel seguente modo dopo aver convertito l'indirizzo esadecimale xx nel corrispondente indirizzo decimale yy:
  indirizzo iniziale del blocco xx = ((yy+1)*256) - 1

indirizzo finale del blocco xx = yy*256

Attribute	From	To
INIT_00	255	0
INIT_01	511	256
INIT_02	767	512
...	...	...
INIT_3F	16383	16128

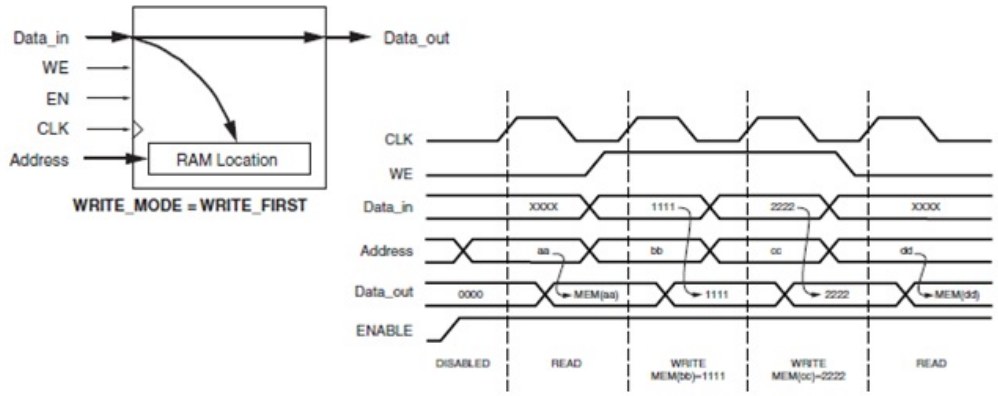
Figura 4.5: Attributi di Inizializzazione del contenuto della Block Ram

INITP_xx sono attributi analoghi che consentono di inizializzare i bit di parit à presenti in memoria (da INITP_00 a INITP_07).

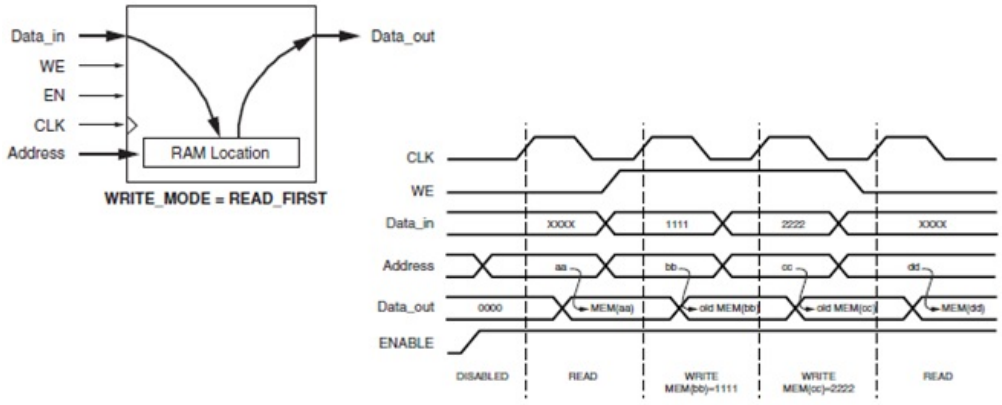
- `INIT` è l'attributo utilizzato in fase di inizializzazione per settare il valore iniziale del registro di output quando viene asserito il segnale GSR.

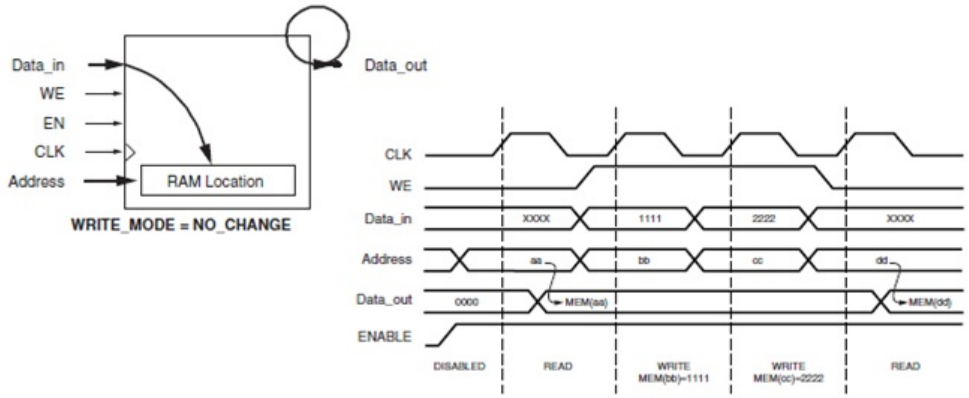
- `WRITE_MODE` è l'attributo che consente di settare il comportamento dei registri in output (relativamente ad una porta) che forniscono il dato sull'Output Data Bus durante un ciclo di scrittura in memoria.

1. `WRITE_FIRST` è il valore di default e comporta un comportamento Read after Write della memoria, ovvero durante un ciclo di scrittura il dato in input viene contemporaneamente scritto alla locazione di memoria indicata dall'indirizzo e portato nel registro di output. Nel caso di utilizzo in Dual-Port si ha l'invalidazione del contenuto del registro di output dell'altra porta.



2. `READ_FIRST` determina un comportamento Read before Write, ovvero prima si carica nel buffer di output il dato (passato) presente alla locazione di memoria specificata dall'indirizzo e poi si sovrascrive tale zona di memoria col dato in ingresso (si effettua la scrittura in memoria).
3. `NO_CHANGE` determina un comportamento classico di scrittura in memoria senza alcun aggiornamento del dato contenuto nel registro in output. Nel caso di utilizzo in Dual-Port si ha come side-effect l'invalidazione del contenuto del registro di output dell'altra porta.





3 Operazioni della Block Ram

Di seguito viene riportato l'elenco delle operazioni che la Block Ram è in grado di gestire e dei relativi segnali impiegati:

- Global Set/Reset: segue la fase di inizializzazione iniziale del contenuto della Block Ram in cui si inizializza la ram o a tutti zeri (default) o ai valori impostati con gli attributi `INIT_xx`. Tale segnale serve per inizializzare lo stato dei flipflop e registri di output che vengono settati in base al valore specificato dall'attributo `INIT` (0 a default).
- RAM Disabled: se il segnale `EN` non è asserito la ram mantiene il proprio stato. Ogni operazione prevede che EN venga asserito affinchè la ram sia attiva.

- Synchronous Set/Reset: è l'operazione conseguente all'asserzione contemporanea dei segnali EN e SSR. Tale operazione comporta la re inizializzazione dei registri di output al valore specificato dall'attributo SRVAL.
- WE + SSR comporta un ciclo di scrittura in cui il dato in input viene salvato in memoria all'indirizzo presente sul bus degli indirizzi, mentre il registro di output viene impostate al valore SRVAL.
- READ: la lettura sulla block ram avviene in modo sincrono, quindi sul fronte positivo del clock qualora sia asserito il solo segnale di EN.
- WRITE: la scrittura sulla block ram avviene in modo sincrono sul fronte positivo del clock e qualora siano asseriti contemporaneamente EN + WE. La scrittura del dato in input sui pin dell'Input Data Bus avviene all'indirizzo specificato e tale operazione è affiancata contemporaneamente dalla lettura del dato alla stessa locazione di memoria che viene reso disponibile in lettura e caricato sui registri di output (naturalmente la politica con la quale avviene tale operazione di scrittura e lettura simultanea è definita dal valore dell'attributo WRITE_MODE visto in precedenza).

La seguente tabella racchiude quanto detto in precedenza e associa ad ogni operazione i valori dei segnali associati.

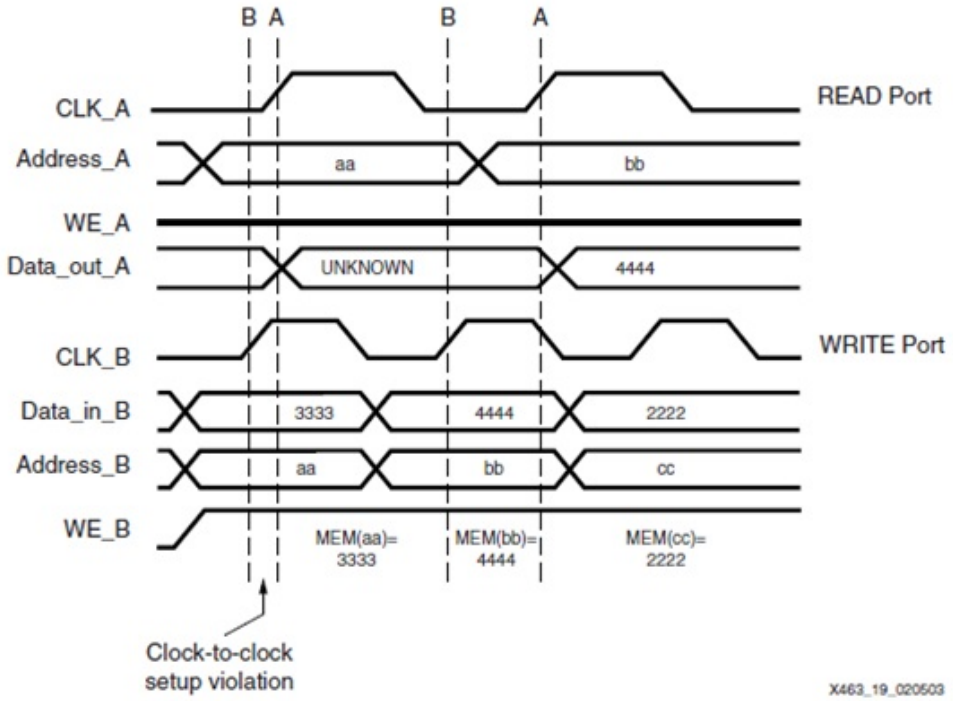
4 Conflitti d'accesso in Block Ram Dual-Port

Utilizzando la block ram in modalità Dual-Port si ha la possibilità di utilizzare contemporaneamente le due porte per accedere alla memoria sia in lettura e scrittura e mentre da un lato ciò consente di aumentare lo throughput complessivo dei dati trasferiti, dall'altro vi sono potenziali problemi di conflitto negli accessi simultanei alle stesse celle di memoria.
Le condizioni di potenziale conflitto si hanno nei seguenti casi:

Input Signals								Output Signals		RAM Contents	
GSR	EN	SSR/RST	WE	CLK	ADDR	DIP	DI	DOP	DO	Parity	Data
Immediately After Configuration											
Loaded During Configuration								X	X	INITP_xx[2]	INIT_xx[2]
Global Set/Reset Immediately after Configuration											
1	X	X	X	X	X	X	X	INIT[3]	INIT	No Chg	No Chg
RAM Disabled											
0	0	X	X	X	X	X	X	No Chg	No Chg	No Chg	No Chg
Synchronous Set/Reset											
0	1	1	0	↑	X	X	X	SRVAL[4]	SRVAL	No Chg	No Chg
Synchronous Set/Reset during Write RAM											
0	1	1	1	↑	addr	pdata	Data	SRVAL	SRVAL	RAM(addr) ←pdata	RAM(addr) ← data
Read RAM, no Write Operation											
0	1	0	0	↑	addr	X	X	RAM(pdata)	RAM(data)	No Chg	No Chg
Write RAM, Simultaneous Read Operation											
0	1	0	1	↑	addr	pdata	Data	WRITE_MODE – WRITE_FIRST[5] (default)			
								pdata	data	RAM(addr) ←pdata	RAM(addr) ← data
								WRITE_MODE – READ_FIRST[6] (recommended)			
								RAM(data)	RAM(data)	RAM(addr) ←pdata	RAM(addr) ←pdata
								WRITE_MODE – NO_CHANGE[7]			
								No Chg	No Chg	RAM(addr) ←pdata	RAM(addr) ←pdata

1. Scrittura simultanea sulle due porte alla stessa locazione di memoria.
   Tale situazione non ha un meccanismo di arbitraggio per far fronte ad accessi in scrittura simultanei, ma l'effetto prodotto Ã¨ quello di comportare l'invalidazione del contenuto dell'area di memoria coinvolta.
2. Conflitti per temporizzazioni clock-to-clock tra le due porte.
   Ciò accade a causa dei clock diversi che comandano le operazioni tra le due porte che sono troppo ravvicinati tra loro e il clock della porta in lettura non rispetta i tempi di setup per l'accesso in scrittura al dispositivo (arriva troppo presto quan-

do ancora non la scrittura in memoria non ha terminato). Un esempio è il seguente:



Nel primo caso, la porta B inizia la scrittura in memoria all'indirizzo aa del dato 3333 e poco dopo, prima che la scrittura abbia terminato, arriva il fronte del CLK_A che fa iniziare la lettura allo stesso indirizzo aa violando il tempo di setup necessario per scrivere il dato in memoria. Nel secondo caso invece si ha la scrittura da parte della porta B all'indirizzo bb del dato 4444 e in questo caso CLK_A rispetta le temporizzazioni di scrittura e la porta A legge il dato correttamente scritto in memoria.

3. Scrittura e Lettura contemporanea sulla stessa zona di memoria in funzione del WRITE_MODE impostato.
   Nei casi di scrittura su una porta e lettura sull'altra, se si utilizza WRITE_MODE= NO_CHANGE o WRITE_FIRST, la scrittura su una porta invalida automaticamente il contenuto del registro di output (in lettura) dell'altra porta, per tale motivo è con-

sigliabile la modalit à di scrittura READ_FIRST per evitare conflitti sulla porta in lettura.

Input Signals								Output Signals			
Port A				Port B				Port A		Port B	
WEA	CLKA	DIPA	DIA	WEB	CLKB	DIPB	DIB	DOPA	DOA	DOPB	DOB
WRITE_MODE_A=NO_CHANGE											
1	↑	DIPA	DIA	0	↑	DIPB	DIB	No Chg	No Chg	?	?
WRITE_MODE_B=NO_CHANGE											
0	↑	DIPA	DIA	1	↑	DIPB	DIB	?	?	No Chg	No Chg
WRITE_MODE_A=WRITE_FIRST											
1	↑	DIPA	DIA	0	↑	DIPB	DIB	DIPA	DIA	?	?
WRITE_MODE_B=WRITE_FIRST											
0	↑	DIPA	DIA	1	↑	DIPB	DIB	?	?	DIPB	DIB
WRITE_MODE_A=WRITE_FIRST, WRITE_MODE_B=WRITE_FIRST											
1	↑	DIPA	DIA	1	↑	DIPB	DIB	?	?	?	?

Per semplicità implementativa la Block Ram non implementa un sistema di arbitraggio per gestire tali conflitti che sono lasciati a cura del progettista e comunque in caso di conflitto dovuto a scritture contemporanee non si verificano danni fisici al dispositivo di memoria.

5 Possibili utilizzi della Block Ram in un progetto su FPGA

La Block Ram può essere utilizzata in un progetto su FPGA per implementare una serie di funzionalità che coinvolgano la memorizzazione di dati. I principali possibili utilizzi sono i seguenti:

1. RAM utilizzata da un microprocessore integrato sull'FPGA per memorizzare dati accessibili in lettura e scrittura.
2. ROM realizzata attraverso l'inizializzazione del suo contenuto all'avvio del sistema e accessibile in sola lettura.
3. Memorie FIFO.

6 Realizzazione di un progetto d'esempio d'uso

Tipicamente per utilizzare la block ram all'interno di un progetto si procede come segue:

1. Si crea un componente Block Ram configurandolo in base alle specifiche di progetto, settando il numero di porte volute, l'ampiezza dei dati da trasferire, la dimensione della ram voluta, etc. Tale operazione può essere fatta o ricorrendo ad una serie di template presenti tra i Language Templates Ram di ISE oppure tramite una configurazione ad hoc tramite Xilinx Core Generator che tramite un wizard consente di personalizzare il componente Ram di cui si ottiene infine il codice VHDL.
2. Si integra il componente all'interno del progetto dichiarandolo nell'Architecture del componente finale e creandone un istanza tramite il `port mapping`.
3. Si utilizza il componente che rappresenta la Block Ram comandando i segnali di input e gestendo opportunamente i valori in output.

Al fine di testare il funzionamento della Block Ram e approfondire le problematiche che vi sarebbero state nel progettare una cache reale che si interfaccia con una Ram esterna il cui tempo di accesso non Ã¨ nullo, abbiamo realizzato un componente Ram ad hoc. Tale componente rappresenta una memoria Ram sincrona (il cui funzionamento è scandito dal clock in ingressso) realizzato con lo scopo di interfacciarsi con il nostro componente cache. Per comodità abbiamo ipotizzato che il nuovo componente, `BlockRam_cmp`, si interfacci alla cache sempre tramite un bus dati dell'ampiezza della linea di memoria da trasferire. Tale ipotesi che ovviamente è semplificativa e porta ad una potenziale complessità del cablaggio del bus dati è tuttavia lecita dal momento che i trasferimenti tra cache e ram coinvolgono sempre linee di memoria. Ciò detto,

il nuovo componente prevede l'utilizzo al suo interno di un componente `BRAM16_s9` capace di leggere e scrivere sulla Block Ram dati da 8 bit (+ 1 bit di parità che non abbiamo considerato). La scelta di tale componente Block Ram Ã¨ derivata dall'ipotesi che le linee di memoria sono di dimensione sempre multipla di 1 Byte e quindi il componente `BlockRam_cmp` ad ogni operazione di lettura o scrittura di una linea dovrÃ provvedere ad un ciclo di trasferimento dei singoli Byte costitutivi la linea a partire dall'indirizzo specificato in ingresso sul bus degli indirizzi che ad ogni accesso dovrà essere incrementato opportunamente. In particolare i casi gestiti sono due:

1. Scrittura di una linea in Block Ram: deve prevedere il campionamento della linea in ingresso al bus dati di input e provvedere al trasferimento della linea byte per byte su block ram tramite una serie di scritture.
2. Lettura di una linea da Block Ram: deve prevedere un buffer (una variabile VHDL `mem_line`) che viene riempito man mano attraverso n letture di byte dalla Block Ram (dove n Ã¨ il numero di byte che compongono una linea); Al termine la linea letta deve essere data in uscita sul bus dati di output verso la cache.

Naturalmente l'accesso alla Block Ram non Ã¨ istantaneo ma comporta un tempo d'accesso necessario che deve essere considerato per generare opportunamento il segnale di `ready` laddove l'operazione di scrittura/lettura dell'intera linea di memoria ha terminato, segnalando ciò al processore e alla cache.

Conclusioni

Non dimentichiamoci di fare le conclusioni.

Bibliografia