

YRT/2007/5

Modello architetturale e prestazionale del processore Cell BE

Roma, 27/07/2007

Redazione

Ing. Marco Pollini

Approvazione

Ing. Paolo Palazzari

[illegible]

1 INTRODUZIONE.....	1
2 UNITÀ DI CALCOLO CBE.....	1
2.1 PPU – POWERPC PROCESSOR UNIT.....	1
2.2 SPU – SYNERGISTIC PROCESSOR UNIT.....	4
3 UNITÀ DI MEMORIZZAZIONE E COMUNICAZIONE CBE.....	5
3.1 PPSS – POWERPC PROCESSOR STORAGE SUBSYSTEM.....	5
3.2 REGISTRI PPE.....	6
3.3 MFC - MEMORY FLOW CONTROLLER.....	7
4 PRIMITIVE DI TRASFERIMENTO DATI E SINCRONIZZAZIONE.....	9
4.1 TRASFERIMENTO DATI DA SPE TRAMITE DMA.....	9
4.2 TRASFERIMENTO DATI DA PPE TRAMITE DMA.....	11
4.3 TRASFERIMENTO DATI SPE – SPE TRAMITE DMA.....	12
4.4 SINCRONIZZAZIONE TRAMITE MAILBOX.....	13
4.5 SINCRONIZZAZIONE TRAMITE MUTUA ESCLUSIONE (MUTEX).....	13
4.6 SINCRONIZZAZIONE CON VARIABILI CONDIZIONALI (CONDITION VARIABLES).....	14
4.7 SINCRONIZZAZIONE CON INTERRUPT.....	15
4.8 SINCRONIZZAZIONE SPE – SPE.....	15
5 ANALISI PRESTAZIONI DI TRASFERIMENTO DATI, SINCRONIZZAZIONE E OPERAZIONI DI CALCOLO.....	16
5.1 PRESTAZIONI DEL TRASFERIMENTO DATI TRAMITE DMA.....	16
5.2 COSTO DELLE OPERAZIONI DI SINCRONIZZAZIONE.....	20
5.2.1 Sincronizzazione con mailbox.....	20
5.2.2 Sincronizzazione con variabili mutex.....	21
5.2.3 Sincronizzazione con interrupt.....	21
5.2.4 Sincronizzazione tra SPE e SPE.....	22
5.3 COSTO DELLE OPERAZIONI DI CALCOLO.....	22
5.3.1 Instruction set PPE (PowerPC).....	22
5.3.1.1 Istruzioni Vector/SIMD Multimedia.....	24
5.3.2 Instruction set SPU.....	25

1 Introduzione

In questo documento verrà creato un modello del processore CellBE nel quale metteremo in evidenza

1. le unità di calcolo, di comunicazione e di memorizzazione dell'architettura CellBE.
2. le primitive di trasferimento dati e di sincronizzazione.
3. il costo corrispondente all'esecuzione delle operazioni di trasferimento dati e comunicazione, di sincronizzazione e di calcolo.

2 Unità di calcolo CBE

2.1 PPU – PowerPC Processor Unit

Analizziamo l'unità di calcolo del PPE (PowerPC Processor Element), denominata PPU, PowerPC Processor Unit. Questa unità si occupa dell'esecuzione delle istruzioni. E' dotata di una cache di 1° livello (L1) suddivisa per dati e istruzioni e di sei diverse unità di esecuzione, come riportato in Figura 1.

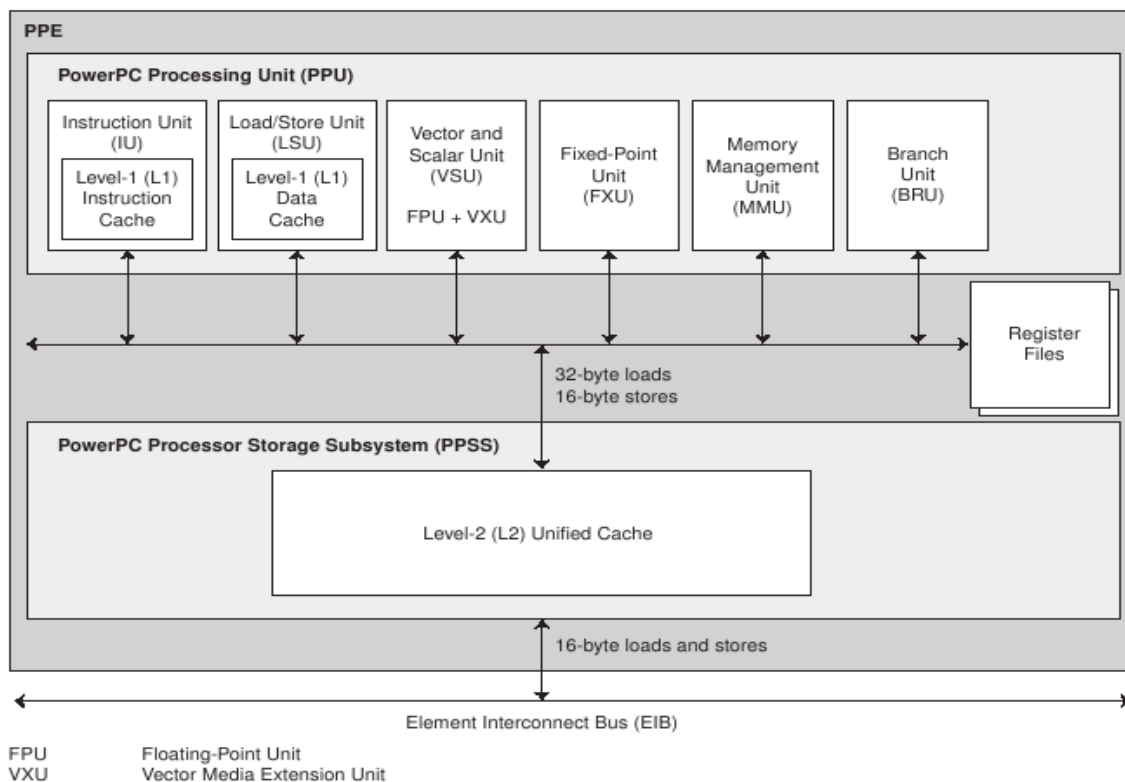


Figura 1: Unità funzionali del PPE

Il PPU esegue le istruzioni appartenenti all'istruzione-set dell'architettura PowerPC e le istruzioni dell'estensione Vector/SIMD Multimedia. E' dotato di un insieme di register-file duplicato in quanto l'hardware supporta l'esecuzione di due thread simultanei, e delle seguenti unità funzionali:

- **Instruction Unit (IU)** – Si occupa delle fasi di instruction-fetch, decode, dispatch, issue e branch. Al suo interno è presente la cache L1 per le istruzioni, di dimensione 32 KB, associativa a 2 vie. Ogni linea di cache è di 128 byte.
- **Load and Store Unit (LSU)** – Si occupa dell'accesso ai dati ed esegue le istruzioni di load e store. Contiene la cache L1 per i dati che ha dimensione 32 KB, associativa a 4 vie. Ogni linea di cache è di 128 byte.
- **Vector/Scalar Unit (VSU)** – Questa unità è suddivisa in due sottounità: la Floating-Point Unit (FPU) e la 128-bit Vector/SIMD Multimedia Extension Unit (VXU) che insieme si occupano dell'esecuzione delle istruzioni floating-point e di quelle Vector/SIMD.
- **Fixed-Point Unit (FXU)** – Esegue le istruzioni fixed-point tra cui le seguenti: add, multiply, divide, compare, shift, rotate and logical.
- **Memory Management Unit (MMU)** – Questa unità gestisce la traduzione degli indirizzi per gli accessi in memoria. E' dotata di un Segment Lookaside Buffer a 64 elementi (SLB) e di un Translation Lookaside Buffer a 1024 elementi (TLB). Supporta la gestione simultanea di tre pagine di memoria di diverse dimensioni: 4 KB più due selezionabili tra 64 KB, 1MB, 16 MB.

L'unità VXU può lavorare insieme alle due unità FXU e FPU come mostrato in Figura 2 ma con le limitazioni legate al dual-issue delle istruzioni nelle pipeline di esecuzione descritto più avanti.

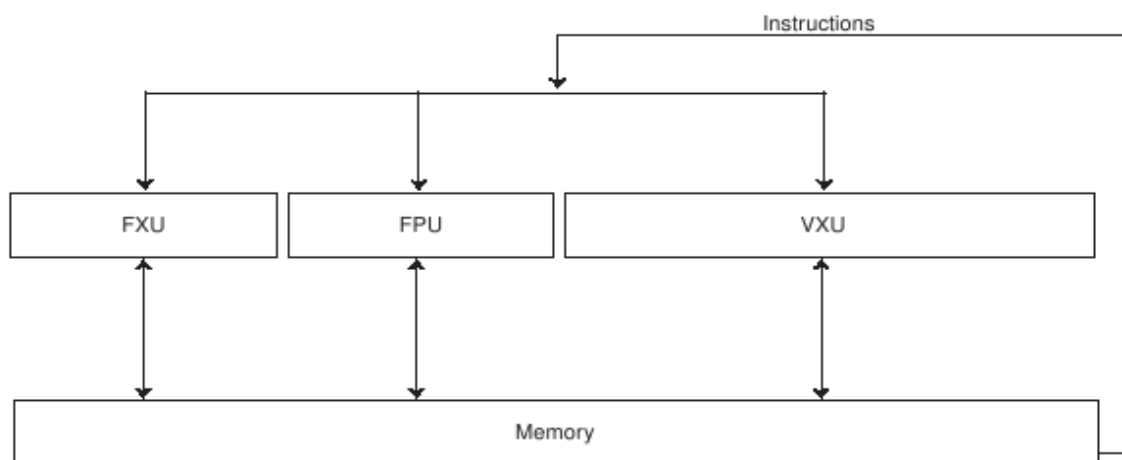


Figura 2 : Esecuzione concorrente nelle unità FXU, FPU e VXU

Queste limitazioni sono essenzialmente legate al tipo di istruzioni che possono essere eseguite assieme. La Figura 3 mostra le possibili combinazioni; i quadrati colorati indicano le combinazioni accettate mentre i quadrati bianchi quelle illegali.

		Slot 1				
		FXU	LSU	BRU	VSU Type 1	VSU Type 2
Slot 0	FXU					
	LSU					
	BRU					
	VSU Type 1					
	VSU Type 2					

BRU	Branch Unit
FPU	Floating-Point Unit
FXU	Fixed-Point Unit
LSU	Load and Store Unit
VSU	Vector Scalar Unit (a combination of VXU and FPU)
VXU	Vector/SIMD Multimedia Extension Unit

Figura 3 : Combinazioni del dual-issue instruction

L'esecuzione parallela di istruzioni Vector/SIMD con quelle fixed-point o floating-point è agevolata dal fatto che le istruzioni Vector/SIMD non generano eccezioni, non supportano gli accessi in memoria disallineati e condividono molte delle risorse con le altre unità del PPE.

L'unità MMU gestisce la traduzione degli indirizzi di memoria virtuale e la protezione degli accessi stessi in accordo con l'architettura PowerPC a 64 bit. Questa unità gestisce inoltre i seguenti spazi di indirizzamento:

- **Real Address (RA) Space** – 242 bytes. Fanno parte di questo spazio di memoria tutti gli indirizzi, compresi anche quelli dei device I/O, nei quali sono stati mappati gli indirizzi fisici da parte del PPE.

- **Effective Address (EA) Space** – 264 bytes. Gli indirizzi generati da un programma in esecuzione appartengono a questo spazio di memoria.
- **Virtual Address (VA) Space** – 265 bytes. Fanno parte di questo spazio gli indirizzi utilizzati dall'unità MMU e dal MFC di ogni SPE per tradurre gli indirizzi EA in indirizzi RA.

2.2 SPU – Synergistic Processor Unit

L'unità SPU fa parte assieme all'unità MFC (Memory Flow Controller) dell'SPE (Synergistic Processor Element). Si occupa di gestire il fetch delle istruzioni dal relativo Local Store ed esegue operazioni di load e store tra il LS e il register-file unificato. Tale register-file è unico per ogni tipo di dato ed è costituito da 128 registri ognuno a 128 bit. La presenza di un register-file di tali dimensioni compensa la mancanza di una memoria cache.

Ogni unità SPU è costituita da 4 unità funzionali, un'interfaccia DMA e una channel-interface adibita alla comunicazione con la MFC, il PPE e gli altri device presenti nel CBE. Ogni SPU è indipendente e dotato di un proprio Program Counter. Le unità funzionali sono mostrate in Figura 4.

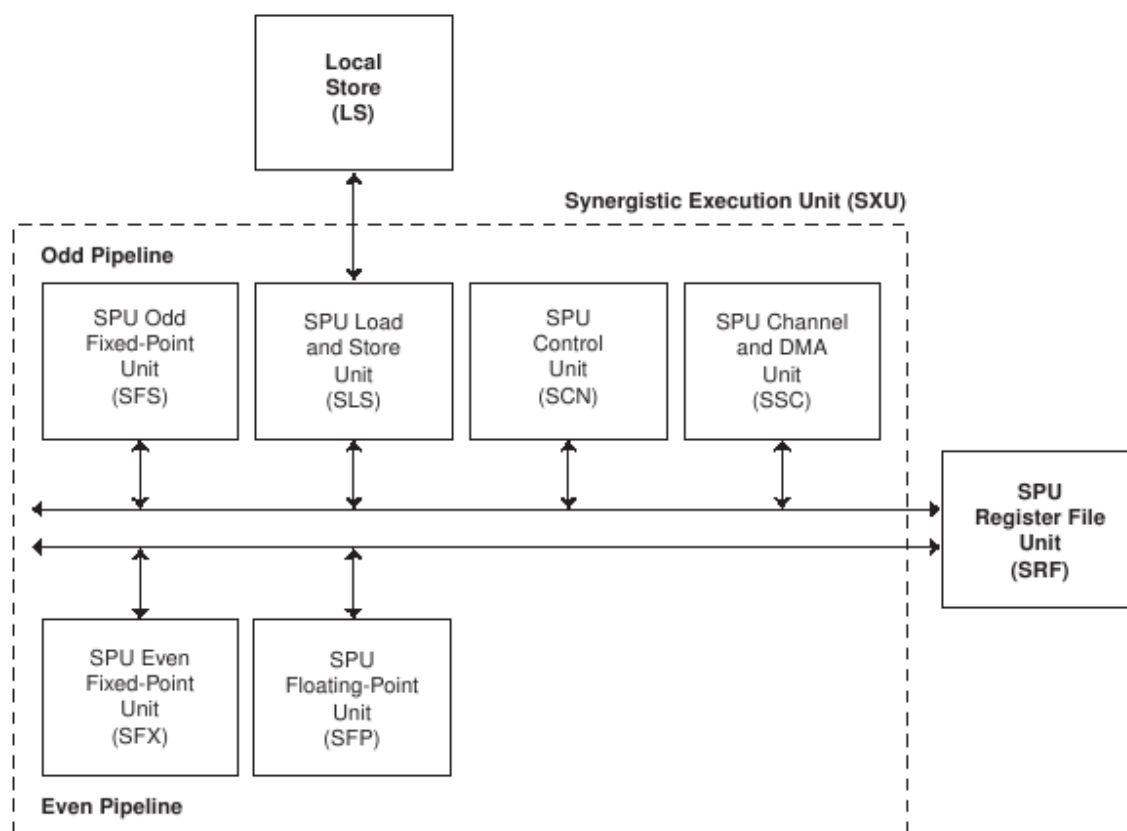


Figura 4 : Unità funzionali di un SPU

L'unità SPU può eseguire fino a due istruzioni per ciclo di clock smistandole sulle due pipeline di esecuzione (Even Pipeline e Odd Pipeline). L'unità funzionale SXU è composta dalle seguenti sottounità:

- ***SPU Odd Fixed-point Unit (SFS)*** – Esegue le operazioni di quadword-shift e rotate-mask su dati di dimensione bit, byte, halfword, word e operazioni di shuffle su dati di dimensione byte.
- ***SPU Even Fixed-point Unit (SFX)*** – Esegue istruzioni aritmetiche, logiche, shift e rotazioni SIMD, floating-point compare e floating point square-root.
- ***SPU Floating-point Unit (SFP)*** – Esegue istruzioni floating-point a singola e doppia precisione e moltiplicazioni tra interi. L'SPU supporta solamente moltiplicazioni a 16 bit.
- ***SPU Load and Store Unit (SLS)*** – Esegue le istruzioni di load e store e gestisce inoltre le richieste DMA verso il LS.
- ***SPU Control Unit (SCN)*** – Si occupa del fetch e dello smistamento delle istruzioni nelle due pipeline, esegue le istruzioni di branch, gestisce gli accessi al LS e al register-file.
- ***SPU Channel and DMA Unit (SSC)*** – Permette la comunicazione e il trasferimento dati da e verso l'SPU.

3 Unità di memorizzazione e comunicazione CBE

3.1 PPSS – PowerPC Processor Storage Subsystem

L'unità PPSS gestisce tutti gli accessi di memoria effettuati dal PPU e le richieste provenienti dal bus EIB. All'interno della PPSS è presente la cache L2 unificata per dati e istruzioni, di dimensione 512 KB, associativa a 8 vie e con codice a correzione di errore (ECC). Come per la cache L1 ogni linea di cache è di 128 byte.

Il PPU si interfaccia con la PPSS tramite una porta di lettura di larghezza 32 byte condivisa tra l'unità MMU, la cache istruzioni L1 e le richieste verso la cache dati L1, e un'altra porta di scrittura di larghezza 16 byte condivisa tra MMU e richieste verso la cache dati L1. L'interfaccia tra PPSS e bus EIB supporta operazioni di load e store a 16 byte.

3.2 Registri PPE

Tutte le operazioni avvengono tra registri; nessuna istruzione può modificare direttamente il contenuto del main-storage. Per poter utilizzare un operando in memoria durante un'operazione di calcolo è necessario caricarlo in un registro, eventualmente modificarlo in base alle operazioni da eseguire e poi scrivere nuovamente il valore nella locazione di memoria. Il set di registri del PPE è riportato in Figura 5.

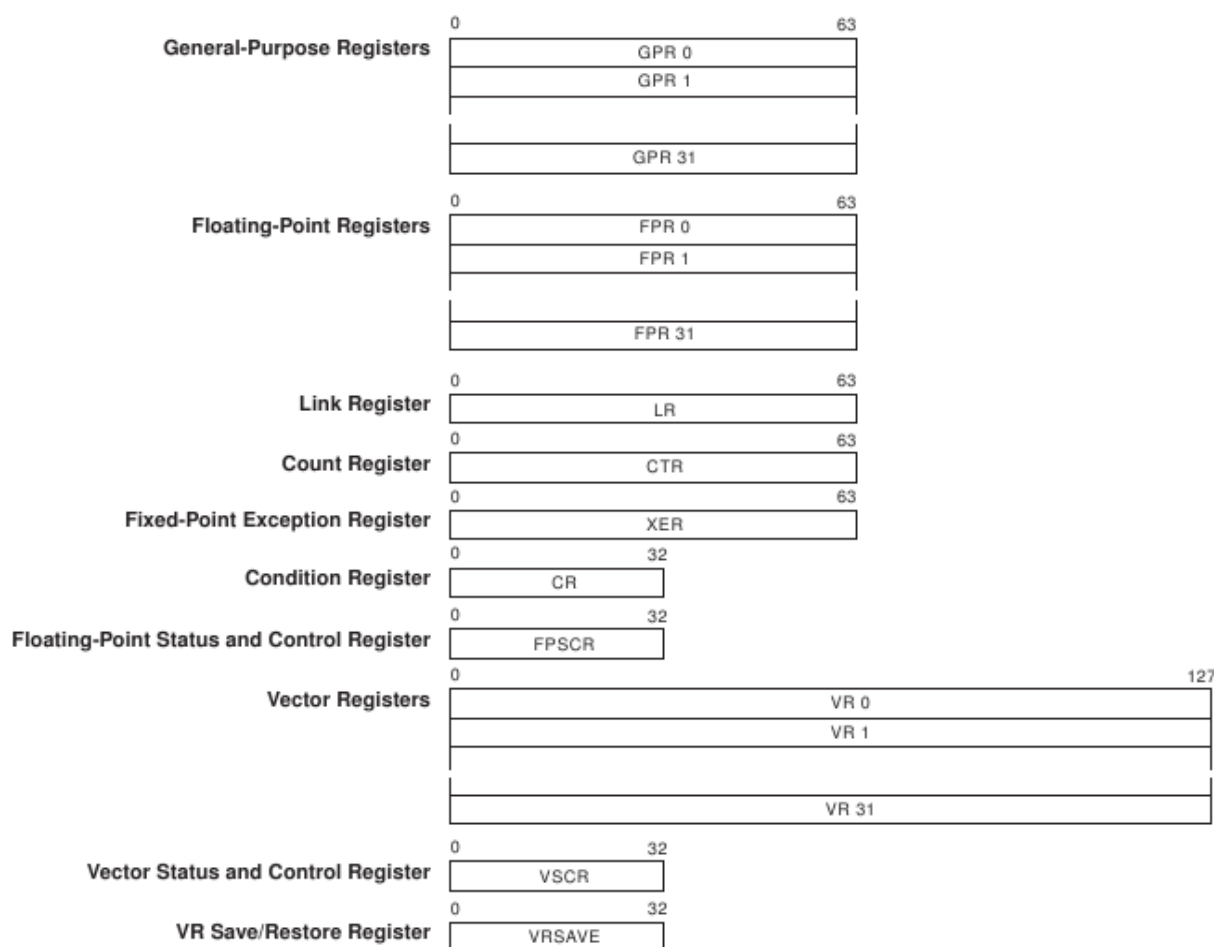


Figura 5 : Set dei registri PPE

- **General Purpose Registers (GPRs)** – Sono 32 registri a 64 bit. Le istruzioni fixed-point operano sull'intero registro a 64 bit.
- **Floating-Point Registers (FPRs)** – Sono 32 registri a 64 bit. I dati floating-point memorizzati in questi registri sono nel formato IEEE 754 a doppia precisione.

- **Link Register (LR)** – Il Link Register a 64 bit può essere utilizzata per memorizzare l'indirizzo di salto di un'operazione di branch.
- **Count Register (CTR)** – Il registro CTR a 64 bit può essere utilizzato per memorizzare sia la variabile contatore di un ciclo che l'indirizzo di salto di un branch. E' utilizzato a volte dalle istruzioni di salto condizionato per memorizzare la variabile condizionale.
- **Fixed-Point Exception Register (XER)** – E' un registro a 64 bit che contiene i bit di riporto e di overflow delle operazioni aritmetiche.
- **Condition Register (CR)** – Le operazioni di confronto condizionale sono eseguite settando prima un valore condizionale nel registro CR a 32 bit che può essere poi verificato da un'istruzione di salto per il controllo di flusso di un programma.
- **Floating-Point Status and Control Register (FPSCR)** – Questo registro a 32 bit viene aggiornato dopo ogni operazione PowerPC floating-point per memorizzare le informazioni sul risultato e su ogni possibile eccezione.
- **Vector Registers (VRs)** – Sono 32 registri a 128 bit utilizzati dalle istruzioni vettoriali del PPE
- **Vector Status and Control Register (VSCR)** – Ha una funzione simile al registro Floating-Point Status and Control Register (FPSCR) ma per le operazioni solo vettoriali.
- **Vector Save Register (VRSAVE)** – Il registro VRSAVE a 32 bit permette al software il salvataggio e il ripristino dello stato del processore durante i context switch.

3.3 MFC - Memory Flow Controller

Ogni SPU ha la propria unità MFC che funziona da interfaccia tra l'SPU il main-storage e gli altri processor elements e device del CBE.

Il compito principale del MFC è quello di interfacciare il Local Store domain con il main-storage domain.

In Figura 6 è riportato lo schema a blocchi dell'unità MFC.

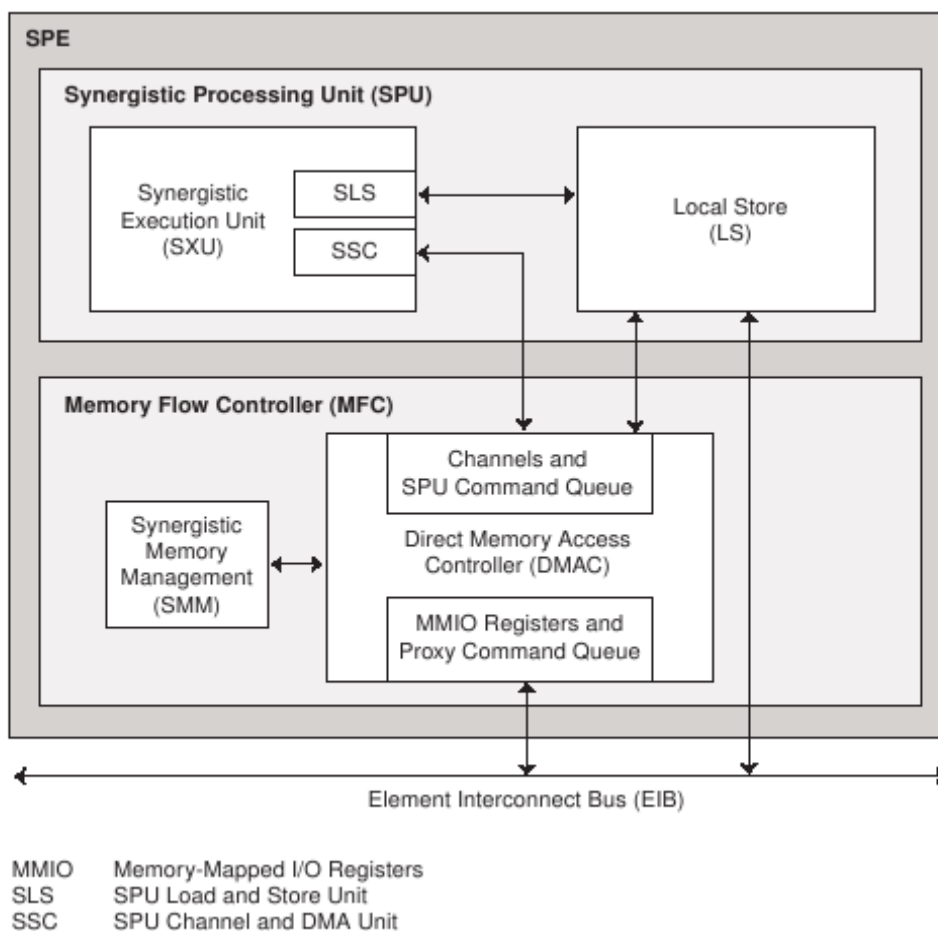


Figura 6 – Diagramma a blocchi dell'unità MFC

I componenti principali sono i seguenti:

- **LS Read and Write Interface** – Collega il LS con il bus EIB. E' utilizzata per trasferimenti DMA, atomici e per le richieste provenienti dal bus EIB.
- **LS DMA List Element Read Interface** – Collega il LS con l'interfaccia Channels and SPU Command Queue. E' utilizzata per implementare efficienti trasferimenti di dati tramite DMA List.
- **Channel Interface** – Collega l'unità SXU e SSC (DMA Unit) con l'interfaccia Channels and SPU Command Queue.
- **EIB Command and Data Interface** – Collega l'unità MFC con il bus EIB. Consiste in un bus ingresso-uscita a 128 bit.

L'MFC di un SPE può essere considerato come co-processore dell'SPU associato che esegue autonomamente le operazioni richieste durante l'esecuzione di un programma. L'MFC è dotato di due code indipendenti che memorizzano la sequenza dei comandi da eseguire; una prima coda di 16 elementi detta MFC SPU Command Queue e una seconda coda di 8 elementi detta MFC Proxy Command Queue. I comandi da eseguire su richiesta dell'SPU sono inseriti esclusivamente nella SPU Command Queue, mentre quelli da eseguire su richiesta del PPE o di un altro device sono inseriti nella Proxy Command Queue. Ovviamente ogni SPU accede al proprio MFC tramite la già menzionata channel-interface.

Gli altri device nel sistema, incluso il PPE, utilizzano registri memory-mapped, detti MMIO register, per accedere al MFC di un SPE e inviare comandi. Ogni MFC ha un proprio controller DMA che implementa i trasferimenti DMA tra main-storage e Local Store. I comandi DMA contengono al loro interno i valori degli indirizzi di memoria del LS (LSA – Local Store Address) e del main-storage (EA – Effective Address).

Ogni MFC traduce i comandi DMA istanziati da un SPU nella coda SPU Command Queue in trasferimenti DMA effettivi e può iniziare e gestire trasferimenti multipli. Ad ogni comando MFC viene associato un attributo numerico a 5 bit detto Tag Group ID che viene utilizzato dal software dell'SPE per monitorare l'esecuzione e il completamento di una particolare sequenza di comandi in coda che abbiano un determinato Tag Group.

4 Primitive di trasferimento dati e sincronizzazione

In questo paragrafo tratteremo le primitive di trasferimento dati e di sincronizzazione che sono state introdotte nel precedente documento riguardante la definizione delle modalità di interazione tra PPE e SPE.

4.1 Trasferimento dati da SPE tramite DMA

Le funzioni utilizzabili, definite nel file SPE_DMA_API.h (da includere nel codice SPE), sono le seguenti:

```
void SPE_dma_read (volatile void *ls_addr, uint64_t *main_storage_addr, uint32_t dmasize, uint32_t tag_id)
```

Trasferisce la quantità di dati specificata da dmasize dall'indirizzo di memoria main_storage_addr del main-storage nel LS all'indirizzo ls_addr.

```
void SPE_dma_write (volatile void *ls_addr, uint64_t *main_storage_addr, uint32_t dmasize,  
                    uint32_t tag_id)
```

Trasferisce la quantità di dati specificata da dmasize dall'indirizzo ls_addr del LS nel main-storage all'indirizzo main_storage_addr.

La versione non bloccante di queste due funzioni, utilizzabile nel caso di trasferimenti multibuffer, è la seguente:

```
void SPE_dma_read_noblocking (volatile void *ls_addr, uint64_t *main_storage_addr, uint32_t  
                             dmasize, uint32_t tag_id)
```

```
void SPE_dma_write_noblocking (volatile void *ls_addr, uint64_t *main_storage_addr, uint32_t  
                              dmasize, uint32_t tag_id)
```

Possono essere istanziati dunque più trasferimenti DMA in parallelo nella SPU Command Queue fino ad un massimo di 16 alla volta.

Nel caso di comandi DMA con specificatori fence e barrier si hanno le seguenti funzioni:

```
void SPE_dma_read_fence (volatile void *ls_addr, uint64_t *main_storage_addr, uint32_t  
                        dmasize, uint32_t tag_id)
```

```
void SPE_dma_read_barrier (volatile void *ls_addr, uint64_t *main_storage_addr, uint32_t dmasize,  
                          uint32_t tag_id)
```

```
void SPE_dma_write_fence (volatile void *ls_addr, uint64_t *main_storage_addr, uint32_t  
                        dmasize, uint32_t tag_id)
```

```
void SPE_dma_write_barrier (volatile void *ls_addr, uint64_t *main_storage_addr, uint32_t  
                          dmasize, uint32_t tag_id)
```

I comandi DMA con fence o barrier sono particolarmente indicati nel caso di utilizzo di buffer di dati condivisi (shared buffers). Questa modalità permette il riutilizzo dello stesso buffer di memoria sia per i dati in entrata da elaborare che per quelli in uscita; il vantaggio è la minore richiesta di memoria LS. In Figura 7 riportiamo un esempio di utilizzo di un comando DMA “fenced” **getf**.

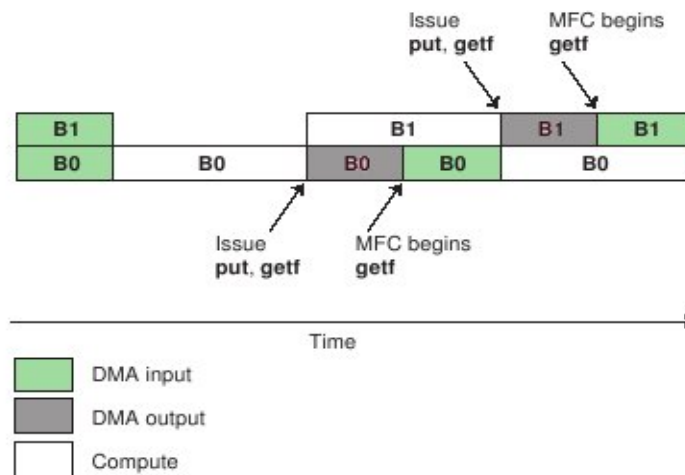


Figura 7 – Shared I/O buffers con DMA fenced

L'ordinamento dei comandi determinato dal fence assicura che l'istruzione **get** non venga iniziata prima che sia completato il trasferimento dati associato al comando **put**. L'utilizzo di comandi get fenced in un codice SPU elimina la necessità di attendere il completamento dei trasferimenti verso il main-storage, mantenendo nel contempo il parallelismo e la semplicità del codice.

4.2 Trasferimento dati da PPE tramite DMA

Le funzioni utilizzabili, definite nel file PPE_DMA_API.h (da includere nel codice PPE) sono le seguenti:

```
int PPE_dma_read (spe_context_ptr_t spe, unsigned int ls_addr, void *main_storage_addr,
                 unsigned int dmasize, unsigned int tag_id)
```

Inserisce un comando DMA get nella Proxy Command Queue per trasferire la quantità di dati dmasize dall'indirizzo di memoria main_storage_addr nel LS all'indirizzo ls_addr.

```
int PPE_dma_write (spe_context_ptr_t spe, unsigned int ls_addr, void *main_storage_addr,
                  unsigned int dmasize, unsigned int tag_id)
```

Inserisce un comando DMA put nella Proxy Command Queue per trasferire la quantità di dati dmasize dall'indirizzo di memoria main_storage_addr nel LS all'indirizzo ls_addr.

La versione non bloccante di queste due funzioni, utilizzabile nel caso di trasferimenti multibuffer, differisce dalle chiamate precedenti solo per il suffisso “noblocking” nel nome della funzione.

Come per i trasferimenti DMA da SPE è possibile utilizzare le seguenti funzioni con specificatori fence e barrier.

```
int PPE_dma_read_fence (spe_context_ptr_t spe, unsigned int ls_addr, void *main_storage_addr,  
                        unsigned int dmasize, unsigned int tag_id)
```

```
int PPE_dma_write_fence (spe_context_ptr_t spe, unsigned int ls_addr, void *main_storage_addr,  
                        unsigned int dmasize, unsigned int tag_id)
```

```
int PPE_dma_read_barrier (spe_context_ptr_t spe, unsigned int ls_addr, void *main_storage_addr,  
                        unsigned int dmasize, unsigned int tag_id)
```

```
int PPE_dma_write_barrier (spe_context_ptr_t spe, unsigned int ls_addr, void *main_storage_addr,  
                        unsigned int dmasize, unsigned int tag_id)
```

4.3 Trasferimento dati SPE – SPE tramite DMA

Le funzioni utilizzabili, definite nel file SPE_DMA_API.h (da includere nel codice SPE), sono le seguenti:

```
void SPE_LS_dma_read (uint64_t ls_source, volatile void *ls_dest, unsigned int dmasize, unsigned  
                    int tag_id)
```

Trasferisce la quantità di dati specificata da dmasize dall'indirizzo LS ls_source nel LS di un altro SPE all'indirizzo ls_dest. In questo caso l'indirizzo ls_source è comunicato all'SPE che esegue la “dma_read” da parte del PPE.

```
void SPE_LS_dma_write (uint64_t ls_source, volatile void *ls_dest, unsigned int dmasize, unsigned  
                    int tag_id)
```

Trasferisce la quantità di dati specificata da dmasize dall'indirizzo LS ls_source nel LS di un altro SPE all'indirizzo ls_dest. In questo caso l'indirizzo ls_dest è comunicato all'SPE che esegue la “dma_write” da parte del PPE.

4.4 Sincronizzazione tramite mailbox

Le funzioni utilizzate per la sincronizzazione dei thread SPE con il PPE, definite nei file `sync_API_PPE.h` e `sync_API_SPE.h` (da includere rispettivamente nel codice PPE e in quello SPE), sono le seguenti:

`void sync_SPE_mailbox (unsigned int num_SPE, ppu_pthread_data_t *thread)`

Sincronizza il PPE con i threads SPE indicati da `num_SPE`. E' necessario passare alla funzione come 2° parametro un puntatore al primo thread SPE creato dal PPE.

`void sync_PPE_mailbox (unsigned int SPU_id)`

Sincronizza il thread SPE indicato da `SPU_id` con il PPE.

4.5 Sincronizzazione tramite mutua esclusione (mutex)

Si utilizza la sincronizzazione tramite una variabile di mutua esclusione nella situazione in cui un thread (sia PPE che SPE) esegue una sezione critica di codice e gli altri thread devono rimanere in attesa del rilascio di questa variabile.

La semplicità delle chiamate alle funzioni di mutua esclusione ci permette di utilizzare direttamente queste funzioni senza un'ulteriore standardizzazione delle chiamate effettuata ad esempio per la sincronizzazione con le mailbox.

Le funzioni sono le seguenti:

`mutex_init (mutex_var)`

Questa routine inizializza la variabile `mutex_var` a zero.

`mutex_lock (mutex_var)`

Questa routine acquisisce il lock aspettando che la variabile `mutex_var` assuma il valore zero e successivamente setta a 1 tale variabile. Se il lock è già presente il thread che ha effettuato la chiamata viene messo in stallo.

`mutex_trylock (mutex_var)`

Questa routine cerca di acquisire il lock sulla variabile.

Se un lock è già presente viene ritornato il valore 0 e il lock non viene acquisito, in caso contrario viene ritornato 1 e il lock viene acquisito.

mutex_unlock (mutex_var)

Questa routine rilascia il lock presente sulla variabile mutex_var.

4.6 Sincronizzazione con variabili condizionali (condition variables)

Anche in questo caso non procederemo ad una standardizzazione delle chiamate a funzione per la sincronizzazione.

Le routine qui riportate operano su variabili condizionali. Le due operazioni principali che si possono effettuare sono “wait” e “signal”. Quando un thread esegue una chiamata wait su una variabile condizionale, l'esecuzione è sospesa fino a quando un altro thread esegua una chiamata signal (oppure broadcast) sulla variabile.

cond_init (condition_var)

Inizializza la variabile condition_var a 0.

cond_signal (condition_var, mutex_var)

E' utilizzata per sbloccare un thread in attesa sulla variabile condition_var.

cond_wait (condition_var)

Questa routine rilascia il lock sulla variabile mutex e mette in attesa il thread sulla variabile condition_var. Tale thread può essere sbloccato da un'altra chiamata cond_broadcast o cond_signal.

cond_broadcast (condition_var)

E' utilizzata per sbloccare tutti i thread in attesa sulla variabile condizionale condition_var. Per sbloccare un singolo thread si utilizza la routine cond_signal.

4.7 Sincronizzazione con interrupt

Le funzioni utilizzate per la sincronizzazione tramite interrupt dei thread SPE con il PPE, definite nei file `sync_API_PPE.h` e `sync_API_SPE.h` (da includere rispettivamente nel codice PPE e in quello SPE), sono le seguenti:

```
void sync_PPE_intr (unsigned int num_SPE, spe_event_handler_ptr_t *event_handler,  
                    spe_event_unit_t *event, ppu_thread_data_t *thread)
```

Il parametro `num_SPE` indica il numero di SPE da sincronizzare con il PPE. Supponendo di ricevere un singolo evento interrupt da ogni SPE, avremo un array di eventi da gestire nel PPE. Il secondo parametro `event_handler` rappresenta un puntatore al primo elemento dell'array dei gestori di eventi di tipo `spe_event_handler_ptr_t`, il terzo parametro rappresenta un puntatore al primo elemento dell'array di eventi da gestire e l'ultimo parametro è un puntatore al primo thread SPE creato dal PPE.

```
void intr_PPE (unsigned int SPU_id)
```

Genera l'interrupt verso il PPE associato al thread SPE indicato da `SPU_id`.

4.8 Sincronizzazione SPE – SPE

La sincronizzazione tra due o più SPE viene effettuata tramite il metodo di comunicazione del signaling. Le due funzioni utilizzate sono definite nel file `sync_API_SPE.h` e sono le seguenti:

```
int read_signal_reg ()
```

Legge il Signal Notification Channel e automaticamente resetta a zero tutti i bit. Se non ci sono segnali pendenti la chiamata a tale funzione mette in stallo l'SPU fino a quando non venga inserito un nuovo segnale. Restituisce il valore del segnale letto.

```
void send_signal (unsigned int SPU_id)
```

Invia un segnale all'SPU indicato dal parametro `SPU_id`.

5 Analisi prestazioni di trasferimento dati, sincronizzazione e operazioni di calcolo

Riportiamo qui i risultati dei test effettuati per valutare le prestazioni nelle operazioni di trasferimento dati tramite DMA, di sincronizzazione e di calcolo.

5.1 Prestazioni del trasferimento dati tramite DMA

In questo paragrafo riportiamo i risultati dei test effettuati per misurare la latenza e la banda dei trasferimenti DMA da e verso un SPE.

Entrambe sono funzioni dipendenti dalla dimensione in byte del trasferimento.

Tutti i benchmark DMA, come anche i successivi test di misura dei tempi di sincronizzazione, sono stati eseguiti nell'ambiente di simulazione fornito con l' IBM SDK 2.1

In Tabella 1 riportiamo i valori di latenza per operazioni di lettura e scrittura da e nel main-storage (rispettivamente istruzioni di get e put).

DMA Size	Latenza DMA get (ns)	Latenza DMA put (ns)
8	130	60
16	130	60
32	130	60
64	130	60
128	130	60
256	140	60
512	150	70
1024	170	90
2048	210	130
4096	330	210
8192	570	370
16384	1060	690

Tabella 1 – Latenza trasferimenti DMA RAM - LS

In Tabella 2 riportiamo i valori della bandwidth ottenibile in funzione della dimensione del trasferimento. In Figura 8 e 9 riportiamo i grafici della latenza e della bandwidth in funzione della dimensione del trasferimento (in scala logaritmica).

DMA Size	DMA get Bandwidth (GB/s)	DMA put Bandwidth (GB/s)
8	0,06	0,14
16	0,12	0,28
32	0,25	0,58
64	0,49	1,13
128	0,98	2,30
256	1,89	4,16
512	3,52	7,27
1024	6,18	11,18
2048	9,96	15,71
4096	12,49	19,36
8192	14,29	22,12
16384	15,41	23,69

Tabella 2 – Bandwidth trasferimento DMA

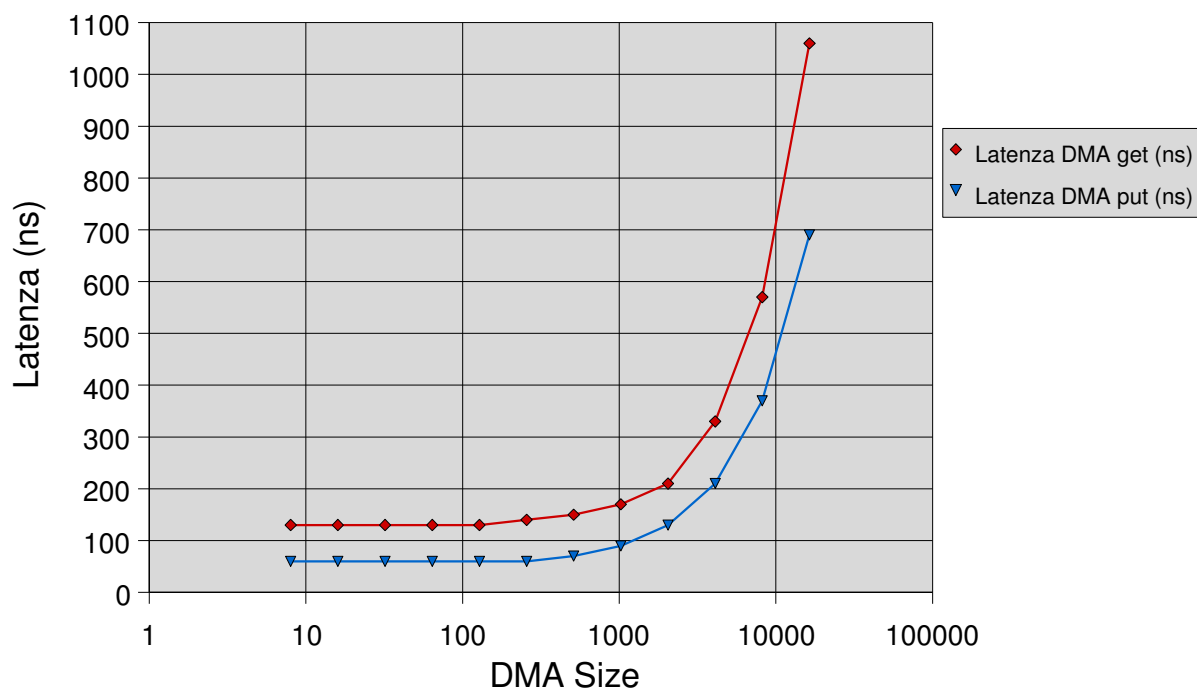


Figura 8 – Diagramma latenza DMA

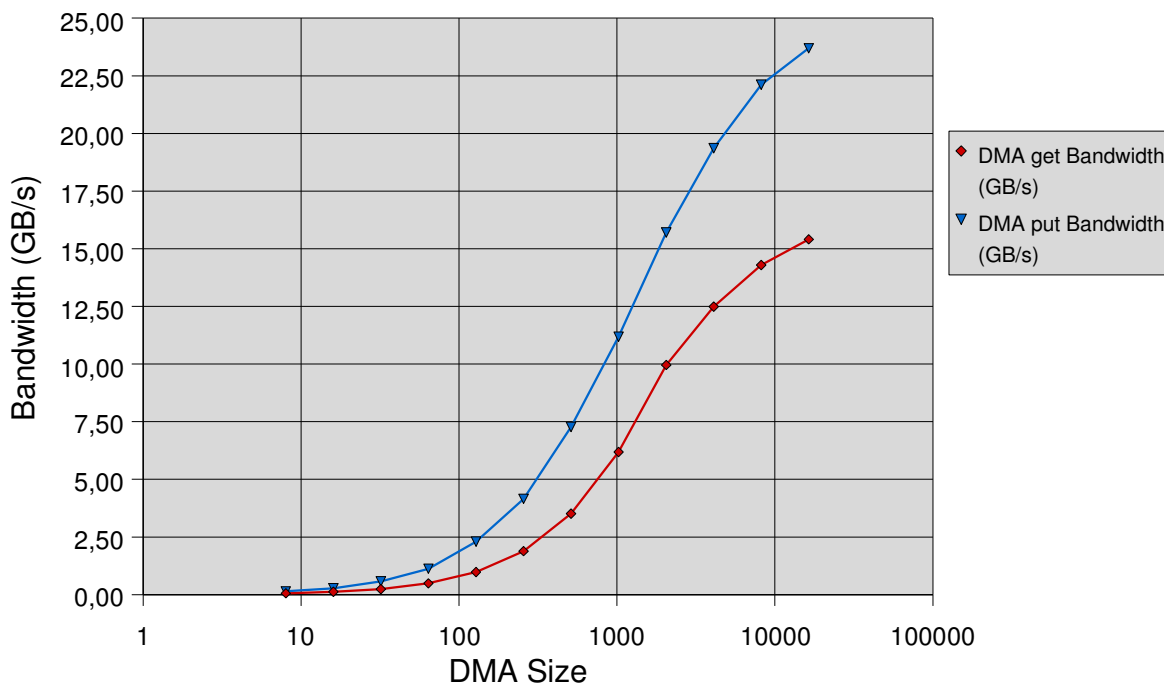


Figura 9 – Diagramma bandwidth DMA

Di seguito, in Tabella 3 e 4 riportiamo poi i valori della latenza e della bandwidth misurati per trasferimenti eseguiti tra i Local Store (LS) di due SPE differenti.

DMA Size	Latenza DMA get LS	Latenza DMA put LS
8	60	50
16	60	50
32	60	50
64	60	50
128	60	50
256	60	60
512	70	70
1024	90	90
2048	130	130
4096	210	210
8192	370	370
16384	690	690

Tabella 3 – Latenza trasferimenti DMA tra LS

DMA Size	DMA get Bandwidth LS (GB/s)	DMA put Bandwidth LS (GB/s)
8	0,13	0,16
16	0,26	0,32
32	0,53	0,64
64	1,06	1,28
128	2,13	2,56
256	4,26	4,26
512	7,31	7,31
1024	11,37	11,37
2048	15,75	15,75
4096	19,50	19,50
8192	22,14	22,14
16384	23,74	23,74

Tabella 4 – Bandwidth trasferimenti DMA tra LS

In Figura 10 e 11 riportiamo i grafici della latenza e della bandwidth DMA tra LS in funzione della dimensione del trasferimento (in scala logaritmica).

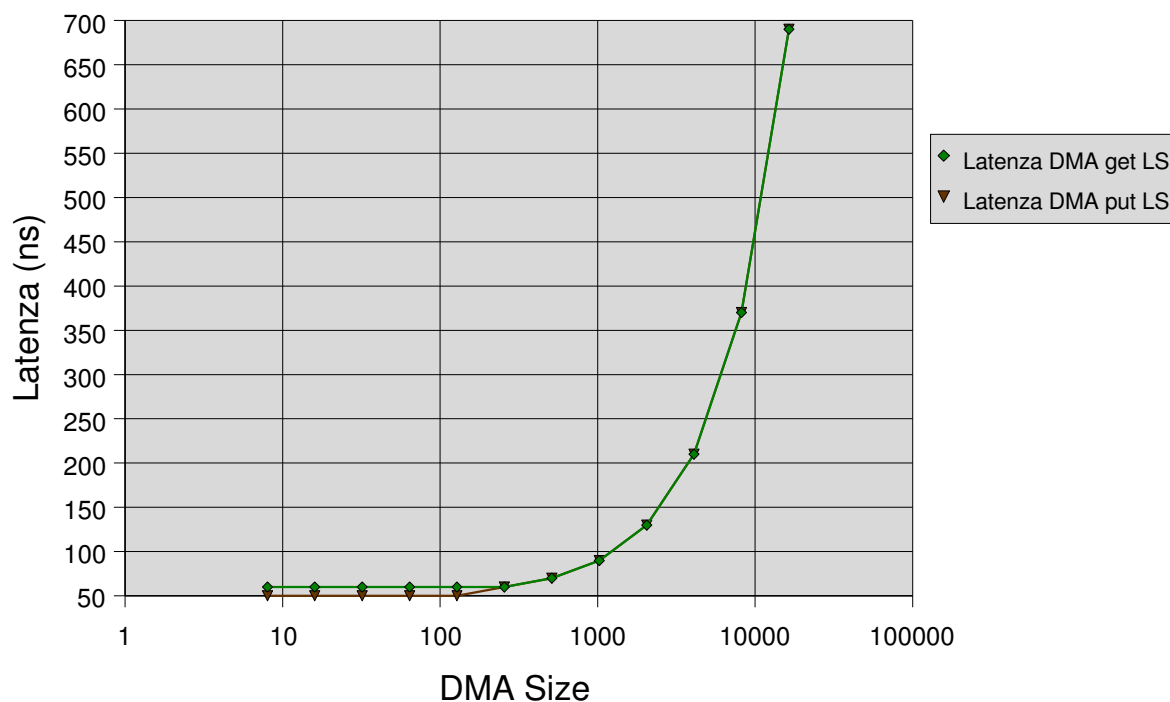


Figura 10 – Diagramma latenza DMA LS

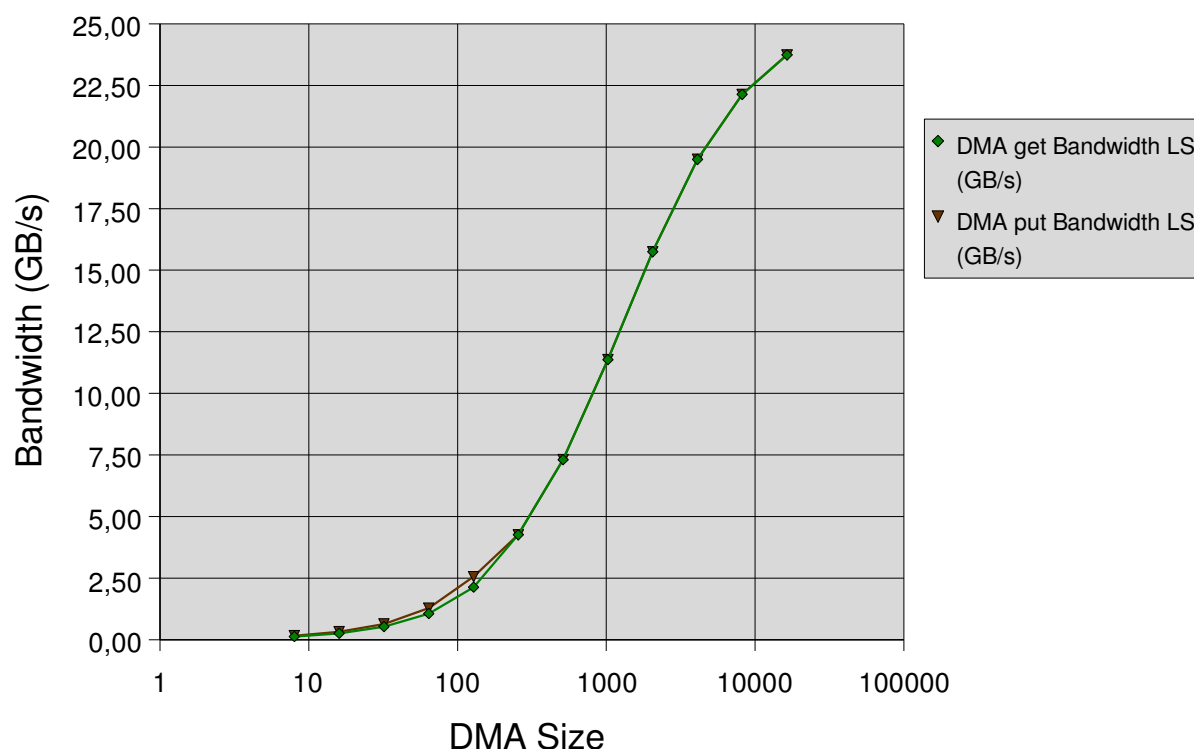


Figura 11 – Diagramma bandwidth DMA LS

5.2 Costo delle operazioni di sincronizzazione

5.2.1 Sincronizzazione con mailbox

In Tabella 5 riportiamo i tempi misurati per sincronizzare i thread SPE con il PPE tramite la comunicazione via mailbox. Separatamente riportiamo il tempo minimo e il tempo medio di sincronizzazione tra SPE e PPE e tra PPE e SPE in funzione del numero dei thread SPE in esecuzione.

	SPE -> PPE		PPE -> SPE	
	T_{min} (μs)	$\bar{T}$ (μs)	T_{min} (μs)	$\bar{T}$ (μs)
1 thread SPE	27	28	12	13
2 thread SPE	41	51	18	19
4 thread SPE	137	145	20	25
8 thread SPE	438	490	19	20

Tabella 5 – Tempi di sincronizzazione con mailbox

5.2.2 Sincronizzazione con variabili mutex

In questo paragrafo sono riportati i tempi misurati nel rilascio (unlock) e nell'acquisizione (lock) della variabile di mutua esclusione da parte del PPE che dei singoli SPE.

In Tabella 6 sono presenti i tempi che intercorrono tra l'operazione di unlock da parte del PPE e il corrispondente lock da parte di un SPE e i tempi richiesti per le stesse operazioni eseguite però tra SPE.

	PPE unlock -> SPE lock		SPE i unlock -> SPE j lock	
	T_{min} (μs)	$\bar{T}$ (μs)	T_{min} (μs)	$\bar{T}$ (μs)
2 thread SPE	186	187	95	95
4 thread SPE	189	190	100	112
8 thread SPE	191	206	70	78

Tabella 6 – Tempi di sincronizzazione con variabili mutex

5.2.3 Sincronizzazione con interrupt

In Tabella 7 sono riportati i valori misurati per la sincronizzazione effettuata tramite interrupt. La dicitura SPE intr -> PPE sta ad indicare il tempo intercorso tra l'evento interrupt generato dal SPE e la gestione effettuata dal PPE.

	SPE intr -> PPE	
	T_{min} (μs)	$\bar{T}$ (μs)
2 thread SPE	39	58
4 thread SPE	49	120
8 thread SPE	70	185

Tabella 7 – Tempi di sincronizzazione con interrupt

5.2.4 Sincronizzazione tra SPE e SPE

In Tabella 8 sono riportati i valori misurati nella sincronizzazione tra singoli thread SPE senza alcun intervento del PPE.

	SPE -> SPE	
	$T_{\min}$ (μs)	$\bar{T}$ (μs)
2 thread SPE	39	40
4 thread SPE	80	89
8 thread SPE	81	96

5.3 Costo delle operazioni di calcolo

5.3.1 Instruction set PPE (PowerPC)

Tutte le istruzioni PowerPC hanno una lunghezza di 4 bytes e un allineamento a 4 byte. La maggior parte di queste possono avere fino a tre operandi e la maggior parte delle istruzioni di calcolo lavorano con due operandi sorgenti e un operando di destinazione. Gli interi con segno sono rappresentati in complemento a due.

L'Instruction set del PPE comprende i seguenti tipi di istruzioni:

- Istruzioni di load/store fixed-point e floating-point
- Istruzioni fixed-point aritmetiche, logiche, shift e rotazione, confronto.
- Istruzioni floating-point aritmetiche, prodotto-addizione, confronto.
- Istruzione di memory-synchronization
- Istruzione di controllo di flusso

In Tabella 9 riportiamo un sommario delle istruzioni PPE assieme alla latenza, al throughput (IPC – Instruction per cycle) e all'unità di esecuzione cui fanno riferimento.

Descrizione	Numero di istruzioni	Latenza (cicli)	Throughput (IPC) per ciclo di latenza	Unità di esecuzione
Load	18	2	1	LSU
Load floating-point	8	1-2	1	LSU,FXU,FPU
Store	17	N/A	1	LSU,FXU
Store floating-point	5	N/A	1	LSU,FPU
Arithmetic fixed-point	59	1-2	1	FXU
Logical	12	2	1	FXU
Shift	14	11	1	FXU
Shift Immediate	8	2	1	FXU
Rotate	13	11	1	FXU
Rotate Immediate	8	2	1	FXU
Compare	5	1	1	FXU
Divide	8	10-70	1 per 10-70 cicli	FXU
Extend Sign	3	2	1	FXU
Multiply immediate	9	6-15	1 per 6-15 cicli	FXU
Arithmetic floating-point	20	10-11	1	FPU
Floating-point compare	2	1	1	FPU
Floating-point divide (IEEE)	2	74	1 per 74 cicli	FPU
Floating-point divide	2	56	1 per 56 cicli	FPU
Floating-point multiply	16	10-11	1	FPU
Floating-point square root	2	66	1 per 66 cicli	FPU
Floating-point square root (IEEE)	2	84	1 per 84 cicli	FPU
Branch	12	1-2	1	BRU
Synchronize	3	N/A	N/A	LSU

Tabella 9 – Sommario istruzioni principali PPE

I campi etichettati N/A indicano che l'istruzione non modifica alcun registro nella sua esecuzione oppure è implementata in “microcodice”. Con questo termine intendiamo quel tipo di istruzioni più complesse della singola istruzione PowerPC che impiegano elevate risorse di sistema e per questo eseguite in microcodice. Ciò significa che, per l'esecuzione, vengono suddivise in istruzioni più semplici dette “microwords”. E' importante sottolineare che sono richiesti sempre 11 cicli di clock per la lettura della prima istruzione microcode dalla memoria ROM che la contiene.

5.3.1.1 Istruzioni Vector/SIMD Multimedia

Le istruzioni Vector/SIMD Multimedia possono essere utilizzate liberamente assieme alle altre istruzioni PowerPC all'interno di un programma. L'unità VXU a 128 bit (Vector/SIMD Multimedia Extension Unit) lavora concorrentemente con l'unità a 32 bit FXU (Fixed-point Unit) e l'unità a 64 bit FPU (Floating-point Unit). Come per le altre istruzioni PowerPC, anche le istruzioni Vector/SIMD hanno lunghezza 4 byte e allineamento a 4 byte. La maggior parte delle istruzioni Vector/SIMD opera su “vettori” a 128 bit, cioè su operandi costituiti da singoli pacchetti di dati i cui elementi possono essere sia valori fixed-point che floating-point. Tali vettori sono detti operandi SIMD oppure operandi “pacchettizzati” e possono contenere i seguenti tipi di dati:

- 16 valori a 8 bit, signed o unsigned
- 8 valori a 16 bit, signed o unsigned
- 4 valori a 32 bit, signed o unsigned
- 4 valori floating-point a singola precisione (IEEE-754)

Le istruzioni Vector/SIMD Multimedia includono i seguenti tipi:

- Vector Fixed-point – istruzioni aritmetiche, di confronto, logiche, rotazione e shift su vettori.
- Vector Floating-point – istruzioni multiply-add, di approssimazione, di confronto e conversione su vettori.
- Load and Store – istruzioni di load e store su vettori a 128 bit.
- Vector Permutation and Formatting – istruzioni di pack, unpack, merge, splat, permute, select e shift su vettori.

In Tabella 10 riportiamo un sommario delle istruzioni Vector/SIMD Multimedia assieme alla latenza, al throughput (IPC – Instruction per cycle) e all'unità di esecuzione cui fanno riferimento.

Descrizione	Numero di istruzioni	Latenza (cicli)	Throughput (IPC) per ciclo di latenza	Unità di esecuzione
Load Vector	10	2	1	VXU,LSU
Store Vector	9	N/A	1	VXU,LSU
Vector Arithmetic	27	4	1	VXU
Vector Compare	40	4	1	VXU
Vector Logical	5	4	1	VXU
Vector Merge, Splat, Permute	13	4	1	VXU
Vector Multiply, Multiply-add	22	9	1	VXU
Vector Pack, Unpack	15	4	1	VXU
Vector Rotate, Shift	14	4	1	VXU
Vector Round, Convert	8	12	1	VXU
Vector Select	1	4	1	VXU
Vector Shift	3	4	1	VXU
Vector FP Single-precision Arithmetic	4	12	1	VXU
Vector FP Single-precision Log-2	2	12	1	VXU
Vector FP Single-precision Reciprocal	2	14	1	VXU

Tabella 10 – Sommario istruzioni principali Vector/SIMD Multimedia

5.3.2 Instruction set SPU

Ogni SPU supporta l'istruzione set descritto nel documento *Synergistic Processor Unit Instruction Set Architecture* (SPU ISA) e un set aggiuntivo di intrinsics C/C++ come descritto nel documento *SPU C/C++ Language Extensions*.

L'unità SPU lavora principalmente con operandi vettoriali SIMD, contenenti sia valori fixed-point che floating-point e supporta le operazioni più comuni su operandi scalari.

L'hardware SPU supporta i seguenti tipi di dati:

- byte – 8bit
- halfword – 16 bit
- word – 32 bit
- doubleword – 64 bit
- quadword – 128 bit

In Figura 12 è rappresentato il layout di un registro SPU in base alla lunghezza del tipo di dato:

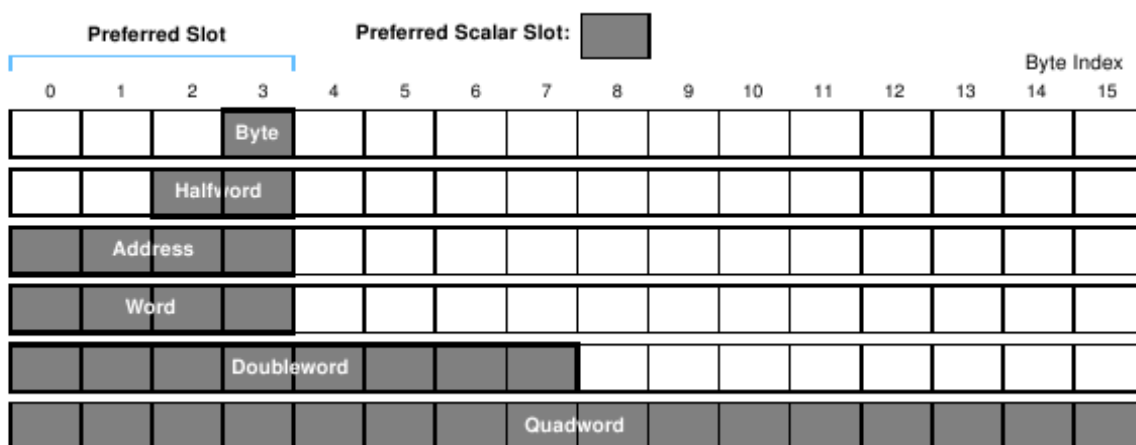


Figura 12 – Layout di un registro SPU

La parola di codice più a sinistra del registro (byte 0,1,2,3) è detta “*preferred slot*” nel *Synergistic Processor Unit Instruction Set Architecture* e “*preferred scalar slot*” nel “*SPU Application Binary Interface*”. Infatti quando un'istruzione utilizza o produce un operando scalare o un indirizzo, il suo valore è memorizzato in tale posizione.

Ogni SPU è dotato di due pipeline di esecuzione dette “even” (pipeline 0) e “odd” (pipeline 1). In ognuna di queste può essere schedata ed eseguita un'istruzione per ciclo di clock. La scelta della pipeline per la schedulazione dipende dal tipo di istruzione.

In Tabella 11 e 12 riportiamo i tipi di istruzioni, con i rispettivi valori di cicli di latenza e di stallo, che sono schedabili nelle due diverse pipeline. Il numero di cicli di clock di stallo riportati nelle tabelle è quello che necessariamente deve intercorrere tra la schedulazione di due istruzioni dello stesso tipo.

Istruzioni Pipeline 0	Latenza (cicli)	Stallo (cicli)
Single-precision floating point	6	0
Integer multiplies, convert between FP and integer, interpolate.	7	0
Load immediate, logical op, integer add-subtract, sign extension, select bit, carry and borrow generate	2	0
Double-precision floating point	7	6
Element rotate and shift	4	0
Byte operations	4	0

Tabella 11 – Tipi istruzioni Pipeline 0 (even)

Istruzioni Pipeline 1	Latenza (cicli)	Stallo (cicli)
Shuffle byte, quadword rotate and shift	4	0
Gather, mask	4	0
Estimate	4	0
Load, store and branch hints	6	0
Branches	4	0
Channel operations	6	0

Tabella 12 – Tipi istruzioni Pipeline 1 (odd)

In Tabella 13 riportiamo invece un sommario delle istruzioni SPU assieme ai valori di latenza, alla classe di appartenenza dell'istruzione, al tipo di pipeline e di unità di esecuzione.

Descrizione	Instruction Class	Numero di istruzioni	Latenza (cicli)	Pipeline	Unità di esecuzione
Load and Store	LS	12	6	Odd	SLS
Branch Hints	HB	4	15	Odd	SLS
Branch resolution	BR	36	4	Odd	SCN
Channel Interface	CH	4	6	Odd	SSC

Descrizione	Instruction Class	Numero di istruzioni	Latenza (cicli)	Pipeline	Unità di esecuzione
Shuffle	SH	35	4	Odd	SFS
No Operation (load)	LNOP	1	0	Odd	LNOP
No Operation (execute)	NOP	1	0	Even	NOP
Single-precision floating point	SP	6	6	Even	SFP
DP floating point	DP	10	13	Even	SFP
Floating point integer	FI	17	7	Even	SFP
Byte operations	BO	3	4	Even	SFP
Simple Fixed-point	FX	69	2	Even	SFX
Word rotate and shift	WS	17	4	Even	SFX

Tabella 13 – Sommario istruzioni SPU