**Carleton University**
**Department of Systems and Computer Engineering**

**SYSC5701 Operating System Methods for Real-Time Applications**
**Class Project**

# Overview of the CELL Broadband Engine Memory Architecture

# Implications for Real Time Operating Systems

**Dated: March 07**

**Dayana Thirukumaran**
**Brian Webb**
**Rich Goyette**

## Table of Contents

# Introduction

This document is a high level study of the memory architecture found on the CELL Broadband Engine (CELL BE, CBE). The motivation behind this study was to review the latest specifications and literature concerning the CELL BE and draw some conclusions concerning its suitability for running a real-time operating system (RTOS).

This document starts with a generic overview of the major CELL BE components by way of introduction and background. A more detailed analysis of specific CELL BE memory architecture is then undertaken. This is followed by an analysis of the CELL BE virtual memory architecture which is a significant component of the overall memory architecture. Finally, this document concludes with an analysis of various memory allocation strategies.

# CELL BE Architecture Overview

The main structural components of the CELL BE are shown in Figure 1. These components include eight *Synergistic Processing Elements* (SPE0-7), one *Power Processor Element* (PPE), a *Memory Interface Controller* (MIC), a *Broadband Engine Interface* (BEI) and a high-bandwidth *Element Interconnect Bus* (IEB).

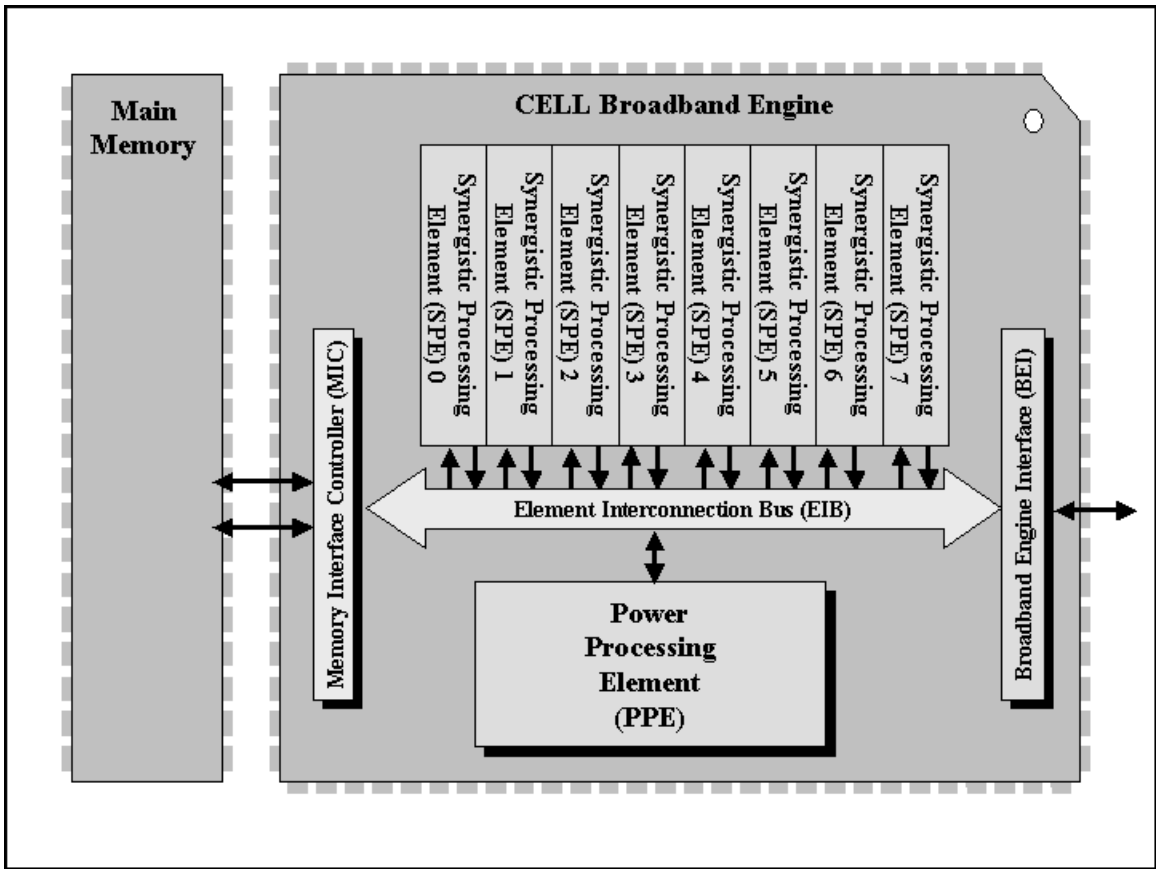


Figure 1: Cell Memory Components [BE06]

A generic description of each of these components is provided in Section 1.2 of [BE06]. In order to situate the discussion of the CELL BE memory architecture, Section 1.2 of [BE06] is summarized below.

## *Power Processing Element (PPE)*

The PPE is a 64-bit, dual-threaded PowerPC-based CPU core. It is generally regarded as the master coordinator for the activities of each SPE [KS06] and is "intended primarily for control processing, running operating systems, managing system resources, and managing SPE threads" ([BE06], p. 37). This is an implicit design assumption that should be considered with the design of any RTOS.

## *Synergistic Processing Element (SPE)*

The SPE is the core innovation of this device. There are eight identical SPEs and each is capable of an independent thread of execution. One of the key features of the SPE is that it contains a 256KB, non-cached local store and a significant register file. Combined

with the SPEs RISC-based, register-register instruction architecture, the SPE is capable of very high execution rates against data-centric applications. The SPE will be the focus of further elaboration.

## *Element Interconnect Bus (EIB)*

The structure of the Element Interconnect Bus (EIB) is shown in Figure 2. The EIB is physically in the center of the CPU die and connects all of the components as shown in the figure.

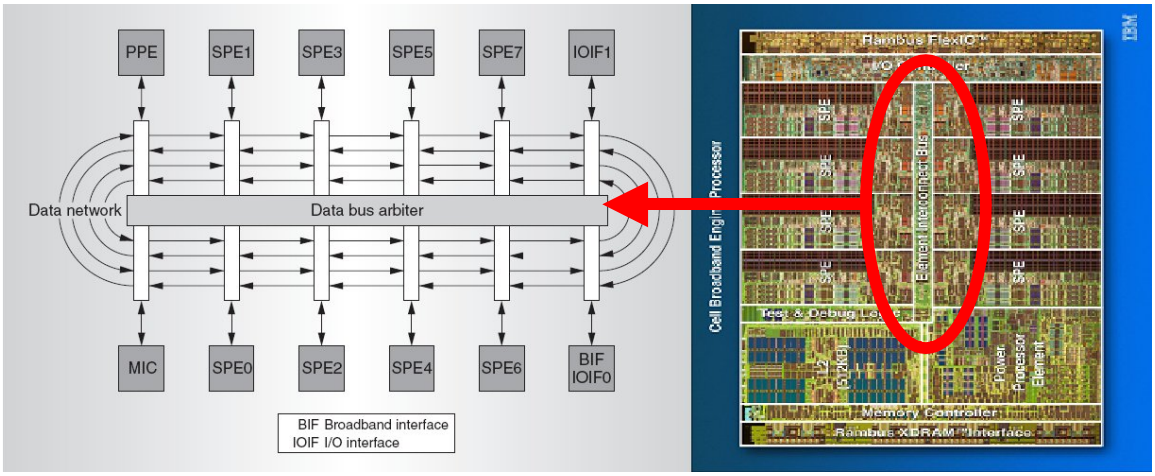


Figure2: Element Interconnect Bus (EIB) Design
(Left: Figure 2 from [KPP06] Right: Page 2 [AE05])

The EIB consists of a central data bus arbiter and four independent 16-byte wide data rings. The bus only operates at half the core system frequency so some references to the EIB only claim an 8-byte wide bus [KRE05]. Two of the rings move data clockwise and the others move data counterclockwise. This was a design decision that limits the total physical travel distance to ½ of the ring perimeter. Because the rings connect between each individual device interface (shown as the white rectangles in Figure 2), it is possible to support up to a maximum of 12 concurrent bus transactions in any given cycle for a peak bandwidth of 96 bytes/cycle (12 transactions X 8 bytes/cycle) [GER06].

Each of the SPE elements has a pair of 128 byte-wide ports connecting to the EIB – one for read access and one for write access. Devices wishing to use the EIB make their request known to the bus arbiter through their respective interface. The bus arbiter selects and assigns an appropriate ring based on round-robin assignment except in the case of reads from to the MIC which are treated as high priority. This is designed to avoid stalling during read access of the main memory [KPP06].

### *Memory Interface Controller (MIC)*

The MIC provides an interface between main memory (referred to in [BE06] as the "main storage domain") and the components connected to the EIB. The MIC can connect up to two Rambus Extreme Data Rate (XDR) interfaces for a total of up to 64GB of externally addressable memory.

### *Broadband Engine Interface (BEI)*

Finally, the BEI provides a mechanism for connecting the EIB to external IO devices or to other Cell BE. In the present analysis, multiple Cell BE will not be considered.

## CELL Memory Architecture

### *Introduction*

The objective of the CELL BE design was "to achieve 100 times the PlayStation2 performance and lead the way for the future." [KAH05] One of the significant architectural roadblocks to achieving this goal was to address the wide disparity between high processor frequencies and off-chip DRAM latencies while trying to maintain low complexity and power consumption. Attempts to hide memory latency in the past have included deeper pipelines and wider busses but these approaches have lead to diminishing returns on performance enhancement due to implementation complexity and the resulting power consumption [BE06]. The Cell BE addresses memory latency, in part, by adopting design goals that "minimize pipeline depth, increase memory bandwidth, allow more simultaneous, in-flight memory transactions, and improve power efficiency and performance." [KPP06] These design goals lead eventually to "ganging" of highly parallel but simple processors, each having their own local store and ….

### *Memory Components*

This section describes the major components of the memory architecture found within each subcomponent of the CELL BE. Figure 3 identifies memory architecture components (light blue).

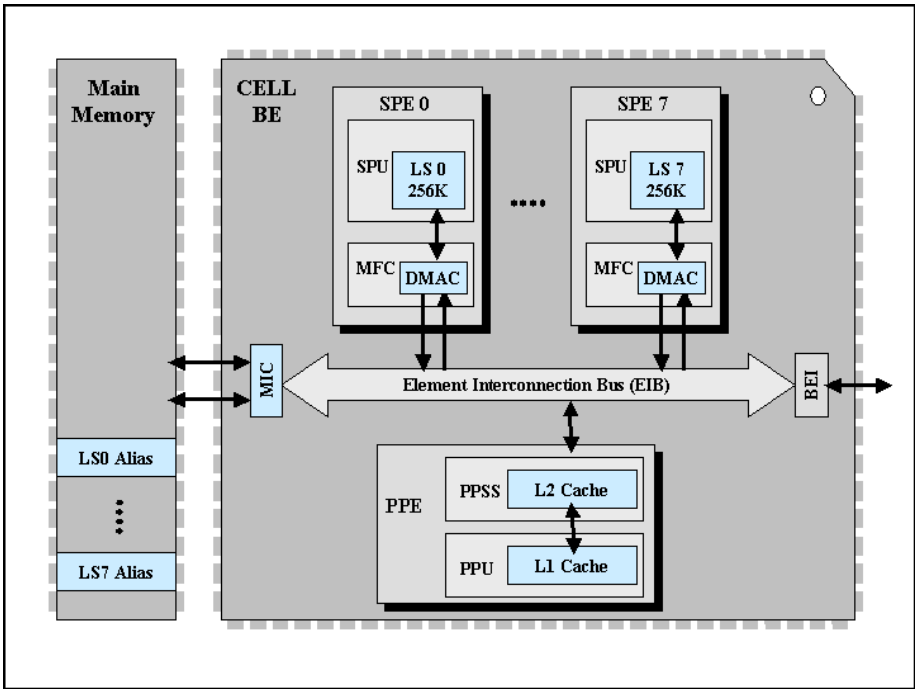


Figure 3: CELL BE Memory Components.

## PPE Memory

The internal memory structure of the PPE is typical of a uniprocessor design. It consists of a two-level cache memory hierarchy that addresses memory latency issues in the traditional fashion.

## SPE Memory

Each SPE consists of a Synergistic Processing Unit (SPU) and a dedicated Memory Flow Controller (MFC). As will be discussed shortly, the architecture of the SPE only allows it to load and store instructions and operands from its local store. Therefore, support is required to fill the local store. This is provided by the Memory Flow Controller (MFC) on each SPE. The MFC is a dedicated DMA controller that can queue up to 16 simultaneous DMA transfers from main storage to the local store concurrently with SPE program execution.

Figure 4 shows a more detailed block diagram of the SPE. The SPU is composed of a Symmetric Execution Unit (SXU), a 128-entry fast register file and an instruction pre-fetch buffer. The SXU is based on a Reduced Instruction Set Computer (RISC) concept with fixed 32-bit instructions [tut]. The SPE is classified as a register-register (or load-store) architecture – all operations take place between 128 bit operands that are located in the register file and results are stored back to the register file. The local store is accessed only to load operands into the register file or to store results from the register file.

Therefore, the only instructions that can modify the local store (besides DMA transfers) are load and store instructions originated by the SPE.

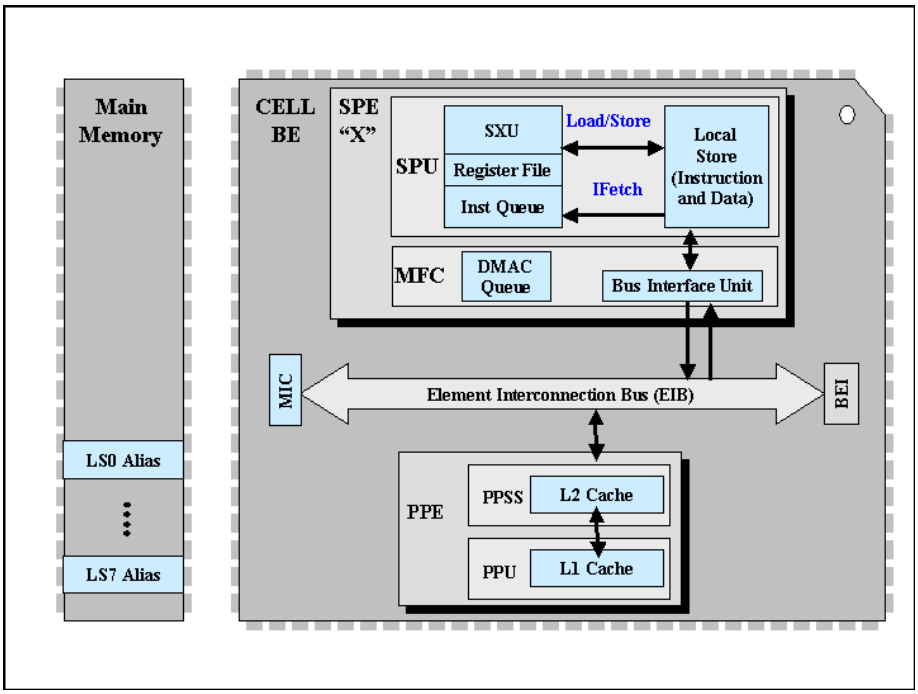


Figure 4: Detailed SPE Architecture

Execution efficiency in the SXU is supported by the design of the local store. Because the LS is *non-cached*, computations of task execution time is deterministic since there are no cache misses to deal with. However, because there are no cache misses, it is essential that application code ensure that all of the required data has been provided or that double-buffering is used to hide main memory latency.

The LS is both single-ported and asymmetric [BE06] in the sense that two different data sizes are supported depending on the type of access. Load and store operations from the SXU are 16 bytes (128 bits – or one register) while instruction fetches and DMA operations make 128 byte accesses. In the case of instruction fetches, this is equivalent to extracting 32 instructions on each local store access. The large register file and relatively high-bandwidth instruction fetch "facilitates highly efficient instruction scheduling and enables important optimization techniques such as loop unrolling" [KPP06].

## *PPE Local Store Access Modes and Aliasing*

As noted previously, the SXU can only access its own LS with load and store operations and to fetch instructions. However, depending on program requirements, other units on the Cell may require access to a particular SPEs local store (e.g, other SPEs, the PPE, or I/O devices). In these circumstances, the PPE can direct an SPE's local store to be aliased into main memory as shown in Figure 4. Once aliased, the PPE can access that

SPEs LS directly using load and store instructions. However, other SPEs must continue to use DMA through their respective MFC to access an aliased SPE's LS because it appears as just another bank of memory.

When a local store has been aliased to main memory, each local store address (LSA) is mapped to a real address. As mentioned earlier, when an SPE makes an access to an SPE's aliased memory space, it does so using DMA transfers so that the exchange is efficient. However, when the PPE or an I/O device makes an access using a load or store instruction, the memory management system fetches the requested item from directly from the associated LS and this is not efficient.

Because the LS is single-ported, there will be access contention between load/store, instruction fetch, and DMA operations. The CELL BE gives the highest priority to Memory Mapped IO (MMIO) register operations. MMIO registers are those within the MFC that can be addressed by the SPU or SPE to initiate DMA operations. DMA read and writes are assigned the next highest priority followed by load/store operations. Finally, instruction fetches are assigned the lowest priority. The reasoning behind this scheme is that it is important to ensure that instructions being executed by the SPE do not "starve" from lack of data nor should they be stalled while waiting to put data away in the LS.

## Main Memory

Main memory is composed of normal off-chip DRAM that is interfaced to the CELL BE via two RamBus Extreme Data Rate (XDR) interfaces within the Memory Interface Controller (MIC).

Main memory is a resource that can be *controlled* via a Resource Allocation Manager (RAM) as shown in Figure 5.

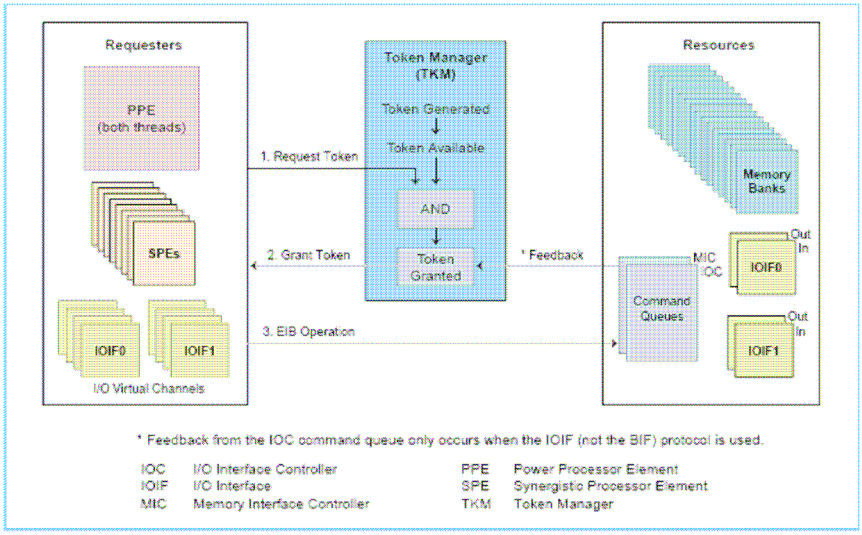


Figure 5: Resource Allocation management (Figure 8-1 from [BE06]

The effect of resource allocation via the token granting mechanism shown is that both resource and bandwidth can be reserved for certain high-priority operations. For example, requestors can request a token from the TKM. If the tokens are available, the TKM will provide them and then (and only then) will the requestor be able to access the EIB. An exception to this rule occurs with feedback from the MIC and IOC; if the granting of a token is blocked by these devices, the TKM will not provide tokens to the requestors thereby guaranteeing both the resource (memory and IO) and the bandwidth (EIB and MIC/IOIF interfaces) are reserved. Obviously, this is advantageous for real-time applications that have high priority, critical deadlines that must be met.

## *Memory Coherence Across CELL BE*

Various data caches are held within the CELL BE. As discussed earlier, the PPE has a two-level data cache and the Memory Management Unit of each SPE contains a 256 entry Translation Lookaside Buffer (TLB) and an Atomic Unit Cache of six 128-byte lines ([BE06], pp. 151). The coherence of these caches is implemented by a hardware-based coherency protocol on the EIB. The caches (assisted by the EIB) perform snoop operations to determine if their cached contents have been invalidated.

All DMA transfers are coherent *with respect to main storage only*. However, the LS of each SPE is **not** coherent with respect to main memory [HOF05] and this is one of the features that makes the CELL BE space and power efficient. When an LS has been aliased, accesses made to the LS by the associates SPE are therefore **not** coherent (they will not be snooped and cannot cause cached values elsewhere to become invalid). Therefore, when performing accesses to aliased memory, the cache-inhibit attribute should be set ([BE06], p. 128).

## *Memory Architecture Operating System Implications*

### Context Switching

Context switching of SPEs is inefficient because of the memory architecture of the CELL BE. While it is possible, [BE06] makes it clear that it is potentially undesirable. In the worst-case context switch (one in which all of the SPE state must be preserved), the entire 256KB LS must be switched out. In addition, since LSes can be aliased for use with I/O devices (a very real possibility for an embedded RTOS), "preemptive context switching of an SPE that interfaces directly with an I/O device should be avoided, unless the I/O device can first be reliably quiesced" ([BE06], p. 351). The act of "quiescing" an I/O device that operates asynchronously may at the very least be time consuming and at worst be impossible in the context switching times required for efficient operation.

## Code Size Limitations

The processes or threads written to operate on the SPE must be fairly small to be efficient. Although it is possible to write code that is larger than the 256KB size of the LS using overlays ([TUT06], p. 139), this leads to significant overhead. For example, in [WIL05], the data space within the LS was limited to about 56KB after the code was generated and stack allocated.

## Data "Starvation"

In [AE05], the fact that SPE instructions were fetched and queued 32 at a time was identified as a potential pitfall if care was not taken by the code or by a scheduler to ensure that data was available at an appropriate rate. Since this source was a presentation and did not quote references, it was difficult to determine how a scheduler would factor into the feeding decision.

## Resource Allocation

As indicated earlier, the Resource Allocation Management facility that acts on the EIB can be a useful tool for guaranteeing bandwidth as well as certain resources for time-critical processes.

## Deterministic Performance

One of the primary advantages of the SPE from an RTOS perspective is the fact that performance can be measured deterministically since caching is not performed on the LS. To wit:

> The SPEs provide a deterministic operating environment. They do not have caches, so cache misses are not a factor in their performance. Pipeline-scheduling rules are simple, so it is easy to statically determine the performance of code and for software to generate high-quality, static schedules. ([BE06], p. 63)

## Event Responsiveness

In [KAH05], responsiveness is sited as an attribute of each of the SPEs. This is provided by the ability of each SPE to "individually and autonomously schedule and receive DMAs as well as interrupts" (p. 601). While this is a positive attribute for an RTOS, it is not clear how it combines with the poor performance perceived for context switching of SPEs.

# CELL BE Virtual Memory Architecture

Virtual memory is an addressing method that allows non-contiguous memory to be addressed as if it is contiguous. Also, it gives an illusion of more physical memory than what is physically available in the RAM by using the disk storage to store information that normally have to be in the main storage for an application to run. This allows more programs to run in parallel.

CBE processor supports virtual addressing scheme similar to PowerPC processors. It allows the operating system to enable/disable virtual-address translation independently for instruction and data. CBE processor hardware and the operating system software together can support a virtual address space larger than the effective or real address space. CBE processor's virtual address (VA) space is 2^65 bytes. Main storage is addressed by effective addresses (EA) thus programs accessing memory have to use the effective addresses. Effective address space in CBE is 2^64 bytes. Real address (RA) space is a sub set of the effective address space that is physically present the in CBE. Real address space includes main memory and devices that are mapped to real address space such as SPE's on-chip LS or an I/O device's off-chip MMIO register or queue. Real address space in CBE is 2^42 bytes.

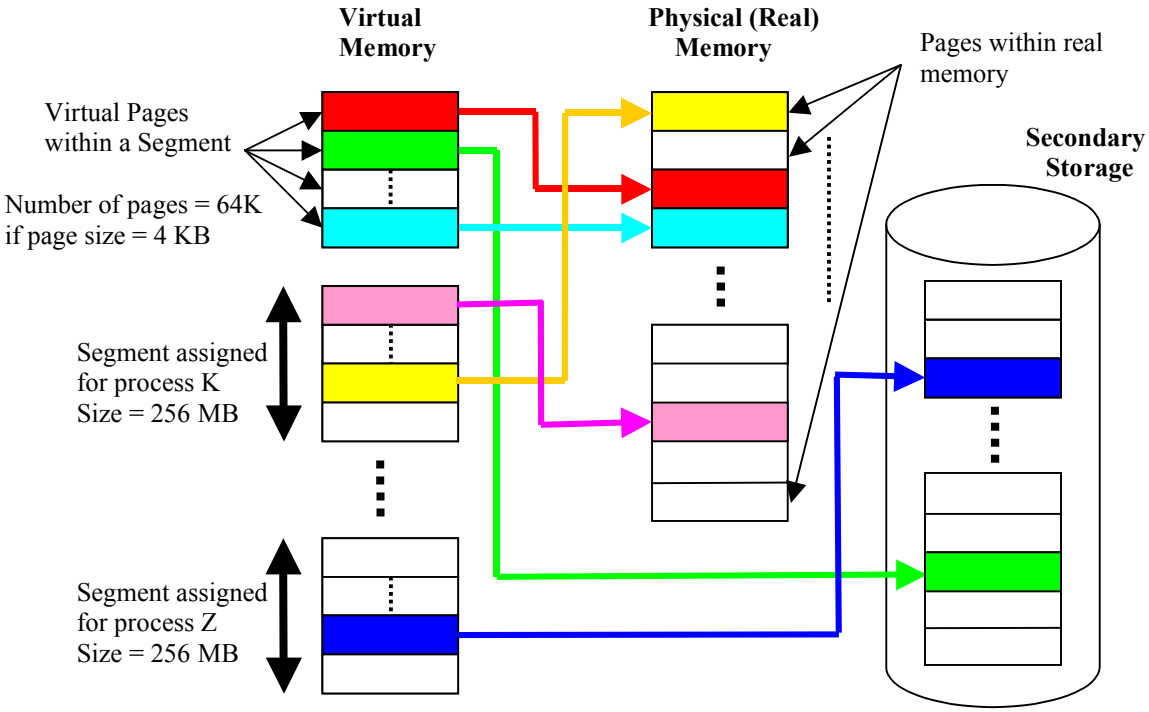


Figure 6: CBE Virtual Memory Space

## Segments

The virtual memory is divided up into 2^ 37 protected, non-overlapping segments, each containing 256MB contiguous addresses. Each process can have one or more segments for its instruction, private data and stack. If a process needs a larger address space, then the operating system can concatenating several contiguous segments to create one big address space.

## Pages

Physical (real) memory is divided up into mostly 4KB pages. CBE allows two additional large pages with the size of 16KB or 1MB or 16MB. Pages are protected, non-overlapping, relocatable contiguous real addresses. Pages can be transformed into virtual pages by the virtual addressing scheme. However this causes a problem when all currently executing programs' virtual pages exceed the available physical memory in the system. So whenever a program tries to access a page containing instruction or data that are not currently in the physical memory would cause a 'page fault'. Operating system has a mechanism to overcome this issue called 'demand paging'. Operating system can bring in page from the secondary storage to the physical memory as demanded by the program. However, before it can bring in a new page, it has to make some room in the physical memory. Pages are replaced using LRU (Least Recently Used) policy, which basically uses the recent past as an approximation of the near future. So a least recently used page will be transferred to the secondary storage before the new requested page is brought in from the secondary storage to the physical memory.

## Page Table

The processor has to have a way of translating from virtual addresses to real addresses before the memory can be accessed. CBE hardware maintains a page table for the whole system or one page table for each logical partition (in hypervisor mode). The operating system uses the hashing algorithm to come up with the table entries. Each page table entry is a 128bit data structure that maps a virtual page number to its currently assigned real page number. This way the page table provides a virtual to real address translation. CBE performs an inverted page table lookup by mapping the addresses from real addresses to virtual addresses. The inverted page table reduces the memory required for page table itself, since page table is proportional to the real address space instead of the virtual address space.

## Address Translation in PPE

CBE contains a 64 bit, dual-thread PowerPC Architecture with RISC core, thus it supports 64 bit memory management model. Hence, the memory is addressed by 64 bit

effective addresses. However, the available physical memory is much less than the effective address space. Therefore effective addresses have to be translated into real addresses before the memory can be accessed. Address translation is performed by means of memory management unit in PPE. It has four hardware components to handle this translation. If virtual addressing mode is turned off, the ERAT hardware unit is used to translate the effective addresses into real addresses. However, if the virtual addressing mode is enabled, the effective addresses have to go through few more hardware units before the address can be translated into real addresses (unless a hit at ERAT HW unit). SLB HW unit translates the effective addresses into virtual addresses. Then the virtual address is translated into real address by the TLB unit. If incase of a TLB miss, the virtual address has to be translated by the page table in the memory.

PPE memory management unit hardware description as follows:

- Effective to Real Address Translation Buffer (ERAT)
  - It has one 64 entry cache for instruction, shared by both threads (2 way x 32)
  - It has one 64 entry cache for data ,shared by both threads (2 way x 32)
  - Each thread maintains its own ERAT entries and cannot access the other threads entries.
  - ERAT entries are identified by the thread ID
  - ERAT contains the recently used ER to RA translation information
  - Each entry holds ER to RA address translation for an aligned 4 KB area of memory.
  - If larger pages are being used, then it can occupy several ERAT entries.
  - This unit is accessed even if the virtual addressing mode is disabled.
  - ERAT miss will result in 11 cycle penalty in order to translate the address in MMU.

- Segment Lookaside Buffer (SLB)
  - Two unified (data & instruction), 62 entry cache, one per PPE thread.
  - It provides EA to VA address translation.
  - PPE supports up to 2^37 segments, 256 MB each.
  - SLB miss creates an instruction segment or data segment exception.

- Translation Lookaside Buffer (TLB)
  - Unified 1024 entry cache, shared by both threads.
  - It stores recently accessed page table entries
  - It provides VA to RA address translation.
  - Pseudo-LRU replacement policy is used.
  - The TLB updates can be managed either by software or hardware.

- Page Table
  - Hardware accessed data structure in main memory
  - Table entries are maintained by the operating system.
  - It provides VA to RA address translation.

- Each entry is 128 bit data structure.

The following flow chart describes the address translation mechanism used by PPE.

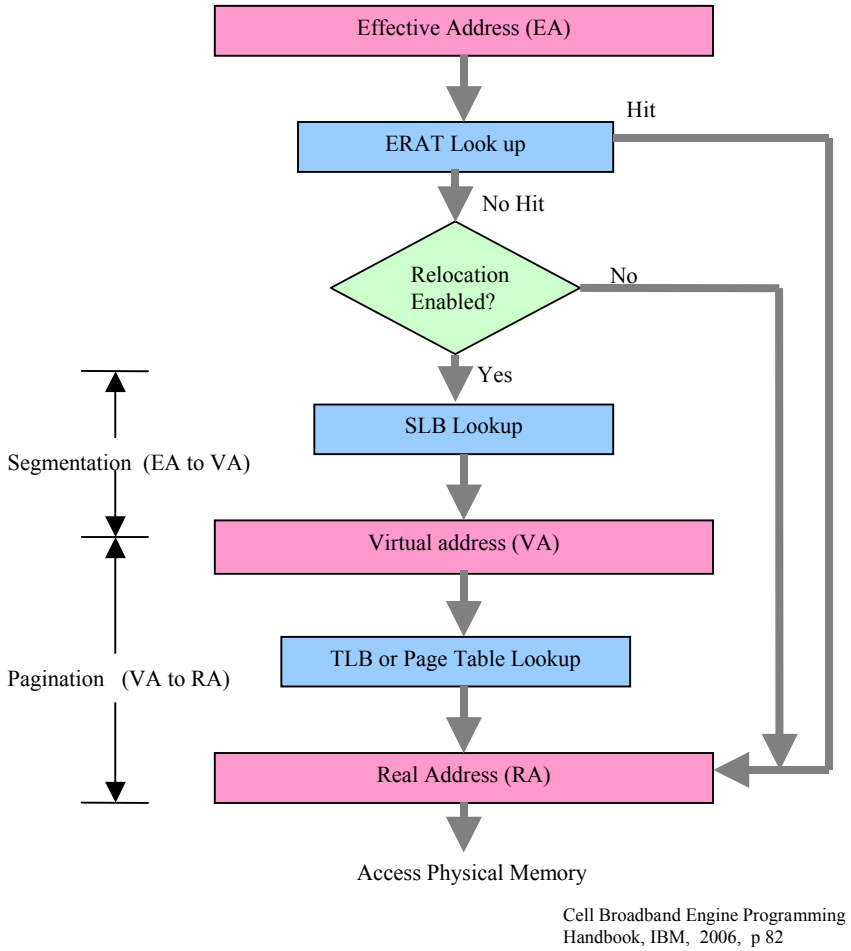


Figure 7: PPE Address-Translation Overview

## *Address Translation in SPE*

SPE access the main memory by DMA commands with proper effective addresses (EA) and local addresses (LS). When virtual addressing is enabled, SPE uses the Synergistic Memory Management (SMM) unit to translate addresses from EA to RA to access the main memory. SMM has two main hardware units (SLB and TLB) for address translation from EA to VA and then to RA.

- Segment Lookaside Buffer (SLB)
  - A unified (instruction and data) 8 entry cache
  - It provides EA to VA translation
  - SLB is mapped to the main memory as MMIO registers.

- It supports 2^37 segments, each 256 MB
- PPE provides the SLB entries to SPE

- Translation Lookaside Buffer (TLB)
  - A unified (instruction and data) 256 entry cache
  - Stores recently accessed page table entries for VA to RA address translation.
  - PPE supplies the TLB entries to SPE.

- Page Table
  - SMM uses the page table entries supplied by the PPE
  - Page size - 4KB plus two large pages (64 KB or 1MB or 16MB)

## *Virtual Memory Operating System Implications*

### Memory protection

Virtual addressing shame has its advantages and disadvantages for CBE. Memory protection and prevention of memory fragmentation are the main benefits apart from providing a huge address space for applications. CBE provides a segmentation mechanism when virtual addressing is enabled. Each process is assigned with one or more segments. So each process can only access virtual addresses assigned to its own segment. This prevents processes from accessing any physical memory that is not mapped to its virtual address range. Memory accessing problems such as over stepping array boundaries, writing into buffers just after freeing them, double freeing, un-initialized or stale pointers and reading beyond boundaries can be reduce/avoided with virtual addressing scheme. Some of these issues may still cause problems for an application's own process space, but with segmentation in place, it cannot damage memory owned by another process. The main memory in CBE is not only accessed by PPE but also by eight other SPE processors. It means chances of memory access issues are much higher with out virtual memory.

### Addressing space

PPE has to manage processes running on SPE, in addition to its own processes. SPEs are optimized to perform compute intensive applications such as steaming data (3D graphics, media, and broadband communications). Thus data and instruction has to be anticipated and transferred into SPE's local store by DMA while the SPE computes using the previously transferred data and instruction. This imposes a huge demand on the main memory. Data and instruction have to be in the main memory well before they are DMAed to SPE and processed. The address space required to satisfy the demand may well be more than the physically available space (2^42). Memory it self may limit the processor's capacity in this case. However to over come this issue, CBE provides the

virtual addressing scheme discussed above. It increases the available addressing space from 2^42 to 2^65. This is one of the main advantages of virtual memory.

## Memory Fragmentation

Segments are continues virtual address space with many continuously addressed virtual pages. It doesn't mean all the virtual pages are mapped exactly the same way (continuously) in the physical memory. The pages can be in different locations of the physical memory or they can be in the secondary storage. However the pages are at least 4KB of blocks in memory, not less. When a process is created it is assigned with a segment (256MB) for its data, stack and instructions even if that process doesn't require that much of memory at that moment. This is similar to allocating memory statically for a process instead of allocating dynamically, it when the need arises. If memory is requested for allocation, it will always be allocated in blocks of 4KB (or more). And when a process frees the memory (when segment get released) it will be freed in blocks of 4KB. It helps to eliminating memory fragmentation in the physical and virtual memory space.

## Can virtual memory be used for real time operating system in CBE?

There are great benefits in using virtual memory in CBE as mentioned above. However is it suitable for real time applications? Virtual addressing mode requires address translation to access the physical memory, which introduces run time overheads. Memory management unit in PPE and SPE provide few hardware caches to speed up the translation. The number of cache entries in each hardware cache unit is small (in PPE , 64 entry for instruction and 64 entry for data in ERAT, two-62 entry for each thread in SLB and 1024 entry in TLB) compare to instruction and data that may need translation on the fly. Cache miss in ERAT invokes 11 cycle penalty. Then ERAT miss entry is flushed and ERAT access is given to other thread. Now the address has to go through other HW cache units to get a hit. If a translation hit occurs in TLB unit, then ERAT entry is reloaded and address translation is attempted again by the original thread. In case of a cache miss in all hardware cache units, the processor has to access the page table in memory for translation. This introduces more clock cycle penalty for the thread. If address translation is successful and page is in the memory, then caches will get updated. However, if the page is not found in the memory means that a page fault has occurred. Now the processor has to find the least recently used page for swapping. Once the LRU page is found it get swapped for the new page from the secondary storage. This will impose more latency on the tread since accessing secondary storage is an extremely slow process.

The cost of cache miss in SPE is even greater, since it involves additional DMA transfers and PPE population of TLB and SLB entries for SPE.

Such runtime overheads may cause the real time applications to miss dead lines. According to cell broadband engine programming hand book (IBM, 2006), TLB hit rate

is more than 95%. However, getting a hit at TLB means that hit miss in ERAT HW unit, and it would already have caused 11 cycle penalty any way.

Since virtual addressing scheme imposes a non-deterministic behavior on address translation, real time applications with hard deadlines cannot run with virtual memory. CBE offers an option of disabling virtual memory altogether.

To reduce the page fault, CBE offers few tricks such as SW TLB management option. However this option is restricted to hypervisor (logically dividing the single system into many partitions) mode only. The benefit of SW managed TLB is that, during a page fault, SW not only can update the missed entry but also it can preload TLB with "translation entries that are anticipated in the future code execution" (IBM, "Cell Broadband Engine Programming Handbook",p.95), which would prevent TLB misses altogether.

So in conclusion, virtual memory offers great benefits to non-real time applications. However, real time application with hard deadline cannot run with virtual memory. Real time applications with soft deadline may run with virtual addressing tuned on, since address translation TLB hit rate is 95%. It means in most cases, applications will be able meet their deadlines.

# Memory Allocation Strategies on CELL BE

Memory management in an RTOS (or in any OS) must be considered at design time. There are two fundamental questions to be asked. These are:

a.) How many addresses does a processes get; and
b.) Can the process dynamically allocate and free memory once it is running?

The answers to these two questions are somewhat intertwined. There are at least three popular models of memory allocation that can be used.

1. Processes have a fixed amount of address space;
2. Processes can request extra memory, but not release it; and
3. Processes can request and free memory.

## *Fixed Amount of Address Space*

If processes have a fixed amount of address space there are four ways to implement it:

1. All processes have the same amount of memory.
2. Memory is divided into blocks of a fixed size, and a process gets the one that fits it best. Ie. there are blocks of size 4KB, 16KB, NKB and so on.
3. Memory is divided up into blocks of fixed and equal size, a processes gets as many of these as it needs.
4. Each processes gets exactly the amount of memory it requires.

All processes having the same amount of memory is the simplest and easiest method to implement. However it is very inefficient and will artificially constrain the number and size of processes that can exist on the system.

Best fit blocks removes some of the inefficiencies of the all equal sizes method, but is still not perfectly efficient in the use of its memory. Using multiple, equal blocks is better still, depending on the size of the blocks. The blocks must also be contiguous, imposing some restrictions on the system.

Giving each process exactly the amount of memory it needs is the most efficient method, but makes it much harder to change the software running on the system at a later date.

If the user of the RTOS cannot change the static sizes of the blocks either by recompiling the OS or changing a configuration file, unnecessary limits may be imposed on them by the RTOS developer.

All of these methods require that you know the maximum amount of memory needed by all processes during the design stage, and are very good from the view of determinism. Some RTOSes, such as RT Linux, use approaches like these for this reason.

## *Processes Request More Memory*

If processes can request extra memory, then this means we have a heap which must be managed. This increases the complexity of the memory management lists of unused areas of memory must be kept and calculations must be done to determine which sections of memory a process will get. There are three ways to handle processes being able to request more memory.

First, the memory can be divided into blocks of fixed and equal size, when more is requested by a process it gets as many blocks as necessary to fill the request. Normally these must be contiguous but if the blocks are equal to the page (or segment) size then they may not have to be, depending on how the MMU works. There can and will be inefficiencies in this method, depending on the size of the blocks.

Second, the memory can be divided into blocks of fixed but unequal size. A processes gets the smallest available block that will fill its request. This has some inefficiencies, especially if the process is requesting a small amount of memory compared to the smallest available block.

Thirdly, exactly the amount of memory requested can be returned, provided that there is enough free, contiguous space. This is the most efficient as far as size goes, but imposes much more overhead for tracking available space.

## *Request and Release Memory*

Allowing processes to release memory they are no longer using increases the overhead. The lists of free blocks or sections of memory must be constantly updated.

Dynamic memory allocation can affect the determinism of a system, reducing confidence in its ability to meet deadlines. However some RTOSes do provided dynamic memory allocation, for example QNX Neutrino.

A system with dynamic memory is more flexible than a system with out, in the end the design decision must be made depending on the final applications the RTOS will be used for and how easily the designers wish to be able to change the final applications.

For the CELL processor idividual DMA transfers have a maximum size of 16KB, but the SPE can queue a list of 2048 such transfers. PPE started transfers are limited to a single transfer. In addition to this, the SPEs have a limited local store of 256KB. It takes 16 transfer operations to completely change the contents of an SPE's local store.

As the SPE's local store is a shared data and code space, this means that ideally code running on an SPE should take up 256KB or less. It is possible to break the code into pieces and have the currently running code load the next piece when it finishes, but this does add to the complexity of the final system.

Also the larger the code, the less room there is for data on the SPE. A large amount of code, taking up more of the SPE's local store would mean more DMA transfers would be required to load all the data into the local store as processing went on.

Because the DMA transfer is independent from the processing hardware on the SPE it is possible to have transfers occuring in the background. This allows for double buffering of either data, code or both. When the SPE is finished with one buffer, it switches to the other while a DMA transfer, or series of them, replaces the old buffer.

When it comes to memory allocation, DMA transfers are optimized for sizes which are multiples of 128 bytes. Using a memory allocation scheme that forced assigned memory into sizes that fit this criteria would help improve the determinism of the system.

## References


[AE05] Echenberger, Alexendre, Kathryn O'Bien, Kevin O'Brien, Peng Wu, Tong Chen, Peter Oden, Daniel Prener, Janice Shepherd, Byoungro So, Zehra Sura, Amy Wang, Tao Zhang, Peng Zhao, Michael Gschwind, "Optimizing Compiler for the Cell Processor", 14th International Conference on Parallel Architectures and Compilation Techniques, St. Louis, Missouri, 17-21 Sept, 2005. Accessed March 4, 2007 at: http://.pact05.ce.ucsc.edu

[BE06] IBM Systems and Technology Group, "Cell Broadband Engine Programming Handbook", Version 1.0, 19 April 2006.

[CAR07] John Carbone (2007), Efficient Memory Protection for Embedded Systems, Express Logic RTC Group, Retrieved March 15 2007, from the World Wide Web http://www.rtcmagazine.com/home/article.php?id=100120

[GER06] Gerosa, Liuigi, "Cell Multiprocessor: supercomputer on a chip," 2006. Accessed March 4, 2007 at: www.elet.polimi.it/upload/silvano/mioweb5/FilePDF/arc_multimedia/Gerosa_cell.pdf

[HOF05] Hofstee, Peter, "Introduction to the Cell Broadband Engine," Technical report, IBM Corp., 2005.

[KAH05] Kahle, J., M. Day, H. Hofstee, C. Johns, T. Maeurer, and D. Shippy, "Introduction to the Cell Multiprocessor" IBM Journal of R&D, 49(4), pp. 589-604, 2005. Accessed March 4, 2007 at: www.research.ibm.com/journal/rd49-45.html

[KPP06] Kistler, Micheal, Micheal Perrone, Fabrizio Petrini, "Cell Multiprocessor Communication Network: Built for Speed," *IEEE Micro*, 26(3), May/June 2006.

[KRE05] Krewell, Kevin, "Cell Moves Into the Limelight," Instat MircroProcessor Report, 14 February, 2005. Accessed March 4, 2007 at: www.mdronline.com/publications/epw/issues/pw_021405.html

[KS06] Shimizu, Kanna, Sanjay Gupta, Tatsuya Koyama, Takashi Omizo, Jamee Abdulhafiz, Larry McConville, and Todd Swanson, "Verification of the Cell Broadband Engine Processor" Proceedings 43rd ACM/IEEE Design Automation Conference (DAC), 24-28 July 2006, pp. 338-343.

[L107] LynuxWorks™, Inc (2007), Using the Microprocessor MMU for Software Protection in Real-Time Systems, Retrieved March 15 2007, from the World Wide Web http://www.lynuxworks.com/products/whitepapers/mmu.php3,

[L207] LynuxWorks™, Inc (2007), Processes, Name Spaces and Virtual Memory, Retrieved March 15 2007, from the World Wide Web

http://www.lynuxworks.com/products/posix/processes.php3

[L307] LynuxWorks™, Inc (2007), Processes, Partitioning Operating Systems Versus Process-based Operating Systems, Retrieved March 15 2007, from the World Wide Web
http://www.lynuxworks.com/products/whitepapers/partition.php

[TUT06] IBM Systems and Technology Group, "Cell Broadband Engine Programming Tutorial", Version 2.0, 15 Dec 2006.

[WIL05] Williams, Samuel, John Shalf, Leonid Oliker, Shoaib Kamil, Parry Husbands, Katherine Yelick, "The Potential of the Cell Processor for Scientific Computing," Proceedings of the 3rd Conference on Computing Frontiers, ACM Press, 2006, pp. 9–20.