Chapter 5
Direct Memory Access (DMA)

Abstract In this chapter we discuss the Direct Memory Access (DMA) functionality of the Cell architecture. Sender and receiver initiated DMA plays a significant role in program optimization. We present the DMA engine, fence and barrier concept for ordering data transfers. The DMA engine can be used as an additional data movement processor using dma-list. Double-buffering and SPE-to-SPE DMA is discussed in the context of examples in the sequel of this text.

5.1 Introduction to DMA in the Cell Broadband Engine

The SPEs DMA engine handles most communications between the SPU and other Cell elements and executes DMA commands issued by either the SPU or the PPE. A DMA commands data transfer direction is always referenced from the SPEs perspective. Therefore, commands that transfer data into an SPE (from main storage to local store) are considered get commands (gets), and transfers of data out of an SPE (from local store to main storage) are considered put commands (puts). DMA transfers are coherent with respect to main storage. Programmers should be aware that the MFC might process the commands in the queue in a different order from that in which they entered the queue. When order is important, programmers must use special forms of the get and put commands to enforce either barrier or fence semantics against other commands in the queue.

The MFCs MMU handles address translation and protection checking of DMA accesses to main storage, using information from page and segment tables defined in the PowerPC architecture. The MMU has a built-in translation look-aside buffer (TLB) for caching the results of recently performed translations.

The MFCs DMA controller (DMAC) processes DMA commands queued in the MFC. For the most part we analyze only the MFC SPU command queue.

In the absence of congestion, a thread running on the SPU can issue a DMA request in as little as 10 clock cycles (we had seen in Chapter 3 that instruction latency for SPU channel read-write is 2 clock cycles). For each of the five parameters
The overall latency of generating the DMA command, initially selecting the command, and unrolling the first bus request to the BIU or, in simpler terms, the flow-through latency from SPU issue to injection of the bus request into the EIB is roughly 30 SPU cycles when all resources are available. If list element fetch is required, it can add roughly 20 SPU cycles. MMU translation exceptions by the SPE are very expensive and should be avoided if possible. If the queue in the BIU becomes full, the DMAC is blocked from issuing further requests until resources become available again. Examples of DMA transfers are present in almost every major SPE program presented in the later chapters. SPE-to-SPE DMA without PPE involvement (except for initial LS address gathering) is shown in Section 19.6, pp. 375.

A transfers command phase involves snooping operations for all bus elements to ensure coherence and typically requires some 50 bus cycles (100 SPU cycles) to complete. For gets, the remaining latency is attributable to the data transfer from off-chip memory to the memory controller and then across the bus to the SPE, which writes it to local store. For puts, DMA latency doesn’t include transferring data all the way to off-chip memory because the SPE considers the put complete once all data have been transferred to the memory controller.

5.1.1 Difference between fence and barrier synchronization

In particular, there are two flags that can be embedded into a command, barrier and fence. Both flags affect only commands in the same tag group. Generally, the embedded fence flag will not allow the command to execute until all commands within the same tag group and issued prior to the command with the embedded fence flag are complete. That is, the fence flag requires that all commands within the same tag group issued prior to the command with the embedded fence be completed prior to the execution of the command with the fence flag. The fence flag does not affect subsequent commands in the queue. Thus, a command issued after a command with an embedded fence flag can execute before the command with the fence.

The barrier flag affects all previous and subsequent commands within the same tag group. Generally, the barrier flag will not allow the command with the barrier flag or any subsequent commands within the same tag group to be executed before the execution of every command in the same tag group issued prior to the command with barrier flag. For example, commands within the same tag group and issued after a command with a barrier flag cannot execute before the command with the barrier flag. Typically, when all commands within the same tag group issued prior to the command with the barrier flag are complete, the command with the barrier flag and subsequent commands within the same tag group can be executed.

The barrier command, as opposed to the barrier flag orders all subsequent MFC commands with respect to all MFC commands preceding the barrier command in