Application Report
SPRAA68 – November 2004

TMS320C64x+ Megamodule

Chad Courtney
Wireless Infrastructure

ABSTRACT

The C64X+ Megamodule supports a wide variety of internal memory configurations by allowing the L1 program and data memory (L1P and L1D) to set as cache only, SRAM only, or a mixture of cache and SRAM. In addition, the C64x+ Megamodule provides new system functionality including: cache freeze, Internal DMA (IDMA), bandwidth management, and memory protection. This document discusses the enhancements made to the internal memory and describes the new features which have been added to support the internal memory architecture's performance and protection.

Contents

List of Figures

1 Introduction

Over the past few years, the C6000 architecture has evolved in order to provide higher performance. One area of evolution in particular is the internal memory architecture, which has evolved from fixed program and data SRAMs to L1 program and data cache with fixed L2 SRAM. More recently, L2 has been made configurable as SRAM and cache.

The C64x+ Megamodule continues this tradition, evolving the existing two-level memory hierarchy to support a wider variety of internal memory configurations by allowing the L1 program and data memory (L1P and L1D) to be set as cache only, SRAM only, or a mixture of cache and SRAM. In addition, the C64x+ Megamodule provides new system functionality including: cache freeze, internal DMA, bandwidth management, and memory protection.

This document discusses the enhancements made to the internal memory and describes the new features which have been added to support the internal memory architecture's performance and protection.

2 Internal Memory Enhancements

2.1 L1P and L1D Cache/SRAM

The C64x+ Megamodule continues the evolutionary process by evolving the existing two-level memory hierarchy to support a wider variety of internal memory configurations.

The main reason for using cache is to reduce the average memory access time. Since programs typically reuse data from the same or adjacent memory locations within a small period of time, it is good to cache this data into a fast memory while allowing the data to permanently reside in a slower memory, such as L2. The normal behavior of the cache improves the average performance of code with very little effort from the programmer. To reach peak performance, some applications can benefit from more direct control of where data is placed in memory. Most data which is cached is only there for a short while and later either becomes invalid because new data is written into the space which the cache is associated, or is evicted/discarded when new data is being cached.

The data that is invalidated after a time but used on multiple occasions during a short period of time are ideal for caching. Data that is discarded due to other things being cached into the same cache-way may not be ideal for caching. Such is the case when this data is not modified or invalidated and need to be accessed again soon, but is evicted prior to being accessed. Some examples would be static lookup tables which are used periodically by programs, functions which are called periodically, and functions which are called frequently by other programs.

When such frequently used data and functions are cached, it often can reduce the overall performance of the DSP, due to cache thrashing between it and other cached data. In such cases, the advantages of having the data or functions in fixed SRAM insure it is always ready when needed.

2.2 Cache Coherence Protocol Changes

The C64x+ Megamodule also evolves the scheme by which it keeps DMA activity coherent with program activity. The new protocol makes two important changes. First, it removes coherence between the L1P cache and DMA activity in L2 and L1P. Second, it introduces a more direct protocol for keeping L1D synchronized with updates occurring in L2. This was done to reduce the complexity of the L1P cache controller and L2 controller interface and in turn eliminate performance penalties associated with snoop invalidates sent to L1P.

The C64x+ Megamodule simplifies the L1P protocol, both to reduce the complexity of the design and to increase its efficiency. The C64x+ L1P treats all program fetches as cacheable, and does not track DMA activity elsewhere in the system. The memory attribute registers (MAR) which control cacheability for L2 and L1D no longer affect the contents L1P. Programs can manage coherence between L1P and DMA activity as needed using software-accessible cache controls.

The L2 controller will maintain cache coherence with L1D cache when DMA performs writes to L2. However, it will not maintain the cache coherence with L1P cache. This is the main difference in cache coherence from a user perspective, because it will require the user to maintain cache coherence of L1P for DMA writes to L2. In most cases this will not be an issue, as users will have program code in L2 SRAM fixed. The issue will arise when using a program paging scheme, in which case the kernel is responsible for maintaining cache coherence between L1P cache and L2.

In contrast to L1P, the C64x+ L1D and L2 controllers more actively manage coherence for DMA and CPU activity in L2 memory. Previous devices managed coherence by moving cached data out of L1D and into L2 in response to DMA activity in L2. On those devices, DMAs can result in subsequent cache misses in L1D, leading to lower overall performance. The C64x+ Megamodule avoids this by redirecting DMA accesses directly to L1D when it makes sense to do so.

For incoming DMAs, the C64x+ detects if the address the DMA writes to is held in L1D. If it is, it forwards a copy of the data to L1D and updates both L1D and L2 in parallel, as opposed to removing the line from L1D and only updating L2. For outgoing DMAs, the C64x+ detects whether L1D holds a modified copy of the data. If it does, it requests the updated data directly from L1D, rather than instructing L1D to write it back to L2 first. The result is less memory traffic between L1D and L2, as well as fewer stalls due to L1D cache misses. This saves energy and cycles.

3 Features Added with C64x+ Megamodule

The following features have been added to the C64x+ Megamodule:

- Cache freeze
- Internal DMA
- Memory protection
- Bandwidth manager

4 Cache Freeze

The purpose of a cache freeze is to prevent data that is in cache from being evicted from the cache. This can be particularly useful in preventing non-periodic interrupt service routines from overwriting cached data (which would need to be re-cached later).

Cache freezing capabilities were added to each of the three caches: L1D, L1P, and L2, and are controlled individually. Cache freezing only affects the cache portions of memory; it does not affect any portion that is set as SRAM.

4.1 L2 Cache

When L2 cache is frozen, it responds to read and write hits by accessing the requested data. It does not, however, allocate new lines if an access misses the L2 cache. It also does not update the LRU state within the cache for hits or misses. The LRU state normally indicates which *way* in a given set was *least recently used*.

Read and write misses are treated as non-cacheable and result in long-distance accesses. New L2 cache lines cannot be allocated while L2 cache is frozen. If something is not in cache and is accessed, it will not be cached by L2 and the long-distance access is required.

While L2 cache is frozen, cached lines can still be evicted from L2 by software and manual cache coherence operations, via the L2 writeback or L2 writeback with invalidate operations.

4.2 L2 Cache Freeze

When L1D cache is frozen, it responds to read and write hits by accessing the requested data. It does not, however, allocate new lines if an access misses the L1D cache. It also does not update the LRU state within the cache for hits or misses.

Read hits result in the data being returned from cache. Write hits result in the data being updated for the cache line and marked dirty if needed. The LRU bit, which indicates the least recently used way for the affected cache line, is preserved so that the cache eviction ordering is not effected by actions which occurred while the L1D cache was frozen.

Read misses are non-cacheable while L1D cache is frozen, thus the L1D cache will not allocate new lines. If data is not cached and an attempt to read this data occurs, it will not be cached by L1D and a long-distance access is required.

L1D cache is a read-allocate-only cache, and does not perform write allocation. Therefore, all L1D write misses will be queued in the L1D write buffer whether L1D is frozen or not.

While L1D cache is frozen, cache-coherence is still required to be maintained. L1D cache will still respond normally to cache-coherence commands issued from L2 including snoop-read and snoop-write, as well as any software and manual cache controls (writeback, invalidate, writeback-invalidate, mode change). This is necessary to ensure that freeze mode does not violate cache coherence and does not override any software and manual cache commands, which are part of the software or is being applied with emulation.

4.3 L1P Cache Freeze

The L1P cache support of freeze mode allows applications to prevent CPU program data accesses from evicting cached program data. A variety of applications may find this useful in an interrupt context so that interrupts do not unduly affect the current cached program data.

When L1P cache is frozen, it responds to read and write hits by accessing the requested data. It does not, however, allocate new lines if an access misses the L1P cache. It also does not update the LRU state within the cache for hits or misses.

Read misses are non-cacheable while L1P cache is frozen, thus the L1P cache will not allocate new lines. If data is not cached and an attempt to read this data occurs, it will not be cached by L1P and a long-distance access is required.

The only time write hits and misses occur with L1P Cache is when the Emulation Data Interface (EDI) accesses the L1P Cache. Write hits via emulation will occur normally when L1P cache is frozen, while write misses will be dropped as usual.

While L1P cache is frozen, cache-coherence is still required to be maintained. L1P cache will still respond normally to cache-coherence commands from software and manual cache controls (invalidate and mode change). This is necessary to ensure that freeze mode does not violate cache coherence and does not override any software and manual cache commands, which are part of the software or is being applied with emulation.

5 Internal DMA

An integral part of the C64x+ Megamodule is the IDMA controller. The IDMA is a simple DMA engine that can be used to perform block transfers between any two local memory-mapped locations. Local memories include L1P, L1D, L2 and peripheral configuration registers.

The IDMA has three ports: port A, port B, and the peripheral configuration port (CFG). Ports A and B connect to L1P, L1D, and L2, while the peripheral configuration port connects to the peripheral configuration registers.

The IDMA controller contains an address map that defines the addresses available through each port and whether or not it is considered local. The IDMA controller allows rapid data paging between all local memories and provides a fast way to page data sections to any local memory-mapped RAM. The key advantage of the IDMA is that it allows for paging between slow L2 and fast L1D data memory, which provides lower latency than the cache controller. Also, the transfers take place in the background of CPU operation, so stalls due to cache can be removed from the system.

In addition, the IDMA facilitates rapid programming of peripheral configuration registers. The IDMA has a windowed view of the configuration space and allows any register within that window to be individually accessed.

The IDMA also provides the ability to perform a block fill of memory, whereby the controller issues a block of writes only, using a user-programmed fill value.

The IDMA transfers data at a rate of 256-bits every L2 memory controller cycle (CPU/2).

There are two IDMA Channels. Channel 0 is meant for configuring or reading the configuration of peripheral registers, and Channel 1 is meant for transferring data from one memory to another.

5.1 IDMA Channel 0

IDMA channel 0 is intended for quick programming of configuration registers through the CFG port of the C64x+ Megamodule. It provides a "window" to the configuration space through which the CPU can quickly access any of 32 registers. To allow this, the IDMA channel has five registers: status, mask, source address, destination address, and window count. The following diagram shows one possible transfer using IDMA channel 0, using a mask to identify which locations to update. The LSB of the mask applies to the first word of a 32-word source, the second LSB to the second word and so on. Applying 1 to a bit in the mask will mask the corresponding word out from being transferred, while applying 0 allows the transfer of the corresponding word.

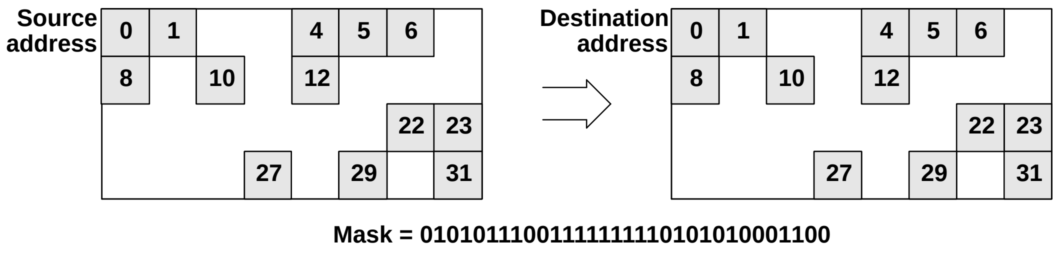

Figure 1. IDMA Channel 0 Transaction

5.2 IDMA Channel 1

IDMA channel 1 is intended for data paging between local memories. It allows the CPU to move data and program sections in the background of CPU operation to set up processing from fast memory. To allow this, the IDMA channel has four registers: status, source address, destination address, and count. All source and destination addresses increment linearly through the transfer, the size of which is set by the count field (in bytes). Following the transfer, a CPU interrupt is optionally set. The following diagram shows a transfer using IDMA channel 1.

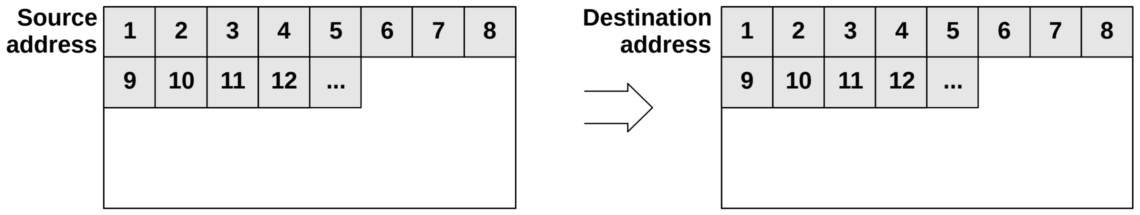

Figure 2. IDMA Channel 1 Transaction

6 Bandwidth Management

One of the new features added to the C64x+ Megamodule is the bandwidth manager. The bandwidth manager is designed to improve the overall throughput of C64x+ Megamodule by better bandwidth allocation of L1D and L2 as well as arbitration between CPU, DMAs, memory controllers, and coherence controllers.

6.1 Bandwidth Allocation and Arbitration

L1D:

Bandwidth allocation for L1D can prove very valuable in improving overall throughput when accessing data. The L1D interface consists of eight independent 32-bit banks, for a total width of 256 bits. On any given cycle, the CPU can access, at most, 128 bits using a combination of LDDW and STDW instructions. This leaves 128 bits per cycle available for use by DMA or IDMA.

With most software kernels that access the L1D on a cycle by cycle basis, you may expect the CPU to stride through the 256-bit memory sequentially, resulting in a nice clean pattern being available for the DMA/IDMA to sneak into. This is relatively simple for bandwidth management. However, certain algorithms, like FFTs, DCTs, and others may access a specific bank of L1D for extended periods of time, requiring a more complicated bandwidth management scheme to allow optimal usage of the memory interface. This is where arbitration is needed to make sure one source does not lock out all other sources from accessing memories.

The following example shows a simple scenario where the CPU is accessing two double words in sequence every cycle, while the DMA/IDMA attempting to perform a 256-bit aligned transfer to L1D. In this case, the CPU has a higher priority than DMA/IDMA transfer, and the CPUs initial accesses on Cycle 0 are to banks 1/0 and 3/2. The DMA/IDMA normally would have wanted to start at bank 0 on cycle 0 since the data being transfer is 256-bit aligned. The DMA/IDMA instead takes advantage of the open banks 4,5,6,7 and sends the second 128 bits on cycle 0, then on cycle 1 while the CPU is using banks 4,5,6,7 the DMA/IDMA sends the first 128 bits utilizing banks 0, 1, 2, 3. This repeats while both CPU and DMA/IDMA activity remains the same.

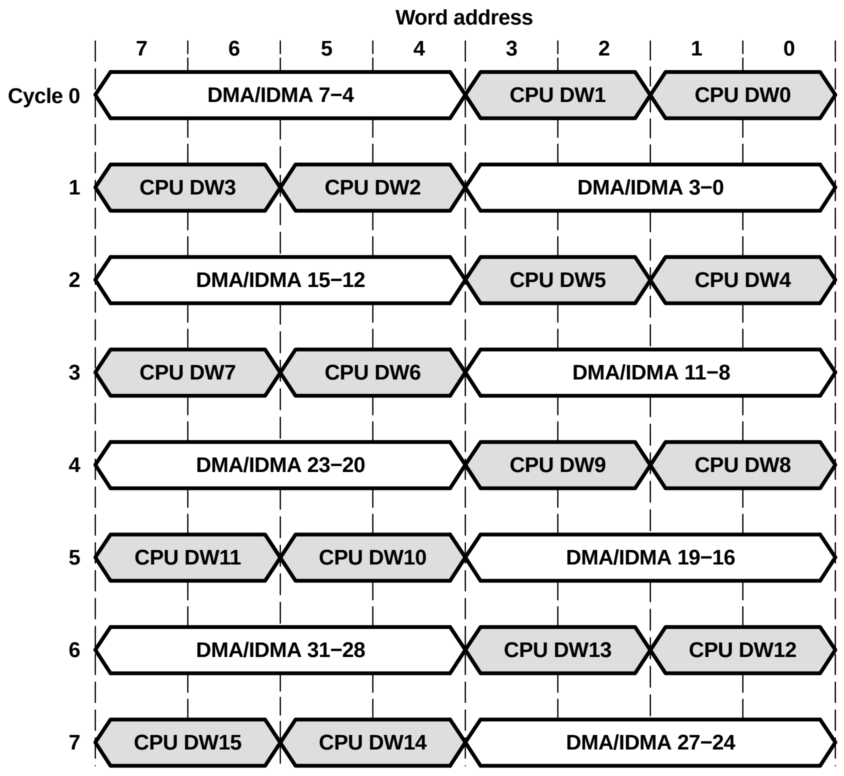

Figure 3. L1D, CPU With Higher Priority than DMA, DMA Accessing Open Bandwidth

A second example shown in Figure 4 has a slightly, more complicated scenario in which the CPU is constantly accessing the same bank/word location on every cycle. Notice how on cycle 0 the DMA/IDMA path was able to utilize banks 7-1, but could not progress any further because the CPU is accessing bank 0 every cycle. Normally, this would have locked out the DMA/IDMA transfer until the CPU was finished accessing Bank 0, assuming CPU is set at a higher priority than DMA/IDMA. In order to prevent this, apply the bandwidth allocation rules where the max wait for the DMA/IDMA arbitration controller is set to 4 cycles. This sets up a stall counter which counts the number of cycles the transfer for the particular controller has been stalled. When this counter reaches its limit, in this case on cycle 4, the controller (in this case DMA/IDMA), is given priority for 1 cycle. The whole process starts over again for the next 256 bits of DMA/IDMA data.

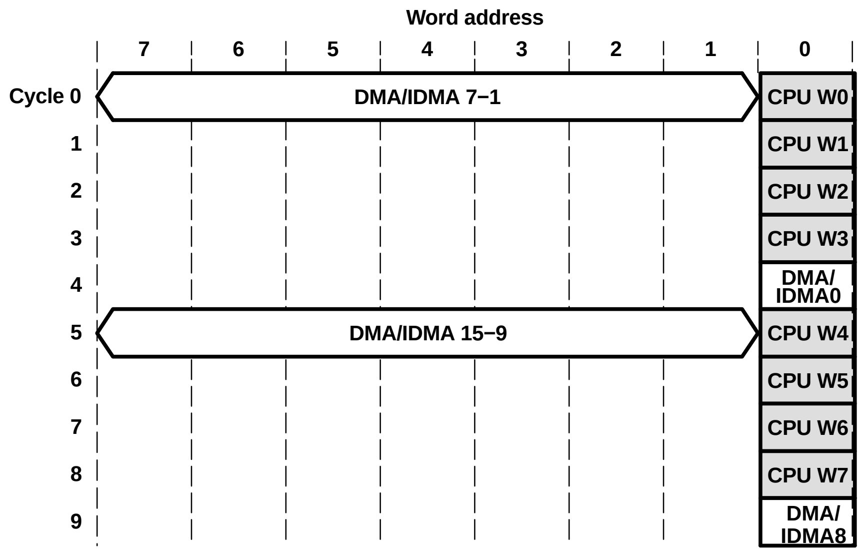

Figure 4. L1D, CPU With Higher Priority than DMA, DMA Accessing Open Bandwidth and Max Wait Set to 4 Cycles

L2:

The L2 memory may vary from device to device, but will use the same C64x+ Megamodule. The megamodule, however, will always be viewed as having logical 256-bit banks. Concurrent accesses from different requestors to different sub-banks of the logical 256-banks are not allowed.

This document will not go into the all the potential configurations for L2, but will focus on what is seen as a typical configuration (as in the case of C6495). A typical configuration will have two 256-bit SRAM banks that are individually capable of throughputs at the rate of the L2 controller/2 speed. This will yield an effective max throughput speed of 256-bits per L2 controller cycle or 128-bits per CPU cycle as the L2 controller will operate at CPU/2.

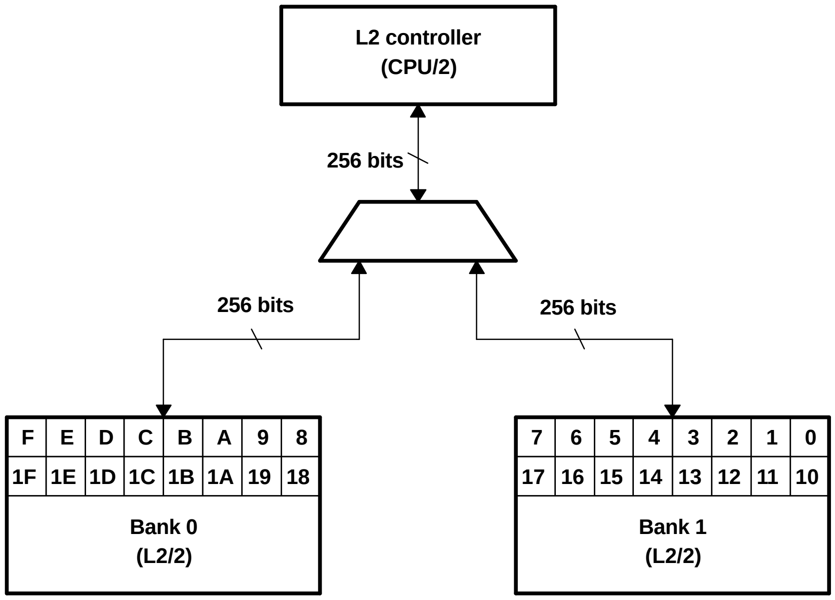

Figure 5. L2 Banking Structure

In the following examples:

- 128-bit peripheral data transfers (PDT) are concatenated into 256-bit transfers on the peripheral DMA interface (DMAIF).
- Pending DMAIF transfer represents the peripheral data transfers that are pending arbitration.
- Pending IDMA transfers represent the IDMA transfers that are pending arbitration.
- Pending transfers which are "stretched" out dataphases represent a loss of arbitration.
- Stalls due to loss of arbitration due to being lower priority level are shown with an incrementing number in the pending transfer column. This is a W if stall is due to a bank conflict.
- The winner of the arbitration is shown in Bank 0 and 1

The examples shown in Figure 6 and Figure 7 have both IDMA accessing L2 as well as the PDT accessing L2 via DMAIF. DMAIF has higher priority than IDMA and IDMA has MAXWAIT=4 for both examples, but the rules do not come into play as the IDMA is able to slip in during the gaps in the DMAIF traffic. This is especially true in the first example shown in Figure 6. Even if MAXWAIT=1, the arbitration is not needed. At the point in which the arbitrations rules would take effect, the IDMA is stalled due to a bank conflict with the pervious DMAIF, as indicated by the W in the IDMA's pending transfer column. On the cycle the IDMA is waiting for the bank conflict resolution, the DMAIF transfer is able to write to the other bank.

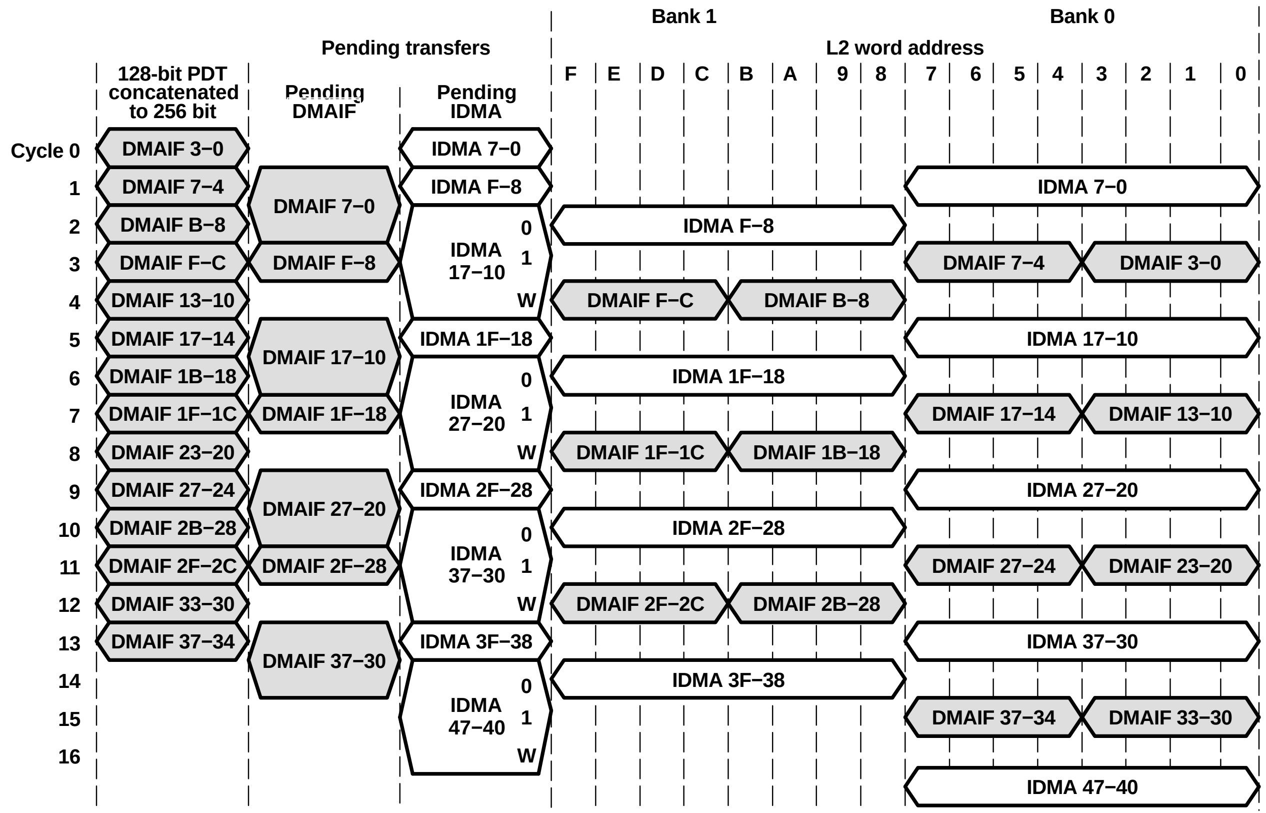

Figure 6. L2 Example, Simultaneous IDMA and DMAIF Transfers when DMAIF is Higher Priority

The example shown in Figure 7 is a slight modification of the previous example. In this example PDT is shifted forward in time by one cycle, which generates a delay of the IDMA transfers to L2 due to an initial bank conflict. On cycle 4, the IDMA is unable to proceed due to the bank conflict with the previous DMAIF transfer, and DMAIF is unable to proceed to the other bank because the request has not yet arrived. On cycle 5, the DMAIF transfer is ready to proceed to Bank 1 and preempts the IDMA transfer to Bank 0. The overall result is a single cycle delay in the transfer to IDMA. The rest of the transfers align nicely and allow full throughput to L2. The overall impact of this single cycle delay will depend upon the length of the transfers.

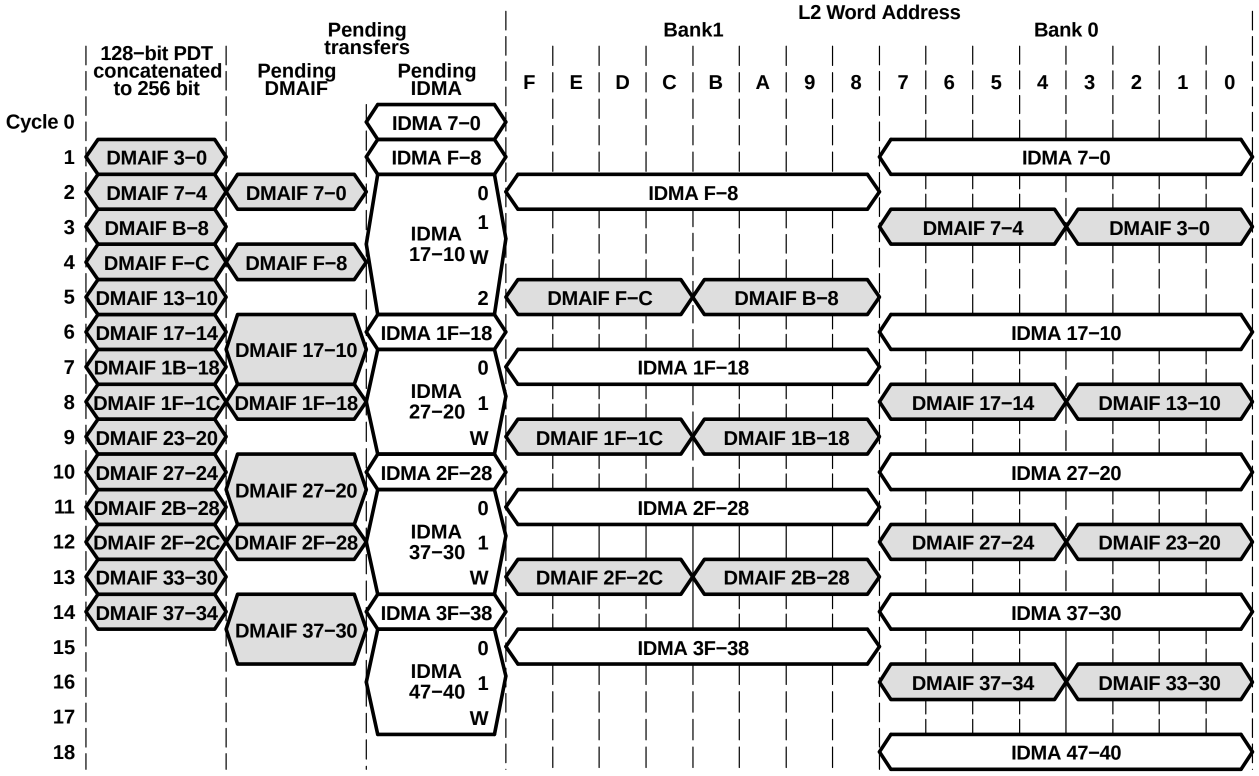

Figure 7. L2 Example, Simultaneous IDMA and DMAIF Transfers when DMAIF is Higher Priority and Time Misalignment

The examples shown in Figure 8 and Figure 9 have both IDMA accessing L2 as well as the PDT accessing L2 via DMAIF. IDMA has higher priority than DMAIF and DMAIF has MAXWAIT=4 for both examples. In this case, the IDMA is capable of using the entire bandwidth so that DMAIF's MAXWAIT=4 comes into play and arbitration occurs. A key thing to point out in these examples, is that the PDT transactions always attempt to stay two 128-bit dataphases (or a single 256-bits dataphase) ahead of the pending DMAIF transaction. This minimizes banking penalties. For example, on Cycle 7, if the "DMAIF f-8" dataphase was not ready, then cycle 8 would have gone unused and the pattern would have repeated until the transfers completed.

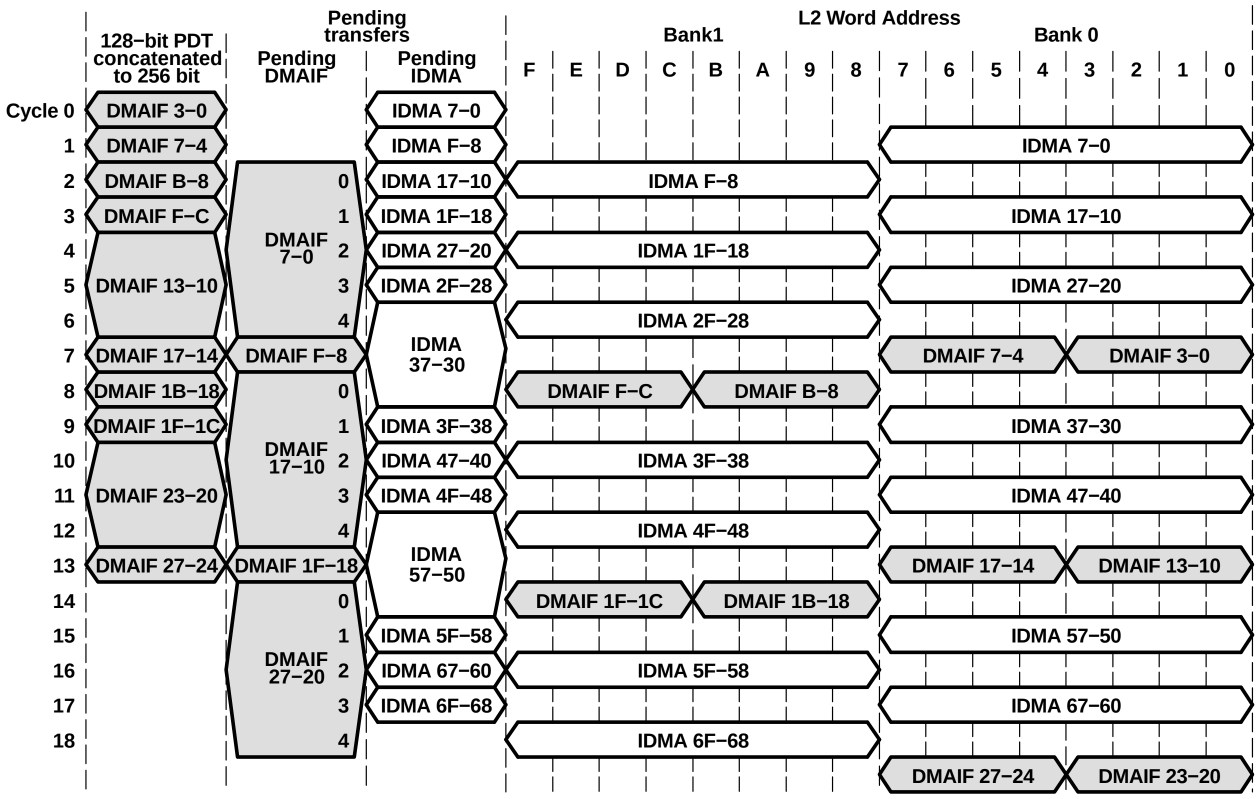

Figure 8. L2 Example, Simultaneous IDMA and DMAIF Transfers when IDMA is Higher Priority

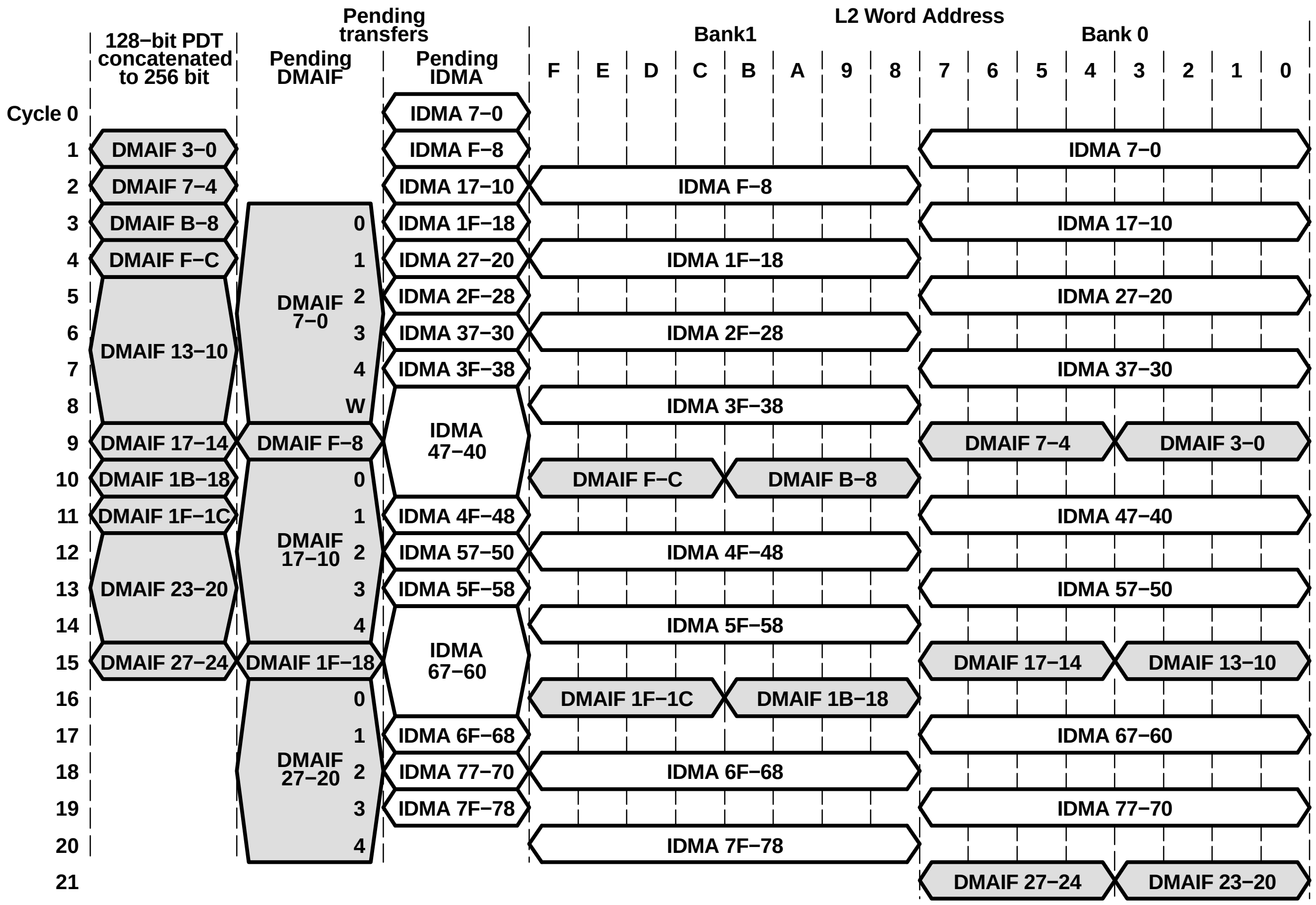

Figure 9. L2 Example, Simultaneous IDMA and DMAIF Transfers when IDMA is Higher Priority and Time Misalignment

L1P:

There is no bandwidth allocation performed for L1P. All 256 bits of the L1P memory interface are granted access to the arbitration winner on a cycle by cycle basis. The reason for this is CPU transfers always consist of a 256-bit read (CPU always fetches 1 packet). Other accesses to L1P would tend to want to use the full bandwidth, as well and no attempts to optimize multiple sub-256-bit accesses, are made.

7 Memory Protection

Another new feature added to the C64x+ Megamodule is the ability to protect memories from unwanted accesses. This may be used to protect vital OS data structures and allow OS to enforce clearly defined boundaries between supervisor and user mode accesses. It will also provide more information about illegal memory accesses when debugging a system.

The C64x+ Megamodule's memory protection provides these abilities using a combination of CPU privilege levels and a memory system protection structure.

7.1 Privilege Levels

Two privilege levels available in the C64x+ Megamodule are the supervisor mode and user mode. Supervisor code is viewed as more trustworthy than user code. Examples of supervisor code include operating system kernels and hardware device drivers. Examples of user code include filters, encoders/decoders and other end applications.

Supervisor mode is generally granted access to peripheral registers and the memory protection configuration. User mode is generally confined to memory spaces which the OS specifically designates for its use.

A privilege bit is provided by the requestor alongside each memory or peripheral register access to indicate the privilege level associated with that access. CPU, DMA, IDMA and other accesses have a privilege associated with them. CPU privilege level is determined by the user mode in which the CPU is currently operating. DMA and IDMA accesses that are initiated by the CPU inherit the CPU's privilege level at the time they are initiated. Mastering peripherals such as rapid IO, HPI, and EMAC generally issue transfers with supervisor privileges.

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265