Application Report
SPRA620 - December 1999

Memory Transfers with TMS320VC5420 and TMS320VC5421 DSPs

Peter Galicki — *Digital Signal Processing Solutions*

ABSTRACT

The TMS320VC5420 and TMS320VC5421 dual CPU DSPs feature two 6-channel DMA Controllers (DMAC) plus Host Port Interface Controllers (HPI) for efficient movement of data without involvement of the CPUs. Internal transfers can move data between the two DSP subsystems, or within each subsystem. Typical internal transfers read and write data between on-chip memories and peripherals. Memory transfers can be initiated and driven by the CPU, on-chip DMAC or the external host CPU across the Host Port Interface. Multiple internal and external channels combined with the multi-channel DMAC enable optimum utilization of all chip resources by de-coupling most Peripheral/Memory transfers from the CPU. The CPU's ability to use most of available cycles to process data, instead of moving data, is a key component of the high performance and power efficient operation of the TMS320VC5420 and TMS320VC5421 DSPs.

Contents

List of Figures

1 Differences Between the ’C5420 and ’C5421 DSPs

Both the TMS320VC5420 and TMS320VC5421 DSPs are dual-CPU implementations of the ’C54x architecture. While the two devices are mostly pin compatible, each one internally contains a different mix of on-chip RAM blocks. Functionally, the ’C5421 is an evolution of the ’C5420 with an improved DMA controller, capable of external accesses, a shared program RAM between the two subsystems and one bootload ROM per Subsystem. The HPI and XIO transfers involving the two subsystems are multiplexed on the same XPORT pins for both devices. While the XPORT (in XIO mode) on the ’C5420 can only be driven with the CPU originated transfers, the ’C5421 has added capability to also drive the XPORT interface with the DMA controller. The following is the summary of the major differences between the ’C5420 and the ’C5421:

MEMORY:

The ’C5420 has 200Kx16 bits of internal RAM (100K per subsystem) versus 256Kx16 bits of internal RAM for the ’C5421. Half of the ’C5421 internal RAM is dedicated to each subsystem, while the other half (program only) can be 2-way shared between both subsystems. The ’C5421 also features 2Kx16 bits bootload ROM per subsystem, for the total of 4Kx16 bits of internal ROM.

DMAC:

While the ’C5420 DMAC can perform internal transfers only between on-chip peripherals and memory, the ’C5421 DMAC can also directly drive external transfers. The cross-subsystem access by a DMAC is restricted to reads and writes from/to one end of the shared FIFOs on the ’C5420, while the ’C5421 DMAC has capability to directly drive transfers across subsystems through a dedicated cross-path.

XPORT:

The external port on both devices can operate in the XIO (External I/O) mode or the HPI (Host Port Interface) mode. In XIO mode, the subsystem selection for external I/O is performed with a dedicated select pin on the ’C5420, while on the ’C5421 the subsystem selection for CPU initiated external transfers uses the Request/Grant bits of the General Purpose I/O register. Additionally, only the ’C5421 EMIF features HOLD/HOLDA handshaking signals for sending requests to the DSP to temporarily release the XPORT allowing another device to access external memory.

OTHER:

- McBSP with128 Channel selection capability on ’C5421
- Modified HPI and DMA memory maps on ’C5421 versus ’C5420
- 5421 internal components are optimized for increased power savings
- JTAG Compliant boundary scan on the ’C5421

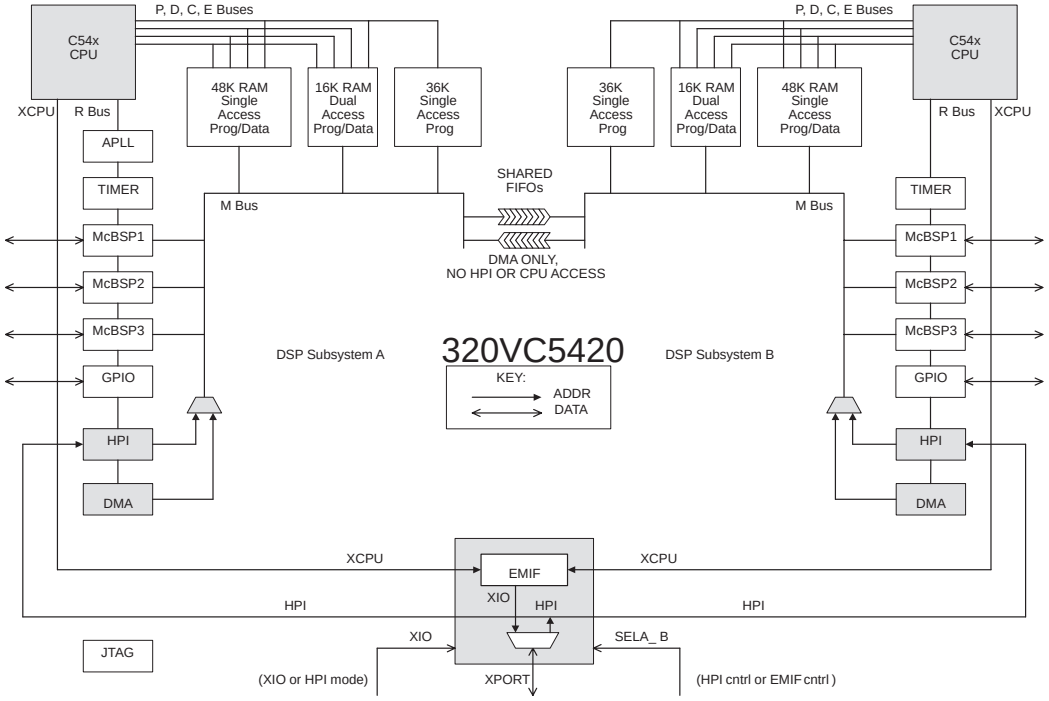

Figure 1. TMS320VC5420 DSP

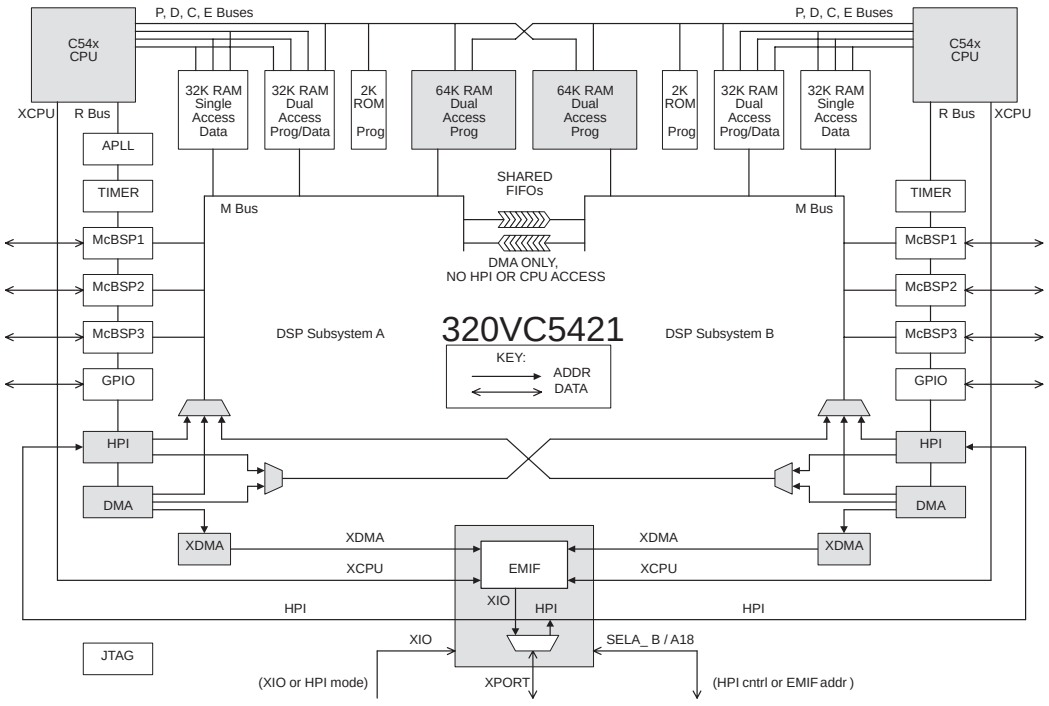

Figure 2. TMS320VC5421 DSP

2 External I/O Terminology

The ’C5421’s external I/O block consists of the XPORT module containing the EMIF sub-block and the HPI multiplexer. The XPORT module can function either in the EMIF mode (master) or HPI mode (slave).

Internally to the DSP, the XPORT module connects to subsystem A and subsystem B. On the outside, the XPORT module connects to the Parallel Port pins. Depending on the state of the input pin called XIO, the Parallel Port pins represent the XIO master memory interface driven by subsystem A or subsystem B, or a slave Host Port Interface (HPI) driven by a host CPU.

In the XIO mode the XPORT module connects the output of the EMIF module to the Parallel Port. On the ’C5421 device, the XIO bus is mastered by the CPU or DMAC read/write cycles originating in subsystem A or subsystem B. Internally, the EMIF uses Hold/Hold Acknowledge handshaking (involving GPIO bits) to arbitrate between CPU requests on XCPU buses and round-robin arbitration for external DMAC requests on the XDMA buses (see the EMIF Arbitration chapter for more information). DMA requests always have priority over CPU requests for access to the Parallel Port. On the ’C5420 device configured for XIO memory transfers, EMIF uses the SELA/B to select which CPU is driving the Parallel Port pins. The ’C5420 DMA controllers have no external I/O reach.

In the HPI mode, SELA/B input pin selects subsystem A or subsystem B HPI module to respond to I/O transfer requests from the host CPU. The host can access all internal resources of the CPU through the HPI module of either of the subsystems. The exception is the internal FIFO connecting the subsystems. It is only accessible by internal DMA cycles.

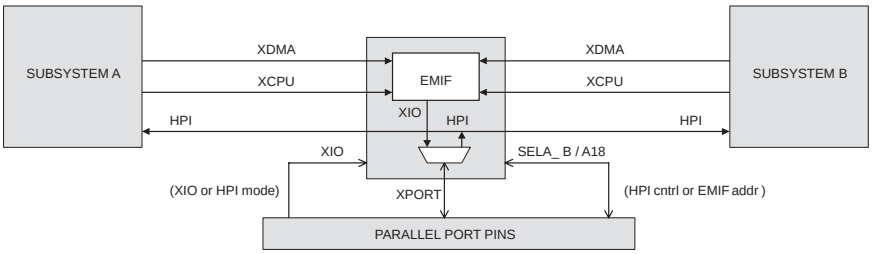

Figure 3. ’C5421 External I/O

3 TMS320VC5420 I/O Transfers

The ’C5420 I/O transfers are driven by the CPU, on-chip DMAC or the external host processor across the Host Port Interface. While the DMAC is limited to internal transfers, the CPU can perform both internal and external transfers. Additionally, the DMAC has the capability to transfer data between the two subsystems (in both directions) across a set of two dedicated internal FIFOs.

3.1 Transfers Driven by the CPU

The CPU transfers are triggered by CPU program fetches from program space and instruction operand loads/stores from and to data or I/O spaces of the memory map. Some CPU instructions transfer data between data and program spaces. Depending on the address location within the memory map, the CPU initiated transfers can be internal or external to the DSP. The internal transfers use the high performance P, C, D and E buses to access internal RAM and a slower R bus to access on-chip peripherals. The CPU of each subsystem uses its local X bus to access external memory or peripherals through a single common EMIF interface.

3.1.1 Internal CPU Transfers

The main types of the ’C5420 internal transfers initiated by the CPU include internal memory program/data accesses and peripheral initialization/servicing. The CPU accesses internal memory over a dedicated set of high performance P,C,D and E buses, intended for simultaneous transfers of one instruction, two operands and a single result. The CPU accesses the on-chip peripherals using the R bus. While the CPU is capable of accessing peripheral data, in most applications it is only used to initialize and control the peripherals. Offloading the peripheral data transfers to the DMAC reduces the overhead associated with interrupt context switches and frees the CPU to spend more cycles processing data.

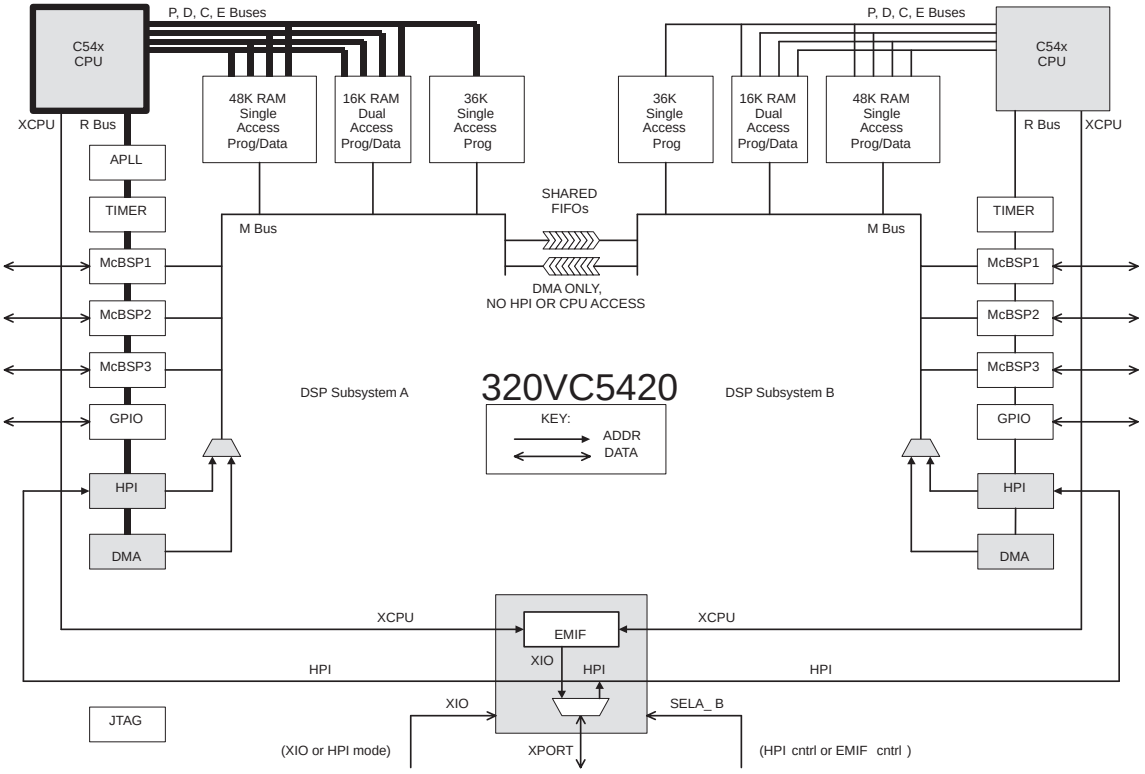

Figure 4. ’C5420 Internal Transfers Driven by the CPU

3.1.2 External CPU Transfers

External CPU Transfers are triggered by the CPU fetching program from the external address space, or by executing load or store instructions involving external data or I/O spaces of the memory map. The CPU access to external memory across the XPORT is allowed only when the XIO pin is set to 1, indicating the XIO mode. (XIO=0 configures the XPORT in the HPI mode, and any attempt to access external memory by the CPU is ignored for writes, and loads undefined data for reads). Since a single EMIF has to be shared by both subsystems of the ’C5420, only one subsystem can drive external transfers across EMIF at any one time. The SELA/B pin selects subsystem A or subsystem B as a current owner of the EMIF.

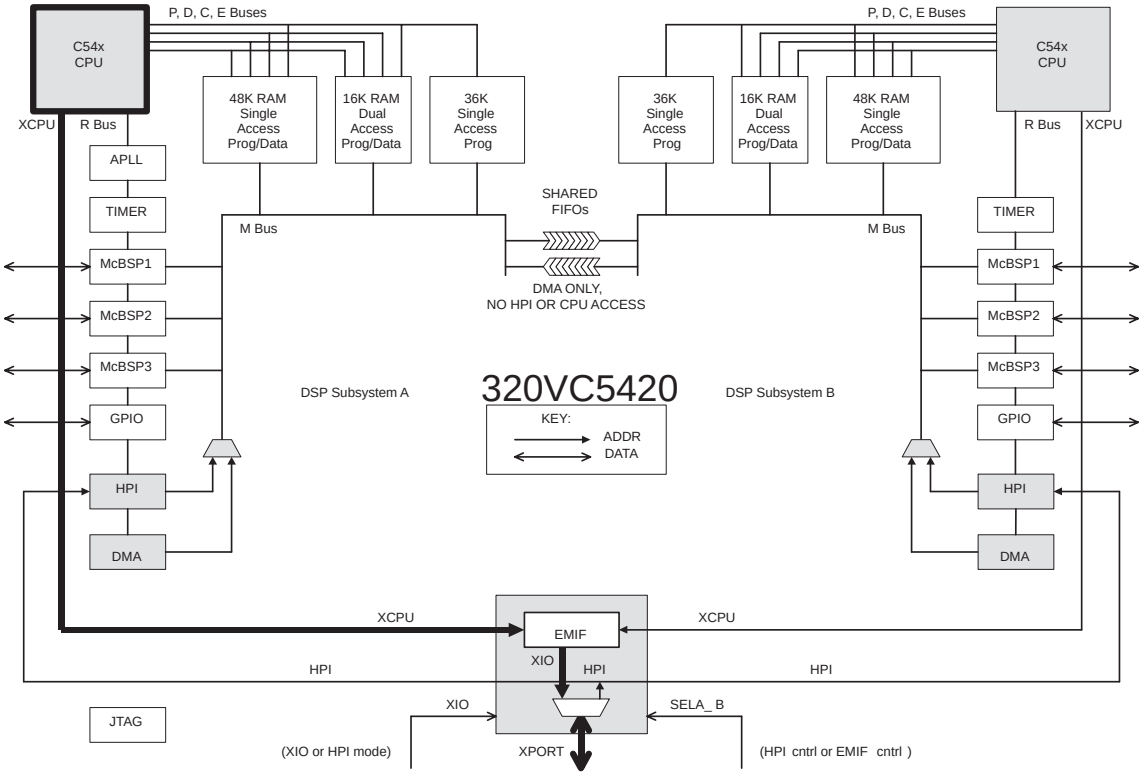

Figure 5. ’C5420 External Transfers Driven by the CPU

3.2 Transfers Driven by the DMAC

The ’C5420 DMAC is primarily intended to offload the CPU from transferring data between internal peripherals and internal memory. The DMAC can also move blocks of data within the internal memory or across the two subsystems through a set of internal FIFOs (8 deep, one for each direction). The DMAC can simultaneously service and keep track of up to 6 separate transfers, each representing a different channel. Every channel is assigned a set of memory mapped registers that fully define all parameters needed to accomplish one transfer.

Before DMA transfers can take place, the CPU has to first initialize and turn on each of the active channels by programming a set of memory mapped registers representing that channel. The DMAC registers are memory mapped using a register sub-addressing scheme, allowing a large number of registers to be accessed using a small number of memory mapped addresses. Once enabled, the DMA channels typically respond to peripheral interrupts to move data elements as soon as they are available inside the peripherals for reads, or as soon as the peripheral is ready to accept new data for writes. The DMAC uses a minimum of 4 CPU clock cycles to transfer one element of data across the M bus (2 cycles to read from the source address, and 2 cycles to write to the destination address). DMA transfers can be occasionally affected by the HPI, due to the fact that the HPI uses the same M bus to transfer data into and out of the DSP under host control. HPI interruptions to the DMA are typically infrequent due to the 6-7cycles/transfer maximum performance of the HPI. This means that the DMA traffic across the M bus can be interrupted at most once every 6-7 cycles, depending on when the asynchronous HPI cycle is initiated relative to the internal DSP clock.

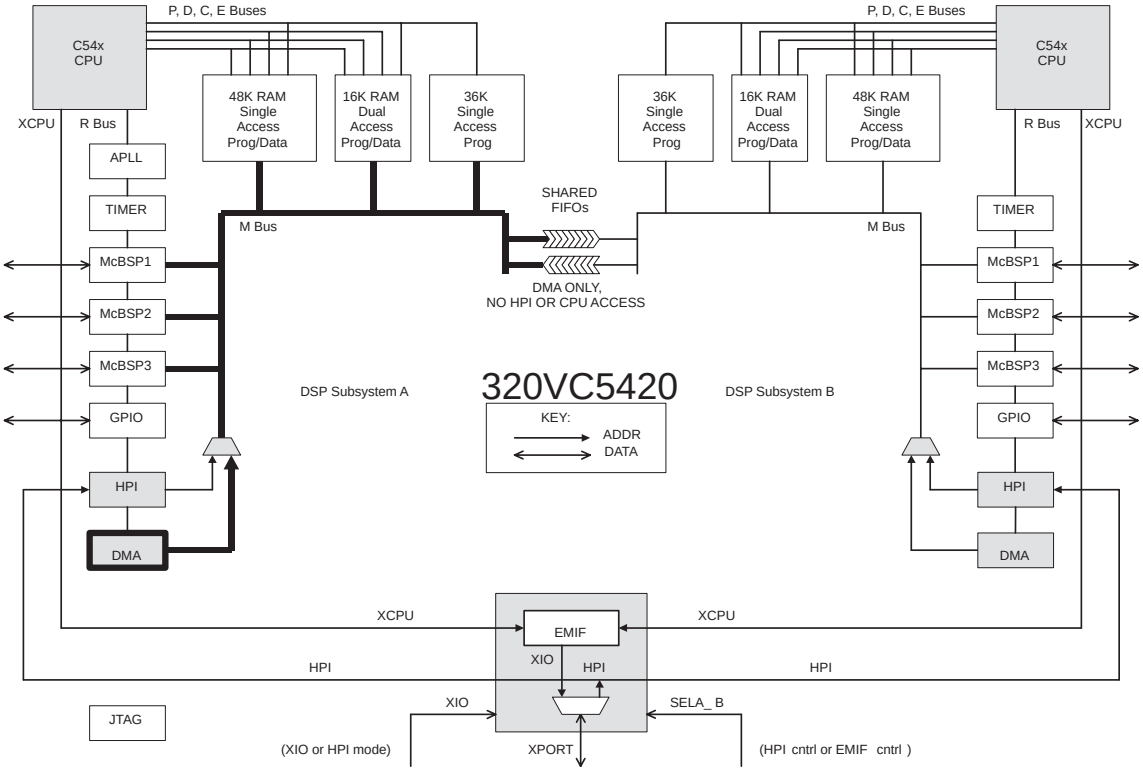

Figure 6. ’C5420 Transfers Driven by the DMAC

3.3 Host Driven HPI Transfers

The ’C5420 Host Port Interface allows an external host CPU to directly access a portion of the DSP’s memory map through the XPORT block, when the XPORT is configured in HPI mode (XIO pin =0). With two DSP subsystems internally connected to a single XPORT, the Host uses the SELA/B pin to activate a single subsystem for subsequent transfers. Since the HPI and DMAC share the same M bus to access internal memory and peripherals, the performance of the HPI transfers can be affected by the current DMAC activity. The best case HPI performance occurs when the DMAC is not active, in which case the host can access internal memory at a rate of at least one 16-bit transfer every seven CPU cycles. With one or more DMAC 16-bit channels active, the HPI throughput becomes at least one transfer every 10 cycles. With one or more DMAC 32-bit channels active, the HPI throughput becomes at least one transfer every 14 cycles. The HPI has the highest priority when competing with CPU or DMAC for access to the same internal memory location. For that reason even as a higher priority DMA channel can lock out a lower priority channel, DMA transfers can never lock out the HPI transfers. The HPI can access the same program and data spaces that the DMAC can, but not the I/O portions of the DMA memory map that control access to the two Shared FIFOs connecting the subsystems. The HPI has no method of distinguishing between program, data or I/O spaces, which is required for access to the Shared FIFOs.

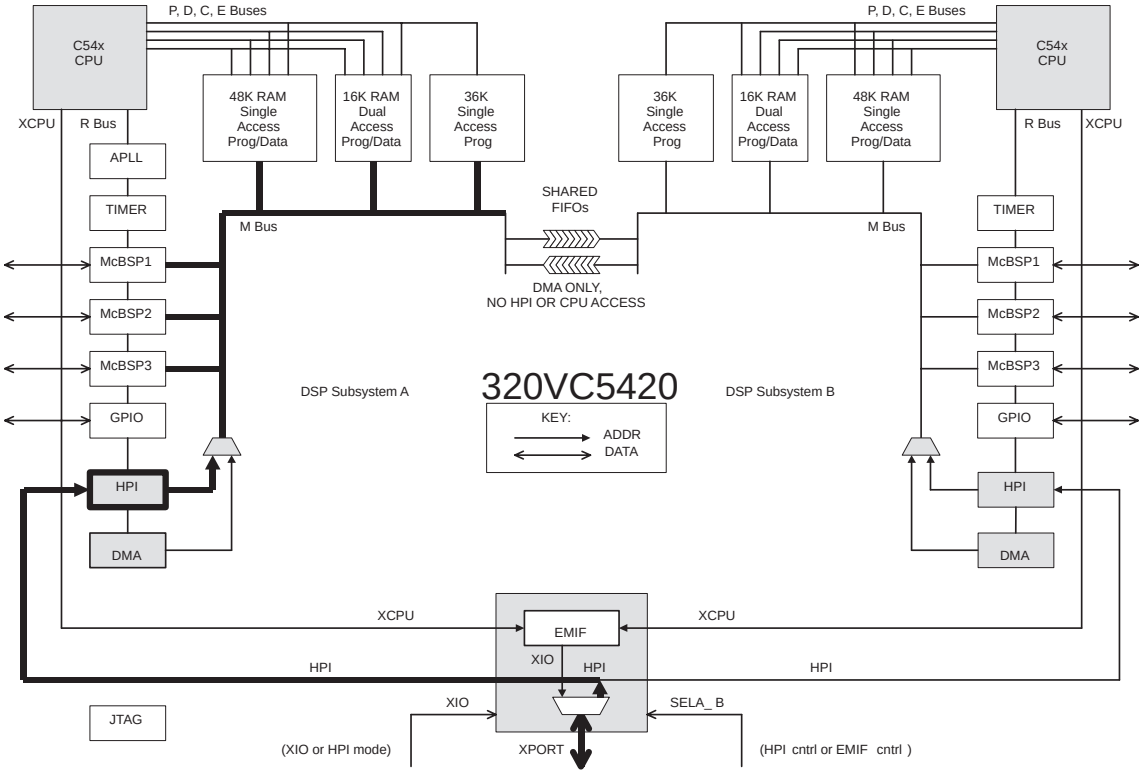

Figure 7. ’C5420 Host Driven HPI

4 TMS320VC5421 I/O Transfers

The ’C5421 I/O transfers are driven by the CPU, on-chip DMAC or the external host processor across the Host Port Interface (HPI). Each subsystem’s CPU can access its own internal memory and on-chip peripherals plus external devices across the XPORT interface. Each subsystem’s CPU can also fetch program words from the internal program RAM block shared by both CPUs. The DMAC Controller can perform internal transfers, external transfers, internal to external and external to internal transfers. In addition to internal transfers within each subsystem, the DMA controllers can also perform transfers across the subsystems using one of two methods. The first method uses internal shared FIFOs (8 deep, one for each direction) for indirect connection between subsystems where each subsystem only sees its own end of the FIFO. The second method allows direct addressing of one subsystem’s memory map from the other subsystem, for either source or destination portions of the transfer. A host can only access the internal memory of the ’C5421 through the HPI. The HPI Controller located in one subsystem can reach internal memory and peripherals in its own local subsystem and the other subsystem as well. HPI Controllers have no access to the Shared FIFOs connecting the two subsystems.

4.1 Transfers Driven by the CPU

The CPU transfers are triggered by CPU program fetches from program space and instruction operand loads/stores from and to data or I/O spaces of the memory map. Depending on the address location within the memory map, the CPU initiated transfers can be internal or external to the DSP. While most internal CPU transfers are contained within one subsystem, internal program fetches can also access blocks of Shared program RAM that is also available to the CPU of the other subsystem. The internal transfers use the high performance P, D, C and E buses to access internal RAM and a slower R bus to access on-chip peripherals. The CPU of each subsystem uses a dedicated interface to access external memory or devices through a single common EMIF interface. In addition to CPU transfers, the EMIF controller also supports DMA transfers involving external sources or destinations.

4.1.1 Internal CPU Transfers Local to a Subsystem

The ’C5421 internal transfers initiated by the CPU include internal memory program/data accesses and peripheral initialization/servicing. The CPU accesses internal memory with a dedicated set of high performance P, D, C and E buses, intended for simultaneous transfers of up to one instruction, two operands and a single result. The CPU accesses the on-chip peripherals using the R bus. While the CPU is capable of directly accessing peripheral data, in most applications it is only used to initialize and control the peripherals. Offloading the peripheral data transfers to the DMAC reduces the overhead associated with interrupt context switches and frees the CPU to spend more cycles processing data.

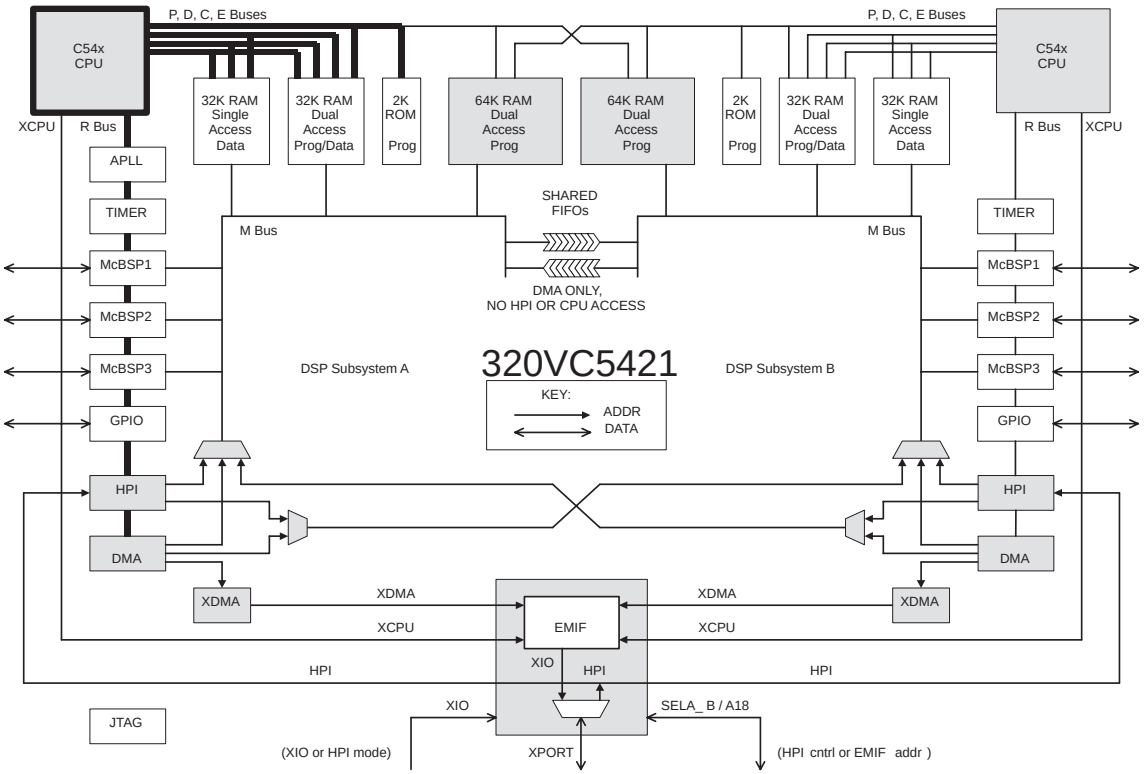

Figure 8. ’C5421 Internal CPU Transfers Local to a Subsystem

4.1.2 Internal CPU Program Fetches from Shared PRAM

Unlike the ’C5420, the ’C5421 features a 128K word block of Shared Dual Access Program RAM that is accessible to both subsystems. Since this memory is shared by the subsystems, the amount of the total available on-chip Program space can be effectively doubled if the applications running on both subsystems are driven with the same code.

Both CPUs can simultaneously access program instructions from any portion of the Shared PRAM. The Shared Program RAM can also be written to or read from by the DMACs of the two subsystems. The two DMA controllers access the bottom 64K words of the PRAM with the subsystem A M-bus, and the top 64K words with the subsystem B M-bus. There is no provision for the CPU to write to the PRAM block.

To the CPU of each subsystem, the 128K words of shared Program RAM appear as 4 32K word memory segments – 0, 1, 2 and 3 located in the upper half of the CPU memory map Program Pages 0, 1, 2 and 3, respectively. To the DMAC of each subsystem, the 128K word of shared Program RAM appear as two 64K word memory segments inside the DMA memory map Program Pages 1 and 3, respectively.

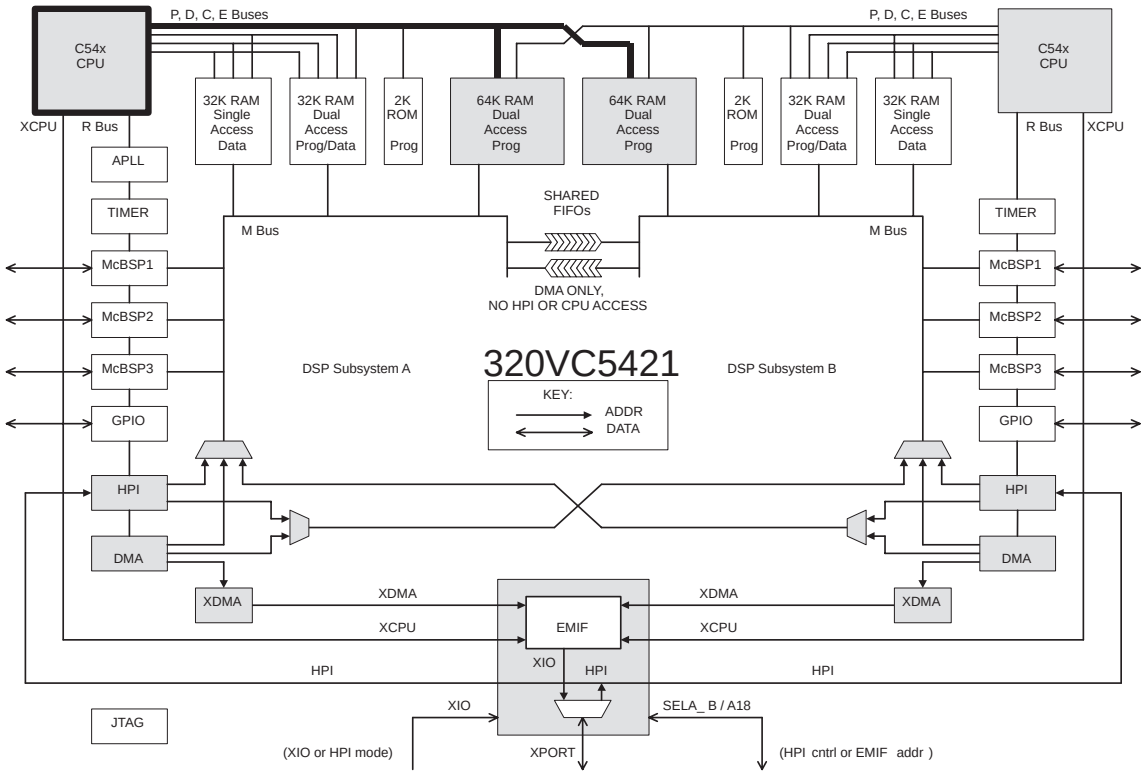

Figure 9. ’C5421 Internal CPU Program Fetches from Shared PRAM

4.1.3 External CPU Transfers

External CPU Transfers are triggered by the CPU fetching program from an external address space, or by executing load or store instructions involving external data or I/O spaces of the memory map. The CPU access to external memory across the XPORT is allowed only when the XIO pin is set to 1, indicating EMIF mode. (XIO=0 configures the XPORT in the HPI mode, and any attempt to access external memory by the CPU or DMAC is ignored for writes, and loads undefined data for reads). Since a single EMIF has to be shared by both subsystems of the ’C5421, only one subsystem can drive external transfers across EMIF at any one time. While the DMAC transfers from both subsystems are automatically interleaved inside the EMIF on per cycle basis, the arbitration involving external CPU transfers relies on software handshaking protocol involving 3 bits of the GPIO register – CORE_SEL, XIO_REQ and XIO_GRANT.

Before issuing a transfer, each CPU needs to verify the XIO ownership by checking the XIO_GRANT bit of the GPIO register. A “1” value of the XIO_GRANT bit means that the requesting CPU already owns the XIO and is free to start external transfers without further arbitration. A “0” value requires the requesting CPU to assert the XIO_REQ bit in the GPIO register and poll the XIO_GRANT bit until the CPU Arbiter inside EMIF asserts it following the release of XIO by the other CPU. The requesting CPU can now perform the transfer and then complete the handshake by de-asserting the XIO_REQ bit. Each CPU should always de-assert the XIO_REQ bit at the completion of each transfer in order to enable the future XIO requests from the other CPU. Another bit of the GPIO register, CORE_SEL identifies to the software which CPU is executing the program. Software referencing the GPIO register, executing on CPU A will return a value of “0” when reading the CORE_SEL bit of side A’s GPIO. Same software, executing on CPU B will return a value of “1” when reading the CORE_SEL bit of the side B’s GPIO register.Once a particular subsystem CPU has established the ownership of the EMIF for external I/O, these transfers can be affected by the on-chip DMA external transfer activity originating in either of the two subsystems. See the EMIF Arbitration chapter for details.

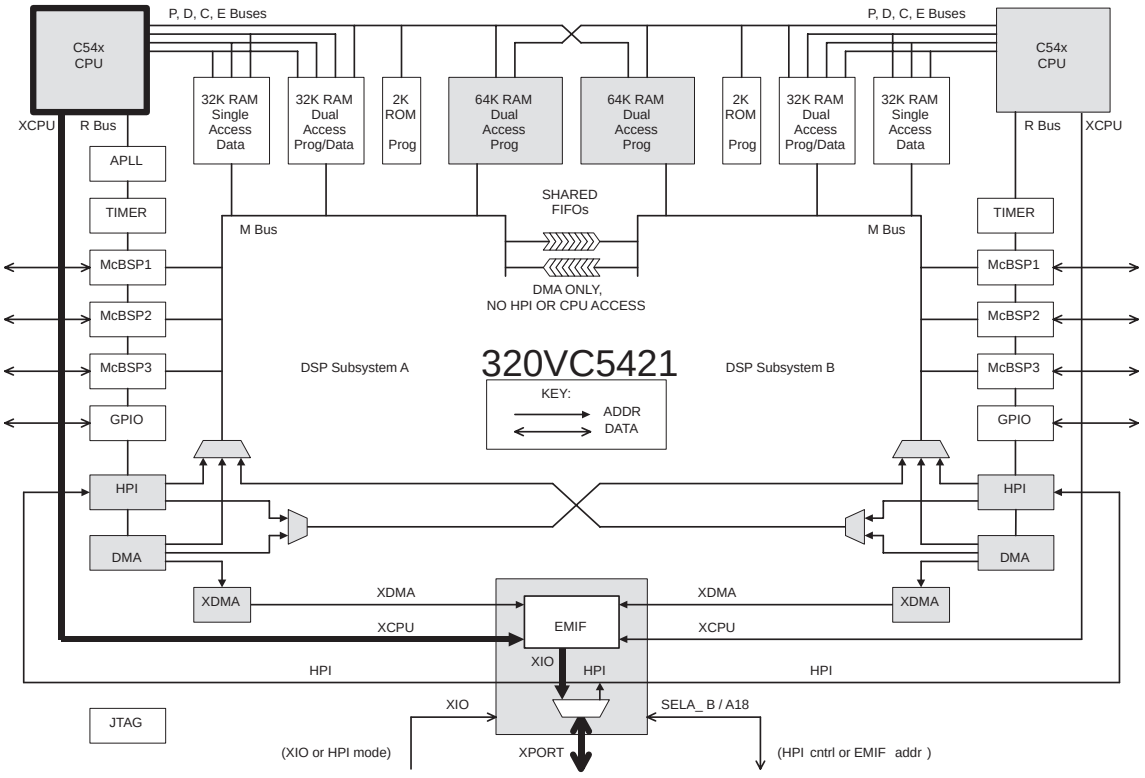

Figure 10. ’C5421 External Transfers Driven by the CPU

4.2 Transfers Driven by the DMAC

The ’C5421 DMA controller inside each subsystem performs data transfers independently of the CPU activity using a dedicated internal bus (the M bus). DMA transfers may be organized in block/frame/element structures with each element transfer broken down into source read and destination write components. While the ’C5420 DMA transfers are only limited to internal sources and destinations on the local M bus, the ’C5421 transfers can also access external program/Data and I/O spaces through the shared XPORT interface. And, while the ’C5420 DMA cross transfers between cores were limited to indirect moves across bi-directional FIFOs, the ’C5421 inter-core DMA transfers can directly access all resources tied to the DMA M buses inside both subsystems, including RAM blocks, Peripheral Registers, and Memory Mapped Registers.

The ’C5421 DMA controllers are primarily intended to offload the local CPUs from transferring data between internal peripherals, internal memory, external memory/devices and across subsystems. The DMAC can simultaneously service and keep track of up to 6 separate transfers, each representing a different channel. Every channel is assigned a set of memory mapped registers that fully define all parameters needed to accomplish one transfer.

Before DMA transfers can take place, the CPU has to first initialize and turn on each of the active channels by programming a set of memory mapped registers representing that channel. The DMA registers are memory mapped using a register sub-addressing scheme, allowing large number of registers to be accessed using a small number of memory mapped addresses. Once enabled, the DMA channels typically respond to peripheral interrupts to move data elements as soon as they are available inside the peripherals for reads, or as soon as the peripheral is ready to accept new data for writes (no synchronization is needed for memory-to-memory transfers). The DMAC uses a minimum of 4 CPU clock cycles to transfer one element of data across the M-bus (2 for read and 2 for write) inside a subsystem. External and cross-subsystem transfers require additional cycles. DMA transfers can be occasionally affected by the HPI, due to the fact that the HPI uses the same M buses to transfer data into and out of the DSP under host control. HPI interruptions to the DMAC are typically infrequent due to the 6-7 cycles per transfer maximum performance of the HPI. This means that the DMA traffic across the M bus can be interrupted for 2 cycles at most once every 6-7 cycles, or less.

4.2.1 Internal/External Addressing

Each DMA element transfer for a given channel consists of a single source read operation followed by a single write to the destination. Whether the source/destination locations are internal or external to the ’C5421 device depends on the settings of the SLAX and DLAX bits in the Transfer Mode Control Register (DMMCR) for each DMA channel. The SLAX bit set to 0 indicates internal source, while SLAX bit set to 1 represents an external source. Similarly, DLAX bit set to 0 indicates external destination, while DLAX bit set to 1 represents an external destination, for a given DMA transfer. These bits must be manually set by software prior to initiating a DMA transfer.

The internal or external DMA sources and destinations can involve program, data or I/O spaces of the memory map. The DMA Transfer Mode Control Register (DMMCR) 2-bit register fields DMS and DMD define whether the DMA transfer source or destination for a particular channel belongs in the program, data or I/O spaces of the memory map. program and data space sources and destinations are also grouped into extended program and data pages. Program pages for internal or external DMA transfers are encoded in the Source/Destination Program Page Address Registers DMSRCP/DMDSTP. Data pages for external transfers are encoded in the Source/Destination Extended Data Page Registers XSRCDP/XDSTDP. Internal and external DMA I/O spaces do not require paging as the corresponding address do not exceed 64K.

For external transfers, the data, program and I/O DMA reads or writes are identified by the corresponding DS, PS and IS XPORT strobes. The 8 possible banks of external program or data transfers are represented by the 3 high order bits of the XPORT address bus (bits 18–16).

For internal transfers, irrespective of the page settings in the XSRCDP and XDSTDP registers, all reads or writes from/to data space are contained within a single 64K word page local to the subsystem containing the DMA controller performing the transfer. Internal program reads and writes, on the other hand, can involve local memory/peripherals, memory/peripherals of the other subsystem or shared PRAM – depending on which one of the 4 internal program pages has been chosen in the DMSRCP and DMDSTP registers for source and destinations. For program transfers involving the shared PRAM, the DMAC always accesses the bottom half of the PRAM using the M-bus inside subsystem A, and the top half using subsystem B M-bus.

4.2.2 Source Addressing Flow

The source portion of a single DMA element transfer identifies a single read location within the DMA memory map by combining the 16-bit DMA source address with the current settings of SLAX and DMS fields of the DMMCR register and the contents of the program and data extended page addresses stored in DMSRCP and XSRCDP registers. While each channel contains its own dedicated SLAX and DMS fields, the Program and Data Extended Page Registers apply to all 6 channels of the DMAC.

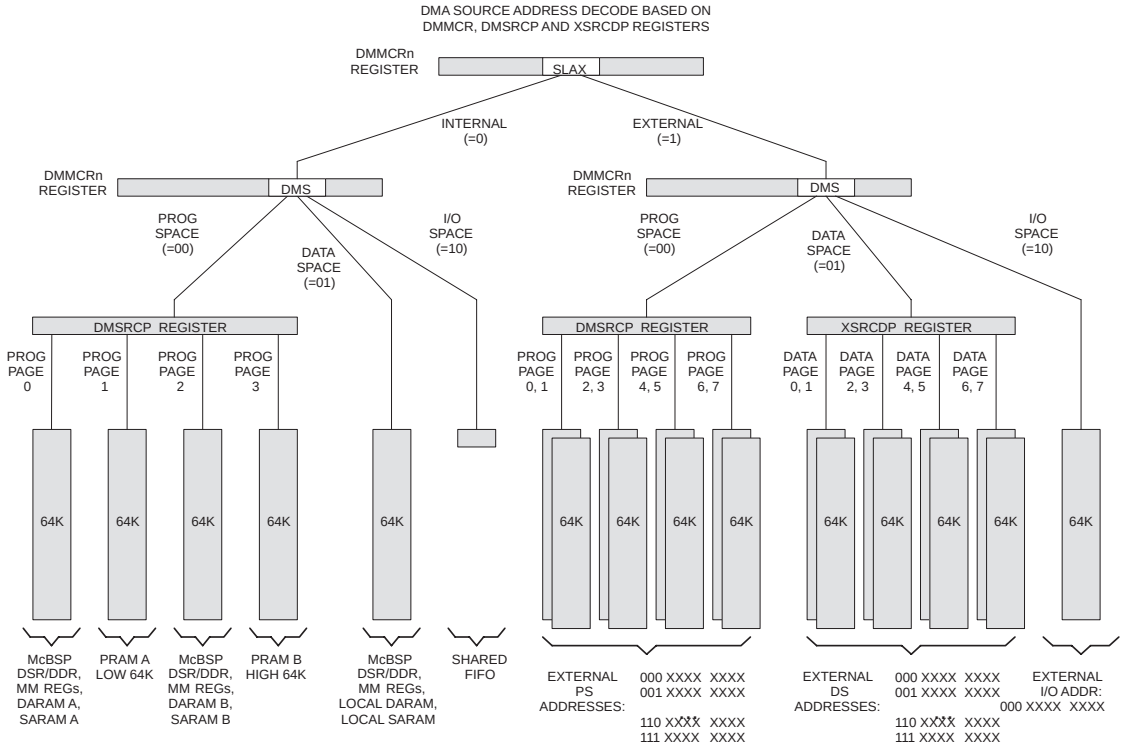

Figure 11. ’C5421 DMA Transfer Element Source Addressing

The single bit SLAX field identifies the source as internal or external to the ’C5421.

The 2-bit DMS field identifies the source space as program, data or I/O read.

If the source exists in the external I/O space, its 16-bit address is presented without any modification on the 16 low order address pins of the XPORT, with the 3 high order XPORT address pins (18–16) set to 0.

If the source exists in the Internal I/O space, it always represents the Shared FIFO, regardless of the actual address. All DMAC reads from any Internal I/O address of the same subsystem transfer a single word from the same Shared FIFO.

If the source exists in the Internal data space, its 16-bit address points to all RAM, Peripheral and Memory Mapped Register spaces local to the subsystem containing the DMA controller that is reading the source.

If the source exists in the external data space, the XSRCDP register identifies one of 8 possible External Extended Data Pages, each containing 64K words of data. The 8 External data Pages are encoded in the 3 high order address bits of the XPORT interface and are identified as data by the active external Data Strobe (DS).

If the source exists in the external program space, the DMSRCP register identifies one of 8 possible External Extended Program Pages, each containing 64K words of program. The 8 External Program Pages are encoded in the 3 high order address bits of the XPORT interface and are identified as program by the active external Program Strobe (PS).

If the source exists in the Internal program space, the DMSRCP register identifies one of 4 possible Internal Extended Program Pages, each containing 64K words of program. The 4 Internal Program Pages together represent all of the addressable internal space of the two ’C5421 subsystems that can be reached by both M buses. Internal Program Pages 0 and 2 each point to all RAM, Peripheral and Memory Mapped Register spaces of subsystem A and subsystem B, respectively. Internal Program Pages 1 and 3 each point to one-half of the 128K Shared Program RAM.The bottom 64K words are accessed using the subsystem A M-bus, while the high 64K words are accessed using the subsystem B M-bus.

Note, that the terms DMA program space and DMA data space do not necessarily identify the same program and data spaces as are visible to the CPU. In the DMA context, the terms program and data spaces are simply used to identify different ways by which the DMAC can reach source and destination addresses.

4.2.3 Destination Addressing Flow

The destination portion of a single DMA element transfer identifies a single write location within the DMA memory map by combining the 16-bit DMA destination address with the current settings of DLAX and DMD fields of the DMMCR register and the contents of the program and data extended page addresses stored in DMDSTP and XDSTDP registers. While each channel contains its own dedicated DLAX and DMD fields, the Program and Data Extended Page Registers apply to all 6 channels of the DMAC.

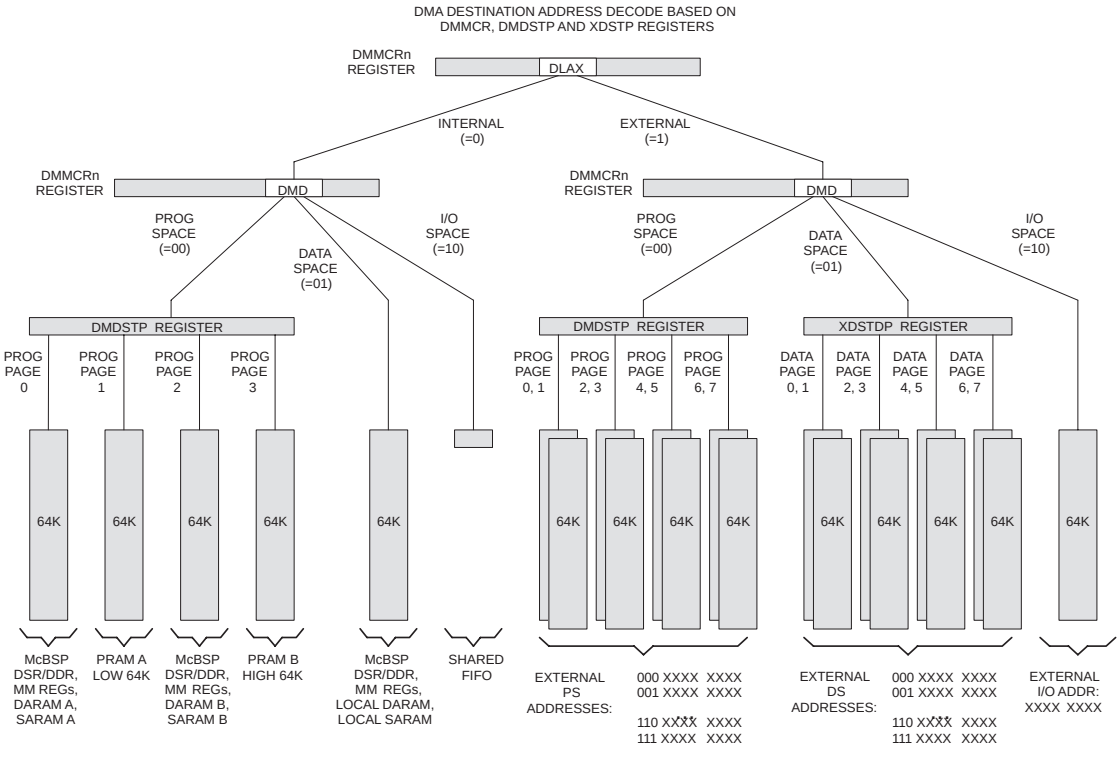

Figure 12. ’C5421 DMA Transfer Element Destination Addressing

The single bit DLAX field identifies the destination as internal or external to the ’C5421.

The 2-bit DMD field identifies the destination space as program, data or I/O read.

If the destination exists in the external I/O space, its 16-bit address is presented without any modification on the 16 low order address pins of the XPORT, with the 3 high order XPORT address pins (18–16) set to 0.

If the destination exists in the Internal I/O space, it always represents the Shared FIFO, regardless of the actual address. All DMAC writes to any Internal I/O address of the same subsystem transfer a single word to the same Shared FIFO.

If the destination exists in the Internal data space, its 16-bit address points to all RAM, Peripheral and Memory Mapped Register spaces local to the subsystem containing the DMA controller that is writing to the destination.

If the destination exists in the external data space, the XDSTDP register identifies one of 8 possible External Extended Data Pages, each containing 64K words of data. The 8 External Data Pages are encoded in the 3 high order address bits of the XPORT interface and are identified as data by the active external Data Strobe (DS).

If the destination exists in the external program space, the DMDSTP register identifies one of 8 possible External Extended Program Pages, each containing 64K words of program. The 8 External Program Pages are encoded in the 3 high order address bits of the XPORT interface and are identified as program by the active external Program Strobe (PS).

If the destination exists in the Internal program space, the DMDSTP register identifies one of 4 possible Internal Extended Program Pages, each containing 64K words of program. The 4 Internal Program Pages together represent all of the addressable internal space of the two ’C5421 subsystems that can be reached by both M buses. Internal program Pages 0 and 2 each point to all RAM, Peripheral and Memory Mapped Register spaces of subsystem A and subsystem B, respectively. Internal program Pages 1 and 3 each point to one-half of the 128K Shared Program RAM.The bottom 64K words are accessed using the subsystem A M-bus, while the high 64K words are accessed using the subsystem B M-bus.

Note, that the terms DMA program space and DMA data space do not necessarily identify the same program and data spaces as are visible to the CPU. In the DMA context, the terms program and data spaces are simply used to identify different ways by which the DMAC can reach source and destination addresses.

4.2.4 Internal DMA Transfers Within a Local Subsystem

Local DMA transfers represent the fastest way of moving data within the same subsystem. For local DMA transfers, the source and destination addresses reside on the same M-bus. Local DMA transfers take a minimum of 4 CPU clock cycles to transfer one element of data across the M bus (2 cycles to read from the source address, and 2 cycles to write to the destination address). Local DMA transfers can be occasionally affected by the HPI, due to the fact that the HPI may use the same M bus to transfer data into and out of the DSP under host control. Local DMA transfers can also be affected by the DMA controller of the other subsystem performing a cross-subsystem transfer.

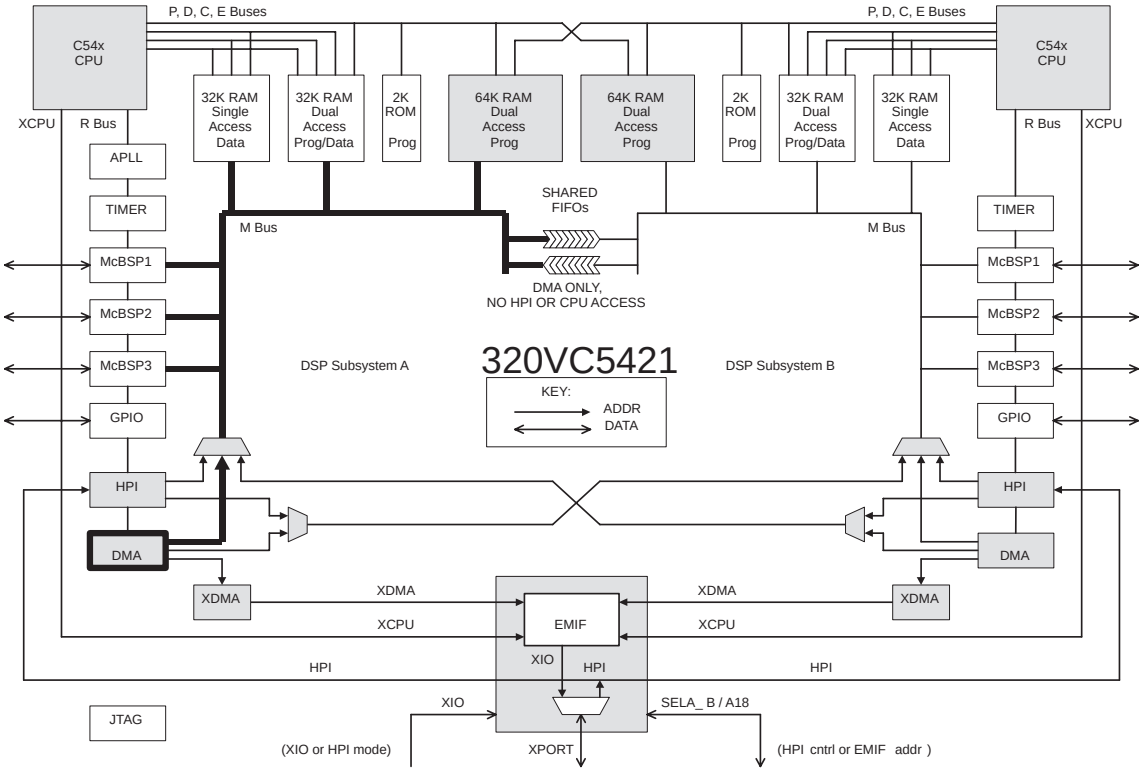

Figure 13. ’C5421 Internal DMA Transfers Within a Local Subsystem

4.2.5 Internal DMA Transfers Involving the Other Subsystem

In addition to accessing peripherals/memory/FIFOs within its own subsystem, the DMA controller can also perform remote transfers involving resources residing in the other subsystem. For example, the DMAC in subsystem A can transfer a block of data from a McBSP in subsystem B to the DARAM in subsystem B. Although it is more efficient to use the local DMA controller to perform local data transfers, some applications may require the DMAC of one subsystem to perform transfers on the M-bus of the other subsystem, for example, if all of the DMA channels inside the other subsystem are already committed to other transfers. These kinds of remote internal transfers take longer to perform than local transfers and can be further slowed down by the local DMA and HPI traffic.

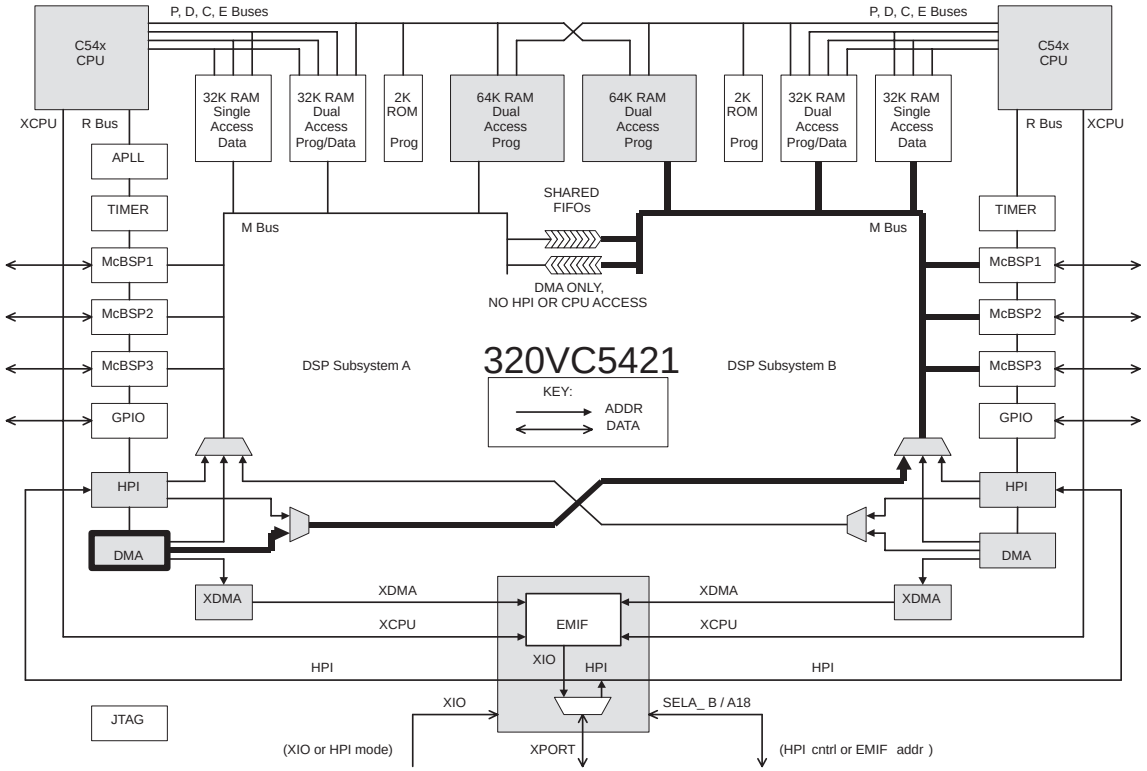

Figure 14. ’C5421 Internal DMA Transfers Involving the Other Subsystem

4.2.6 Internal DMA Transfers Across Subsystems

DMA transfers across subsystems read the source elements from a location on one subsystem and write them into a destination located in the opposite subsystem. This type of communication is typically necessary if both subsystems are simultaneously working on the same task. The DMA controller performing such a transfer can reside in either of the two subsystems. The portion of the transfer residing in the same subsystem as the DMA controller takes a minimum of 2 cycles to perform. The portion of each element transfer residing in the other subsystem takes longer than the local portion. Either local or remote portion of each DMA transfer across subsystem can also be delayed by the activity of the other DMA or HPI. DMA transfers across subsystems effectively connect the M-buses and all of their associated resources on both sides of the chip. Although Shared FIFOs can indirectly facilitate some types of inter-subsystem communication, cross-subsystem DMA transfers represent the most efficient and direct method of transferring data from one subsystem to another.

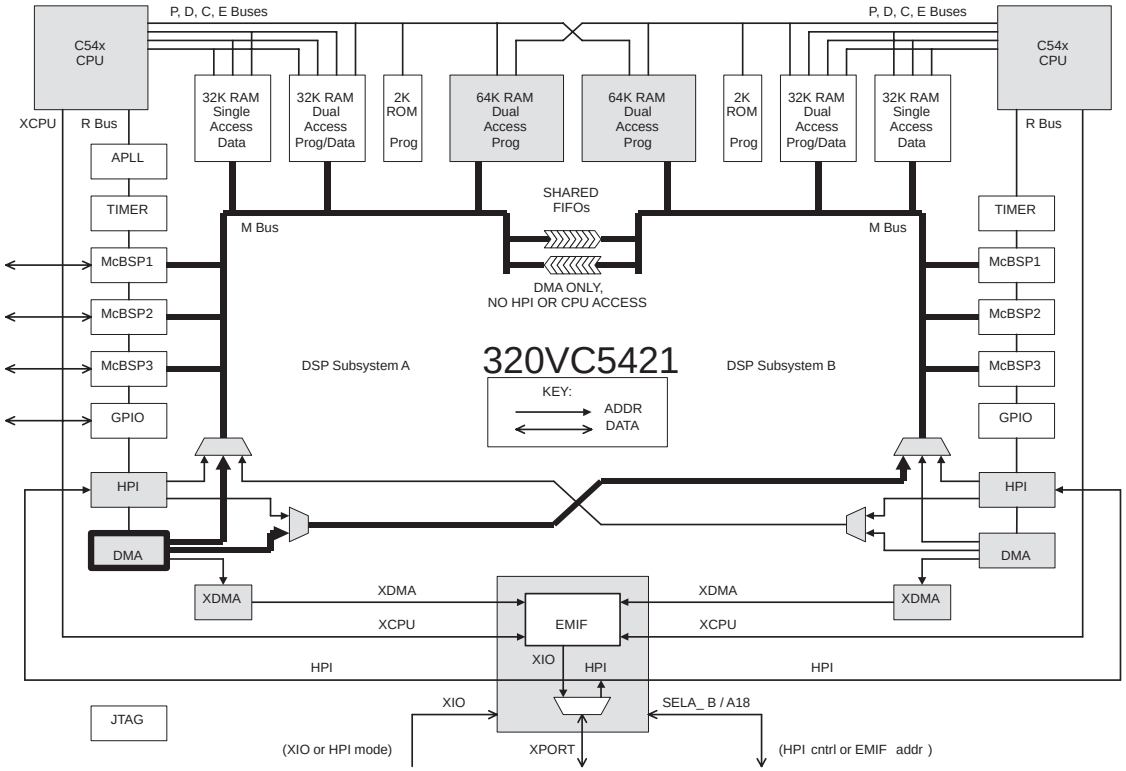

Figure 15. ’C5421 Internal DMA Transfers Across Subsystems

4.2.7 Internal/External DMA Transfers

While the ’C5420 DMA controller is restricted to internal transfers only, the ’C5421 DMA controllers can also perform external transfers. Both of the ’C5421 DMA controllers feature a path to the external bus that allows them to perform transfers between on-chip RAM/peripheral/FIFO resources and external sources or destinations. The DMA controllers can also perform transfers where both sources and destinations are external. Moving blocks of data between internal and external locations is necessary for applications that require more memory than what is available on the DSP. In such cases the DMAC may be loading the next application frame from external memory and storing the previous results in memory, while the CPU is computing the current frame. The CPUs and DMA controllers of both subsystems access external resources through a single XPORT interface that is also shared by the two Host Port Interface Controllers. The XIO input pin determines whether the XPORT interface is a slave HPI port allowing the external host to access internal DSP resources, or whether it is a master interface driving the external system resources with CPU or DMAC generated bus cycles. Thus, while the internal portion of the internal/external transfer may take as few as 2 cycles, the external portion will take longer, depending on the status of the other DMA controller and external activity of CPU A and CPU B. DMA access to external memory across the XPORT is allowed only when the XIO pin is set to 1, indicating the XIO mode. (XIO=0 configures the XPORT in the HPI mode, and any attempt to access external memory by the DMAC is ignored for writes, and loads undefined data for reads).

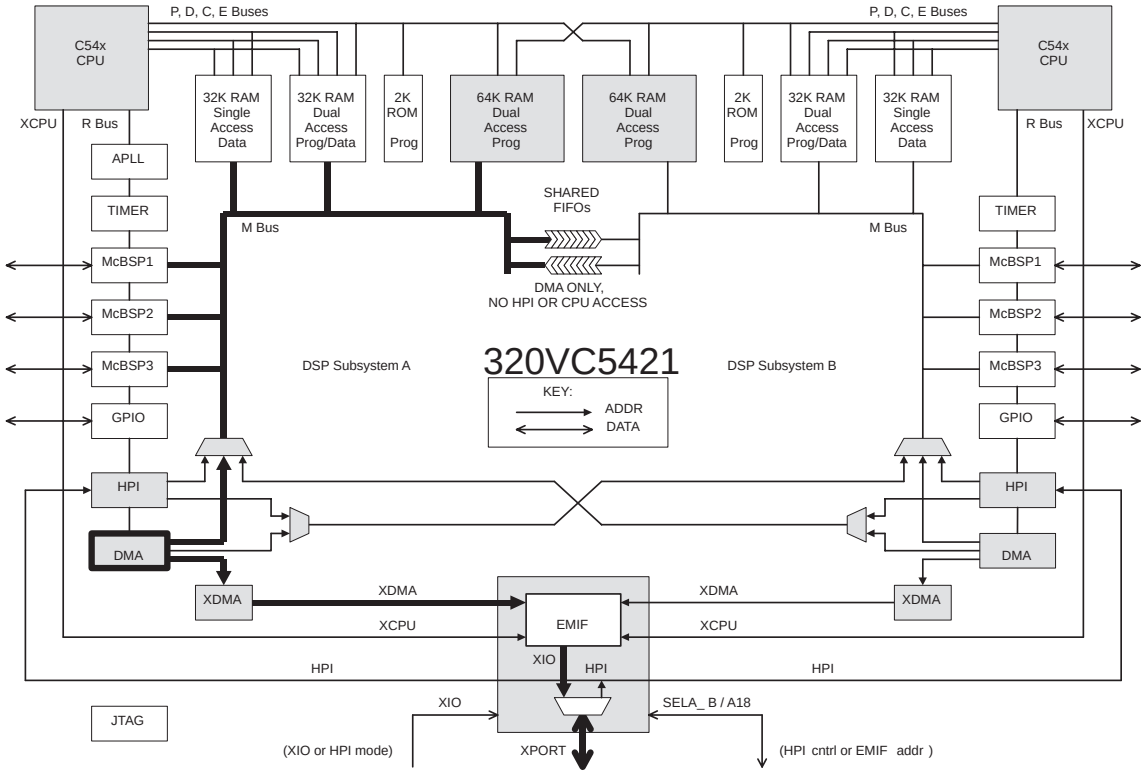

Figure 16. ’C5421 Internal/External DMA Transfers

4.2.8 Internal/External DMA Transfers Across Subsystems

In addition to driving transfers between the external resources and the local subsystem sources or destinations, each DMAC can also connect external resources with the internal resources located inside the opposite subsystem. In such cases, the internal part of each transfer will take longer then if the internal sources or destinations were located in the subsystem local to the DMAC performing the transfer.

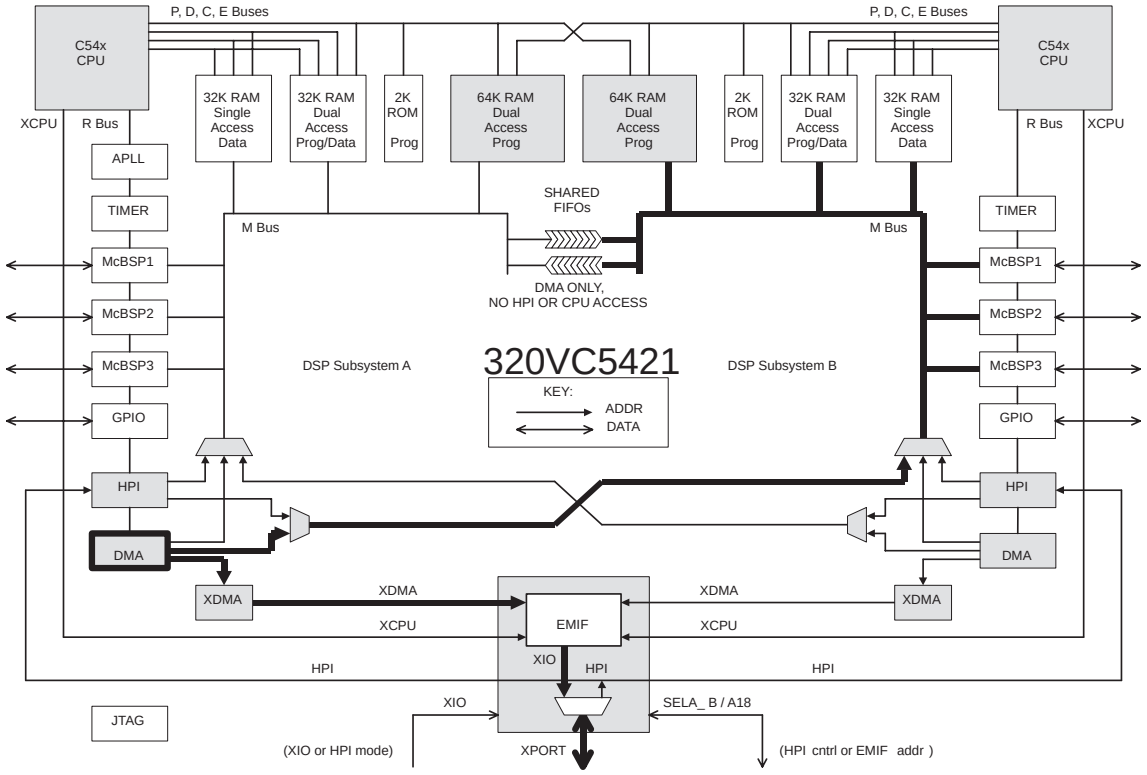

Figure 17. ’C5421 Internal/External DMA Transfers Across Subsystems

4.2.9 External DMA Transfers

In some applications it may be necessary for the DSP to perform external system house-keeping transfers without actually engaging any of the potential resources inside the DSP. In such cases, both the source and destination of each element transfer are external to the DSP and are not exercising the DSP M-buses. Even though these DMA transfers involve external sources and destinations, each element read and write still has to travel across the common XPORT to and from the DMA controller. Such transfers are subject to delays if the other DMAC, CPU A, CPU B are competing for the same pins. See the following sections for detailed arbitration analysis of the EMIF module.

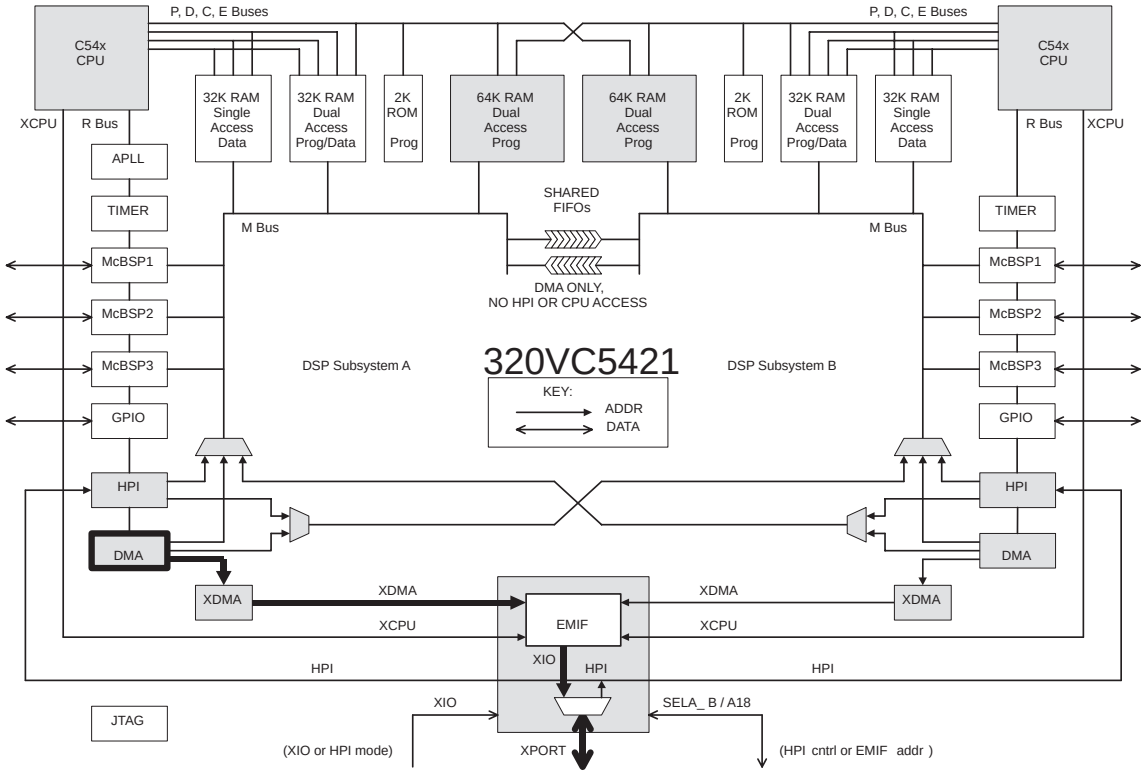

Figure 18. ’C5421 External DMA Transfers

4.3 Host Driven HPI Transfers

The ’C5421 Host Port Interface allows an external host CPU to directly access a portion of the DSP’s memory map through the XPORT block, when the XPORT is configured in HPI mode (XIO pin =0). With two DSP subsystems internally connected to a single XPORT, the host uses the SELA/B pin to activate a single subsystem for subsequent transfers.Host Driven HPI Transfers can engage DSP’s RAM and peripheral resources local to the subsystem containing the active HPI Controller, or can reach the other subsystem, depending on where the transfer addresses are positioned within the HPI memory map.

4.3.1 HPI Internal Addressing

The host CPU can access all internal M-bus peripherals and RAM of both subsystems (same as DMA internal space) except the Shared FIFOs. The host CPU cannot access external DMA space, since the XPORT is dedicated to HPI and the EMIF is disconnected from the XPORT.

During HPI transfers the host has access to four 64K word pages of internal DSP resources representing subsystem A and B RAM and Peripherals. All 4 pages can be reached through subsystem A HPI Controller or subsystem B HPI Controller, as determined by the SELA/B input pin. The top 2 bits of an 18-bit HPI address identify the page, while the remaining bottom 16 bits point to an address within a given page.

HPI page 0 points to subsystem A RAM blocks, peripherals and memory mapped registers. HPI page 2 points to subsystem B RAM blocks, peripherals and Memory Mapped Registers. HPI pages 1 and 3 represent the 128K words of on-chip Shared PRAM. The bottom 64K words are routed through subsystem A M-bus and the top 64K words are routed through subsystem B M-bus.

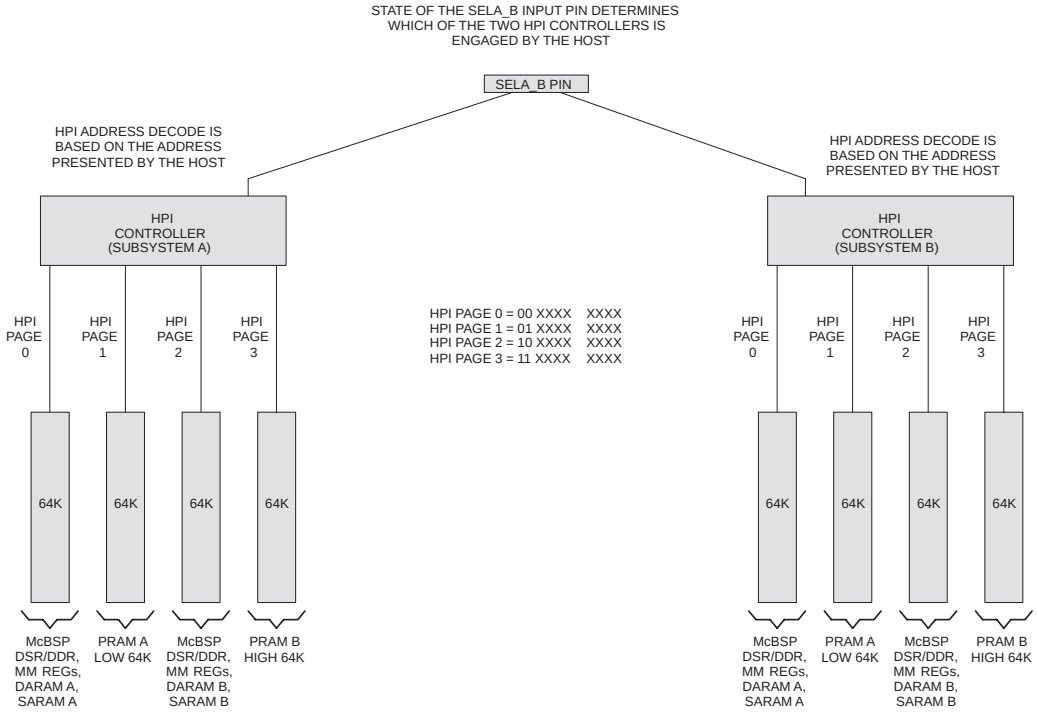

Figure 19. ’C5421 HPI Internal Addressing

4.3.2 HPI Transfers Within a Subsystem

Since the HPI and DMA share the same M bus to access internal memory and peripherals, the performance of the HPI transfers can be affected by the current DMA activity. The best case HPI performance occurs when the DMAC is not active, in which case the host can access internal memory at a rate of at least one 16-bit transfer every seven CPU cycles. The HPI has the highest priority when competing with CPU or DMAC for access to the same internal memory location. For that reason a higher priority DMA channel can lock out a lower priority DMA channel. However, DMA transfers can never lock out HPI transfers. The HPI can access the same program and data spaces that the DMAC can, but not the I/O portions of the DMA memory map that control access to the two shared FIFOs connecting the subsystems.

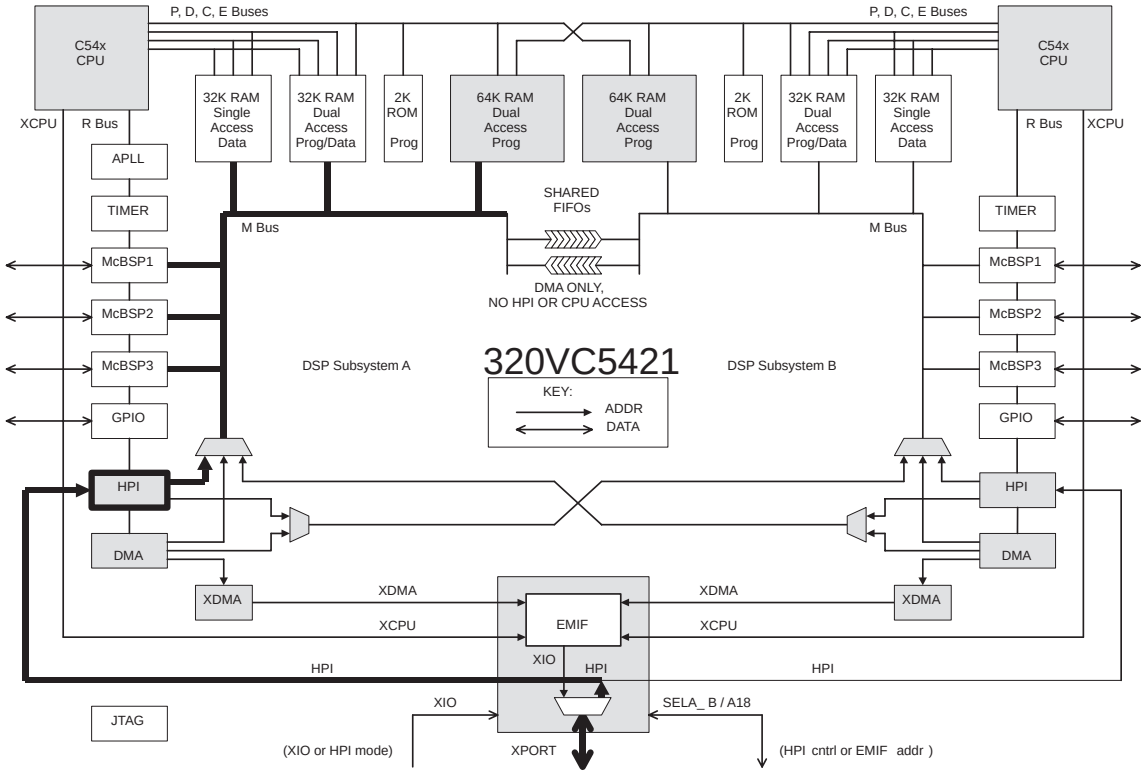

Figure 20. ’C5421 HPI Transfers Within a Subsystem

4.3.3 HPI Transfers Across Subsystems

In addition to local visibility, the HPI of one subsystem also has access to the RAM and peripherals of the other subsystem. The ability of either HPI controller to reach both subsystems is similar to the internal DMA reach except that the HPI cannot access the shared FIFOs. This flexible internal reach capability of both HPI Controllers combined with the ability of the host to drive the HPI controller of either subsystem via the SELA_B pin effectively enables the host to reach the internal DSP resources of a particular subsystem through two different paths. While typically the subsystem A HPI Controller would be used to access subsystem A’s RAM and peripherals, some applications may use the subsystem A HPI Controller to perform cross-transfers involving subsystem B. This could be the case, for example, if the cycle cost in reconfiguration of HPI B for a short host transfer is higher than the extra cross-delay to access subsystem B from an already configured HPI A.

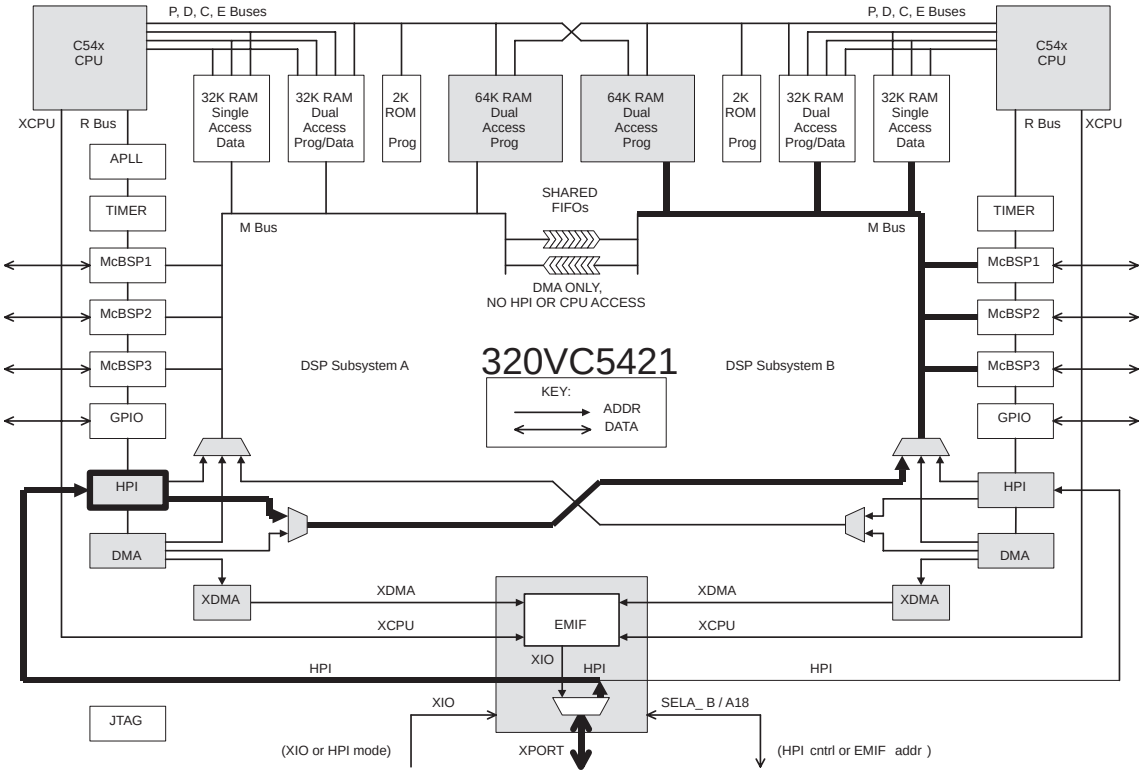

Figure 21. ’C5421 HPI Transfers Across Subsystems

4.4 M-Bus Arbitration

DMAC and HPI Controllers use two M buses to reach internal DSP RAM blocks and peripherals. In addition to local visibility, the DMAC and HPI Controllers of one subsystem can also use the second M bus to reach RAM blocks and peripherals belonging to the opposite subsystem, across the cross-subsystem paths.

Each M bus can be driven with one of 3 requesters – local DMAC, local HPI or a cross-subsystem path. The M bus arbiter rotates between the 3 requests after every element transfer so that all 3 requesters have equal access to the M bus. Depending on the state of the M bus arbiter at the time of the transfer request, a requester may be granted immediate access to the M-bus, or alternately could be forced to wait for one or both of the other two requesters to complete an element transfer first. Each cross-subsystem transfer is driven by the DMAC or the HPI Controller of the other subsystem on a rotating priority basis. The cross-path adds one extra cycle each time the address or data crosses from one subsystem to another. Thus, as compared to 2-cycoe local transfers, the minimum time required for a DMA to perform a cross-subsystem write is 2+1 cycles (one extra cycle for the outgoing address/data), and for a cross-subsystem read is 2+2 cycles (one extra cycle for address and one for returning data). HPI transfers incur additional delays due to communication and synchronization overhead internal to the peripheral and transport delays between the HPI peripheral and the pins. When no DMA channels are active, the host CPU can write to the DSP memory via the HPI at a maximum rate of one 16-bit word every 6-7 DSP cycles.

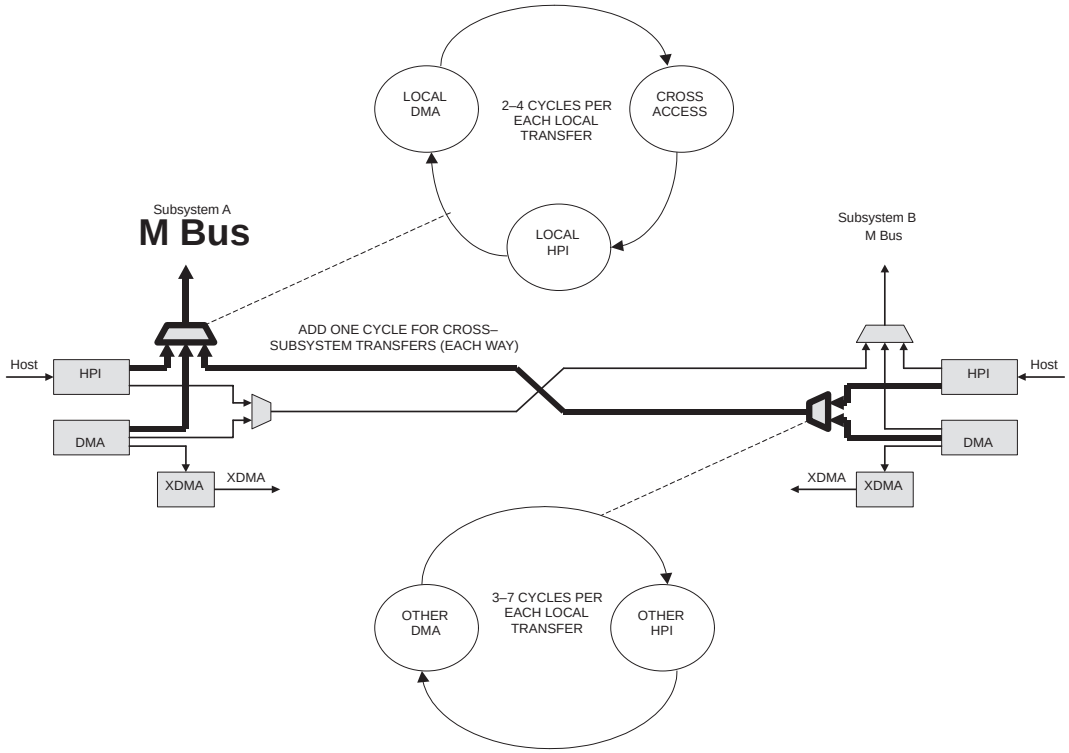

Figure 22. ’C5421 M-Bus Arbitration

4.4.1 Local DMA Transfers

The DMA controller performs data element transfers in two steps – source read and destination write. The best case for the local DMAC to access the M-bus occurs when there is no other activity of the M-bus, implying an idle HPI and cross-subsystem DMA controller. In such cases, the local 16-bit DMA destination writes and source reads each take 2 cycles to complete. The local 32-bit DMA destination writes and source reads each take 4 cycles to complete.

The next best case for a local DMA access occurs when a local HPI Controller is reading or writing to/from a resource on the M-bus (an internal memory location, for example). The local DMA access will be stalled until the current local HPI cycle completes. Depending on when the local DMAC request arrives inside an active local HPI access, it will be stalled for a minimum of 1 cycle and a maximum of 2 cycles before gaining access to the M-bus. Any additional HPI transfers pending at the completion of the local DMA reads or writes cause the M-bus controller to interleave local DMA and local HPI requests on per-element basis. Since the HPI as a whole is much slower than the DMAC, the DMA/HPI interleave will most likely happen at a rate of one HPI transfer every few local DMA transfers.

The next best case for a local DMA access occurs when the cross-subsystem DMAC or HPI Controller is reading or writing to/from a resource on the M-bus. The local DMA access will be stalled until the current cross-subsystem read or write completes. Depending on when the local DMAC request arrives inside an active cross-subsystem access, it will be stalled for a minimum of 1 cycle and a maximum of 7 cycles before gaining access to the M-bus. The 7 cycles represents the longest cross-subsystem access (a 32-bit DMA source read). The 7 cycles break down as follows – 1 cycle for the address to cross between subsystems, 3 cycles to return the first 16 bits and 3 cycles to return the second 16 bits of data. Any additional HPI transfers pending at the completion of the local DMA reads or writes cause the M-bus controller to interleave local DMA and cross-subsystem requests on per-element basis. This means that each additional element of the local DMA transfer will be delayed by the same number of cycles as the first element, when the local DMAC and cross-subsystem DMAC or HPI take turns reading or writing from/to the M-bus.

The worst case for the local DMA accesses occurs when the M-bus is already being shared by both the local HPI and the cross-subsystem DMA transfers. Before gaining access to the M-bus, the local DMAC has to first wait for the local the cross-subsystem DMA transfer to complete and then for HPI transfer to complete before gaining access to the M-bus. The worst case cross-subsystem 32-bit DMA accesses take 6 cycles for destination writes and 7 cycles for the source reads. This delay is added on top of the 2 cycles that the local DMAC has to wait for the local HPI to complete, before gaining access to the M-bus.

4.4.2 Cross-Subsystem DMA Transfers

The DMA controller performs data element transfers in two steps – source read and destination write. The best case for the cross access of the DMAC from one subsystem to the M-bus resources of the other subsystem occurs when there is no other activity of the M-bus, implying an idle local DMAC and HPI. In such cases, the cross subsystem DMAC 16-bit destination writes complete in 3 cycles (1 for address/data to travel across subsystems and 2 for M-bus write cycle), while the 16-bit source reads take 4 cycles (1 for address to travel across subsystems, 2 for M-bus read cycle and 1 for the data to return across the subsystems). The best case 32-bit cross-subsystem DMA transfers add 2 cycles inside the M-bus (32-bit accesses on the 16-bit M-bus take two 2-cycle M-bus reads or writes) and one extra cycle to transfer 32-bit data across the 16-bit bus connecting the subsystems. Adding the cycles yields the best case 32-bit cross-subsystem destination writes of 6 cycles and best case 32-bit source reads of 7 cycles.

The next best case for DMA cross-access occurs when a local DMAC is reading or writing to/from a resource on the M-bus (an internal memory location, for example). The cross-subsystem DMAC trying to get through to the same M-bus will be stalled until the current local DMA cycle completes. Depending on when the other DMAC request arrives inside an active 16-bit local DMA access, it will be stalled for a minimum of 1 cycle and a maximum of 2 cycles before gaining access to the M-bus. If the cross-subsystem DMAC request arrives inside an active 32-bit local DMA access, it will be stalled for a minimum of 1 cycle and a maximum of 4 cycles before gaining access to the M-bus. Additional local DMA transfers pending at the completion of the cross-subsystem reads or writes cause the M-bus controller to interleave local and cross-system accesses on per-element basis. This means that each additional element of the cross-system DMA transfer will be delayed by the same number of cycles as the first element, when the two DMA controllers take turns reading or writing from/to the M-bus.

The next best case for DMA cross-accesses occurs when the M-bus is already being shared by both the local DMAC and the local HPI Controller. Before gaining access to the M-bus, the cross-subsystem DMAC has to first wait for the local HPI to complete and then for the local DMA to complete before gaining access to the M-bus. Once at the M-bus Arbiter, the local HPI takes 2 cycles to read or write data. This delay is added on top of the time that the cross-subsystem DMAC has to wait for the local DMA to complete, before gaining access to the M-bus.

The worst case for the DMA cross-accesses occurs when the M-bus is already being shared by both the local DMAC and the cross-subsystem HPI Controller. Before gaining access to the M-bus, the cross-subsystem DMAC has to first wait for the cross-subsystem HPI transfer to complete and then for the local DMA to complete before gaining access to the M-bus. Similar to the cross-subsystem DMA transfer, the cross-subsystem HPI accesses take 3 cycles to write 16-bit data and 4 cycles to read 16-bit data. This delay is added on top of the time that the cross-subsystem DMAC has to wait for the local DMA to complete (2-4cycles), before gaining access to the M-bus.

4.4.3 Local HPI Transfers

The best case for the local HPI to access the M-bus occurs when there is no other activity of the M-bus, implying an idle local DMA controller and the cross-subsystem DMA controller. In such cases, the local HPI transfers take a maximum of 6-7 cycles to complete (as viewed by a host from outside of the DSP).

The next best case for a local HPI access occurs when a local DMA controller is reading or writing to/from a resource on the M-bus (an internal memory location, for example). The local HPI access will be stalled until the current local DMA cycle completes. Depending on when the HPI request arrives inside an active local DMA access, it will be stalled for a minimum of 1 cycle (16-bit DMA) and a maximum of 4 cycles (32-bit DMA) before gaining access to the M-bus. Any additional local DMA transfers pending at the completion of the local HPI reads or writes cause the M-bus controller to interleave local HPI and local DMA requests on per-element basis. Since the HPI as a whole is much slower than the DMAC, the DMA/HPI interleave will most likely happen at a rate of one HPI transfer every few local DMA transfers.

The next best case for a local HPI access occurs when the cross-subsystem DMAC is reading or writing to/from a resource on the M-bus. The local HPI access will be stalled until the current cross-subsystem DMA read or write completes. Depending on when the local HPI request arrives inside an active cross-subsystem DMA access, it will be stalled for a minimum of 1 cycle and a maximum of 7 cycles before gaining access to the M-bus. The 7 cycles represents the longest cross-subsystem access (a 32-bit DMA source read). The 7 cycles break down is as follows: 1 cycle for the address to cross between subsystems, 3 cycles to return the first 16 bits and 3 cycles to return the second 16 bits of data. Any additional cross-subsystem DMA transfers pending at the completion of the local HPI reads or writes cause the M-bus controller to interleave local HPI and cross-subsystem DMA requests on per-element basis. This means that each additional element of the local HPI transfer will be delayed by the same number of cycles as the first element, when the local HPI and cross-subsystem DMAC take turns reading or writing from/to the M-bus.

The worst case for the local HPI accesses occurs when the M-bus is already being shared by both the local DMA and the cross-subsystem DMA transfers. Before gaining access to the M-bus, the local HPI has to first wait for the local DMA transfer to complete and then for the cross-subsystem DMA to complete before gaining access to the M-bus. The worst case cross-subsystem 32-bit DMA accesses take 6 cycles for destination writes and 7 cycles for source reads. This delay is added on top of the up to 4 cycles that the local HPI has to wait for a local DMA transfer to complete, before gaining access to the M-bus.

4.4.4 Cross-Subsystem HPI Transfers

The best case for access of the HPI from one subsystem to the M-bus resources of the other subsystem occurs when there is no other activity of the M-bus, implying an idle local DMAC and cross-subsystem DMAC. In such cases, the cross subsystem HPI 16-bit destination writes complete in 3 cycles (1 for address/data to travel across subsystems and 2 for M-bus write cycle), while the 16-bit source reads take 4 cycles (1 for address to travel across subsystems, 2 for M-bus read cycle and 1 for the data to return across the subsystems). The HPI peripherals also uses a total of 5 cycles to synchronize to the local clock, for transport delays and internal operation. Adding the cycles yields the best case HPI cross-subsystem destination writes of 8 cycles and best source reads of 9 cycles.

The next best case for HPI cross-access occurs when a local DMAC is reading or writing to/from a resource on the M-bus (an internal memory location, for example). The cross-subsystem HPI Controller trying to get through to the same M-bus will be stalled until the current local DMA cycle completes. Depending on when the HPI request arrives inside an active 16-bit local DMA access, it will be stalled for a minimum of 1 cycle and a maximum of 2 cycles before gaining access to the M-bus. If the HPI request arrives inside an active 32-bit local DMA access, it will be stalled for a minimum of 1 cycle and a maximum of 4 cycles before gaining access to the M-bus. Additional local DMA transfers pending at the completion of the cross-subsystem HPI reads or writes cause the M-bus controller to interleave the local DMA and cross-system HPI accesses on per-element basis. This means that each additional element of the cross-system HPI transfer will be delayed by the same number of cycles as the first element, when the cross-subsystem HPI and local DMA controller take turns reading or writing from/to the M-bus.

The worst case for the HPI cross-accesses occurs when the M-bus is already being shared by both the local DMA and the cross-subsystem DMA controllers. Before gaining access to the M-bus, the cross-subsystem HPI has to wait for the cross-subsystem DMA transfer to complete, and for the local DMA to complete before gaining access to the M-bus. Similar to the cross-subsystem HPI transfer, the cross-subsystem 16-bit DMA accesses take 3 cycles to write 16-bit data and 4 cycles to read 16-bit data. The 32-bit cross-subsystem DMA reads take 6 cycles for destination writes and 7 cycles for source reads. This delay is added on top of the time that the cross-subsystem HPI has to wait for the local DMA to complete, before gaining access to the M-bus.

4.5 EMIF Arbitration

The external parallel interface, XIO can be driven by one of 4 sources – the subsystem A CPU, subsystem B CPU, subsystem A DMAC and subsystem B DMAC. The EMIF block arbitrates between the 4 possible requesters and allows one request at a time to propagate out through the XPORT pins to external memory or peripherals. The XPORT interface is shared between the XCPU or XDMA XIO transfers and the HPI transfers driven by a host. The input pin named XIO configures the XPORT interface to either HPI or XIO mode.

Barring any DMA requests, the EMIF arbiter uses a software handshaking protocol to allow one of the two CPUs to drive the XPORT pins with XIO reads or writes. Before issuing a transfer, each CPU needs to verify the XIO ownership by checking the XIO_GRANT bit of the GPIO register. A “1” value of the XIO_GRANT bit means that the requesting CPU already owns the XIO and is free to start external transfers without further arbitration. A negative result requires the requesting CPU to assert the XIO_REQ bit in the GPIO register and poll the XIO_GRANT bit until the CPU Arbiter inside the EMIF asserts it following the release of XIO by the other CPU. The requesting CPU can now perform the transfer and then complete the handshake by de-asserting the XIO_REQ bit. Each CPU should always de-assert the XIO_REQ bit at the completion of each transfer in order to enable the future XIO requests from the other CPU. Another bit of the GPIO register, CORE_SEL identifies to the software which CPU is executing the program. Software referencing the GPIO register, executing on CPU A will return a value of “0” when reading the CORE_SEL bit of side A’s GPIO. The same software executing on CPU B will return a value of “1” when reading the CORE_SEL bit of the side B’s GPIO register.

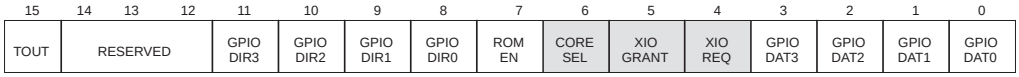

Figure 23. ’C5421 GPIO REGISTER

An active XDMA request from any of the two subsystems causes an internal REQUEST signal to be asserted to both CPU A and CPU B. As soon as the EMIF block recognizes that both CPUs have returned their respective ACKNOWLEDGE signals (communicating that the CPUs have suspended their XIO transfers), the requesting XDMA is allowed to perform its transfer. Simultaneous XDMA requests from both subsystems are resolved by the DMA Arbiter inside EMIF using rotating priority scheme after each transferred word.

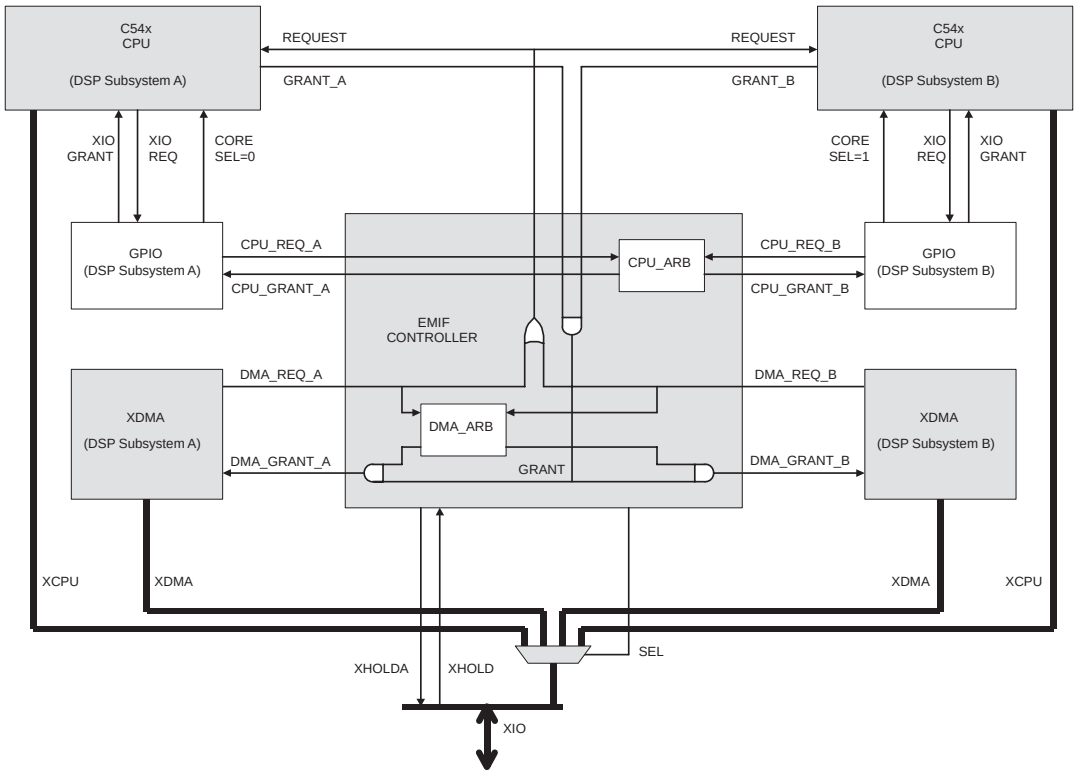

Figure 24. ’C5421 EMIF Arbitration

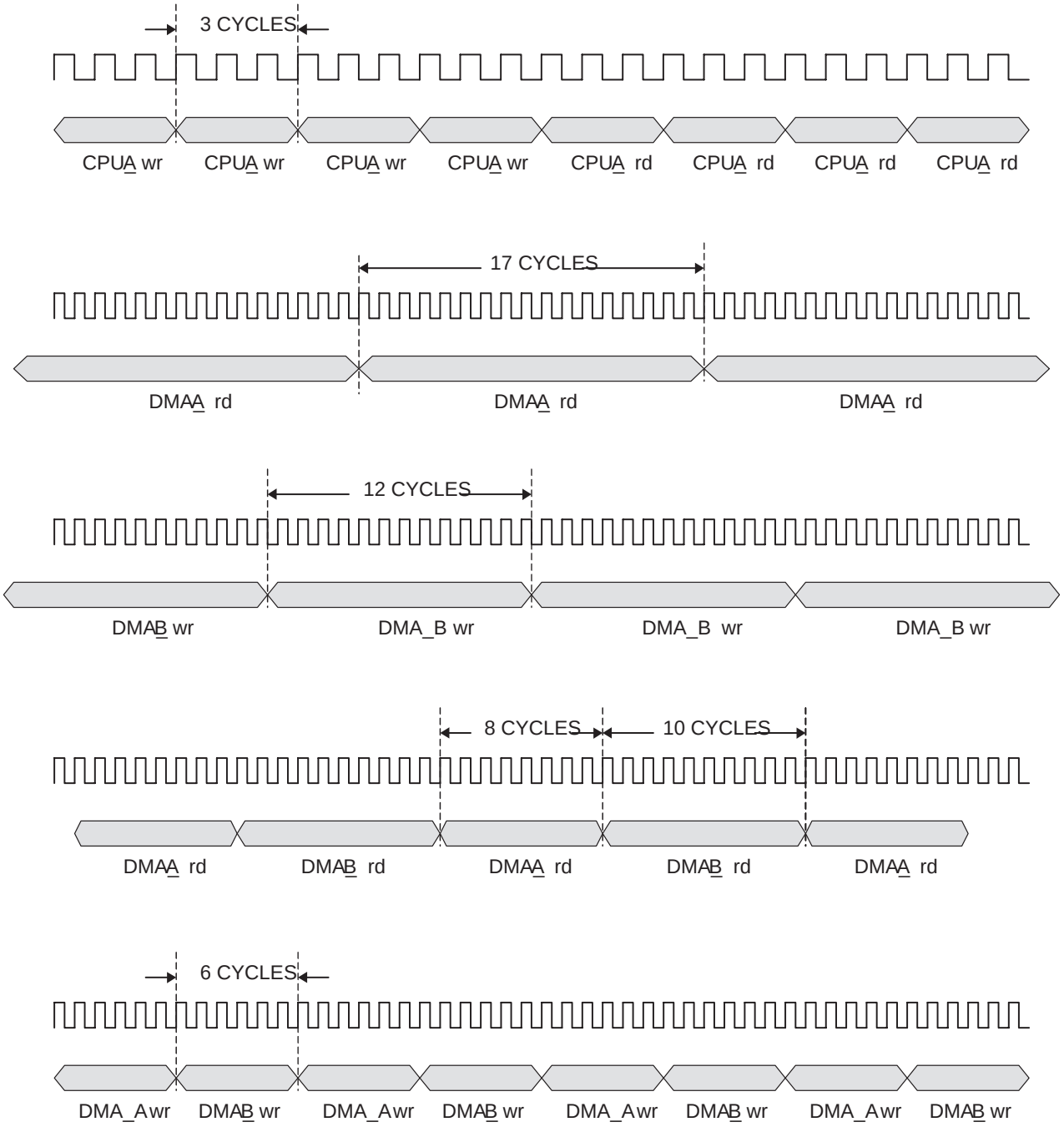

Figure 25. ’C5421 CPU and DMA Systained External Data Transfer Rates

4.5.1 CPU Transfers

Provided that there are no pending external DMA transfers, subsystem A's CPU or subsystem's B CPU can read or write data from/to external memory. Before starting transfers, the CPU has to first establish the ownership of the XIO port with s/w handshaking.

Once allowed to access the external space, the CPU can read or write 16-bit data elements at a maximum rate of one element per 3-4 cycles, depending on the number of wait states (minimum 2-3) reflecting the speed grade of the external memory device.

All external CPU transfers can be interrupted by an external transfer request from any of the two DMA controllers.

4.5.2 DMA Transfers

Any of the two subsystem's DMA controllers, requesting XIO ownership for external access, causes the suspension of any active CPU A's or CPU B's external transfers that may be in progress. The hardwired DMA/CPU handshaking protocol temporarily suspends CPU external transfers in progress (on a word boundary), and grants the XIO ownership to the requesting DMA. In order to prevent the DMA from locking out the CPU, the DMA/CPU handshaking protocol is repeated after every DMA word access.

In the case where only one subsystem is using its DMA controller to access external space, the 16-bit external DMA transfer elements are transferred at a maximum rate of 17 cycles for reads and 12 cycles for writes.

If both DMA controllers (from subsystems A and B) are simultaneously requesting external access, the two DMAs share the XIO interface in an interleaved fashion, one element from each side at a time. The maximum sustained DMA transfer rates for both subsystems are 8-10 cycles between reads and 6 cycles between writes (one word every 18 and 12 cycles per subsystem, respectively).

Only 3-4 cycles of all DMA accesses actually carry a valid address and data. Segments of the remaining time between the DMA reads or writes can be simultaneously used for CPU transfers, if any are pending in between the consecutive DMA accesses.

4.6 Internal RAM Arbitration

The internal RAM of the 'C5421 accessible to a given subsystem consists of one 32K word block of Single Access Data RAM, one 32K block of Dual Access Program/Data RAM and two 64K blocks of Shared Dual Access Program RAM. All of the internal RAM space is also accessible to DMAC and HPI Controllers through the M-bus. Each M-bus read or write takes a minimum of 2 cycles.

Individual Internal RAM blocks are composed of four or eight 8K word sub-blocks. Simultaneous access of different sub-blocks with different buses always guarantees no conflicts. DARAM and PRAM memories can also support sustained simultaneous accesses within the same 8K word sub-blocks, based on the ability of DARAM and PRAM arrays to perform half-cycle internal CPU reads and writes.

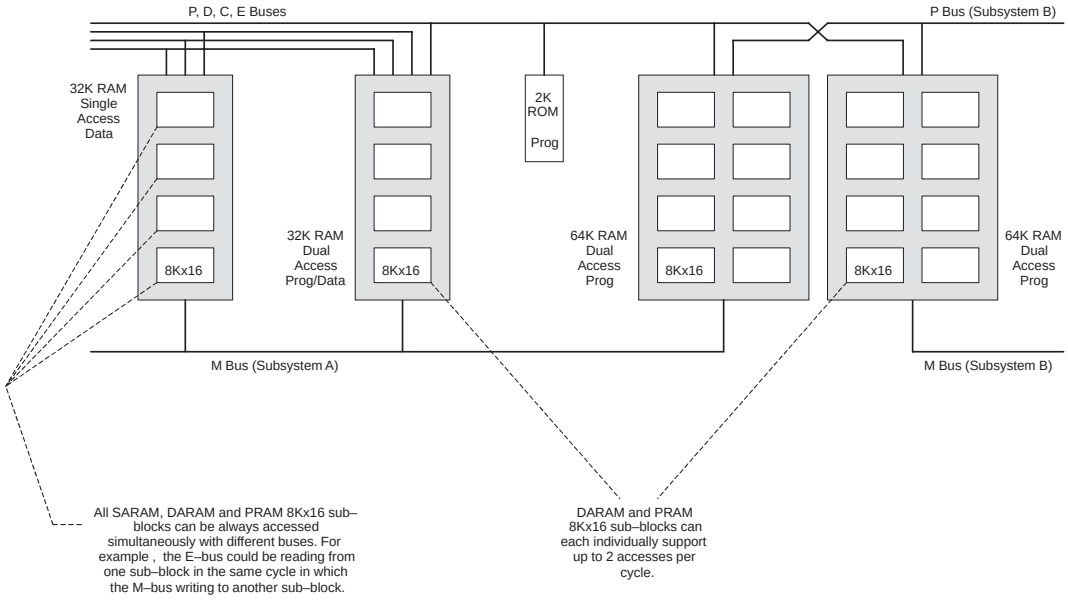

Figure 26. ’C5421 Internal RAM Arbitration

4.6.1 Single Access Data

The primary purpose for the SARAM is to provide data storage for the CPU using the D, C and E buses. The SARAM can also be accessed by the DMAC or HPI Controller using the M-bus, one read or write at a time. The 32K SARAM is organized in 4 independent 8K word blocks, each block able to support read or write requests independently from the other 3 blocks. As is the case with the DARAM and PRAM, the maximum sustained performance is achieved when individual 8K blocks are serviced by different buses (through s/w). For example, block 0 could contain CPU read data, block 1 could contain another section of CPU read data, block 2 could be assigned to CPU write data and block 3 could represent data driven by the DMAC to and from external memory. This arrangement guarantees the maximum sustained performance of 2 words/cycle for CPU data reads, 1 word/cycle for CPU data write and 1 word every 2 cycles for DMA reads or writes.

If more than one address is requested to propagate through a single 8K word block of SARAM in the same cycle, the memory chooses the highest priority access to take place first, while stalling the remaining ones. Any read or write requested by the M bus (driven by DMAC or HPI) has priority over any CPU access. The M-bus reads or writes always take 2 cycles to complete. If there is not an M-bus request present, a C-bus read is always scheduled before a D-bus read and a D-bus read is always scheduled before an E-bus write.

The C/D/E bus priority changes when there is already one write pending when another write is requested. At that time the E bus priority temporarily becomes greater then C or D bus priority, and the writes are processed first until there are no more pending writes.

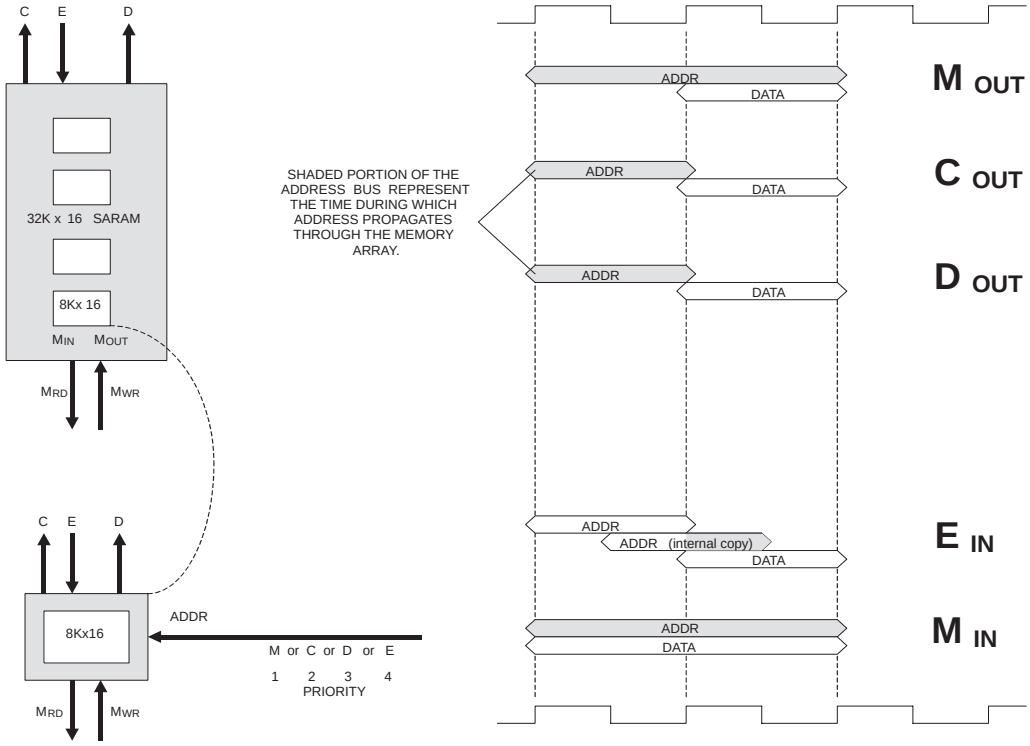

Figure 27. ’C5421 SARAM Timing

4.6.2 Dual Access Program/Data

The major difference between SARAM and DARAM memories is that while SARAM is capable 1 read or write per cycle, DARAM allows 2 reads or a read and a write each cycle.

The primary purpose for the DARAM is to provide data and program storage for the CPU using the D, C, E and P buses. The DARAM can also be accessed by the DMAC or HPI Controller using the M-bus, one read or write at a time. The 32K DARAM is organized in 4 independent 8K word blocks, each able to support read or write requests independently from the other 3 blocks. Similar to the case with the SARAM and PRAM, the maximum sustained performance can always be achieved when individual 8K blocks are serviced by different busses (through s/w). For example, block 0 could contain CPU read data, block 1 CPU write data, block 2 CPU opcodes and block 3 data driven by the DMAC to and from the external memory. This arrangement guarantees the maximum sustained performance of 2 words/cycle for CPU data reads, 1 word/cycle for CPU data write, 1 word/cycle for program fetches and 1word every 2 cycles for DMA reads or writes.

Note, that while the SARAM is only able to access a single word per cycle inside a given 8K block, the Dual Access RAM can fetch up to 2 words per cycle from the same 8K block, using dedicated C and D buses. While the C and D addresses are active outside of the memory blocks throughout the entire cycle, internally they propagate across the memory arrays in different half-cycles. Thus, two word reads can be sustained during every clock cycle. For single word reads, bus D carries the data and bus C in not active. Since the E bus writes take place internally on the same half-cycles as C bus reads, the less frequent write operations can fill the C-bus holes for any cycle with only one read. For example, every even cycle the CPU can read 2 words of data and write one word, as long as during every odd cycle the DARAM is not asked to perform more than a single read, even if all of that data is located in a single 8K word block.

If more than one address is requested to propagate through the DARAM during the same half-cycle, the memory chooses the highest priority access to take place first, while stalling the remaining ones. Any read or write requested by the M bus (driven by DMAC or HPI) has priority over any CPU access. The M-bus reads or writes always take 2 cycles to complete. If there is not an M-bus request present, a C-bus read is always scheduled before an E-bus write during the first half-cycle, and a D-bus read is always scheduled before a P-bus fetch during each second half-cycle.

The C/E bus priority changes when there is already one write pending when another write is requested. At that time the E bus priority temporarily becomes greater then C bus priority, and the writes are processed first until there are no more pending writes.

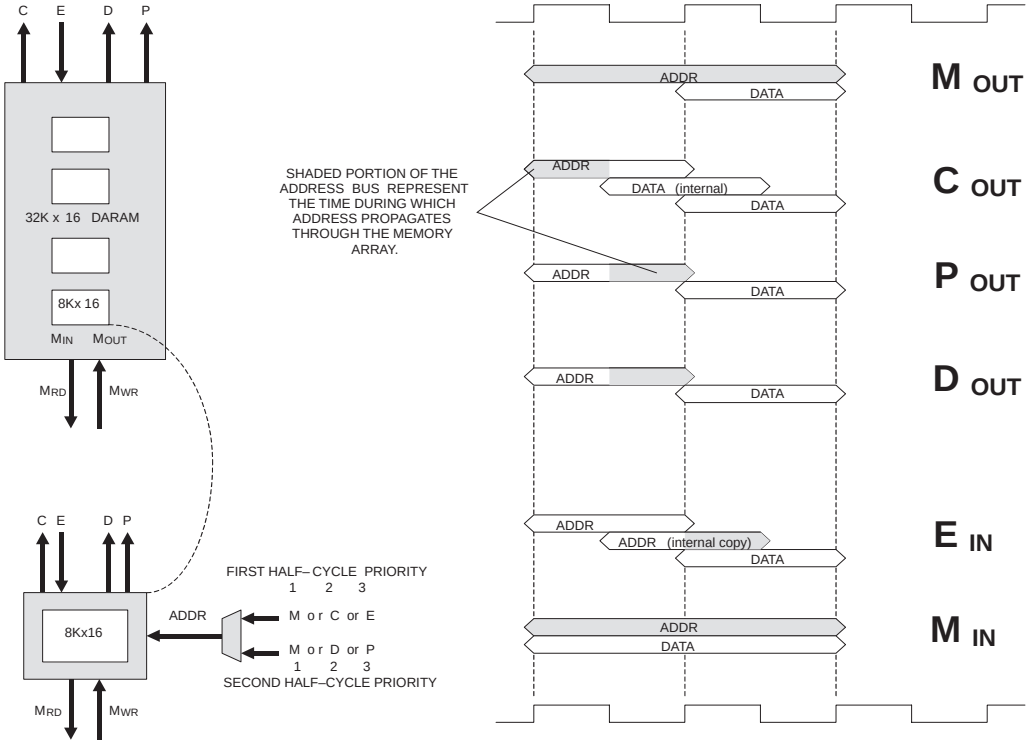

Figure 28. ’C5421 DARAM Timing

4.6.3 Shared Program RAM

The 128K of the Shared PRAM are organized as two 64K blocks, each composed of eight independent 8K word arrays. The access to the two 64K blocks is identical for program fetches of both CPUs. The DMAC and HPI Controllers of both subsystems have identical full visibility to the entire 128K of the Shared PRAM. Each side, however, uses different M-buses to access the low and high halves of the PRAM. subsystem A DMAC/HPI use their local M-bus to access the low 64K words of the Shared PRAM, and the remote M-bus of the subsystem B to access the high 64K words. subsystem B uses its local M-bus to access the high 64K words of the Shared PRAM, and the remote M-bus of subsystem A to access the low 64K words. The remote access involves additional busing across the chip and takes 1 extra cycle in each direction. This means that cross-subsystem writes by DMAC or HPI will take 1 extra cycle and cross-subsystem reads will take 2 extra cycles (one for outgoing address and one for returning data).

The primary purpose for the PRAM is to provide shared program storage for both CPUs using two P buses. For similar applications running on both CPUs, shared program memory effectively doubles the available on-chip program storage. The PRAM can also be accessed by the DMAC or HPI Controller using the M-bus, one read or write at a time. The 32K PRAM is organized in 8 independent 8K word blocks with dedicated address multiplexers, each able to support read or write requests independently from the other 7 blocks. Just as is the case with the SARAM and DARAM, maximum sustained performance is always guaranteed to be achieved when individual 8K blocks are serviced by different buses (through s/w). For example, block 0 could be assigned to CPU A program fetches, while block 1 is servicing CPU B program fetches and block 3 the I/O data driven by the DMAC. This arrangement guarantees the maximum sustained performance of 2 words/cycle for program fetches and 1word every 2 cycles for DMA reads or writes.

Since the individual 8K blocks of PRAM are identical to the ones used by the DARAM, two program fetches per cycle can even be achieved if CPU A and CPU B are simultaneously accessing the same 8K block. Internally, this is done by connecting the P-bus of one of the CPUs to the C-bus port of the memory block, while keeping the P-bus of the other CPU connected to the P-bus port of the memory block. Since the P-bus and C-bus addresses propagate through the memory arrays on different half-cycles of the clock, the Shared PRAM can support simultaneous program fetches from both CPUs regardless whether the program addresses point to different 8K blocks or are located in the same 8K block.

If any of the CPU program fetches are requested at the same time as an M bus transfer request, the CPU is stalled for a minimum of two cycles until all I/O transfers are completed. In other words, any read or write requested by the M bus (driven by DMAC or HPI) has priority over the CPU A or CPU B program fetches. The M-bus reads or writes always take 2 cycles to complete.

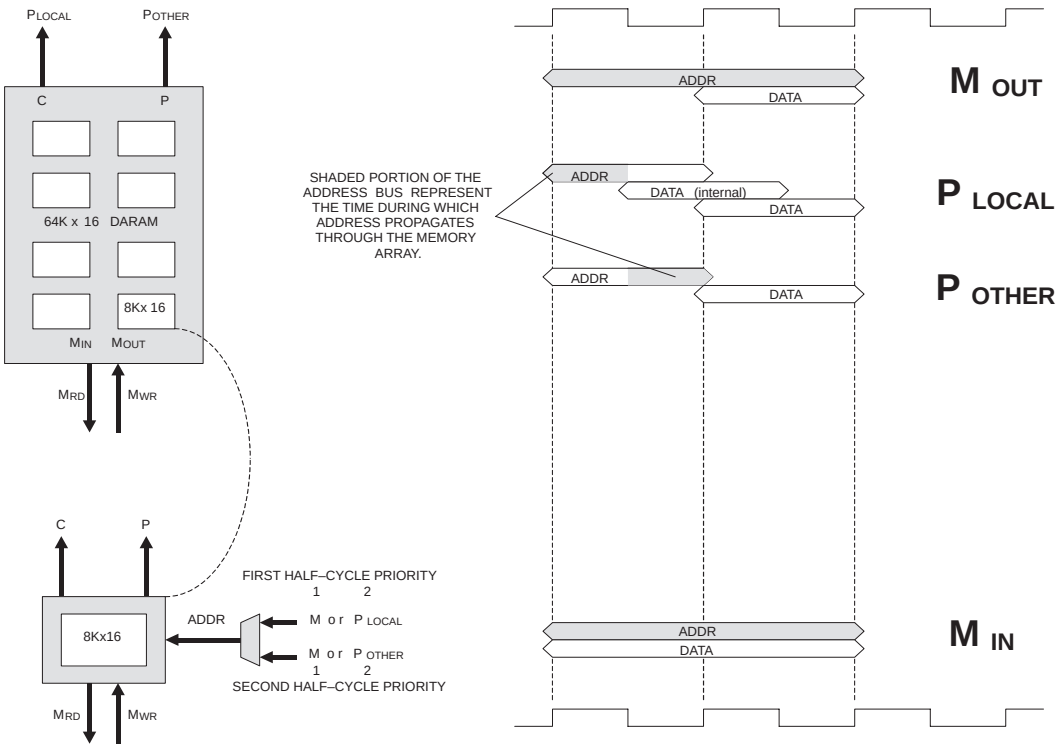

Figure 29. ’C5421 PRAM Timing

5 Begin Writing Code for the TMS320C5421 Today

Similar architectures of the ’C5420 and ’C5421 devices allow for many system-level issues to be resolved prior to obtaining ’C5421silicon. The high level of compatibility between these DSPs allows for system development to begin now. By taking advantage of the existing ’C54x software and hardware tools, ’C5421 systems can have a running start at the time of silicon availability. The identical CPUs in the ’C5420 and ’C5421 devices allow for code to be written for the ’C5421 using existing ’C54x tools. ’C5420 code will require no modification to use on the ’C5421 except for re-linking program and data to different memory map ranges of the internal RAM blocks. All peripheral-specific code will also be able to run unchanged on the ’C5421. The ’C54x compiler may be used for all members of the ’C54x device platform.

Code development for the ’C5421 may begin using the ’C54x simulator. The simulator provides a cycle-accurate account of device. The simulator provides a good environment to learn the ’C54x CPU and peripheral architecture. The peripherals on the ’C5421 are identical to those modeled by the current simulator.

For a development start in hardware, the ’C5410x EVM may be used to understand the ’C5421 system level functionality. In this environment, C and assembly code can be debugged while running in real time. All of the peripherals, except the XIO, on the ’C5421 are identical to those of the 5410 so the EVM is a good tool to understand how to incorporate the peripherals into a real-time system.

In addition to the EVM, the low cost C5402 DSK can also be used for device familiarity and evaluation. Using these development platforms, as well as the ’C54x literature currently available will enable ’C5421 systems to be designed before ’C5421 silicon is made available.

5.1 ’C54x Tools Support

’C54x tools are available now for use in all ’C54x designs. ’C5421 will be added to the development tools in early fourth quarter 1999. The ’C54x development tools available today for the ’C54x are:

- ’C54x Simulator Software
- ’C54x Optimizing C Compiler/Assembler
- ’C5402 DSK (Design Starter Kit)
- TMS320C5410 Evaluation Module (EVM)
- XDS510 ’C54x C Source Debugger Software
- XDS510 Emulator Hardware with JTAG Emulation Cable
- Third Party Hardware and Software

5.2 ’C54x Literature Available

A great deal of literature is available today for the ’C54x devices.

- TMS320C54x CPU and Peripherals Reference Set
- TMS320C54x Enhanced Peripherals Reference Set
- TMS320C54x Technical Brief
- TMS320C54x Programmer’s Guide
- TMS320C54x Evaluation Module Reference Guide
- TMS320C54x Assembly Language Tools User’s Guide
- TMS320C54x Optimizing C Compiler User’s Guide
- TMS320C54x C Source Debugger User’s Guide

See http://www.ti.com/sc/docs/dsps/products/c5x/index.htm for more information.

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

CERTAIN APPLICATIONS USING SEMICONDUCTOR PRODUCTS MAY INVOLVE POTENTIAL RISKS OF DEATH, PERSONAL INJURY, OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE ("CRITICAL APPLICATIONS"). TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS. INCLUSION OF TI PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE FULLY AT THE CUSTOMER'S RISK.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.