Application Report
SPRAB25–January 2009

How to Approach Inter-Core Communication on TMS320C6474

Juergen Mathes

ABSTRACT

Today's digital signal processor (DSP) architectures are confronted with the tough requirement of addressing a wide-range of standards and meeting a cost-effective performance/power trade-off. Increasing raw million instructions per second (MIPS) performance just by running at a higher frequency is not possible anymore since leakage is becoming a dominant factor with shrinking silicon geometries. One vector in scaling modern processing architectures is the number of cores on a single piece of silicon. It becomes crucial to find the sweet spot of performance and power consumption.

Having said this, the question is how to ease the handling of the three cores that are present on the C6474? What features are supported and how can they be used? How do the cores communicate effectively with each other on the chip? How is scalability allowed on board level? This application report offers some answers to those questions.

Contents

List of Figures

List of Tables

1 Why Move Towards Multi-Core?

The trend for more performance and smaller geometries in the semiconductor industry is unbroken. However, increasing performance by *simply* increasing the device frequency is not meeting the power requirements of the next generation anymore. One answer to that challenge is to move to multi-core architectures. Why is that?

The main reason why clock speeds cannot increase infinitely is the power consumption per square mm of silicon. The heat generated per square mm is steadily increasing and with shrinking geometries and dissipating heat becomes an expensive exercise.

The contribution of leakage power grows going down process nodes, which is the static portion of power consumption; that means more power is consumed without having done anything useful. Clock speed is the dynamic portion of the total power consumption observed on a device, so reuse of logic helps to reduce the domination of static power consumption. Multiple cores sharing memory, peripherals, and accelerators can help to achieve that. But, leakage also increases when a process node is optimized for performance rather than power consumption. This means that the capability to reduce clock-speed without sacrificing performance helps as well, e.g., multiple cores running at lower speed achieve the same performance as a single core running at higher speed, but meeting the power budget goals.

The frequency increase is not the main performance contribution anymore. Memory speeds are not increasing as quickly as logic does. The performance gap between core and memory eats up parts of the performance gained by the frequency increase. To be able to support the higher frequency changes in the architecture becomes necessary, e.g., a longer instruction pipeline. Changes like this can take away parts of the gained performance increase also.

2 C6474 Architecture Overview

The TMS320C6474 was designed to meet the requirements for next-generation high performance DSPs. The DSP is comprised of three C64x+™ DSP cores (running at 1GHz), high-speed interfaces, and base-band accelerators. All cores and EDMA masters on the device have full access to the memory mapped random-access memory (RAM), including L1 memories if configured as static random access memory (SRAM), L2 memories, and external DDR2 synchronous dynamic random access memory (SDRAM) and all other resources on the device like accelerators and peripherals.

Access restrictions to memory pages and some chip-level components can be enforced through memory protection hardware, which configures read, write, and execute access to resources on the device.

The three cores can arbitrate over shared resources through direct signaling, or through utilization of the semaphore peripheral block.

To ease signaling, a very flexible event routing is possible on the C6474. The system events (e.g., a frame sync) can be routed to interrupts or exceptions per core and they can be used to trigger the EDMA or the emulation logic.

The switch fabric provides enhanced on-chip connectivity between the DSP cores and the peripherals and accelerators.

For CPU independent data movements, there are three local DMA's: one per core called internal direct memory access (IDMA) and a single central enhanced direct memory access (EDMA3) available on the C6474.

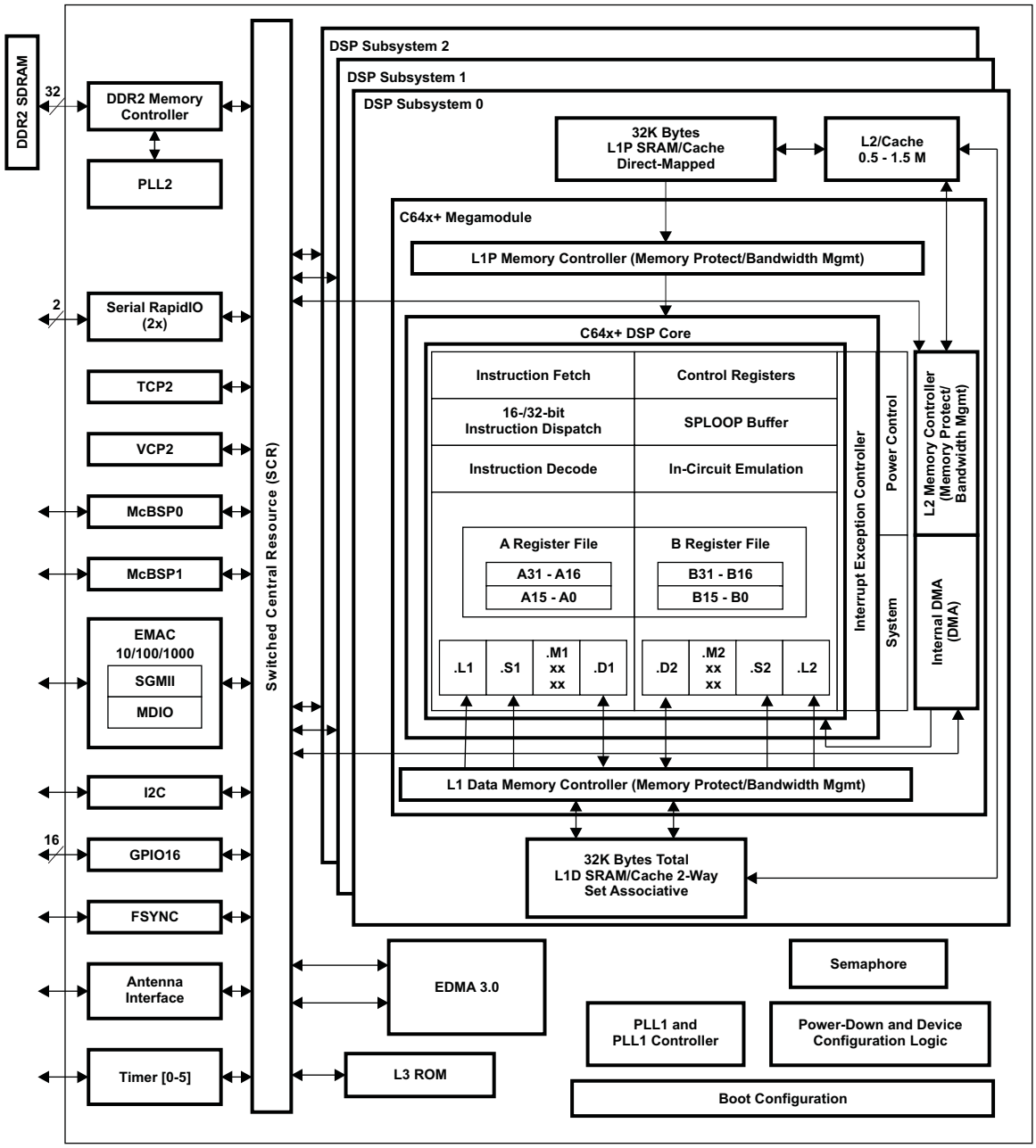

Figure 1. C6474 Block Diagram

For more detailed information, see the *TMS320C6474 Multicore Digital Signal Processor Data Manual* (SPRS552).

3 Features That Support a Multi-Core Architecture on the C6474

Every application has different communication needs that might span from rather large chunks of data that need to be processed jointly from multiple cores to load balancing models or just simple synchronization to run the same tasks in parallel on all cores. That is why the device is not limited to a certain use model and why the architecture is kept generic enough to allow you or 3rd parties to come up with innovative ideas. Look at the individual hardware features that the C6474 is offering to support a multi-core architecture since these are the *ingredients* that allow you to define your own recipe to a full custom inter-core communication model. This section describes the modules available and outlines different scenarios in which they can help dealing with a multi-core architecture. Appendix A discusses specific example scenarios in greater detail.

3.1 Hardware Semaphores

The semaphore module allows resource sharing amongst the three cores on the C6474. There's no fixed assignment from a semaphore to a resource; therefore, the DSP software designers have the freedom to decide upon the mapping. All the DSP cores should follow the same convention that has been decided on. The resource that has to be shared could be any peripheral, but also memory regions that are accessed by more than one core.

The hardware semaphore allows read-modify-write operations that are atomic and are not interruptible by other operations that try to access the same resource. This helps to keep coherency between the cores. Three flexible accessing modes are offered: direct, indirect, and combined accesses.

The direct access mode grants ownership if the semaphore is free, otherwise, nothing else happens. That means a core needs to utilize a polling mechanism to check if the semaphore flag is free and come back when blocked.

When using the indirect mode, a core is notified by interrupt when the requested resource is free. There is a two level deep request queue per semaphore resource base available to manage the three cores. This means that there are 2 pending and 1 accepted access. Once an indirect request has been posted to the queue, this request cannot be cancelled and remains queued. This also means that the semaphore module cannot be used to arbitrate accesses within the same core. For a single core, software semaphores offered within DSP/BIOS™ could be used. The SEMFlag is set and an interrupt is generated to the originating CPU If the semaphore becomes available. Upon receiving the interrupt, the CPU should first check the SEMFlag to find out which semaphore has become available.

Best resource sharing is offered with the combined access mode. In case a request sees a free semaphore, it basically becomes a direct access. Access is denied and the request is queued in case the semaphore is locked by another core. This means you only have to do polling once and then wait for an interrupt, in case the semaphore was blocked.

In all modes, once a master has completed accessing a reserved resource, it must free the resource again.

The hardware semaphore module allows faster arbitration of the shared resource compared with software semaphores. The semaphore module supports 32 flags. There is support in DSP/BIOS for software semaphores in case the number of shared resources exceeds 32.

For more information, see the *TMS320C6474 DSP Semaphore User's Guide* (SPRUG14).

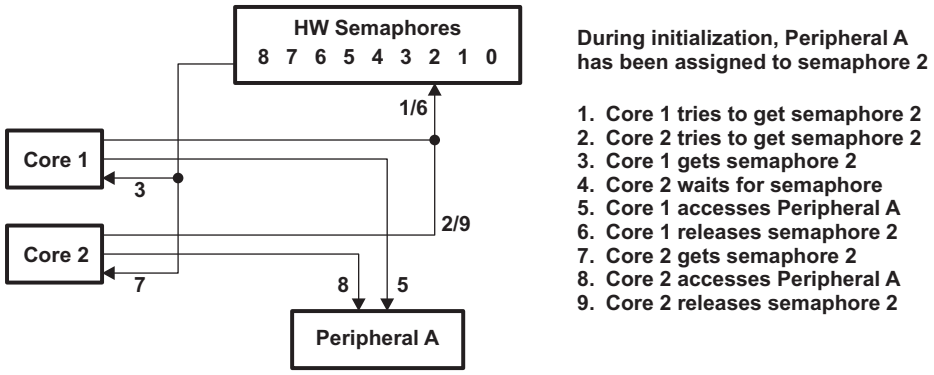

Figure 2. Example of Two CPUs Trying to Access the Same Peripheral

3.2 Global and Local Memory Map

Each of the three cores has its own local memory map. These local memory addresses are only visible from its corresponding core. The same local memory locations are also mapped into the global address space using distinguishable global addresses.Therefore, every core is capable to access the other cores memory, unless blocked by memory protection.

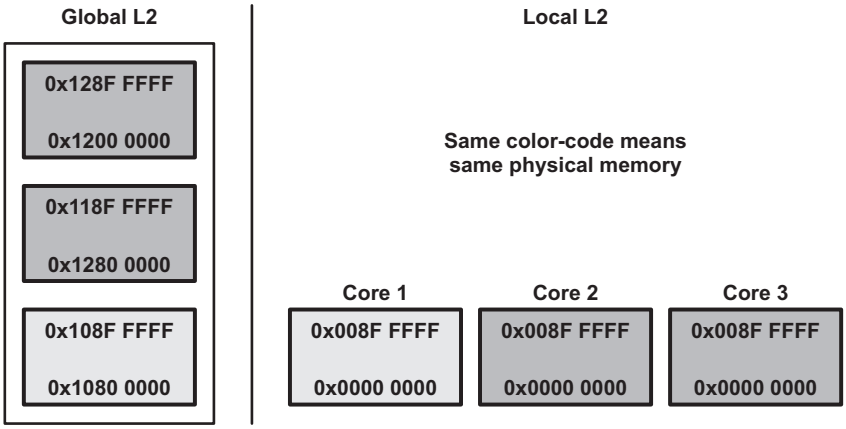

Figure 3. Global/Local Memory Map for L2

The build process for code and the linker command file remain the same for each core since the local addresses are the same for every core. Already built software modules can be run on any core. The same principle applies for memory mapped L1 and L2. Memory accesses to the local memory map will be faster than accesses to the global memory map since global accesses will be routed through the switched central resource (SCR). Exact cycle counts are dependant on the data traffic, cache settings, etc.

For more information, see the *TMS320C6474 Multicore Digital Signal Processor Data Manual* (SPRS552).

3.3 Memory Protection

Memory protection allows controlling access to certain memory pages. That becomes a very useful feature when multiple masters are present in a device-architecture. Within the C6474, there are obviously the three cores that can be distinguished and permitted by a certain type of access or not. Besides that, there is also the EDMA3 that acts as a master for the peripherals Serial RapidIO® (SRIO) and Ethernet media access controller (EMAC). These can, of course, also be controlled with the memory protection. The memory protection has a requestor-ID based access control for global accesses. Each master has an access ID assigned except for the EDMA3/IDMA. It inherits the access ID of the core that has set up the transfer. That way the originator of a transfer is always known. It is also possible to allow or deny local access (see also proxy memory protection in Appendix A).

The C6474 memory protection architecture divides the DSP internal memory (L1P, L1D, L2) into pages and each page has an associated set of permissions. The 16 pages in L1P and in L1D (granularity of 2k) and up to 64 pages in L2 (granularity of 32k in symmetrical memory configuration).

The C6474 offers two privilege levels: supervisor mode and user mode. Code executed in supervisor mode is considered trusted and has more access privileges than code executed in user mode. A typical example for supervisor mode code would be an operating system kernel or a device driver. Code running in user mode would be the application itself.

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

CPU accesses, internal DMA (IDMA), and other accesses have a privilege level associated with them. In addition, the allowed type of access can be defined. Individually read, write or execute accesses can be enabled or disabled.

For more information, see *TMS320C64x+ DSP Megamodule Reference Guide* (SPRU871).

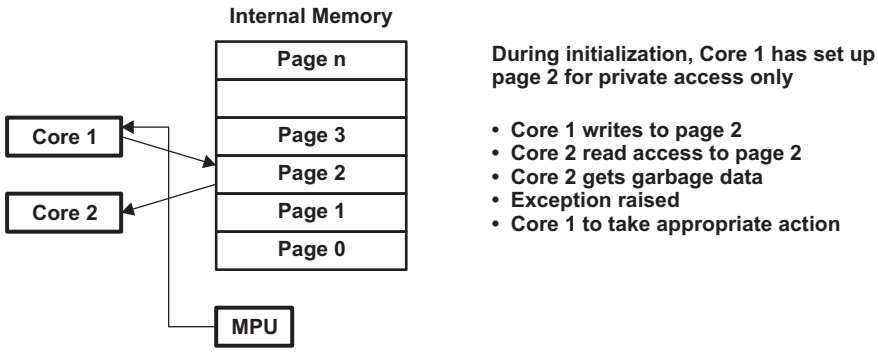

Figure 4. Example of Access Violation

3.4 EDMA3, IDMA and SCR

In terms of data movement, the heart of the C6474 is the EDMA3, together with the three IDMA's per core and the switch matrix. The switch matrix has two independent instances: one for the data movements and one for configuration of the peripherals. That way re-configuration of peripherals will not get into the way of data transfers. The IDMA comes in handy for memory-to-memory transfers between L2 and L1 and automatic peripheral configuration.

The EDMA3 is shared between all three cores so one of the first questions that arise is how do I manage sharing in order to avoid configuration conflicts between them. To ease the use for multiple masters, the EDMA3 channel controller divides its address space into eight regions. Individual channel resources are assigned to a specific region, where each region is typically assigned to a specific EDMA3 programmer (this may be a specific core or even a specific task). Active memory protection can be used in conjunction with regions such that only a specific EDMA3 programmer with a specific requestor-ID and/or privilege level is allowed access to a given region, and thus to a given EDMA3 or quick direct memory access (QDMA) channel. This allows robust system-level EDMA3 code where each EDMA3 programmer only modifies the state of the assigned resources. Then for notification there is region interrupts associated with every shadow region. These individual interrupt lines can be routed to the respective EDMA3 programmer or the core that is interested in the data being moved. This way the corresponding EDMA3 programmer or core receives the interrupt when the initiated transfer is completed.

An alternative approach would be to ignore the regions altogether and use just the global region. Then the channel's resources need to be agreed on and split between the cores. The software is responsible to respect the agreed resource split. That is a simpler approach but also bears the risk of unwanted re-programming of EDMA3 channels. For notification there's also a global completion interrupt that can be routed to an individual core or more cores.

For more information, see the *TMS320C6474 DSP Enhanced DMA (EDMA3) Controller User's Guide* (SPRUG11).

On the C6474, the C64x+ Megamodule, the EDMA3 transfer controllers, and the system peripherals are interconnected through two switch fabrics. The switch fabrics allow for low-latency, concurrent data transfers between master peripherals, and slave peripherals. Through a switch fabric, the CPU can send data to the viterbi-decoder co-processor (VCP2) without affecting other data transfers (e.g., from SRIO to the DDR2 memory controller). The switch fabrics also allow for seamless arbitration between the system masters when accessing system slaves.

Two types of buses exist in the C6474 device: data and configuration buses. Configuration buses are mainly used to access the register space of a peripheral and the data buses are used mainly for data transfers. However, in some cases, the configuration bus is also used to transfer data. For example, data is transferred to the VCP2 and turbo-decoder coprocessor (TCP2) via their configuration bus interface. Similarly, the data bus can also be used to access the register space of a peripheral. For example, the DDR2 memory controller registers are accessed through their data bus interface. The C64x+ Megamodule, the EDMA3 traffic controllers, and the various system peripherals can be classified into two categories: masters and slaves. Masters are capable of initiating read and write transfers in the system and do not rely on the EDMA3 for their data transfers. Slaves on the other hand rely on a master to perform transfers to and from them. Examples of masters include the EDMA3 traffic controllers, SRIO, and EMAC. Examples of slaves include the multichannel buffered serial port (McBSP) and inter-integrated circuit (I2C). The C6474 device contains two switch fabrics through which masters and slaves communicate. The data switch fabric, known as the data switch central resource (SCR), is a high-throughput interconnect mainly used to move data across the system. The configuration switch fabric, also known as the configuration switch central resource is mainly used by the C64x+ Megamodule to access peripherals. Note that the data SCR also connects to the configuration SCR.

For more information, see the *TMS320C6474 Multicore Digital Signal Processor Data Manual* (SPRS552).

The IDMA plays a minor role in terms of inter-core communication since it is only operating locally on memory. However, since it is very useful for peripheral configuration, it touches indirectly on the aspect of resource sharing. For usage examples, see Appendix A.

Each core is assigned its own bandwidth manager. The purpose of the bandwidth management is to assure that some of the requestors do not block the resources (L1 and L2 memory plus the memory mapped registers) that are available in the C64x+ Megamodule for extended periods of time. To avoid prolonged lockout of any one requester, (this can be CPU, IDMA, an externally-initiated slave DMA transfer or cache coherency operations) each of these requesters can be assigned a maximum waiting time. A minimum amount of bandwidth (50%, 33%, 20%, 11%, 6% and 3%) can be assured, if using this mechanism. The internal bus structure is dimensioned in a way so that the CPU and EDMA can concurrently access L1D or L2. 256-bits are available every cycle (L1 runs at CPU speed, L2 runs at CPU/2).128 bits are the maximum requested by the CPU data path, which leaves 128-bit per cycle of bandwidth for EDMA transfers coming in.

3.5 Exceptions, Interrupts and Events

Inter-core interrupts can be used to facilitate fast signaling between cores. IPC registers are provided at chip-level register space to facilitate inter-core interrupts. These registers can be utilized by external hosts or the three cores to generate interrupts to other cores. A *Source ID* is provided by which up to 28 different sources of interrupts can be identified. The allocation of source bits to source processor and meaning is entirely based on software convention. Since the registers are located at chip level, any master that has a connection to this module can write to these registers (all traffic controllers except TC1, RapidIO and the three cores).

The CPU interrupts on the device are configured through the C64x+ Megamodule interrupt controller (INTC). The interrupt controller allows for up to 128 system events to be programmed to any of the twelve CPU interrupt inputs, the CPU exception input, or the advanced emulation logic. Since the C6474 offers a lot of system events there's a chip tnterrupt controller (CIC) to pre-select some of the events before going to the INTC. With the three cores on the C6474, there is also 3 times the INTC + CIC present; Figure 5 shows a single instance. The EDMA also has an additional CIC before its channel controller to select from the available events that could be used as triggers.

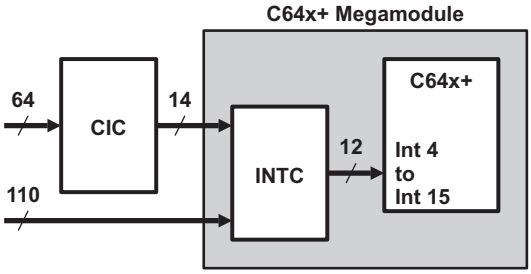

Figure 5. Event Architecture

Events can also be used to trigger EDMA3 transfers or the advanced event triggering (AET) to begin, e.g., a program counter (PC) trace.

Exceptions are used when memory protection detects an access violation. The other possible source for an exception would be an unrecoverable error, e.g., an illegal instruction. Exceptions can also be enforced by software.

4 Software Support

4.1 Operating System

DSP/BIOS is a scalable real-time kernel, designed specifically for the TMS320C5000™ and TMS320C6000™ DSP platforms. DSP/BIOS has been proven in thousands of customer designs and is an integral part of the Code Composer Studio™ development tools. DSP/BIOS requires no runtime license fees and is backed by Texas Instruments worldwide training and support organizations. On the C6474, you would run an instance of DSP/BIOS on each core.

The MSGQ module was specifically designed to be used for homogeneous or heterogeneous multi-processor messaging (available from DSP/BIOS 5.10 onwards). It allows for the structured sending/receiving of variable length messages and enables them to be passed transparently between DSP/BIOS executions threads, such as tasks and software interrupts. The MSGQ module internally handles message passing between threads on the same processor. To pass messages between discrete processors, a message queue transport (MQT) is required for the specific physical inter-processor communication mechanism used, such as peripheral component interconnect (PCI), MCBSP or SRIO.

4.2 Drivers

The SRIO MQT enables DSP/BIOS message passing between multiple devices via the SRIO bus based on the MSGQ module. The MQT module does not support the direct IO protocol.

5 On-Board vs On-Chip Communication

Boards in high performance applications typically have more than just one DSP on a board. There is a need to look beyond the device internal core to core communication and extend the model to inter device communication. Ideally this is kept as transparent as possible since a modular approach makes the lives of hardware and software designer much easier.

In addition to transparency, high performance applications require ever increase data communication rates. The need for increased input/output (I/O) capacity will result in a substantial increase in the amount of communication among processors on the board and also via the backplane interconnecting the boards. SRIO is an ideal peripheral to match these criteria.

5.1 Serial RapidIO (SRIO)

On the C6474, there is no host port interface (HPI) anymore. The HPI from previous devices in the C6000™ DSP platform has disappeared on the C6474. The DirectIO protocol of RapidIO can achieve the same purpose since it allows direct access to the memory map of the targeted device, just as the HPI did but in a much more effective way. This is true in terms of pin-count and through-put. There are also SRIO boot-modes present on the C6474.

Since SRIO is a point-to-point, there are two principle topologies that need to be considered in a multi device environment. The first option is to daisy chain all the devices. The advantage of this architecture is that no additional component is needed.

The second approach is to utilize an SRIO switch. The advantage of this approach is that there is no latency that would occur with a daisy chain and the improved bandwidth of the overall interconnect architecture.

6 References

- *TMS320C6474 Multicore Digital Signal Processor Data Manual* (SPRS552)
- *TMS320C6474 DSP Semaphore User's Guide* (SPRUG14)
- *TMS320C64x+ DSP Megamodule Reference Guide* (SPRU871)
- *TMS320C6474 DSP Enhanced DMA (EDMA3) Controller User's Guide* (SPRUG11)
- *TMS320C6474 DSP Serial RapidIO (SRIO) User's Guide* (SPRUG23)
- *TMS320 DSP/BIOS User's Guide* (SPRU423)
- *TMS320C6000 DSP/BIOS Application Programming Interface (API) Reference Guide* (SPRU403)

Appendix A Example Scenarios

A.1 Sharing Peripherals vs Dedicating Peripherals to a Single Core

SRIO:

The SRIO standard offers two protocols that need to be distinguished when discussing how to share the peripheral among the core or dedicating it to a single core.

When the DirectIO protocol is used, you need to distinguish between master and slave devices. The slave operation is more of a memory/buffer organization discussion since the master writes directly into the memory map of the slave device. The memory allocation of the slave device needs to be known by the master. It is important to note that you need to use the global memory map in conjunction with the SRIO peripheral.

The best way to share the SRIO peripheral in terms of resource usage is to dedicate an LSU to a core. This also makes it easy to dispatch interrupts to the according core. The cores can configure them individually while reusing the common physical SERDES configuration.

When the Message passing protocol is used, mailboxes come into play. With message passing, a destination address is not specified. Instead, a mailbox identifier is used within the RapidIO packet. The mailbox is controlled and mapped to memory by the local (destination) device. The advantage of message passing is that the source device does not require any knowledge of the destination device's memory map. The SRIO peripheral contains Buffer Descriptions Tables for each mailbox. These tables define a memory map and pointers for each mailbox. Messages are transferred to the appropriate memory locations via the local DMA (within the SRIO peripheral). So the best way to share the SRIO peripheral is to assign mailboxes to cores.

EMAC:

The EMAC uses four 32-bit words as buffer descriptors that point to different buffers in the DSP memory. The CPUs create and maintain these buffer descriptors, and reads them as they are updated by the EMAC. The EMAC supports up to eight channels for transmit and up to eight channels for receive, and maintains transfer descriptor lists for each. Only four of the eight receive channels are required on C6474, one for each CPU plus a broadcast channel, each of which has its own MAC address. Transmit channel utilization is under software control, and typically involves one or two channels per core. The EMAC reads and writes to these buffer descriptors as it transfers data to or from the buffers.

Each of these channels can be mapped to one or more independent MAC addresses via the address RAM. The EMAC can have up to 32 MAC addresses assigned to it in the RAM, which can be used for unicast, multicast, broadcast traffic. Then, for each channel you have a chain of receive buffer descriptors that each point to a buffer in the actual memory where the packets received on that channel will be stored. An application has to check the destination MAC address of each packet received, and depending on that, put it in the appropriate queue.

On receive, the eight channels represent eight independent receive queues with packet classification. Packets are classified based on the destination MAC address. Each of the eight channels is assigned its own MAC address, enabling the EMAC module to act like eight virtual MAC adapters. Also, specific types of frames can be sent to specific channels. For example, multicast, broadcast, or other (promiscuous, error, etc.) frames can each be received on a specific receive channel queue.

McBSP and I2C:

Since the McBSP/I2C is best serviced by the EDMA3, sharing the peripheral is more a question of an initial configuration. Assume two cores want to send out messages via the same McBSP/I2C. There are two options that can be achieved: one is to implement a job queue that gets serviced by the EDMA3 and the other is a ping pong buffer scheme where one core owns the ping and the other core the pong buffer.

GPIO:

There is no generic recommendation here; it depends a lot on what purpose the GPIOs are used. Assume the pins are used to switch things on and off or read status information. In that case, no special care is needed since via the Set and Clear registers of the outputs can be individually controlled easily without the need for a read-modify-write operation. Reading the input pins status again does not raise an issue.

Timer:

In case a timer is used to synchronize, the access to this timer is only to configure it. Assume the synchronization is needed only for a limited amount of time and it could be shared in a sense to free the timer to be used by another core. When the synchronization is needed continuously, it makes more sense to assign a master to the timer that is responsible for its configuration.

In case a timer is used to measure time, it should be dedicated to a core to avoid access latency.

Antenna Interface/Framesync:

Here the data transfer happens autonomously from the cores. It can be very well defined where the transfer data should end up. From a data processing perspective, there is no special care to be taken.

DDR2:

It makes sense to dedicate the complete external memory to a single core. One example would be if one core runs a huge application that exceeds the capacity of the internal memory. L2 would then be partially configured as cache. To allow maximum performance, other accesses to external memory should be avoided. Code that is not performance critical, but needs to run on multiple cores, is ideal to be placed in external memory.

Another example would be if all cores need to work on a big database. In that case, the L2 should not be configured as cache since there is no automatic cache coherency protocol in place (snooping) between L2 and external memory. To maintain data coherency, the semaphore module is the best option in this case. This avoids race conditions when certain masters write data to the DDR2 memory controller. For example, if master A passes a software message via a buffer in external memory without waiting for an indication that the write is complete, master B could bypass the master A write when it attempts to read the software message and read stale data; therefore, master B could receive an incorrect message.

In contrary, an EDMA3 transfer will always wait for the write to complete before signaling an interrupt to the system, avoiding this race condition.

A.2 Memory Copy/Move Approach vs Memory Sharing

Basically, this discussion can be drilled down to access time vs memory usage (see Table A-1).

Table A-1. Comparison of Data Copy/Move vs Sharing

Sharing memory or passing the pointer	Physical copy or data move
Physical copy or data move	Use EDMA3
Need to manage shared access	Setup done by sending CPU
Support trough atomic instructions or hardware semaphores	Send transfer complete notification to receiving CPU
Pros and cons: • Save memory space • No latency for transfer – Access to memory could be slower – Only one CPU at time can work on data	Pros and cons: • Data can be moved to fast access memory • Local buffer could be freed (data move) – Transfer latency

A.3 Notification Scenarios

This topic is discussed using the very simple example of two cores working on the same data.

The first approach to this example uses a single fixed-memory location that is accessed by both cores in a strictly interleaved fashion. Here the inter-core interrupts are used to signal the availability of the buffer to the other core immediately after. In case the cores need random access to the buffer, it would be perfect to use the indirect or combined mode of the hardware semaphore.

As a second approach to the example above, copy the original data to a second location so that both cores can work in parallel. The question here is if the data can be independently worked from an application point of view. The notification that the data is ready can be achieved by an inter-core interrupt in case the processed data needs to be merged in the end.

Another way to approach this would be to move the buffer from the first CPUs local address space to the second CPUs local address space using the EDMA3 or QDMA. That way, accesses to the buffer will always be fast and the buffer in the first CPUs local address space can be freed up. This proposal would be favorable if the data can be processed in a purely sequential manner.

A.4 Using Memory Protection

Memory protection is not only useful to manage access, but can also be a very useful tool for debug. For example, it could be used to protocol all accesses to a certain memory region. If a region is observed by the memory protection module, but no one is allowed access, all access to that region would end up as an exception. By reading the Memory Protection Fault Status Register (MPFSR) and saving the content to a buffer, all access requesters and the type of access can be recorded. However, this might not be practical in case the access rate is very high. For that reason it would be recommended, for example, to narrow down the access requestor's that should be looked at.

A.5 Proxied Memory Protection

Proxied memory protection allows an EDMA3 transfer programmed by a given CPU to have its permissions travel with the transfer through the transfer request and also through the EDMA3 transfer controller for the read and write transfers to the source and destination endpoints. The PRIV bit (supervisor versus user) and PRIVID bit in the channel options parameter (OPT) is set with the EDMA3 programmer's privilege value and privilege ID values, respectively, when any part of the parameter RAM used by the EDMA (PaRAM) set is written. These options are part of the transfer request submitted to the transfer controller so that the target endpoints can perform memory protection checks based on these values.

Consider a parameter set programmed by a CPU in user privilege level for a simple transfer with the source buffer on an L2 page and the destination buffer on an L1D page. The PRIV is 0 for user level and the CPU has a privilege ID of 0.

The PRIV and PRIVID information travels along with the read and write requests issued to the source and destination memories. For example, if the access attributes associated with the L2 page with the source buffer allow only supervisor read, write accesses (SR, SW), the above user level read request would be refused. Similarly, if the access attributes associated with the L1D page with the destination buffer allow only supervisor read, write accesses (SR, SW), the above user level write request would be refused. For the transfer to succeed, the source and destination pages should have user read and user write permissions, respectively, along with allowing accesses from a PRIVID 0. Because the programmers privilege level and privilege identification travel with the read and write requests, EDMA3 acts as a proxy.

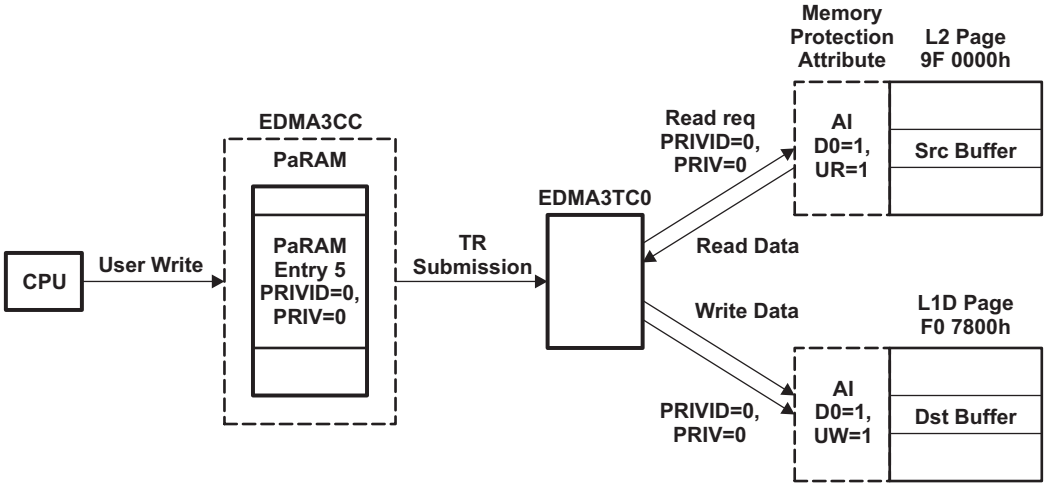

Figure A-1. Proxied Memory Protection

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

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

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DLP® Products	www.dlp.com	Broadband	www.ti.com/broadband
DSP	dsp.ti.com	Digital Control	www.ti.com/digitalcontrol
Clocks and Timers	www.ti.com/clocks	Medical	www.ti.com/medical
Interface	interface.ti.com	Military	www.ti.com/military
Logic	logic.ti.com	Optical Networking	www.ti.com/opticalnetwork
Power Mgmt	power.ti.com	Security	www.ti.com/security
Microcontrollers	microcontroller.ti.com	Telephony	www.ti.com/telephony
RFID	www.ti-rfid.com	Video & Imaging	www.ti.com/video
RF/IF and ZigBee® Solutions	www.ti.com/lprf	Wireless	www.ti.com/wireless

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