Interprocessor Message Passing Performance Issues for Multiple DSP-Based Applications using a Real-Time Operating System

Green, David; Sauks, M.

https://doi.org/10.4224/8914314

*Interprocessor Message Passing Performance Issues for Multiple DSP-Based Applications using a Real-Time Operating System**

Green, D.

* published in NRC/ERB-1100. NRC 45788.

National Research Council Canada
Conseil national de recherches Canada

Institute for Information Technology
Institut de technologie de l'information

ERB-1100

NRC-CNRC

Interprocessor Message Passing Performance Issues for Multiple DSP-Based Applications using a Real-Time Operating System

Green, D.

NRC 45788

Interprocessor Message Passing Performance Issues for Multiple DSP-Based Applications using a Real-time Operating System.[1]

D.A. Green[2],

Institute for Information Technology
National Research Council of Canada
Ottawa, Ontario Canada K1A 0R6

M. Sauks
ARC Internernational Inc.
515 Legget Drive, Suite 300
Kanata, Ontario, Canada K2K 3G4

Abstract

Today's advanced digital signal processors (DSPs) are supported by sophisticated development environments, high level languages, and real-time operating systems (RTOSs). As a result, the role of DSPs in systems design is changing as they assume more of the functionality previously addressed by general purpose processors.

For multiple DSP systems that run under an RTOS and which contain no shared memory, message passing is vital. In this report we examine the performance of message passing amongst DSPs that run under an RTOS. We describe different forms of interprocessor communications (IPC) that support message passing. We compare two commercial implementations involving different generations of DSPs manufactured by Texas Instruments, a TMS320C44-based system that uses chip-level communication ports and a TMS320C6x01-based system that uses board-level FIFO ports. We present a detailed comparison of message passing performance for these two mechanisms.

Keywords

Real-time operating systems, message passing, interprocessor communications, multiple DSPs.

[1] NRC No. 45788
[2] David Green, Institute for Information Technology, National Research Council of Canada, 1200 Montreal Rd., Ottawa ON Canada K1A 0R6, Tel: 613-993-6572, Fax: 613-952-0215, Dave.Green@nrc-cnrc.gc.ca

1. Introduction

Digital signal processors (DSPs) are now being widely used in multiple processor systems. To meet the ever-growing demand for new and more complex systems, DSP chip manufacturers are responding with new devices that incorporate many advanced features. These include not only performance-related features such as parallel execution units and on-chip cache but also development-related features that allow the use of high level languages, typically C and C++ for application development. The line separating general purpose processors (GPPs) from DSPs is beginning to blur. System designers who in the past may have opted for hybrid GPP/DSP implementations, may now consider all-DSP implementations.

Multiple DSP systems like multiple GPP systems can benefit from the use of a real-time operating system (RTOS). An RTOS provides the infrastructure that supports the meaningful partitioning of the application into tasks, the management of tasks (creation, scheduling and destruction), and the communication between tasks. For tasks residing on different processors, and especially in the case where no shared memory is available, intertask communication relies on message passing [1].

Message passing systems have been used for some time [2] to support task-to-task communication and synchronization. Transferring messages between tasks on different processors requires some form of underlying hardware: the interprocessor communications (IPC) mechanism. In multiprocessor systems, task-to-task communication performance is vitally important and is highly dependant on IPC performance.

In this report, we will examine the IPC that was developed for two commercially available processor boards using two generations of DSP devices: the first board used TMS320C44-based (C44) DSPs, and the second, TMS320C6x01 (C6x) DSPs, both from Texas Instruments Inc. We will examine how message passing performance is affected by the underlying IPC hardware. We have implemented our system using Precise/MQX™[1], a commercial real-time operating system.

The work described here is being applied to the design of a PC-embedded, high precision 3D Laser Range Sensor System (LRSS). The prototype LRSS is currently composed of 6, C6x DSPs. Two are dedicated for low-level control of opto-mechanics and video signal processing requirements. The remainder will be used for real-time data calibration and post-filtering, geometrical 3D processing, and geometrical object feature data extraction. The PC will support the user interface, data storage and network requirements.

[1] Precise/MQX is a trademark of ARC International Inc. All other trademarks and registered trademarks belong to their respective owners

2. Message Passing

Multiprocessor systems which lack shared memory must rely on message passing for intertask communications. Blocking semantics have been developed for message passing [2] which combine communications and synchronization. Fig. 1 shows an example of message passing between two tasks on different processors. Task A on processor 1 sends a message to Task B's message queue on processor 2. In this example, the msgq_send() function is assumed to be non-blocking whereas the msgq_receive() function will block the calling task until the message transfer has completed. After sending a message to Task B's queue, Task A calls msgq_receive() and blocks awaiting an acknowledgement message from Task B. After receiving the initial message, Task B unblocks and sends an acknowledgement message to Task A's message queue. When the acknowledgement message is received by Task A, it unblocks and both tasks are ready for dispatch by their respective task schedulers. In this report we refer to the above message cycle as a "Send/Receive/Acknowledge" or SRA cycle. It represents a fully synchronized or interlocked message exchange between two tasks. Of course, messages may be transferred without synchronization, i.e. without the use of the acknowledgement message; this is determined by the application.

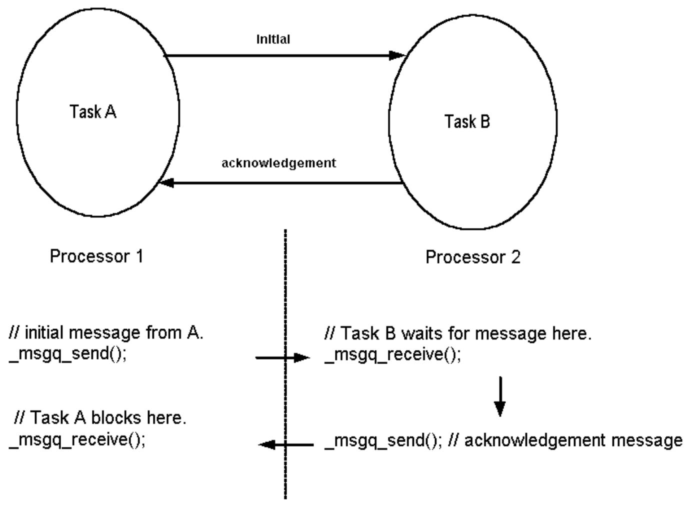

Figure 1 Message passing between tasks.

A fully interlocked Send/Receive/Acknowledge (SRA) cycle occurs when a task sends a message to its correspondent's message queue (not shown) and the correspondent replies with an acknowledgement message. A task calling _msgq_receive() may chose to 'block' until an incoming message arrives.

2.1 The MQX Implementation of Message Passing

The following discussion illustrates the implementation of message passing using MQX as an example [3]. In MQX, a message is passed to a message queue. Each message starts with a header that indicates the size of the message and contains a target *queue_id*. A *queue_id* is a unique object which identifies the destination processor and message queue for the message. Messages are allocated and freed using message pools created by the application. Any task wishing to receive messages will create a message queue during task initialization and will explicitly call the *_msgq_receive()* function in order to receive an incoming message from that queue. A task wishing to send a message obtains a message from a message pool using *_msg_alloc(),* fills in the information, then sends the message using *_msgq_send()*. The operating system kernel will adjust the states of the communicating tasks, put the message on the target message queue and prepare the receiving task for dispatch by the task scheduler. After the task has been dispatched, it can directly access the contents of the new message using the message pointer returned by *_msgq_receive()*. The task may then put the message back into the message pool by calling *_msg_free()*, or alternatively re-send the message to a different queue.

Multiple tasks running on a single DSP will exchange messages directly without the use of IPC. In this simple case, optimizations can be employed to reduce the execution time of message passing. Optimization may include the queuing and dequeuing of messages without having to copy the message.

For message passing between tasks on adjacent processors, IPC is required in order to transfer the message. Access to the IPC mechanism is embedded within the operating system and is completely transparent to the user. The function call used to send the message is identical to the single processor case. The only difference is that the message header indicates a target message queue on a remote processor. The IPC mechanism is used to relay the message from the source task to the destination queue on a different processor.

When MQX determines that the message being sent is not intended for a queue on the local processor, it places the message onto an IPC message queue (Fig. 2). This causes the IPC to initiate a message transfer by requesting access to the communications channel. The IPC mechanism then transfers the contents of the entire message onto the communications medium, and frees the local message.

In preparation for receiving messages from the communications medium, the IPC allocates a message from a pool of messages created by the IPC service at initialization time. When a message transfer actually occurs, the IPC driver copies the incoming message from the communications medium directly into a message object. When the IPC transfer completes, this newly copied message is then sent to its destination queue as if the message had been created locally.

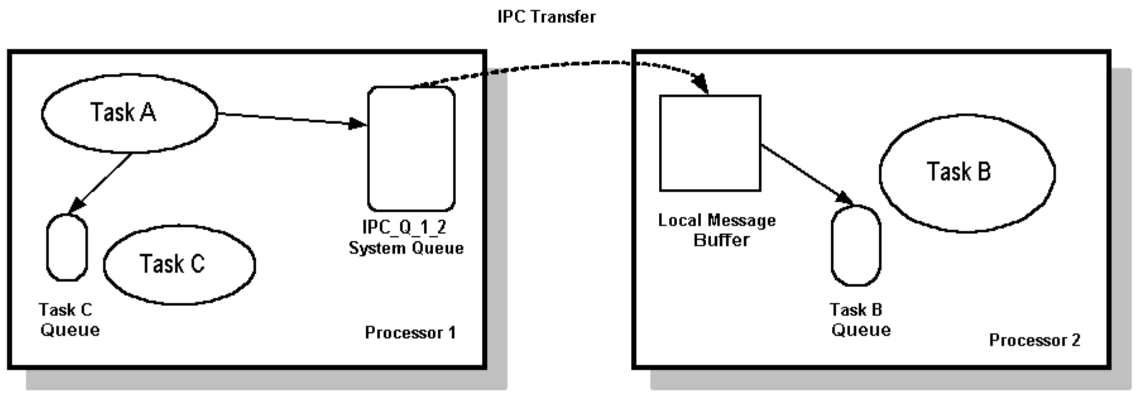

Figure 2 Interprocessor message transfers.

Task A can send messages directly to the local message queue of Task C. However messages sent from Task A to Task B's message queue require interprocessor communications services. System queue IPC_Q_1_2 handles messages on processor 1 that must be routed to remote processors.

In the case of message passing between tasks on non-adjacent processors (Fig. 3), a routing mechanism will relay the message through a sequence of IPC message queues on intervening processors until the destination processor is reached.

Initialization of system-wide message routing uses a routing table. For each processor, this table contains a list of entries for all possible correspondent processors. Each entry collects the correspondent processors into groups that share a common IPC route. For example, in Fig. 4, all messages from processor 1 intended for processors 2 through 4 will be routed to processor 2. Messages intended for processor 5 will be directly routed to a queue on processor 5.

3. Interprocessor Communications (IPC)

IPC has generally been implemented using shared memory, local networks, serial communications links or FIFO ports, Fig. 5. The performance of shared memory systems is limited by the multiprocessor bus bandwidth. As processors are added to the system, message traffic may overload the bus and performance may degrade. In cases where bus bandwidth is not a limiting factor, shared memory systems offer the advantage that globally accessible data are directly

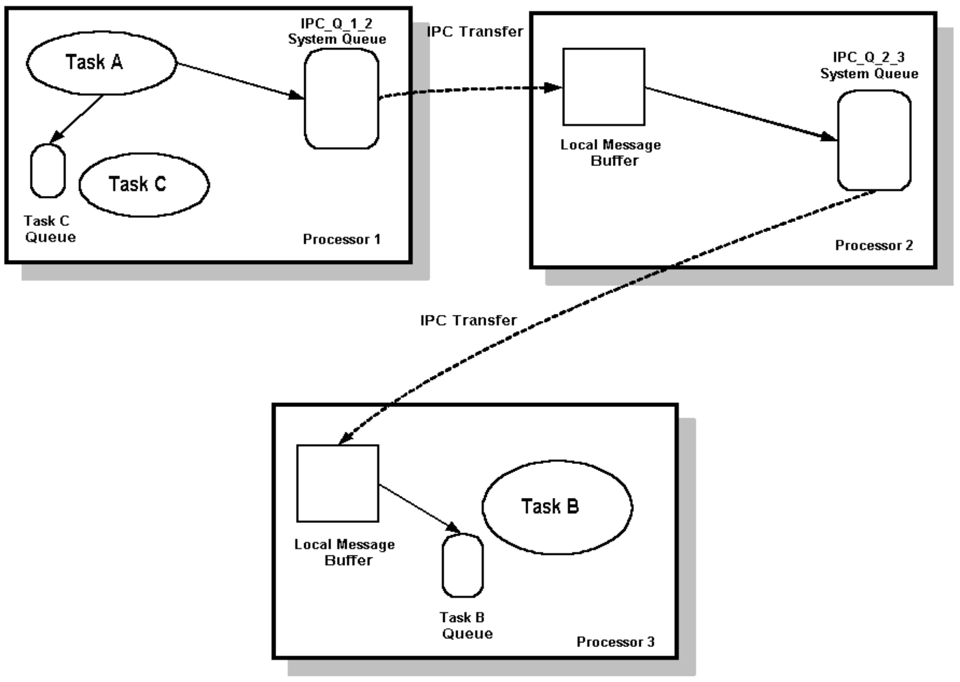

Figure 3 Interprocessor message routing.
Interprocessor message routing conveys a message from the source task, Task A to the target message queue of Task B using a sequence of IPC transfers.

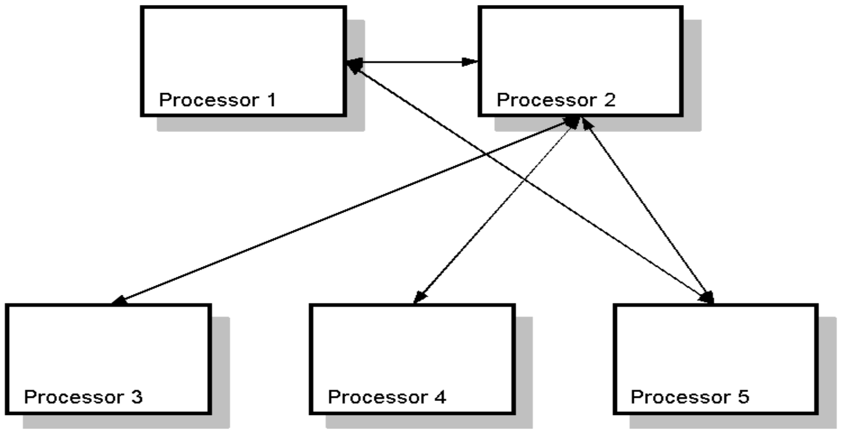

Figure 4 An example of interprocessor message routing.

Processor 1 can only communicate directly with processors 2 and 5. Messages sent from processor 1 to processors 2 to 4 are routed via processor 2. Processor 2 can communicate directly with all processors.

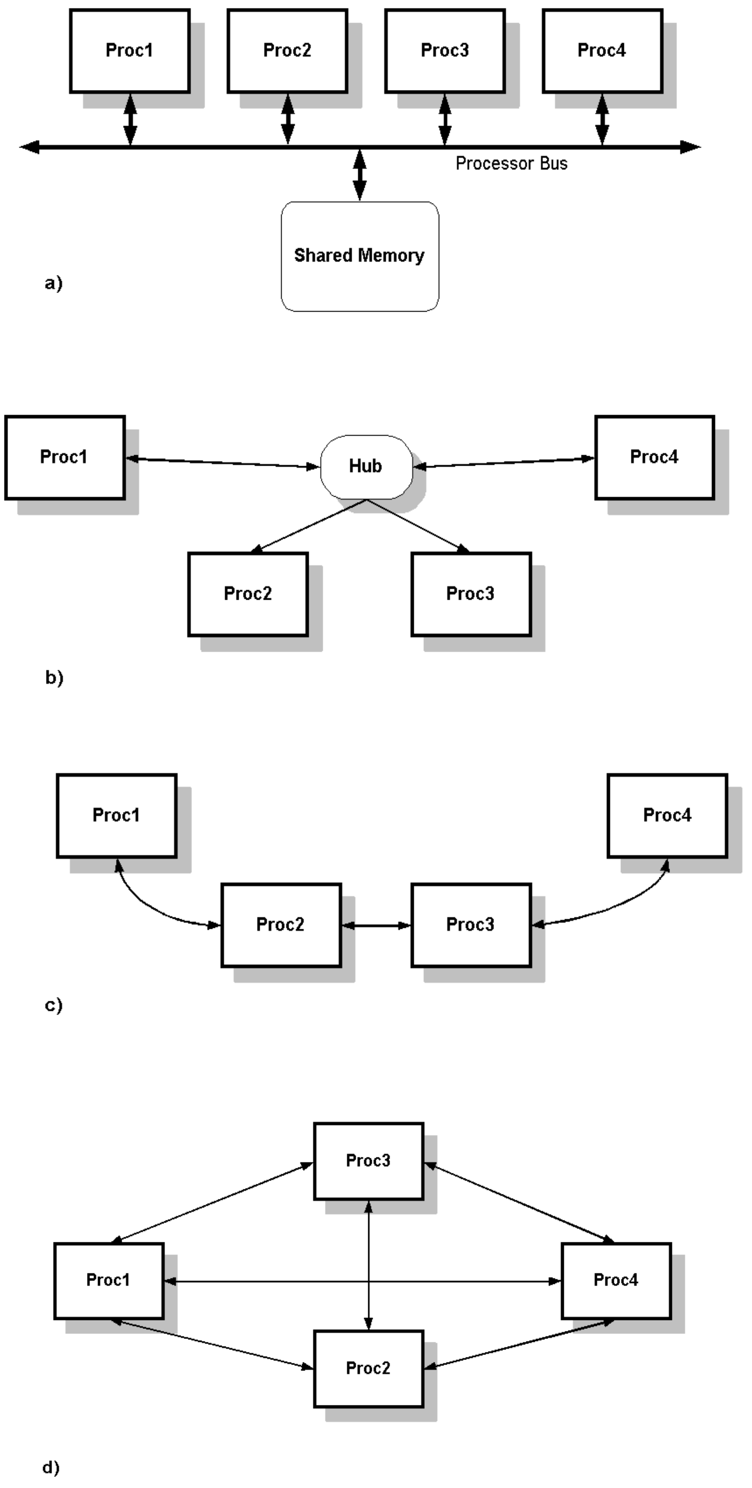

Figure 5 Interprocessor communications alternatives.

Common implementations of interprocessor communications: a) shared memory using a common bus, b) local network, c) bit-serial communications with point-to-point connections, and d) dedicated interconnection topology using communications links or FIFO ports.

available to every task. Instead of passing data from task to task, pointers may be efficiently transferred using messages.

In the past, hard real-time systems have avoided networked multiprocessors for deterministic reasons. Bit-serial communications links use dedicated point-to-point communications links but offer relatively low data rates (e.g. in the 2 Mbits per sec range). FIFO communications links have been directly integrated into the Texas Instruments TMS32C4x DSP series, and the Analog Devices ADSP-2106x SHARC DSP series. These links offer transfer rates in the order of 20 to 100 Mbytes/s and may be configured to produce arbitrary point-to-point interconnection topologies. These First-In-First-Out ports (FIFO ports) use a combination of FIFO buffers, direct memory access (DMA) controllers and fixed interconnections to provide dedicated high-speed links between processors.

Of the four mechanisms described, shared memory and FIFO port IPC mechanisms offer the highest performance. While FIFO port IPC systems, unlike shared memory systems, need to explicitly transfer data between remote tasks, the interconnections are immune to bus loading. This can be a significant factor in system design.

3.1 Interprocessor Communications (IPC) on the C4x and C6x

This section will examine in some detail message passing and IPC performance using byte-serial communications links and FIFO ports. The “Comm Ports” to be analyzed are the communications ports found on the Texas Instruments TMS320C4x DSP (C4x). The FIFO port example will be based on a board-level design developed by Innovative Integration Inc. that uses a TMS320C6701 DSP (C6x). The C6x provides no Comm Ports.

3.1.1 IPC Using TMS320C4x Comm Ports

The C44 DSP contains four Comm Ports [4]. These bidirectional ports provide an 8-bit internal data path and four control signals. Comm Ports on neighbouring devices can be directly interconnected without any intervening logic. Each Comm Port is equipped with an 8-deep, 32-bit wide FIFO buffer and can be directly synchronized with an on-chip DMA controller. Flag logic can be used to trigger DMA transfers into or out of a Comm Port without any CPU intervention.

The “PCI44” DSP “carrier” board used in this project was manufactured by Innovative Integration Inc. [5]. This board, which can be installed in a desktop PC, can be populated with three DSP daughter boards known as “TIM modules”. Each TIM module used in this system contained a 50 MHz C44 DSP and was connected to each of its two neighbours. The actual interconnection topology for the PCI44 was fixed by the manufacturer, Fig. 6. Each C44 shares a single communications channel with each of its two neighbours. Furthermore, each C44

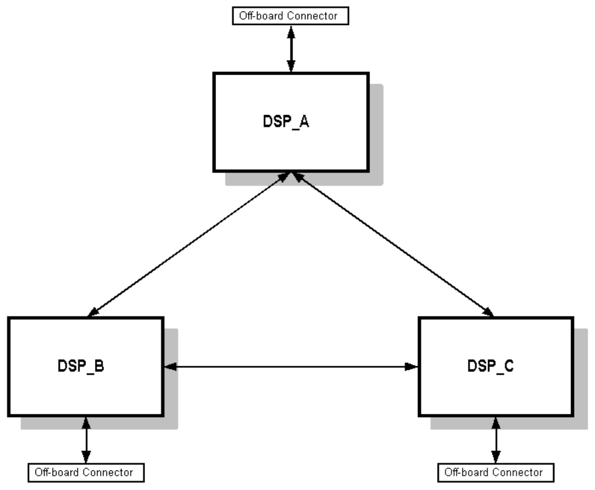

Figure 6 PCI44 communications links.

Byte-serial communication port configuration for the Innovative Integration PCI44 DSP carrier card. A single bidirectional port links each DSP to its neighbour. Off-board connections allow for system expansion.

is connected to an off-board connector for system expansion. In our system, the fourth Comm Port has not been allocated.

To implement the most efficient IPC, a pair of Comm Ports with associated DMA controllers should be assigned to each processor pair in full duplex fashion - one dedicated to data traffic in each direction. Because of the fixed interconnections on the PCI44 described above, this is not possible. Instead it was necessary to rely on a shared communications strategy in which the Comm Ports, at each end of a link, switch between sending and receiving messages as required. Clearly, performance using a shared Comm Port is less than the full duplex case as a communications-level protocol must be employed to negotiate access to the shared link and to resolve any access conflicts that might arise.

Synchronized DMA transfers are used on the PCI44. With synchronized DMA transfers, an event is used to trigger each read or write element transfer within a DMA sequence without any CPU intervention. A write transfer is triggered when the output buffer becomes not full and an input transfer is triggered when an input buffer becomes not empty. In this manner, each full transfer sequence is automatically adjusted to match the length of the current message. When the transfer is completed, an interrupt occurs at each end of the link and the two state machines return to an initial state in preparation for the next transfer.

The IPC transfer is initiated when a message arrives at an IPC message queue (Fig. 2). A handshake takes place with the receiving processor to detect and negotiate any pending access conflicts. The actual data transfer is then accomplished using DMA. The conclusion of the transfer is identified by a DMA interrupt at each end of the link, and the state machines are restored to their initial state.

3.1.2 IPC Using TMS320C6x01 and External FIFO Ports

The processor board chosen for this work is a "Quatro67" manufactured by Innovative Integration [5]. This board contains four 160 MHz TMS320C6701 DSPs that are fully interconnected by bidirectional 32-bit FIFO ports (Fig. 7); each processor can directly communicate with each of its three neighbours. In addition, three of the DSPs also offer 16-bit off-board links for system expansion. We will only consider the on-board FIFO ports.

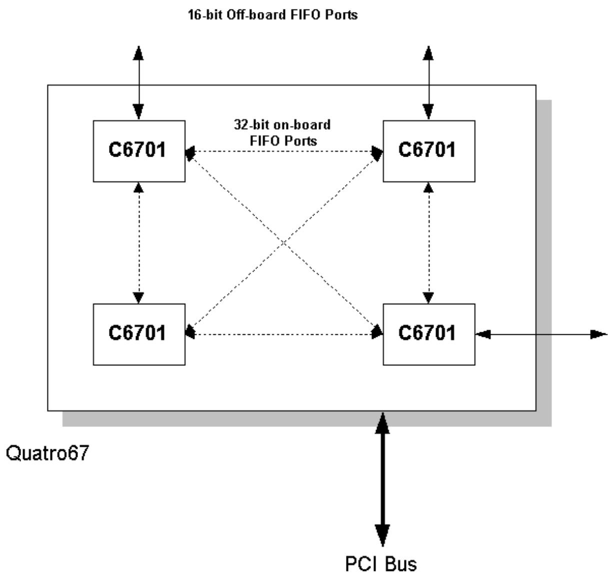

Figure 7 Quatro67 communications links.

Dedicated high-speed FIFO-based links used on the Quatro67. The processors are fully interconnected and three ports are available for off-board expansion.

Each FIFO port contains a 512 x 32 FIFO at the receive end of each link (Fig. 8). FIFO status indicating empty, almost full, and full conditions are available to the receiver as Rx_EMPTY, Rx_AF and Rx_FULL flags. The remote processor, is able to observe the same status as Tx_EMPTY, Tx_AF, and Tx_FULL flags. These flags are available for either polling or for interrupt generation purposes.

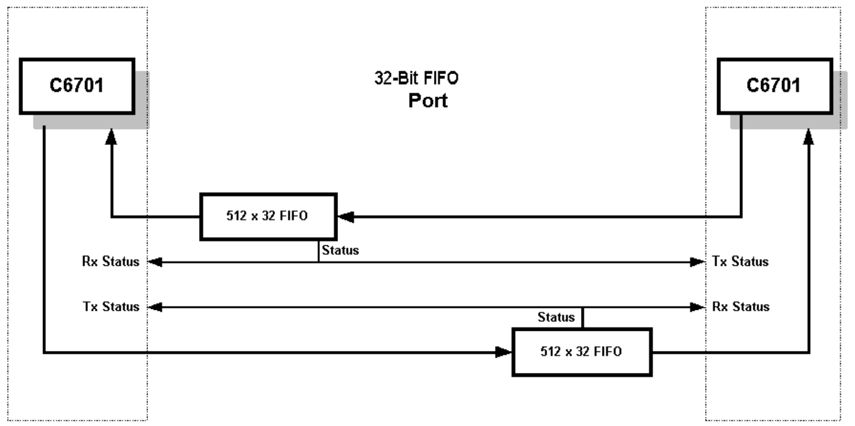

Figure 8 Quatro67 FIFO port communications.

Bidirectional FIFO Ports used on the Quatro67 DSP board. FIFOs appear at the receive end of each link. FIFO status is returned to the receiver (Rx status) and to the transmitter (Tx status) for each FIFO. Status can be used for polling or interrupt generation purposes.

As was the case with the Comm Ports, each FIFO Port must be paired with a DMA controller for efficient data handling. The most efficient implementation requires two dedicated DMA controllers at each end of each link, one for receiving and one for transmitting (Fig. 9). For the Quatro67, three of the processors must be able to communicate with four FIFO ports (three on-board neighbours and one off-board port, see Fig. 7). This requires 8 DMA controllers for an efficient implementation. Since the C6x01 only provides four DMA controllers it becomes necessary to share a DMA controller for both sending and receiving (Fig. 9b). Newer members of the C6x family of DSPs now offer additional DMA controllers.

Interrupts are used as a means to efficiently activate the IPC mechanism. Ideally a pair of interrupts should be allocated to each FIFO port: a “receive” interrupt to signal the arrival of a new command during IPC startup, and a “transmit” interrupt to signal when a new command or acknowledgement may be issued. While a full interrupt implementation capable of handling four FIFO ports requires eight interrupts, our system hardware limited us to select no more than four. We chose the four “receive” interrupts to signal incoming commands and rely on polling to determine when outgoing command transfers can be initiated. We did not attempt to measure the impact of this limitation.

3.2 DMA Transfers

Three forms of DMA transfers were examined and compared for IPC implementation: burst transfers, read/write synchronized transfers, and frame synchronized transfers.

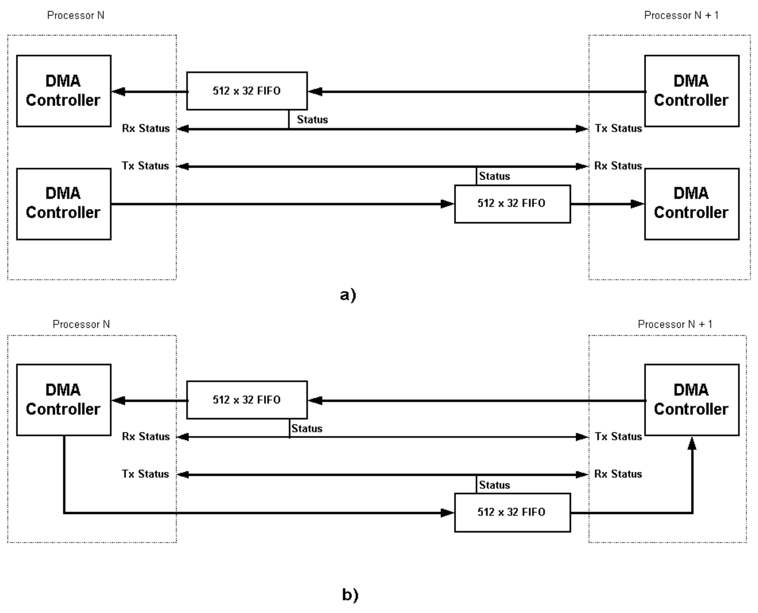

Figure 9 FIFO ports and DMA controllers.

DMA controller access to the FIFO ports: a) the highest performance is achieved by dedicating 2 DMA controllers to each end of each FIFO port, b) with only four DMA controllers available on the C6x01, each controller has to be shared for both sending and receiving messages. Access conflicts are avoided by a link negotiation handshake.

The IPC execution is triggered by an interrupt associated with the startup preamble. The data to be transferred are accessed directly by the IPC code. In burst mode, messages that are shorter than the FIFO buffer size (512 x 32 bits), are sent in their entirety in a single DMA burst. The length of the burst is adjusted to the length of the short message. Long messages that exceed the length of the buffer are partitioned into half-buffer-sized packets that are sent sequentially.

For read/write synchronized DMA transfers, the DMA controller at each end of a FIFO Port can be configured to automatically pace each transfer using an external DMA event. This allows long DMA transfers to occur with no CPU intervention. In this mode, each data element is transferred in response to an event.

For frame synchronized DMA transfers, the DMA controller at each end of a FIFO Port is configured to automatically transfer a complete frame of data containing a specified number of data elements using an external DMA event. In this mode, a complete data frame is transferred in response to a single event.

4. IPC Performance: C4x vs. C6x

We performed measurements of message passing performance on both systems using short (24 byte) and long (4092 byte) messages. We began by measuring the performance of message passing between two tasks running on a single processor. This provides a baseline measurement without IPC overhead. The measurements were repeated for the case where the tasks reside on two adjacent processors.

Tables 1 and 2 summarize the message passing performance using 50 MHz C44 DSPs and 160 MHz C6701 DSPs. The tables list the time required to send a message and then to receive an acknowledgement from a correspondent task. We refer to this as a Send/Receive/Acknowledge (SRA) cycle that measures the end-to-end cost of sending a message and then receiving an acknowledgement message. In the single processor case message passing on MQX only passes a pointer to a message, data are not actually transferred. This is evident from the similar performance for the short and long message transfers on a single processor.

	Short Messages (Except as noted.)		
	1 DSP	2 DSPs (SRA)	2 DSPs (Estimated IPC)
C44 R/W Sync	64 μs	184 μs	60 μs
C6701 Burst	49 μs	209 μs	80 μs
C6701 R/W Sync	49 μs	167 μs	59 μs
C6701 Frame Sync	49 μs	255 μs [1]	103 μs [1]

Table 1

A summary of IPC performance for C44 and C6701 DSPs using short, 24 byte messages. Note 1. This measurement is for a 1 kB packet transfer.

For the two processor case, we can estimate the incremental cost of sending a message using IPC by subtracting the message passing overhead (single processor SRA cycle time) from the two processor SRA cycle time and dividing by two (the SRA cycle time contains two message transfers). We can also estimate a meaningful IPC rate for long message transfers by dividing the message size (4092 bytes) by the IPC time. The results appear in Table 2.

	Long Messages			
	1 DSP	2 DSPs (SRA)	2 DSPs (Estimated IPC)	Estimated IPC Rate
C44 R/W Sync	71 µs	692 µs	310 µs	13.2 MB/s
C6x Burst mode	48 µs	642 µs	297 µs	13.8 MB/s
C6x R/W Sync	48 µs	1083 µs	517 µs	7.9 MB/s

Table 2

A summary of IPC performance for C44 and C6x DSPs using long, 4092 byte messages.

For comparison purposes, Table 3 summarizes the performance of just the message send and message receive function calls. IPC makes extensive use of DMA transfers that largely take place independent of processor execution. While DMA transfers are occurring, the operating system task scheduler will dispatch another ready task that will then run concurrently with the DMA operations. In other words, while the SRA cycle time is a measure of end-to-end communications between tasks, it hides the fact that the processor may continue to perform useful work during the transfer. The times listed in Table 3, are a measure of how long a *processor* is occupied when sending or receiving a single message.

	Short				Long		
	1 DSP	2 DSP Burst	2 DSP R/W Sync	2 DSP Frame Sync	1 DSP	2 DSP Burst	2 DSP R/W Sync
Send	7 µs	70 µs	68 µs	84 µs	7 µs	121 µs	61 µs
Receive	4 µs	3 µs	4 µs	2 µs	4 µs	3 µs	4 µs

Table 3

Performance of C6701 for simple message send and receive operations for short (24 byte) and long (4092 byte) message transfers.

5. Discussion of IPC Performance

5.1 Raw DMA performance of the C6x

Before measuring IPC behaviour, the performance of basic DMA transfers was studied for the C6701 using three types of memory that are found on this design. The memory types included on-chip internal data memory (IDM), off-chip synchronous dynamic RAM (SDRAM) and off-chip synchronous burst static RAM (SBSRAM).

In each case, a 1 kB DMA transfer was initiated between selected memory and a FIFO Port. The performance of both memory read and write DMA transfers was measured. The results are presented in Table 4. From the table, it is clear that off-chip DMA rates offer less than half the performance of on-chip memory.

160 MHz TMS320C6701 DMA Performance			
Memory Type	Read Rate (MB/s)	Write Rate (MB/s)	Avg. R/W Rate (MB/s)
On chip	100	100	100
SDRAM	38.5	52.6	45.6
SBSRAM	40	52.6	46.3

Table 4

Raw DMA performance of the C6701 using internal and external memory. “Avg. R/W Rate” is the average of the read and write DMA rates for each memory type. Access to on-chip memory is about 2 times faster than off-chip memory.

5.2 C44 vs. C6701 – IPC Using Read/Write Synchronized DMA Transfers

The IPC developed for MQX using the C44 Comm Ports uses read/write synchronized DMA transfers. From Table 1, the estimated short message IPC times for the C44 and C6701 processors is about the same: 60 µs. For long messages, the C6701 implementation is significantly slower: 517 µs vs. 310 µs for the C44. This performance reflects the highly efficient design of the Comm Ports on the C44 [4] and in particular, the use of level-triggered interrupts.

5.3 C6701 IPC Using Burst Transfers

DMA burst transfers with the C6701 achieve IPC transfer rates of 13.8 MB/s. This is only slightly faster than the 13.2 MB/s that the C44 achieves using Comm Ports with read/write synchronization.

Burst transfers are explicitly triggered by a function call. The message is partitioned into half buffer-sized packets that are transmitted one at a time. A DMA interrupt is generated at the conclusion of each packet transfer. If more packets are to be sent, the DMA interrupt handler will poll the transmit FIFO's half-full flag and will initiate the next transfer when the FIFO becomes less than half full.

The receiving DSP receives an interrupt when the receive FIFO becomes half full. The interrupt service routine calls a function to transfer the contents of the FIFO to memory using DMA. At the conclusion of each packet transfer, a DMA interrupt is taken by the receiving DSP and the interrupt is discarded.

Therefore each outgoing packet transfer using the burst transfer mechanism requires the servicing of one DMA interrupt as well as a polling loop to determine when the next DMA transfer can be started. For the receiver, two interrupts must be serviced for each incoming packet.

In summary, burst transfers require extensive interrupt handling, the use of explicit function calls to start each send and receive DMA transfer per packet, and polling to decide when the next outgoing packet transfer can be started. While the DMA transfers are fast, the overhead with this method is high.

5.4 C6701 IPC Using Frame Synchronized Transfers

We investigated frame synchronized transfers with hopes of achieving optimum IPC performance. The objective was to combine the high performance of burst DMA transfers with the low overhead of synchronized DMA transfers.

An outgoing frame synchronized DMA transfer requires a single function call that initializes the DMA controller with the frame and element counts. In this case, the message is divided into an integral number of frames, each containing a half buffer-size (1 kB) number of elements. Each frame contains a single packet. Since message size is arbitrary, the final packet will almost certainly contain padding, but this presents only a minor overhead. The corresponding incoming DMA controller must be initialized in a similar manner.

Each DMA controller requires a synchronizing event to trigger the next DMA frame transfer. In this implementation only the FIFO almost-full flag, initialized to just below the half-full condition, was available for DMA synchronization. An outgoing frame synchronized transfer is started explicitly by the function call. An incoming synchronized transfer at the receiver is triggered by the half-full condition. Thus, single frame messages can be transferred. Table 1 indicates that the SRA cycle time for a single packet frame synchronized transfer is 255 μs. Note that this represents a 1 kB packet, not a 24-byte packet as is used for the other entries in Table 1. The equivalent performance for burst transfers of 1 kB is 291 μs. It is apparent that frame synchronized transfers offer a significant savings due to reduced overhead when compared to burst transfers. Unfortunately the existing hardware for handling DMA events does not support additional outgoing frame transfers, that are required for long messages.

While we were unable to measure the performance of long message transfers using the frame synchronized DMA method, it is possible to provide an estimate. The measured SRA cycle time of a 1 kB message using frame synchronized transfers is 45 μs faster than when burst transfers are used. Therefore a four frame transfer, consisting of 4092 bytes, will see an overall improvement in the SRA cycle time of about 4 x 45 = 180 μs. The expected SRA cycle time is therefore about 642 – 180 = 462 μs. The expected long IPC time is therefore (462 – 48)/2 = 207 μs. This is about 30% faster than the measured IPC rate of the burst transfer mode and corresponds to an effective IPC data rate of 19.8 MB/s.

6. Conclusions

This report has described in some detail our experiences with the implementation of interprocessor communications in support of message passing in a real-time, multiprocessing operating system. We have demonstrated how the use of dedicated data links based on either byte-serial communications ports or high-speed FIFO-based ports can support intertask communications. For the C6x board, we measured raw DMA transfer rates to external memory (Table 4) to be approximately 45 MB/s whereas the maximum estimated IPC rate is close to 20 MB/s or about 43% of the maximum DMA rate. For the C4x, the specified data rate on the Comm Port is 20 MB/s. We measured an IPC rate of 13.2 MB/s which is about 66% of the Comm Port data rate.

FIFO port IPC performance could be improved by changes to the DMA event handling hardware, by using dedicated, send/receive DMA channels, and by modifying the RTOS kernel to allow the use of on-chip memory for message buffers.

In a non-shared memory system, message passing is the only way for tasks on remote processors to communicate and to exchange data. There are some interesting implications on systems design of these results. For example, applications that generate a large amount of message traffic may become message bound particularly if long messages are being transferred between tasks on different processors. In this situation it is better to cluster, as much as possible, communicating tasks onto a common processor and to limit interprocessor message passing to task synchronization and to short messages. This helps minimize the impact of IPC on overall system performance.

7. References

1. Gauthier, C., “Design of a Thinwire Real-time Multiprocessor Operating System”, AGARD Conf. Proceedings 545, Aerospace Software Engineering for Advanced Systems Architectures, Paris France. May 10-13, 1993. pp. 18-1 – 18-10.

2. Gentleman, W.M. “Message Passing Between Sequential Processes: the Reply Primitive and the Administrator Concept”, Software - Practice and Experience, Vol. 11, No. 5, May 1981, pp. 435 – 466.

3. Precise/MQX User’s Guide, Precise Software Technologies Inc. July 1999.

4. TMS320C4x User’s Guide, SPRU063B, Texas Instruments Inc. March 1996.

5. http://www.innovative-dsp.com.