Designing Real Time DSP systems - What are the crucial issues?

Peter Warnes, Technical Director, HUNT ENGINEERING (UK) Ltd.

Steve Bradshaw, President, TRAQUAIR Data Systems, Inc.

July 9th, 1998

Copyright © 1998, Hunt Engineering (UK) Ltd.

Copyright © 1998, Traquair Data Systems, Inc.

Abstract

The issues involved in the system level design of real time DSP systems, particularly high performance multiple processor systems will be outlined. We take a detailed look at the system requirements for I/O and inter-processor communications hardware, looking at the common methods of implementing such systems.

It will be seen that recognisable methods do not actually satisfy our requirements and we need to look further for a better solution.

The novel ideas that we end up with can provide groundbreaking improvements for DSP system design and performance, for a broad range of low medium and high bandwidth applications.

We then consider how such a communications architecture reflects on the other important issues in the system design.

Hunt Engineering (UK), Ltd.,
Chestnut Court, Burton Row,
Brent Knoll, Somerset, TA9 4B9, UK
Phone: (01278) 760188
Email: pete@hunteng.demon.co.uk

Traquair Data Systems, Inc.
114 Sheldon Road,
Ithaca, NY 14850, USA
Phone: (607) 266 6000
Email: sjb@traquair.com

1. INTRODUCTION

Why are we presenting this paper?

The authors have been working in the real-time DSP systems business for many years, providing systems that need to deal with the applications problems of real customers. These customers are from a varied background but without exception they are producing real solutions to real problems.

This places the following discussion into context: **we have an interest in real world solutions**.

Today we provide these solutions based on one (TMS320C4x Comport based) architectural principle, which has allowed us to successfully deliver multiple processor DSP systems in a wide variety of (inter-compatible) board level implementations for over six years.

We see radically new processor architecture like the TMS320C6000 family and wonder what is necessary to make a system that can utilise the new levels of performance it brings.

We have an interest in finding a system architecture that will serve us as well in the future with different DSPs as they become available.

2. REAL TIME

What does real time mean?

People seem to use the term "real time" more and more these days, often without really knowing what they mean. "I have to do my processing in real time" is often heard, which really means 'at the same rate as my data flow' or perhaps 'not off-line'.

This is actually closer to the truth than the definition that most people would give if asked, which is "really fast".

For this paper we will use my personal definition of "real time" which is **deterministic**. We could expand this to include 'with enough bandwidth for the application' or 'with low latency', but these can be covered separately from the use of "real time".

3. DSP

What is a DSP system?

It seems silly to ask such a question, but we need to be clear about the type of systems we are considering. Anybody can tell you what DSP stands for, but it is not always considered that a DSP needs **Digital** **Signals** to **Process**.

Such a system must have inputs and outputs, and these I/Os must be generated by real world events. Until there is both input and output, there is no processing to do.

4. REAL TIME DSP SYSTEMS

What are the issues?

In a system like we have been describing there are a number of important issues that need to be considered, and unfortunately they all interact with each other. The main categories that we will use are: -

1. DSP Processor Chip
The actual processing power of the system, fixed or floating point, 16 bit 32 bit….
2. System Software strategy
The programming model that you will use, single tasking/multi-tasking, SIMD/MIMD…
3. I/O interfaces
An I/O bus type or dedicated A/D, D/A etc….
4. System Communications strategy
How the processors and I/O will transfer data, how the programming model will pass Control flow ……

We need to start the search for an architecture that supports what we want to do today and what our customers will ask us to do tomorrow. It is important for our customers that anything we do will cope with future DSP technology and system requirements so that they can continue to benefit from the clear progression path that we can provide them.

To this end, we must first abstract away from the specific details of particular DSP processors, board level designs, and products, and to focus on the real issue of what sort of system level support is actually needed when attempting to utilise multiple DSP processors efficiently in real-time applications.

This means that we will not discuss items 1 or 3 in at this time.

There is a wealth of information within the computer industry, resulting from the significant commercial use and experimentation of a wide variety of multiple processor systems for well over a quarter of a century.

The system software strategy is something that can be specifically chosen for a particular application. Standard texts address the issues involved in this field with reference to the general computing environment, the same theories can be applied to real time DSP systems so it is not interesting for us to address them here.

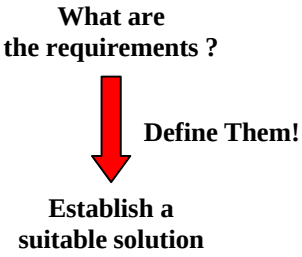
This leaves the System Communications Strategy for us to talk about. The DSP industry is a relatively new player on the multi-processor block, perhaps only seriously committing to multiple processor architectures as a result of the introduction of some explicit technology support - in the form of the TMS320C40 processor by Texas Instruments.

Blindly following multiple processor architectural techniques which have worked for general computing problems with no real-time operational constrains can however be quite hazardous to a development project. Just because there is familiarity with particular architectures, doesn't mean they are appropriate for real-time applications. It is essential that whatever architectures are utilised for supporting real-time DSP related applications, are capable of accommodating the needs of these applications.

The System Communications Strategy is one key piece of the system that everything depends on. You can see that a bad communications strategy can render high performance DSP processors useless.

If there is no support for I/O how can a system be called a DSP system?
If there is no determinism how can a system be called Real time?

The philosophy that we use every day in our business is based on deciding what you need to solve a problem, committing that to record that we will not change, and then judging possible solutions against those requirements. It seems simple but it is all too easy to twist the original requirements to fit a particular solution unless we consciously discuss and record the original requirements.



The next part of this paper is dedicated to forming an opinion about the requirements we have for a communications system, validating our theory with some example systems, then judging the Communications strategies that we see in the marketplace today, against those requirements.

5. NODAL BASED SYSTEMS

What are they?

Whether the system has lots of processors and a small amount of I/O, or a small number of processors and lots of I/O, the system can always be considered as a system of nodes that require to communicate with each other in some way. Hence it can be a sensible strategy to consider the new concept of "nodes" where a node can be a processing node, an I/O node or even both.

6. INTER NODE COMMUNICATION

What is it?

People are usually comfortable with the idea of data flow in a system, and can usually accept the concept of control flow, especially with I/O nodes. What can sometimes be a major leap is to realise that the requirements for both of these are actually the same.

So we will begin our explorations of System Communication strategies, by trying to consider Communicating Nodes as a general case of Processors, I/O, Data flow and Control flow.

7. THE COMMUNICATION REQUIREMENTS OF A REAL TIME DSP SYSTEM

What do communicating Nodes need?

When we are dealing with "real world" data it does not magically appear correctly formatted at the node, as is often assumed when running academic benchmarks. It is the job of the communications system to format and present the data to the node in that way, so that it can be handled efficiently.

Usually "real world" data from I/O devices is a constant stream of data at some fixed rate, rather than a short high speed burst of data at the full bandwidth of the communications system. It is however not uncommon that data processing involves a block of consecutive data items in order to provide a single result. This is sometimes a swing buffer type processing that moves from one block to the next, or is sometimes a 'sliding window' of data, where when a new data sample is added, the oldest can be discarded. What is common to both of these techniques is that access to stored data samples is required.

So we can state that it is a requirement of the communications system to present the data to a node in a way that makes this possible. -- **Transfer of data in packets**

Definitely in the swing buffer case, and probably in the sliding buffer case, it is inefficient for a node to be informed about the arrival or departure of each individual data item. It is preferable that the node is informed after a "block" has been transferred. As the block size is increased, the overhead of informing the node becomes negligible. The same requirements are true of all communications in the system. -- **Transfer of data in packets with notification of completion.**
It is possible though that the node requires access to a large amount of stored data, or a small amount - perhaps even single data items. Varying block sizes in this context usually offer a trade

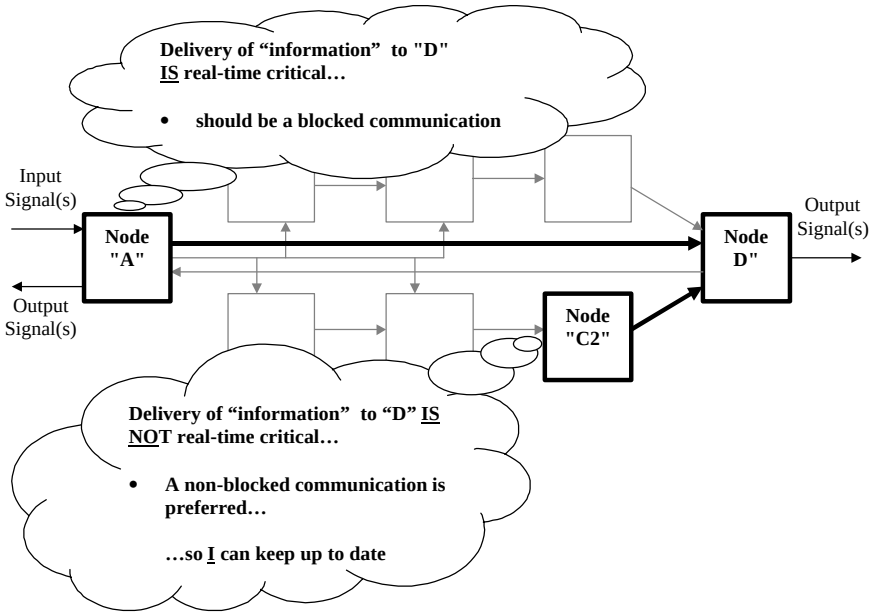
off between throughput and latency. The size of the storage available at a particular node will also need to be taken into consideration. Each system will have its own constraints for these parameters.

So we can state the next requirement is **the communications system must not impose a particular packet size on the node**

As a side issue from those requirements we can suggest that a processing node should not be constantly polling and copying data, so if possible some kind of **DMA transfer** would be desirable. *This could be less important at an I/O node as the hardware would be dedicated to transferring the data.*

We have already stated that for a real-time system, it is important that the communication system should be deterministic, i.e. it must be able to guarantee bandwidth whatever the magnitude of the requirement. It should be the case that the guarantee is independent of the application software that is running on the "system"; i.e. one node is not affected by the behaviour of other nodes not involved in the same communication. The latency is also an important issue, most applications can tolerate a latency to some extent, minimising it is important, but making it deterministic is more important. – **The communications system must guarantee bandwidth and latency**

Nobody would ever argue against the statement that the communications system must be able guarantee delivery of data. This can be described as a blocking communication. If the recipient of some data is temporarily unable to receive data, then the transmitting node should have a mechanism to suspend transmission so that no data loss occurs.
It is however possible to imagine situations where the blocking is undesirable, such as would occur with a data stream that has to be monitored infrequently, and only when a trigger condition occurs, high resolution data is captured. In this case it would be useful to have a non-blocking communication. The data pipe simply carries data all of the time, the node that is monitoring "picks" a data item or burst at regular intervals to check for the trigger condition. If a blocking communication is used, during the time that data is not being read, the system buffers will simply fill up, eventually overflowing at the 'far' end. In a non-blocking communication the data that is not being read is simply discarded, so that the next time the node "picks" a sample it is fresh. When a trigger condition is sensed the node can begin to capture data continuously for analysis. In the non-blocking situation the capture of data can begin immediately, knowing that the next sample is fresh and that there are no discontinuities caused by buffer overflow. Use of a blocking communication in this case, could mean that there are one or more discontinuities in the data where the buffer overflows. The only way to cope with this is for the application to have knowledge of the buffer size and perform a 'flush' operation by discarding the first n samples – less than ideal!
So we can form our next wish as being – **we need the choice of blocking or non-blocking communications.**



Multi-Node System highlighting Blocking and Non-Blocking communication

It is often desirable in a system that several nodes can act upon the same data set. This could be achieved by copying data and re-transmitting it, but if the design of the communications allowed this to happen, without intervention by the node, it would be a highly desirable feature. – **Multi-casting is desirable.**

If this feature were to be used to transmit data from one source to two sinks, should the transmission be blocked be either of the recipients, or one of them, or neither? This can only be answered by intimately knowing the details of the application, so any of those schemes should not be imposed by the communications system. This reinforces our statement that we need the choice of blocking and no blocking communications.

The communications system should be **simple to use**. This means that its use is not just efficient but also that it fits the application requirements rather than imposing constraints on the application. Having a full feature set that is hard to program will prevent users from utilising all of the features provided.

It has also been assumed all along but not explicitly stated that the communications system should be able to provide a high sustainable throughput at a **viable system cost**. This does not mean cheap, but more having an acceptable price/performance ratio.

Now we can make two lists of requirements, the first a "must have" list:

- **Transfer of data in packets with notification of completion.**
- **the communications system must not impose a particular packet size on the node**
- **the communications system must guarantee bandwidth and latency**
- **we need the choice of blocking or non-blocking communications.**
- **simple to use**
- **viable system cost**.

And a second list of "would like" items:

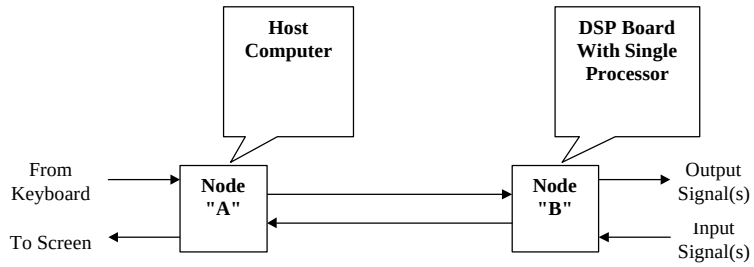
- **Use of DMA at processing nodes**
- **multi-casting is desirable.**

Having produced this list of requirements we should validate them against some recognisable systems:

7.1 A simple communicating Multi-node system

Most early experiences with multi-node DSP systems are with simple DSP cards, having a single DSP chip, plugged into a host computer. In this system the host computer is one node, and the DSP chip is another node.

The host computer is probably used to provide input from the user and output to the user either via the screen or the file system. This makes this node a Processing and I/O node. If the DSP has some kind of I/O it is usually one or more memory mapped A/Ds and one or more memory mapped D/As. This makes the second node also a Processing and I/O node. We can test our list against this system:



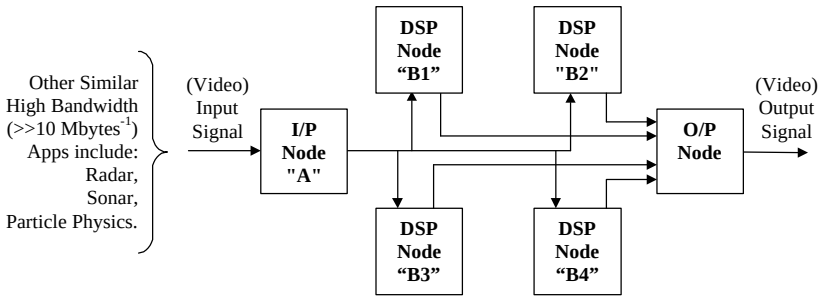
Two-Node System (Single Processor DSP Board in Host Computer

It can be seen that even with the simple system, having only two communication possibilities most of the requirements are desirable depending on the needs of the actual application. The only exception is the multi-casting, which is clearly not useful.

7.2 A multi-processor system with single input and single output nodes

This type of system could just as easily be a control system, a data storage system or many others.

For the purposes of this illustration, lets assume a system that enhances images in real time. It has a camera providing input images, and a display for outputting processed images. Perhaps several processors are needed to process the images. Consider a four processor configuration, with each responsible for processing a quadrant of the image.
If we assume that the image input and output devices do not contain any processing capability, this system has two I/O nodes and four processing nodes.

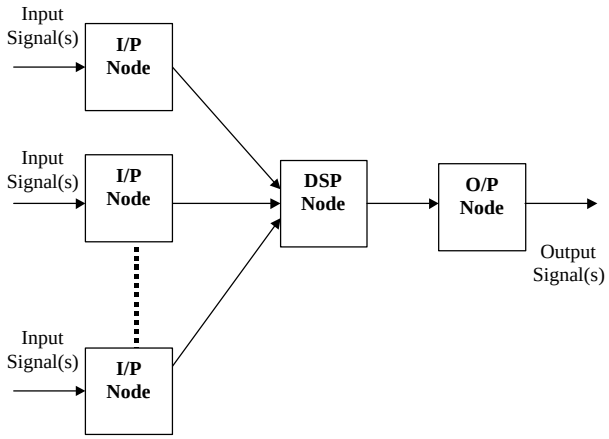


Multiple Processor System with single input and output nodes

We can test our list against this system and find that all of the requirements are used, particularly the multi-casting.

7.3 A single processor system with multiple I/O nodes

This type of system could just as easily be a control system, a data storage system or many others, but for the purposes of this illustration lets assume a system that is monitoring safety levels of multiple inputs. It has several input nodes all of which feed data to a common processing node.
This processing node could be comparing inputs, or testing each input for a spurious signal. This system has multiple I/O nodes and a single processing node.



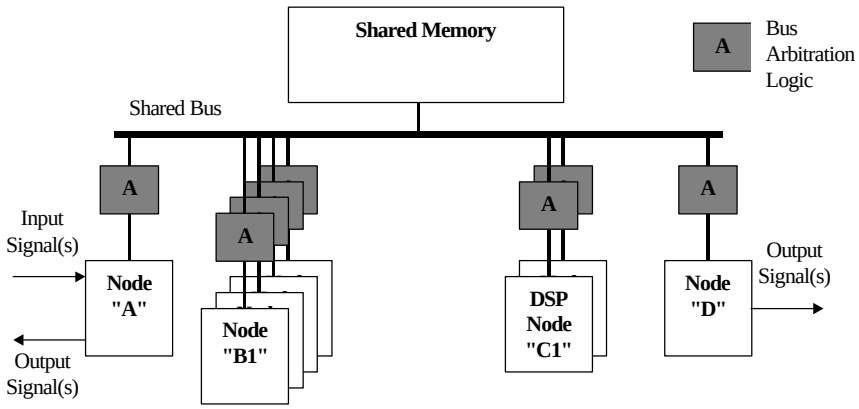
Multiple I/O nodes connected to single DSP node

We can test our list against this system and find that again, depending on the application requirements, all features we have stated are useful. In this example it is particularly easy to see the requirement for the non-blocking communications.

It is encouraging that these quite different system configurations can be addressed with a common set of requirements. We can happily state that the theory that we developed has been validated and will be a useful set of requirements against which we can judge some traditional approaches for providing system communications.

7.4 Shared memory

Most people's first experiences of multi-node DSP systems use shared memory as the interface between the nodes. This involves using an area of memory that is addressable by all nodes.



Multi-Node System using Shared Memory as Communication Medium

The main feature of such a system is the arbitrator. The main function of this is to prevent accesses from the different nodes clashing. There are many standard techniques for arbitrating this type of system, but judging each type is not interesting here.

What we can say is that an arbitration design could be made that allowed transfer of packets, gave notification of completion of the transfer, did not impose a packet size, gives choice of blocking/non-blocking, allowed DMA to be used, has viable cost, and supports multi-cast (in fact very well!).

However……. a fundamental problem is this system is its use of a shared resource. The guaranteed access speed of a buffer will always be less than 1/nth of the maximum access speed of the memory devices (n = number of nodes). This is because in any communication both the transmitting and the receiving node requires access to the memory in order to complete the communication, and some finite time will be 'lost' in order to arbitrate the accesses.

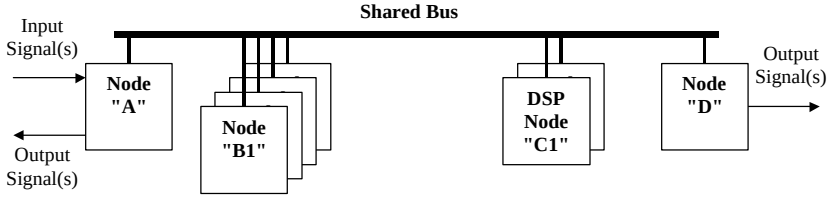
There is also a significant amount of management that has to be performed by software, making the complexity of the system (and hence the time to create, test and support that system) grow as the number of nodes is increased. Resulting from this management, any one communication requires many additional accesses in order to gain allocation of the buffer, flag that it is queued for use, find out its purpose and finally to acknowledge the action is complete and to release the buffer. This makes the use of small buffers very inefficient, and means the system is not easy to use.

Additional comments:

1. How simple is it for an I/O node to exist in a shared memory system? There would need to be hardware arbitration for the buffers, thus fixing the algorithm used by the system, or an additional processor added to perform this task.
2. Shared memory is only practical in a single board system, unless each board is in fact a single node, as in the next system implementation that we will discuss.

7.5 Bus Communication systems

Sometimes architectures are considered where a shared Bus is used for the communications path. This could be separate boards communicating over a back plane like VME, or multiple nodes on a board communicating over an internal bus.



Multi-Node System using Shared Memory as Communication Medium

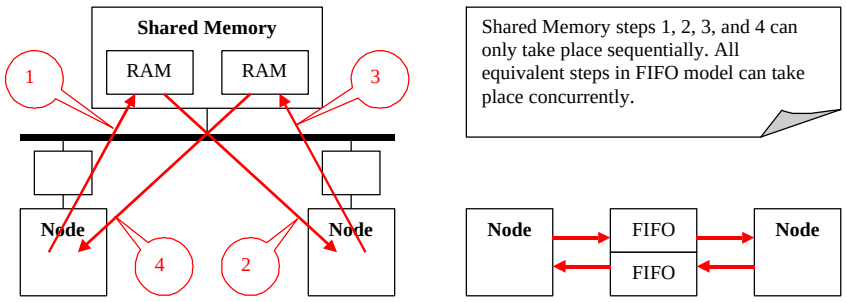
There are many standard bus definitions that people often choose for such a system, simply because of the availability of standard components. It is usual however that the standard bus definitions do not provide the features that we need, simply because they were designed for other purposes such as providing I/O for general purpose computing.

Like the shared memory system we have just considered, the actual bus design could be customised to give the features we need if we realised that the standard busses do not give us what we want.

However we still have the same problem as in the shared memory system, the use of a shared resource, with some kind of arbitration. This arbitration could work on a first come first served basis, in which case one communication could be completely blocked by another, or it could work on an equal share basis, in which case the available bandwidth depends on how many communications are taking place. Either way the bandwidth is either not guaranteed, as more nodes are added to the system.

7.6 Communication using FIFOs

Sometimes system designers realise that shared memory management algorithms are often attempting to implement a FIFO in software.

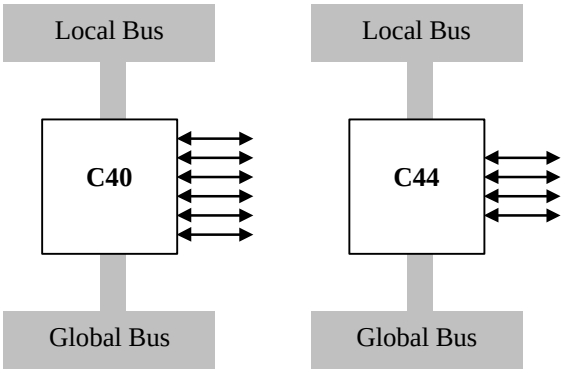


FIFOs support efficient transfers of data between connected nodes

So why not use actual hardware FIFOs?

The FIFO flags give the information about the availability or non-availability of space, or data to read, and the two ends of the FIFO can be read and written simultaneously. Some multi-node implementations have used this technique, where a system has a dedicated FIFO placed between two nodes that need to communicate. Each end is accessible by only one node, so the FIFO offers a point to point connection between the nodes.

Against the requirements that we started with this is the best fit yet, and in fact we have some experience with this approach as this is in fact the same approach that Texas Instruments used in their ‘C4x DSPs to implement the Comports. This is the basis of the architecture of our ‘C4x products.



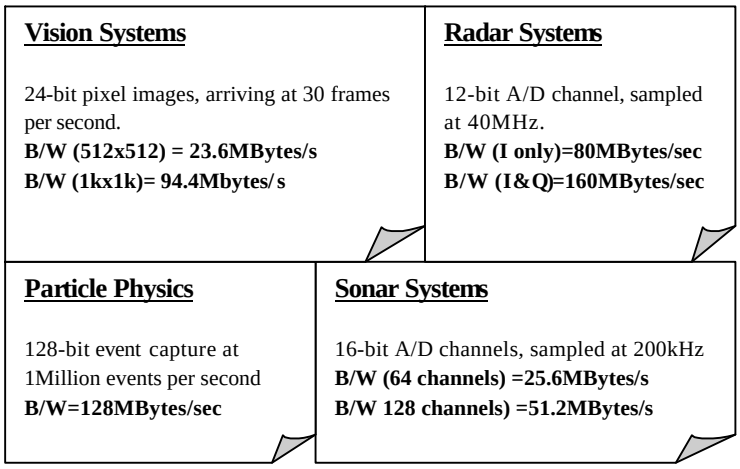
TMS320 Processors which support the FIFO point-to-point communations model

It is an interesting aside that many early ‘C4x products used shared memory and Comports, whereas we use comports as the only available inter-node communication, even with the host machine that we have already discussed as being just another node. It is also interesting that we are one of the few vendors that recognise the concept of having a node in the system that provides I/O only, as is demonstrated by our GDIO range of I/O products.
Almost all of those early products on the market that used shared memory have converted to be comport only products now, perhaps because of customer pressure, or perhaps because when those companies started to use the comports they realised what an elegant solution they were.

So the market actually accepted this new architecture because it brought benefits.

With our ‘C4x products today, we see four common additional needs:

1. It is possible to require more connections than are available on the processor silicon.
2. When more connections are required than can be supported by the hardware, it is sometimes "desirable" is to utilise a means of using intermediate nodes to forward data. This technique can cause the guarantee of bandwidth to be lost because of the bandwidth used to forward communications.
3. The multi-cast is a requirement of some systems
4. The bandwidth is often too low for the application, especially when the node is required to forward data onto a subsequent node.



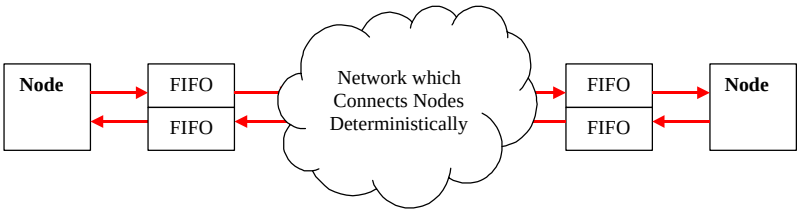
Back of envelope calculations for possible high bandwidth applications

It is clear however that in the modular multi-node system business that we are in, the comports are the most elegant solution available today, but we should take some time to search for a better solution starting from a "clean sheet". Perhaps we can find:-

- An architecture that is independent of any particular processor type (unlike the 'C4x comport), so we can use it for our existing 'C4x product range and for our 'C6000 product range, where there is nothing like the comport provided by the silicon manufacturer.
- An architecture that satisfies our original wishes, of at least has justifiable compromises.

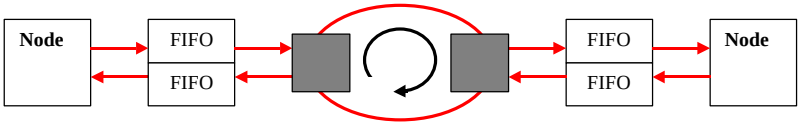
8. THE SEARCH FOR A BETTER COMMUNICATIONS SYSTEM

Taking on board what we have been discussing, it is clear that having a FIFO interface at each node is an elegant way of inexpensively and simply connecting to the communications system. What would make this even better would be if all communications could be routed through a single bi-directional FIFO regardless of the number of nodes in the system, regardless of the ultimate source or destination of the communication. This requires that the other end of the FIFO can have its data routed to any other node in the system.



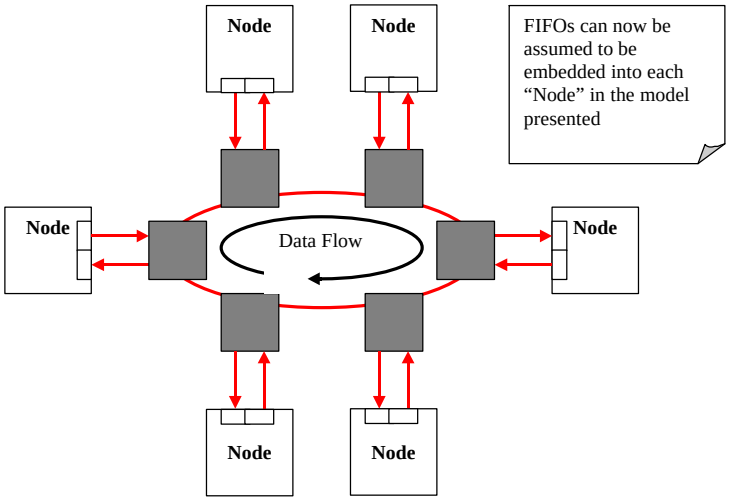
Pair of Nodes communicating via FIFOs connected to "Transparent" Network

If we propose that the communications system takes the input and output of the FIFO and chains them together to form a ring, made of point to point connections, then there is a possibility to reach any other node in the system



Pair of Nodes communicating via FIFOs connected to Ring

As we add more nodes to the system we get a picture like :-



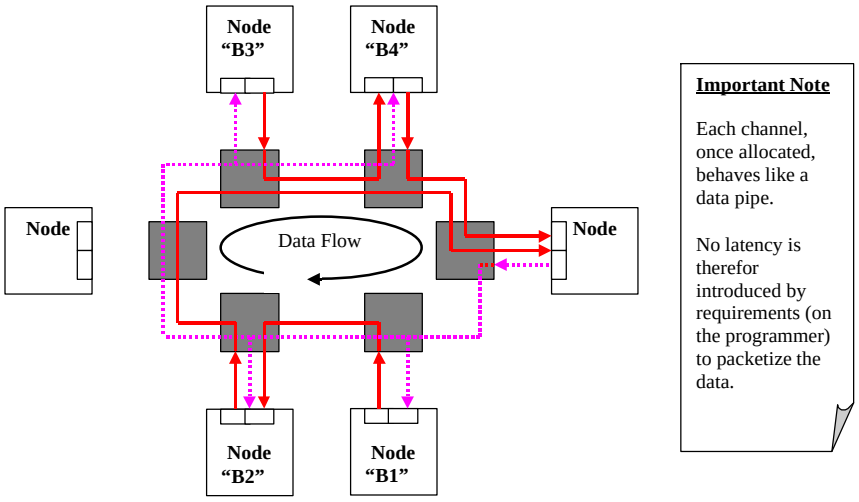
Group of Nodes communicating via Ring

Taken simplistically this means that each node has to handle the data for all other nodes in the system. This is clearly undesirable. Following our "clean sheet" approach let us state that the grey box is not just a FIFO but can also determine if the communication is to be received by this node or to be simply passed on. If the system is clever enough the node could receive and re-transmit the data, providing support for multi-casting.

The usual train of thought when this diagram is considered is to use a packet-switching ring. In this case a packet would be assembled, and have routing information added to it, then the hardware would wait for an available packet slot to arrive, claim it and transmit the whole packet.

STOP! This has destroyed our most fundamental requirement, which is the guarantee of bandwidth and latency. A system implemented in this way is not deterministic, and probably imposes the packet size and other problems.

There must be some mechanism for ensuring that these separate communications cannot cause deadlock. It is a requirement of a real time system that they can both proceed at the same time. This allows us to achieve our need for guaranteed bandwidth availability for each data path in the system. Thus it is necessary that at any point on the ring, multiple communications can be occurring simultaneously.
In the case of the ring communication system, it must be possible to pre-allocate “channels” which will then be utilised by the system.
Changing our model to a channel-switched model gives :-



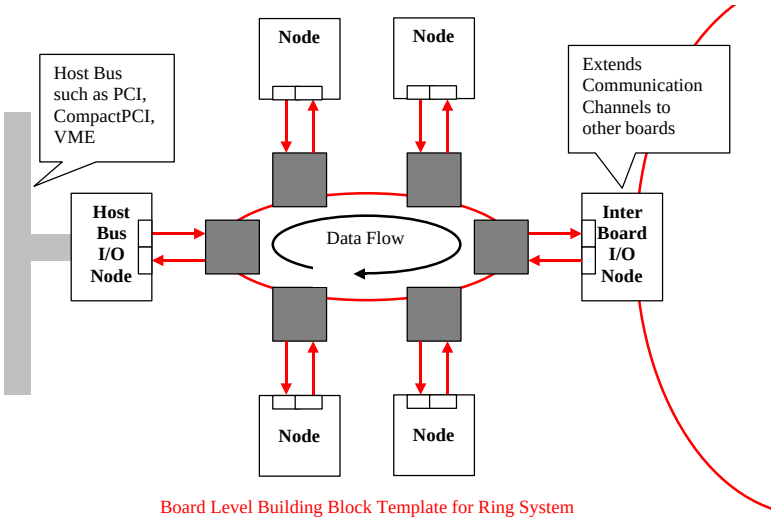
Support for separate concurrent “Channels” across communication system

This shift in thinking immediately shows advantages, in the diagram it can be seen that multiple channels are allocated, one of them is a multi-cast connection, and some are passing through the nodes communication block without requiring interaction from the node.

To the application, the communication system should simply be there, and work deterministically, so that the application doesn’t need to worry about it. Using a pseudo-static allocation of the communications resources at system design time enables the application to simply use the system, and be guaranteed the availability of bandwidth.

Reality check!
I am able to state that this type of system is achievable, we are in fact developing such a system with the following features:-

- Synchronous FIFO connection to nodes, offering DMA support and interrupts from flags.
- 6 nodes on an "on board" ring, including the host interface, and the inter-board connection.
- Up to 400 Mbytes per second communication, allocated in 66.6Mbyte/second increments.
- Application level programming of routing paths, dynamically configurable during system operation.
- Application level selection of blocking/non-blocking communication.
- Application level selection of multi-cast groups.
- No imposition of block size on the application.
- "Inter board" ring, connected via 14 way connectors (one for ring input, one for ring output) that supports the same bandwidth as the "on board" ring.
- Guaranteed maximum latency of three 32-bit words in the "on board" ring, plus three more for the off board ring.
- Guaranteed deadlock free.



Board Level Building Block Template for Ring System

9. THE EFFECTS OF THIS STRATEGY HAS ON THE OTHER SYSTEM CHOICES

Returning to our original system components

Processor

Simply put the processor must have a memory bus, that's all! It is very hard to imagine a DSP chip being designed without a memory bus, so we are left with a free choice of DSP processor.

This choice may be effected by items like the availability of DMA or multiple memory busses, but these are advantages not requirements.

System Software strategy

This communications strategy is very flexible; thus it does not impose any restrictions on the programming model that you will use.

I/O interfaces

The treatment of nodes, rather than processors, means that I/O interfaces are supplied with the same rich features that the processors get. Coupled with a simple to use FIFO interface this means that flexibility of I/O interfaces is actually enhanced by such a communications architecture.

System Communications strategy

This is evidently a very crucial issue in such systems. Getting it right can make the rest of the issues become simple to solve. Blindly continuing with a strategy simply because it is available, or was used before can severely damage your system!