INTERFACES FOR EMBEDDED PARALLEL MULTIPROCESSOR NETWORKS

SIMON TRIGER

A thesis submitted in partial fulfilment of the requirements of The Nottingham Trent University for the degree of Doctor of Philosophy

Department of Electrical and Electronic Engineering
School of Engineering
The Nottingham Trent University
Burton Street
Nottingham, United Kingdom

December 2002

Collaborating Establishment: IC Routing Ltd, UK

Abstract

This thesis documents research to improve tolerance to faults of an embedded parallel network. This resulted in the development of two building blocks of a novel embedded communications system with enhanced fault detection and recovery.

A review of embedded inter-processor communications was initially performed. The research aimed to expand the potential of embedded parallel systems in three main areas: improving bi-directional throughput; implementing a distributed fault detection, isolation and recovery mechanism; and the implementation of hardware virtual channels utilising Context Addressable Memory (CAM) to reduce processor intervention.

The embedded multiprocessor network comprises off-the-shelf custom hardware message routers. An interface between a StrongArm SA-110 microprocessor and the embedded routing network was developed using VHDL. This was simulated and synthesised, with post-synthesis simulations used as a means of gauging performance. An interface was also developed between a PC and the network, utilising the PCI bus standard for communication. The research resulted in a fully operational hardware prototype, whose results were compared and contrasted with both the previous non-fault tolerant PCI interface and theoretical expectations.

The routers, StrongArm processors, PCs and their respective interfaces form the building blocks of a robust, embedded network with improved tolerance to faults. The StrongArm and PCI interfaces allow RISC and general-purpose processors to operate as processor nodes in the same network, thus increasing system flexibility and applications. The possibility of adapting the interface design to other processors offers further possible increases in system flexibility. The new protocol allows a much greater degree of tolerance to faults in the system, reducing the dependence on external intervention in the event of network failure.

Acknowledgements

This thesis is dedicated to my father, who I am sure would have been proud of me and to Richard and Michael, whose circumstances were a major influence in my decision to remain in academia and who will always be sadly missed. Finally to my grandfather and uncle Keith both of whom saw me begin my studies but sadly are not present to witness their completion. Special thanks, first and foremost to Mum and Sara for absolutely everything.

I would like to thank the following people for the limitless amounts of help and support over the duration of my studies and beyond. Firstly to my supervisors, Professor Brian O'Neill and Dr Steve Clark for being the most supportive and understanding supervisors imaginable. Also thanks to Steve for dragging me out of the office on cold, damp days to climb up some godforsaken, green, slimy, vegetated choss masquerading as a rock climb. To my colleagues, Dr Robin Hotchkiss, Dr Kar Leong Wong and Dr Jien Hau Ng without whom there would have been no research for this project to build on, and to Dr Kenny Liew for his 'alternative' sense of humour. I would also like to thank Jean, Lynn, Iain and Ali for not letting me take things too seriously. Thanks are also due to all my climbing, running, swimming and drinking partners from the last three years, who were always ready and more than willing to provide a welcome distraction.

Lastly, to anyone I've not mentioned, thanks.

Table of Contents

List of Acronyms and Abbreviations

AE	Almost Empty
AF	Almost Full
AMBA	Advanced Microcontroller Bus Architecture
ASIC	Application Specific Integrated Circuit
BEOP	Bad End Of Packet
CAM	Context Addressable Memory
CAS	Column Address Strobe
Cat 5	Category 5 unshielded twisted pair
CBFC	Credit Based Flow Control
CONREQ	Connection Request
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
CSMA/CD	Carrier Sense Multiple Access / Collision Detect
CSP	Communicating Sequential Processes
DMA	Direct Memory Access
DS	Data Strobe
EAB	Embedded Array Block
EEPROM	Electrically Erasable Programmable ROM
EISA	Extended ISA
EOM	End Of Message
EOP	End Of Packet
EOPE	End Of Packet Error
EPROM	Electrically Programmable ROM
ESB	Embedded System Block
ESP	Embedded Standard Product
FEC	Forward Error Correction
FIFO	First In First Out
FIQ	Fast Interrupt Request
FRES	Forward Reset
FT-PCI-OSLi	Fault Tolerant PCI-OSLi
FT-SARNIC	Fault Tolerant SARNIC
FT-SARNet	Fault Tolerant SARNet

IC	Integrated Circuit
ID	Identification
IECR	Interrupt Enable Clear Register
IER	Interrupt Enable Register
IESR	Interrupt Enable Status Register
I/O	Input / Output
IOE	Input / Output Element
IP	Intellectual Property
IRQ	Standard Priority Interrupt Request
ISA	Industrial Standard Architecture
ISR	Interrupt Status Register
IRSR	Interrupt Raw Status Register
JTAG	Joint Test Action Group
LAB	Logic Array Block
LAN	Local Area Network
LE	Logic Element
LUT	Look Up Table
MPP	Massively Parallel Processor
MSB	Most Significant Bit
NRZ	Non-Return to Zero
OS	Over Sampling
PBFC	Permission Based Flow Control
PC	Personal Computer
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
PCI-OSLi	PCI OS Link Interface
PLD	Programmable Logic Device
PQFP	Plastic Quad Flat Pack
RAM	Random Access Memory
RCSMA/CA	Reservation Carrier Sense Multiple Access / Collision Avoidance
Rhop	Router hop
RISC	Reduced Instruction Set Computer
ROM	Read Only Memory
RTL	Register Transfer Level
SARNet	StrongArm Router Network

SARNIC	StrongArm Router Network Interface Controller
SARNode	StrongArm Router Network processing Node
SDRAM	Synchronous Dynamic RAM
SRAM	Static RAM
SMT	Surface Mount Technology
SOC	System On a Chip
SOPC	System On a Programmable Chip
UART	Universal Asynchronous Receiver / Transmitter
USB	Universal Serial Bus
UPT	Unique Token Protocol
VccInt	PLD Internal Voltage
VccIo	PLD I/O Voltage
VCMS	Virtual Channel Message Store
VHDL	Very high speed integrated circuit Hardware Description Language
VLSI	Very Large Scale Integration

List of Figures

1 INTRODUCTION

Parallel processing is based upon the principles of task division and concurrency with multiple processors striving to achieve a solution to a problem that is too complex or time consuming for a single processor [1]. Whilst the operating speed and processing power of computers continues to improve, there will always be computationally intensive problems that are beyond the capabilities of a single processor [2]. Computer aided VLSI design [3] and the simulation of weather patterns used in long term weather forecasting are two examples of parallel processing applications.

One of the drawbacks of parallel processing is the limited ability to handle real time applications, which often require parallel task executions. The computers used in parallel processing systems were originally expensive, custom processors [4]. The critically short task execution times required by real time applications are more suited to RISC processors, such as the Transputer [5], which was developed specifically for use in parallel systems. Interconnected multiple single processors [6] can also form the basis of parallel systems, utilising advantages such as high performance, low cost, availability, relative ease of development and possible scope for upgrades, which led to their use in certain niche applications. The performance is often lower than that of custom parallel systems but for some applications their versatility and accessibility can offset this, particularly in embedded systems.

Many real-time applications require processors to be 'embedded' within a larger system and to act on responses from within the larger system or in its external environment. Embedded systems are often housed within a small area, such as inside a medical instrument [7] or in a military or vehicular application [52]. Many large-scale parallel systems have sufficient processing power but are too large and relatively expensive to be embedded for use in real-time applications. Embedded systems often require portability, deriving power from a battery (a prime example being a mobile phone) and hence requiring minimal power consumption.

There exists a need to minimise failure in embedded processing systems due to the nature of such systems. Embedded systems often perform complex and sometimes safety critical tasks in applications and frequently operate in remote or inaccessible locations. Embedded parallel systems are more susceptible to failure as the network size and number of nodes increases [8] and the systems tolerance to faults becomes increasingly important, however performance is a key aspect of embedded systems and the techniques aimed at improving fault tolerance should minimise overheads so as not to greatly compromise throughput.

The research described in this thesis resulted in the design and realisation of building blocks for a multiprocessor routing network with improved tolerance to faults. The NTR-FTM08 router was previously developed to provide a robust replacement for the commercial 16-channel ICR-C416 router [9]. The serial OS (Over Sampling) links based protocol [5] (similar to that employed by the first generation Transputer parallel processors) utilised by the ICR-C416 was replaced with a new protocol, modified to permit the transmission of control information required for the implementation of fault detection and recovery functions between processors. The new protocol meant that the NTR-FTM08 had no means of interfacing to the processing nodes and external PCs, requiring the redesign of these interfaces. A key aim of the research was to enhance the systems tolerance to faults through alterations to the interface architecture, giving a robust communications environment, to reduce network down-time and the systems reliance on external intervention to recover from faults.

1.1 The Transputer

The Transputer was a microprocessor designed for use as a parallel processing building block. It was often used to construct low cost embedded parallel computing systems due to its ability to support real time programming [10]. This ability was partly due to its RISC architecture, instruction set, reduced silicon usage and state-of-the-art performance for processors at the time. Additionally, the Occam Network Description Language [11, 12] was a vital component in the process of the construction of Transputer networks.

Communication with other processors was mainly via four OS serial links [5]. Communications responsibilities were devolved to a co-processor providing a physical point-to-point connection between processing nodes, based on the Communicating Sequential Processes (CSP) model proposed by Hoare [13]. These four full-duplex bi-directional serial communication links allowed for connections to four neighbouring processors, as the two-dimensional mesh topology shows in Figure 1.

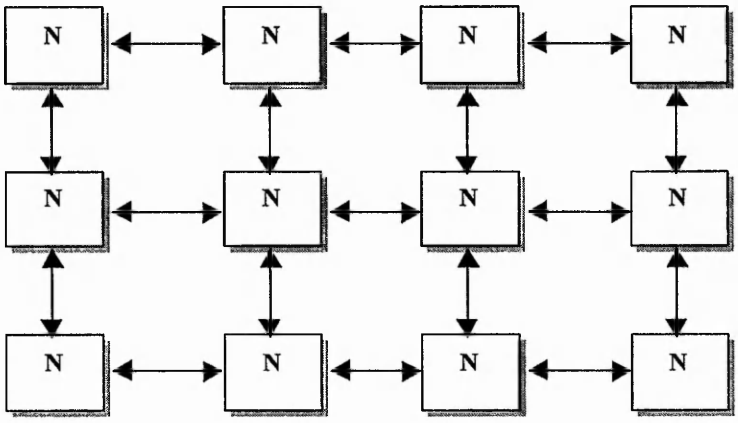


Figure 1: 2D 'mesh' network topology utilised by first generation Transputers

Networks utilising more than five processor nodes encountered problems with messages destined for nodes that were not adjacent to the transmitting node. These messages required forwarding by the intermediate nodes, thus reducing their efficiency as they devoted resources, incurred latency and increased message overheads to message handling tasks required in the forwarding of other processors messages. This results in a reduction of processor performance as the computation: communication ratio [14] increases in favour of communications as processors handle communications from other nodes as well as their own.

Such 'store and forward' methodologies produce message delays proportional to the product of message size and distance [15], giving a wide variation in communication times between processors, based on their relative distances. Large

interprocessor latencies could reduce the performance benefits obtained through parallel processing.

Key features that contributed to the success of the Transputer were the efficient transfer of information between processor and network and the minimal processor intervention. The Transputer achieved the former through the integration of its tightly coupled, built-in communications controller onto the same silicon as the processor. However this approach used up space on silicon, which could have been used for extra processor functionality, in addition to requiring extra engineering effort to design, and build a dedicated interface [16]. Study of the Transputer family history has shown that upgrading either the microprocessor core or the interconnection network or network interface was time consuming and costly [17] due to ASIC design techniques and costs.

1.2 Research History and Objectives

The parallel processing research group at The Nottingham Trent University has for some years focused on hardware routing devices used to link embedded parallel processors. The research group developed a distributed processing system consisting of end nodes (microprocessors) linked together using high-speed serial interconnections. Messages sent between nodes use hardware switches (routers) to reach their destination.

The research began with the development of the NTR-08 [18, 19], a prototype 8-link router that utilised serial transmission links to form interconnections between Transputers. This allowed a processor to form connections with any other Transputer linked to that router without the message needing to be routed via intermediate Transputers, relieving the intermediate processors of the burden imposed by other communications and increasing their efficiency. The NTR-08 allowed up to 8 simultaneous bi-directional communications between its attached entities. More Transputers could be added by including more routers, permitting large increases in network size for relatively small latency increases – hence creating a scalable network. Figure 2 shows a simple routing network utilising NTR-08 routers. The

communications links of the NTR-08, as with all other routers developed by the research group, could form independent and concurrent communications channels between all other links, permitting 8 concurrent bi-directional data transfers.

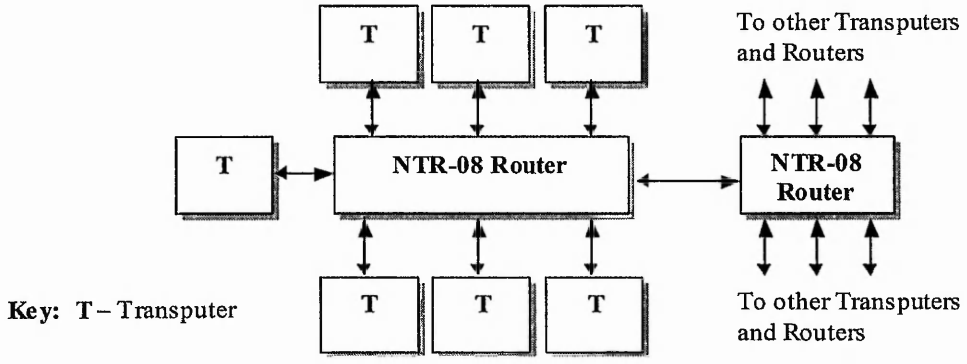


Figure 2: Processing nodes interconnected using NTR-08 routers

The NTR-08 was developed further, resulting in the fabrication of a commercial 16-channel dynamic hardware routing switch, the ICR-C416 [9, 20]. This device demonstrated the efficiency of a routing device as a simple solution to medium scale, low cost, high performance inter-processor communications. One of its major applications has been the hub of a control network for Quantel's CLIPBOX Video Server [21]. The ICR-C416 router formed the backbone of an embedded distributed multiprocessor system as shown in Figure 3.

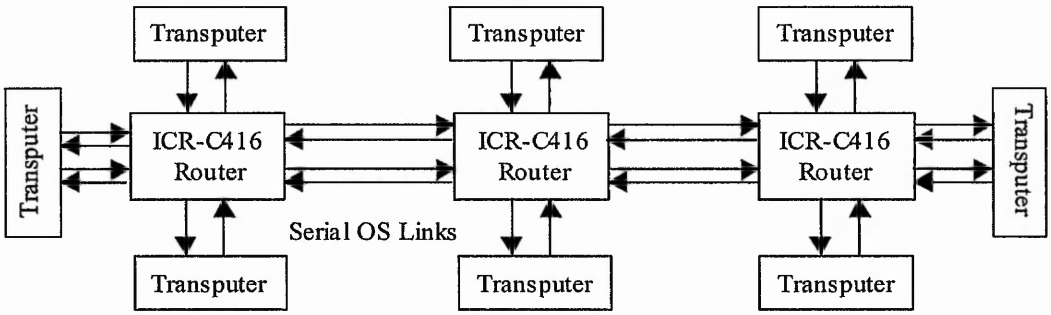


Figure 3: ICR-C416 and Transputer Routing Network

Further developments improved the features of the research groups original router, the NTR-08, via the addition of extra functions resulting in the NTR-M04, a prototype

4-channel router. A new function was Multicasting: the ability to support in hardware the transmission of a message to multiple destinations, whilst simultaneously supporting other unicast messages [22]. Multicasting was added without compromising existing important features of the router. Other enhancements included the Split Channel Link, which allowed for effective doubling of link bandwidth [23] and access to the control port via any of the communication links.

The rapid increase in microprocessor performance was the spur that prompted the need to upgrade the processors used in the ICR-C416 network in order to keep pace with other embedded networks. The first generation Transputers were superseded by the less successful T9000 Transputer family [24, 25]. The demise of both generations of Transputer proved to be the catalyst in switching attention to the design of routing networks for other processors, whilst maintaining the successful features of the Transputer parallel network. The main focus was on state-of-the-art Reduced Instruction Set Computer (RISC) processors [26]. Technology upgrades to both processors and interconnection networks were also necessary to take advantage of power consumption improvements, which in embedded networks is of much greater importance than in large-scale multicomputer systems.

The StrongArm SA-110 microprocessor [27] was chosen to replace the Transputer in the ICR-C416 network, as it was a low cost, easily available processor offering state-of-the-art performance. Transputer networks, such as those constructed using ICR-C416 routers were designed to enable efficient execution of embedded applications, not to rival high-end supercomputers. The 'processor node' architecture of the ICR-C416 network lends itself to multiple low cost processors. The 32-bit SA-110 microprocessor can support core and data bus frequencies of up to 233MHz and 66MHz respectively. The processors on-chip cache and write buffer increased average execution speed and reduced average bandwidth required for memory accesses. This permitted the memory bus to be used to transfer data to and from the communications interface, improving throughput, but at a cost of a processor-specific solution.

The transfer of messages to and from the SA-110s memory by the communications controller was done during periods when the processor is not accessing the memory using Direct Memory Access (DMA) transfers. Cycle-stealing

DMA is a special hardware arrangement utilising idle processor cycles to transfer data to and from memory very quickly without the overheads incurred in accessing the I/O bus [28]. A loosely coupled custom processor interface, utilising the built-in cache memory of a processor can achieve a highly efficient and seemingly transparent memory transfer [29]. This is of particular importance in fine-grain computation systems where the communication tasks of a processor are much larger than the computation tasks [30].

The research group subsequently developed the StrongArm Router Network Interface Controller (SARNIC) [31], which was designed to perform the communication interface for the chosen processor. The research resulted in the design, development and hardware implementation of a PLD based interface controller, integrating a bus-based network interface controller, a memory interface controller and a processor interface controller in a single chip mounted on a processor node PCB with dedicated, distributed SDRAM. This interface, the SA-110 and an SDRAM memory module, formed a StrongArm Router Network (SARNet) processing node (SARNode), as shown in Figure 4. The SARNode could be used as a building block in a scaleable distributed parallel processing system, interconnected by ICR-C416 routers.

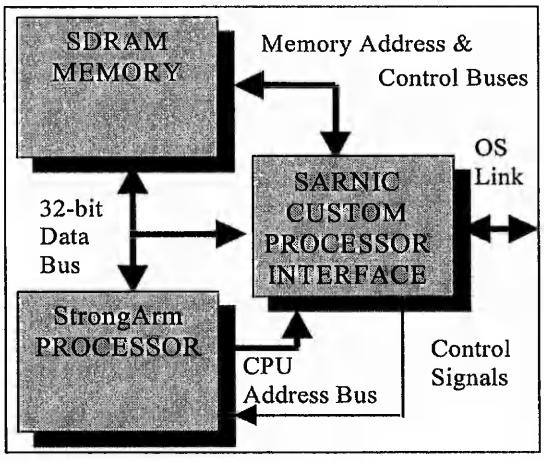


Figure 4: SARNode Block Diagram

The idea was for the processor node in Figure 4 to directly replace the Transputer in a parallel router based network, as shown in Figure 5. The purpose of the SARNIC is to take control of the communications to and from the network, allowing the processor to devote its resources to program execution.

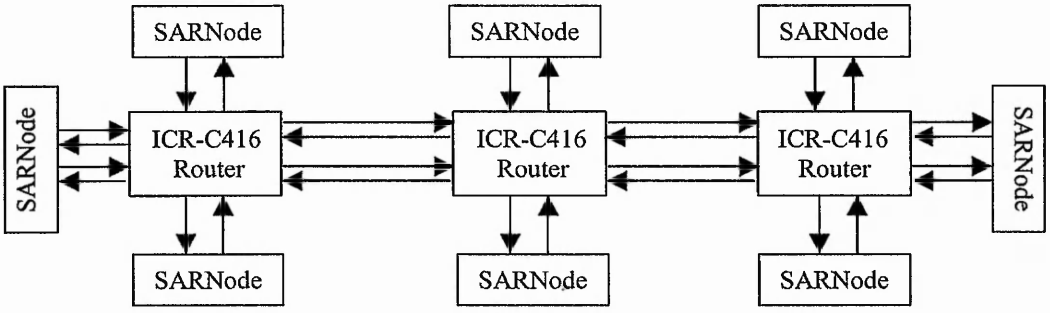


Figure 5 : SARNet Routing Network comprising ICR-C416, SARNIC and SA-110

The SARNIC included the following features [17, 32]:

- Minimised processor intervention.
- It attempted to supply continuous data transfer between memory and network interface whilst utilising minimal buffering.
- A memory bus arbitration system, to guarantee bandwidth for each message channel whilst minimising any interference in the CPU operation.
- DMA assisted message channels, packetising and depacketising messages in hardware.
- Two serial communication links doubled the effective network interface bandwidth. This increased the range of possible network topologies and provided a backup data path in event of a faulty link.
- Link resource allocation, implemented in hardware instead of software, allowing hardware virtual channels that can be allocated to either of the two physical communications links and increasing header-matching efficiency by reducing the degree of software involvement in this process.
- ICR-C416 compatible control link to monitor communications link status.

- Option of booting the interface and real-time processor node reconfiguration possible via the serial communications link. Internal ROM could also be used to boot the interface. These options facilitated real-time system reconfiguration.

Whilst the NTR-M04 attempted to address some of the shortcomings of the ICR-C416, it still utilised a centralised control port, to monitor faults in the event of a failure. In addition the NTR-M04 had no coherent recovery strategy. The wormhole routing mechanism [33, 34] (see section 3.2) employed by the ICR-C416 reduced the message buffering requirements but stretched the message across the network. The credit based flow control mechanism (see section 4.4.2) employed by the ICR-C416 resulted in a communication link stalling in event of a fault. One part of the message stalling would result in the rest of the message that follows to stall, as it could not progress further in the network until resources were free to take the message. Features such as Adaptive routing [35] and Virtual channels [36] helped other messages to bypass the network resources occupied by the stalled links but could not free those resources for use by other messages. Such situations resulted in the effects of a fault spreading across the network within a very short space of time.

Tolerance to network failure with the ICR-C416 was limited and centred on a single, centralised monitoring and intervention solution. This had little scalability as the efficiency decreased as the number of network elements increased [11]. The fault monitoring and intervention strategy was provided via a separate control port and, as such, required an extra bi-directional communications link. As the mean time to failure was high, the solution was acceptable for board level systems, where the network nodes were situated very close together. As the network could tolerate transmission distances of over 100m, an extra link becomes proportionately expensive. The aspects of the design that enhance the systems response to failure must be conveyed across the data transmission line itself, rather than a dedicated control link.

A new router was designed with a view to improving the systems response to failure, in order to form the backbone of a more robust routing network. The NTR-FTM08 [37, 6] was a prototype 8-channel router with increased fault handling capabilities, whose aim was to provide a fast response to faults, implementing fault

tolerant functions in hardware, which had previously been performed using software (if at all). The NTR-FTM08 was designed with a view to containing an integrated detection method for basic faults, which would isolate them, to minimise their effects and enable normal network operation in as large a part of the network as possible. The system would then recover from the failure by removing the faulty message, freeing the resources held by it and would attempt to re-establish any lost network connections. The network should be able to recover from failure without requiring user intervention, whenever possible.

General-purpose microprocessor systems, such as PCs are often better suited to dealing with diverse applications rather than dedicated tasks such as distributed real-time control. A potential use. for the SARNet is the ability to enhance the computational abilities of a general-purpose processor by providing the real-time processing advantages gained through RISC processing. The creation of such a heterogeneous network required the development of a dedicated interface to connect to a general-purpose processor and the embedded SARNet system. Commercial general-purpose processor interfaces have been developed, such as the BBK-PCI [38] and BBK-PCI Light [39] but these are not aimed at supporting communications between embedded systems. Such devices are not optimised to the communications protocols utilised by the SARNet, requiring a software implemented communications mechanism, introducing extra overheads and reducing the efficiency of the communications. Embedded system communications are built on a core principle of efficiency where latency and overheads are minimised to maximise performance.

A custom interface was designed, tested and built, with its communications optimised for the ICR-C416's OS Links based routing network and linked to the PCI bus [40, 41] of a general-purpose computer (for example a PC). This interface, termed the PCI-OSLi [42, 43] would allow connection to any other network that utilised a similar PCI interface and allows construction of flexible systems, in terms of wider, ranges of applications such as that shown in Figure 6.

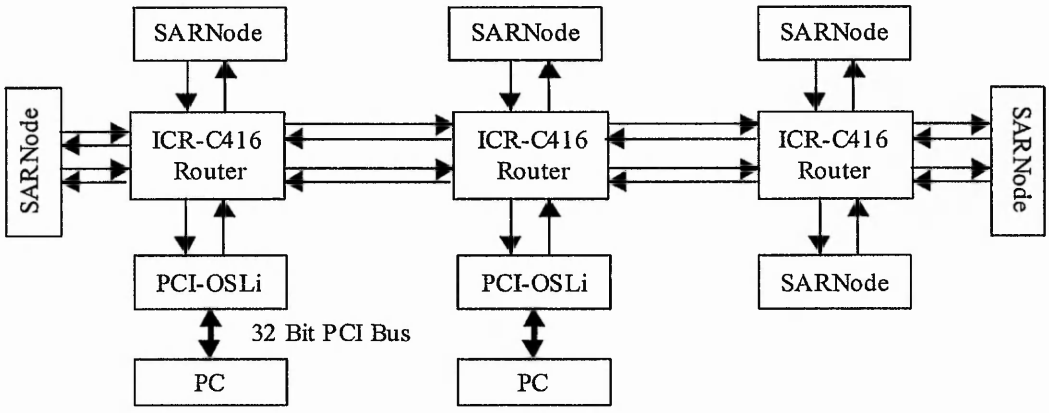


Figure 6 : SARNet Routing Network with a link to a generic PC via the PCI-OSLi hardware interface

Additionally, a novel Operating System that supported inter-processor communications [44] was developed. The operating system addressed some real time control requirements and was optimised for embedded, distributed parallel processing system applications. The operating system was complemented by the following two tools that were developed to aid the user in parallel programming: a Network Specifier allowed the network mapping to be captured graphically, using the graphic user interface, and an NTU-Configurer used Artificial Intelligence to generate routing headers automatically.

1.3 The FT-SARNet Routing Network

As the SARNIC and the PCI-OSLi were developed for use with the OS Link based protocol of the ICR-C416 these interfaces were incompatible with the NTR-FTM08, with its protocol aimed at conveying status information across the communications links, in addition to data. The focus of this project is to implement a range of features aimed at improving the systems response to faults, into devices similar to the SARNIC and the PCI-OSLi to enable the construction of an NTR-FTM08 router network linking StrongArm processors and PCs. These interface

devices are called the Fault Tolerant SARNIC (FT-SARNIC) and Fault Tolerant PCI (FT-PCI-OSLi) respectively.

Fault tolerant in this respect signifies the enhancment of fault detection and recovery in comparison to the designs resulting from the previous research.

The more robust communications system, which could be constructed with these devices is called the Fault Tolerant SARNet (referred to hereafter as the FT-SARNet) and is shown in Figure 7. The ultimate aim of the development of a network with increased fault tolerance, and its full implementation, including software support, is beyond the scope of this work. The main building blocks are the NTR-FTM08, the FT-SARNIC and the FT-PCI-OSLi. The project aims to achieve for the SARNet the same improvements in the systems fault tolerance that the NTR-FTM08 achieved for the ICR-C416. The FT-SARNIC and the FT-PCI-OSLi would be vital building blocks in a StrongArm parallel processing system with enhanced fault tolerance.

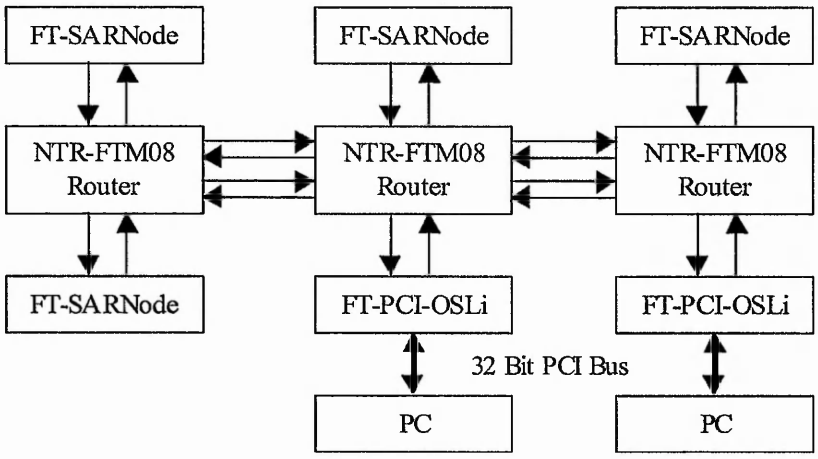


Figure 7: Fault Tolerant SARNet Routing Network

The FT-SARNet would ensure that when communications failure occurs, the effects are confined to as small an area of the network as possible. Initial work on the project identified the salient features of embedded multiprocessor networks, how these features affect the potential causes of network failure and methods of recovering from such failures. Background research involved a review of several types of hardware failure common to such networks and identified a strategy for detecting,

isolating and recovering from each of these faults when they occur. This led to a specification for the FT-PCI-OSLi and FT-SARNIC interface device and an investigation into how these could be realised in hardware.

A low-level, Register Transfer Level (RTL), VHDL model of the FT-PCI-OSLi and FT-SARNIC designs were created. The design was described using RTL level VHDL and partitioned into a modular, top-down hierarchy. VHDL was the chosen tool for design entry as it allowed developers to obtain information about the designs functionality with far greater ease than with schematic design entry. VHDL still provided compilation tools with enough low level information concerning the designs implementation and functionality. Being a recognised design entry standard, VHDL allowed portability between different target technologies, such as PLDs, gate arrays and ASICs as well as between different manufacturers' families within these technologies.

VHDL specified design functionality as opposed to implementation, which was left for interpretation by individual synthesis tools. The development software used throughout the design cycle was Alteras generic MaxPlus II and Quartus II software, based on PCs. Model Technology's ModelSim and Exemplar Logic's LeonardoSpectrum synthesis tools were also used. Post-synthesis simulations of the designs verified their functionality and timing analyses gave an indication of whether the timing requirements were met before implementation of the device in hardware.

The FT-PCI-OSLi interface design was implemented in a Programmable Logic Device (PLD). The latest generations of PLD offer a high density of logic functions with programmable features, and large amounts of programmable embedded memory. The FT-PCI-OSLi design was not fully optimised. This was to allow for a more generic solution, which can be targeted towards other PLD families with a minimum of alterations and to permit further developments. The FT-PCI-OSLi test results obtained from hardware testing were then compared for those of the PCI-OSLi device, simulations, predictions, and current research developments.

The FT-SARNIC design was not implemented in hardware due to the significant hardware developments and software support required evaluating its performance.

The design is ready for programming onto a PLD and post-synthesis simulations have been conducted in order to gauge its performance. Identical tests were performed on a post-synthesis simulation of the original SARNIC design in order to enable comparisons to be made.

1.4 Key Achievements

The work described in this thesis makes four main original contributions to the area of research:

- The FT-PCI-OSLi outperformed the PCI-OSLi in terms of efficient use of communications link bandwidth, due mainly to the adoption of the new flow control protocol.
- The implementation of a scalable autonomic distributed fault detection and recovery strategy that devolved responsibility for link monitoring and intervention to the nodes at either end of the communications link.
- Novel implementation of hardware Virtual Channel capabilities via the use of Context Addressable Memory [142].
- Significant improvement in the understanding of the host system interface of the FT-PCI-OSLi and PCI-OSLi devices. This was because previous research during the development of the PCI-OSLi did not observe the behaviour of the interface to the same degree of accuracy due to a different measure of DMA transmission duration being used, leading to a less accurate representation of the DMA transmission characteristic.

1.5 Structure of the Thesis

Chapter Two gives an overview of the subject area. An introduction to the relevant inter-processor communications is preceded by a review of the characteristics of such systems. This review identifies parameters used to characterise network performance and how they affect this performance, taking into account the requirements of the target network. Thirdly, three router networks used in multi-

processor systems, the ICR-C416, the STC-104 and the Myrinet routing networks are compared and contrasted. Another system, the Reliable Router, is also studied, despite differing target applications and an alternative approach to the previous three systems, because it has features which can be applied in the field of distributed fault tolerance. Finally, the PCI local bus, the interface between the PC and the communications network, is introduced.

Chapter Three examines fault tolerance from a systems level, investigating some of the possible faults that can occur in a distributed, asynchronous, multi-router communications network, why they occur, their effects and possible solutions. The faults detailed were prevalent in the ICR-C416 based router network. The protocol of the NTR-FTM08 based router network has provided the ability to detect and recover from, or in certain cases, prevent these faults. Firstly, the concept of network failure is discussed. Secondly, a study of the different types of network failure covered by the FT-SARNet network protocol is undertaken. Finally, the fault detection and recovery mechanisms in the three router networks under review are discussed.

Chapter Four discusses the FT-SARNIC and FT-PCI-OSLi interface designs from both theoretical and functional levels. It builds on chapter 3 by describing the means by which the features aimed at improving the systems tolerance to faults discussed in that chapter are to be realised. The chapter begins with a review of the design strengths of the SARNIC and PCI-OSLi interfaces and specifies the requirements for the FT-SARNIC and FT-PCI-OSLi devices. The areas of the design that require improvement in order to realise the fault tolerant features are researched and discussed. A synthesis review looks at the target technology of the FT-PCI-OSLi and how it contributes to the realisation of this device. The results from the synthesis of this design on to the target device, in terms of resource usage, timing analysis and power consumption are detailed and analysed in retrospect.

Chapter Five discusses the FT-SARNIC and FT-PCI-OSLi designs on a modular basis and identifies the functions performed by each module. Both designs are broken down in order of their modular hierarchy, with details of the functionality of each module, how they fit into the design and to which modules each interfaces.

Chapter Six includes results and details of the tests performed to obtain these. Post-synthesis simulations of the FT-SARNIC design are contrasted with comparative tests performed on a post-synthesis simulation of the SARNIC and with the estimated theoretical performance of the FT-SARNIC. Results obtained from the hardware testing of the FT-PCI-OSLi design are compared with results obtained from comparative hardware tests performed on identical hardware on the PCI-OSLi design and are also contrasted with the estimated theoretical performance of the FT-PCI-OSLi.

Chapter Seven discusses the implications of the results and the effects of the interfaces on the network performance. Several key points that affect the performance of the FT-SARNIC and FT-PCI-OSLi interfaces are discussed and compared, both between the interface devices and their non-fault tolerant predecessors and, in the case of the FT-PCI-OSLi, with commercially available IP.

Chapter 8 concludes the thesis, noting the achievements of the research and summarising how the addition of distributed system-wide fault tolerance expands on the previous interprocessor communications network. Finally, it details potential avenues of further work.

2 TECHNOLOGY REVIEW

2.1 Fundamental Principles of Interprocessor Communications

Three of the main characteristics used to define the overall performance of the communications in embedded real time parallel systems are efficiency, scalability and reliability.

- Efficiency defines how the system performs relative to the theoretical maximum. As discussed previously, performance is often a key requirement in embedded applications.
- Scalability dictates the ease with which the system can expand and thus determines its flexibility in handling a wide range of applications and network configurations.
- Reliability determines the robustness and stability of a system [45]. A crucial factor given the inaccessibility of some embedded networks and their use in safety critical applications, such as those used in the aerospace industry.

2.1.1 Efficiency

An efficient inter-processor communications model could be said to be one capable of performing communications duties without reducing its computational abilities significantly. A solitary processor can dedicate most of its processing power towards computational tasks, as it is not required to communicate with any other entities. The addition to the network of other communicating entities requires the processor to dedicate an increasing proportion of its processing power towards communication tasks, to the detriment of its computational capabilities.

A processor's performance depends on the compute : communicate ratio [14], which expresses the amount of communication overhead associated with each computation. Some programmes opt for coarse grain computation methods, reducing communications where possible in the interests of boosting computational throughput. Such an approach results in tasks that could be executed in parallel on multiple nodes

being executed sequentially on a single node, reducing the amount of parallelism and therefore defeating the object of parallel processing. Ultimately this leads to a reduction in efficiency as it takes longer to perform tasks sequentially and also because this approach leads to task duplication as data is not shared between processors. Others opt for fine grain parallelism, utilising large and complex communication networks to ensure that communications bottlenecks do not occur and devolving communications tasks to a communications co-processor to alleviate the microprocessor of some of the communications related overheads.

The efficiency of a systems inter-processor communications is often gauged by assessing the three parameters of bandwidth, latency and processor overhead.

2.1.1.1 Bandwidth

Bandwidth indicates the maximum amount of information that can be transferred across a communications medium in a given time and is often seen, rightly or wrongly, as the vital parameter in assessing the performance of a communications system [11]. The bandwidth of a communications network is primarily determined by the maximum clock rate at which the system can run without causing timing problems. Whilst a higher bandwidth communications medium allows faster data transfer, an inter-processor communication can only operate at the speed of the slowest component. As cost limits the available network bandwidth, protocol efficiency, in terms of the number of message bits relative to the raw network bandwidth, is crucial.

The bandwidth of the interconnection network should be high enough to prevent the communications network becoming the bottleneck of the system. An excessively high network bandwidth would be a waste of resources [45]. A high bandwidth, whilst being desirable, can lead to an increased number of transactions to and from memory in a given time (assuming that the processor is fast enough) and care must be taken to ensure that the incursion of overheads on the processors memory bus is not excessive. The maximum clock rate of the network depends on its implementation on silicon. Advances in silicon fabrication technology have led to a reduction in feature sizes, allowing designs to occupy less area than before, reducing the skew limitations

on clock speeds. The performance bottleneck of a circuit can be located with the aid of timing analysis by revealing the longest delay path between two clocked signals. Extra design effort in the areas identified can yield performance benefits.

2.1.1.2 Latency

The definition of latency is the time taken to transfer an empty message between source and destination [46]. Latency depends on the 'background' processes that are required in order to prepare the medium for the proposed message transfer. The latency in a communications interface is due to the overheads incurred in the setting up and receipt of a message transfer. When the transfer has been initiated, the data must pass through the transmission circuitry before it is outputted onto the transmission medium. Whilst this happens the transmission line is idle, reducing the efficiency of the communication and lowering the actual bandwidth from its theoretical maximum.

Network latency depends on the latency of the communications interfaces, their speed and the number of stages or hops the message must traverse in order to reach its destination. Bus based systems have low latency as there is only one stage between source and destination although access contention increases this significantly. Early switch based systems, utilising store and forward routing, incurred large latencies but the introduction of wormhole [47] and virtual cut through [48] routing methodologies have reduced hop related latency dependencies significantly. The latency of a device can also be reduced with the aid of considered circuit design and logic optimisation. Techniques such as pipelining can allow data to progress through the circuit more rapidly. Simplification and a reduction in clocked logic can also reduce latency but care must be taken to avoid timing problems associated with the latter.

Latency is an important factor for short communications such as control, synchronisation, acknowledgement and error signals as the latency incurred is large in comparison to the data contents of such transmissions. Sending multiple short messages is less efficient than sending one long message, as each message incurs its own latency period [45]. In short, a high bandwidth system with high latency can be

less efficient and thus have a lower throughput than a slower system with lower latency.

Real time applications require deterministic latency, in particular worst case scenarios to ensure that the process is completed within the given time despite such delay [11].

2.1.1.3 Processor Overhead

Communications related processor overhead is the time the processor dedicates to initiating or receiving a message. The latency of current communications is affected directly and contributes obliquely to those of subsequent communications. The design of the network interface hardware directly affects the overhead as it dictates to what extent communications related functions are off-loaded from the processor and thus the overlap between computation and communication tasks. This is best achieved via the use of a communications co-processor, freeing the processor to concentrate on program execution. A communications co-processor takes control of the tasks involved in transferring data to and from the communications network formats it appropriately and transfers it to memory where it is fetched when needed by the processor. Software overheads can also be reduced with the relatively recent introduction of direct support for communications at the user level [49, 50].

2.1.2 Scalability

A system can be said to be scaleable if it is able to retain its efficiency as it expands in size [11]. It dictates the ease with which a system can tolerate alterations to the number of processing elements and the arrangement of these which may occur as a result of a change in application. Scalability is affected by network topology [45] and it affects flexibility, which is an important consideration, given factors such as the cost, durability, lifespan and time and effort that are put into selecting, installing and maintaining a multiprocessor network. An architecture is said to be scaleable if it continues to yield performance increases in proportion to the number of processors in the network, whether that number is two or two hundred. Expansion should be

possible with minimal disruption to the system, ideally involving as few alterations as possible to both the system's hardware and software.

Scalability is also important as it affects the access to the resources required by the network's constituent nodes. As the number of nodes in the system increases, the amount of access each node has to the resources of the system decreases. This leads to the formation of performance bottlenecks, hence, reducing the efficiency of the system. The demand for scalability has led to a design philosophy in which no single resource is assumed to be in restricted supply [51]. This is achievable by replicating the resource that is in demand. For example, shared memory becomes a performance bottleneck in systems that utilise this method of data storage. Providing multiple memory stores increases the effective amount of memory bandwidth per processor and distributed memory guarantees the bandwidth for each processor irrespective of the network size.

Scalability is highly desirable in real time embedded parallel communications due to the need to guarantee task completion within a given time frame.

2.1.3 Reliability

Reliability is crucial to achieving an efficient communications medium. The system must ensure the integrity of both the communications medium and the information transmitted over it. Reliability can be defined as the probability of correct system operation occurring for a long enough time for the system to be useful [52]. There are several other terms in addition to the likelihood of the system ceasing to perform its function [45]. These include:

- Availability: The measure of how much of the system will be affected by faults. A more available system experiences minimal network 'down-time' following faults.
- Performability: Quantifies to what extent the systems performance is degraded by the occurrence of faults. A more performable system loses less of its capabilities following a fault.

- Maintainability: Measures the ease with which faults can be corrected. A more maintainable system can recover from faults more easily.

Interconnection failure will result in messages arriving either with errors or not at all. The former scenario can be detected via the use of error detection algorithms such as parity or checksum bytes. Error correction mechanisms can also be employed to ensure that erroneous data can be detected and amended without requiring retransmission. Communications protocols such as Bluetooth [53] employ 1/3 rate forward error correction (FEC) [54] which involves transmitting each data bit three times and interpreting the most common logic level as the correct one. In the case of wireless protocols such as Bluetooth, this is because free space transmission is more prone to interference.

The probability of single bit errors in the target network is of the order of 1.74×10^{-12} and 9.76×10^{-13} for OS link speeds of 10 Mbits/s and 20 Mbits/s respectively [55]. This was considered low enough for error detection and / or correction codes to be an excessive overhead / use of bandwidth.

Disconnected links and failed processors affect all transmissions and can be detected using time-outs or acknowledge tokens, which also incur overheads. A system may be robust enough to get faults very infrequently but failure can always occur and it is impossible to achieve a truly error free medium. Consideration must be given as to the appropriate response after the occurrence of a fault.

Faults must firstly be detected: detecting errors using parity or cyclic redundancy checks (CRC) once the message has been retrieved from the multiprocessor network can be ineffective, as such error checksums are usually appended to the tail of the message. In multi-router networks, ordering a message retransmission is impractical as the receiving node does not have the origin of the erroneous message and therefore does not know where to make the request. In addition, by the time the error is detected at the receiver, the transmitting node is already executing another task and would need to 'step back' to the task involving the transmission of the erroneous message, which may be impractical. A better solution would be to flush the erroneous message from the system before it causes further problems, such as when a processor attempts to process the faulty data.

Following detection, the next stage involves isolating the error to prevent its effects from disrupting the system. If the fault cannot be isolated, the entire network may need to be reset (which is always viewed as a last resort as it is more desirable to maintain system operation, even if part of the system is unusable due to the fault). It is then necessary to enable the system to recover to a normal operational state, often initiated by the user. Convenience dictates that it is far more efficient to have a system capable of resetting itself, as it is impractical to expect the user to be on standby to perform such a task that occurs rarely. Despite this, resetting the entire multiprocessor network, irrespective of whether or not it is performed automatically or otherwise, causes significant disruption. Network down-time can be significantly reduced if the effects of the fault can be localised, allowing the remainder of the network to continue functioning whilst the affected entities are restored to an operational state, via automatic reset.

This research recognised that there is more to system reliability than minimising the occurrence of failure and that the manner in which such failures are handled can be just as influential to a system's reliability as the frequency with which they occur. The probability of failure can be minimised, but not eradicated and a network with an effective error detection, isolation and recovery mechanism can operate more efficiently and with less network down time, than a network with a lower frequency of failure but also possessing a less effective error handling procedure.

2.2 Characteristics of Interprocessor Communications Networks

2.2.1 Shared and Distributed Memory Multiprocessor Systems

The efficiency of an embedded multiprocessor processing system is affected in part by the need to provide the computational elements (processors) with the correct data for manipulation in time for it to be processed. The data must be stored elsewhere before and after manipulation to allow the processor to work on the next task. This storage usually takes the form of some sort of memory. A processing element must be able to gain fast access to the required information in memory. There are two different ways of using a systems memory to achieve this, dependent on the systems requirements. These are Shared and Distributed memory.

Shared memory systems contain a single, centralised memory resource that can be accessed by any processor. This offers fast transfer of information with very low latency, as all information is only a single memory access away. As it is a shared resource, it suffers from a lack of scalability. As more entities have access to the memory, each has access to it for less time, reducing the effective bandwidth of each node. Another disadvantage is shared resources require careful access arbitration in the form of some sort of central arbiter to prevent any user from monopolising memory access at the expense of others. This must manage access to the resource, synchronise tasks and resolve access contentions. Programmers must take care to ensure that multiple resources are not attempting to access memory simultaneously as this will lead to processors wasting bandwidth as they stand idle whilst waiting for data from memory. Memory bus management in shared memory systems must be performed efficiently in order to maximise available bandwidth and to prevent memory access becoming the performance bottleneck of the system. Scalability is ultimately controlled by the nature of the communications medium.

The largest problem encountered with memory that is shared amongst multiple processors is that of data validity. As every processor can access all memory locations, there is no way for a processor to know whether another processor has modified memory location contents.

Distributed memory affords each processing node a dedicated memory resource, to have direct and unlimited access to, thus requiring no arbitration. This allows for a flexible and scalable solution as the effective bandwidth of each node remains constant, irrespective of network size, topology or architecture. The principle behind this method is a processor will access its own memory far more frequently than it will need to access the memory of another processor. As parallel processing is based on the division of labour between multiple computational elements, such a situation is highly probable as process's input variables can depend on the output of a previous process. When this occurs, the required data must be passed between processing nodes in the form of messages. Message passing solves many of the problems incurred by shared memory, as proven by Hoare [13] but can suffer from increased latency depending on the length of the message path. Such a solution can require the programmer to be aware of which messages need sending between which processors

and must ensure that the receiver does not stall while waiting for them. The research described here is concerned with the development of new protocols with the aim of reducing message latency and increasing reliability.

There also exist hybrid systems, such as the Cray T3D [56], which have logically shared memory whilst having a distributed memory architecture. Shared data is moved around the system on a high-speed 3 dimensional Torus interconnect [57] (a popular distributed memory architecture) similar to the cache or virtual memory operations of recent mainstream processor's systems. This system allows the software designer to think of the processors as sharing memory, whilst the network gains all the benefits scalability offers. Such systems are aimed at the higher end of the distributed systems market and lack other factors such as flexibility, efficiency and cost offered by smaller scale parallel and distributed systems.

Real time embedded multiprocessor applications typically utilise distributed memory, to allocate storage to each processor for data unique to that node, utilising interprocessor communications to transport data required by other nodes. Real time applications cannot afford to hold up task execution due to memory access contention. Whilst the efficiency of memory access is important, the means by which it is transported to and from memory also plays a part in the performance of interprocessor communications.

2.2.2 Serial Vs Parallel Communication Links

As microprocessor technology has developed over the past few years, the width of the microprocessors data bus has increased from 8 through 16 and 32, to 64 bits wide. The decision on the optimum width of the communications links between processors depends on many factors, such as the network topology, architecture, speed, application and cost. Physically distributed systems minimise cost and clock skew, which can adversely affect speed, by utilising serial based communications. Embedded networks that occupy a very small space, for example communicating nodes situated on the same PCB, may benefit from parallel interconnections. Data throughput, a key feature in many embedded systems, is maximised by sending as much data in parallel as is practical.

As clock speeds increase, synchronisation of the parallel data becomes harder as set-up times fall. This highlights an advantage of asynchronous communications, which are more suited to high-speed serial communications. Asynchronous data requires either an encoded clock signal to be sent with the data [58, 59] or receiver over-sampling at a higher data rate to recover the data [60]. Parallel communication links result in increasingly complex circuit board design and increased wiring space and costs, important factors in embedded system design. Whilst the medium used to transport data (serial or parallel) is important, one must also consider how the medium is utilised in terms of how data travels from inception to destination in the most efficient fashion.

2.2.3 Bus and Switch Based Network Topologies

Early embedded multiprocessor systems generally had a static or predefined network layout [5, 56] and used point-to-point connections between processors. The number of entities a processor could connect to depended on the numbers of communications links it possessed, a factor limited by practicalities, available silicon area and cost. Such networks were suited to particular applications dependent on the number of communications links per node. As this parameter could not be altered, there was little scope for flexibility, making these networks application specific. Regular network topologies benefited from reductions in cost and complexity but were only suitable for systems with specific communications patterns. An example of such a network [57] is shown in Figure 8. The processing entities are arranged in a 2 dimensional 'mesh' topology to which each entity can connect 4 adjacent nodes via its communications links. Communications to other nodes must be forwarded via these and any other processors on the desired communication path. Early embedded networks used such 'store-and-forward' mechanisms. A message travelling from node 1 to node 16, say, must be sent via 5 intermediate nodes, all of which must dedicate time and effort to forwarding this message instead of dealing with their own tasks. It is more desirable to have a dynamic network in which nodes can be placed anywhere in the system to suit the intended application. Such networks require arbiters or switches to make the decisions relating to routing messages to their required destination.

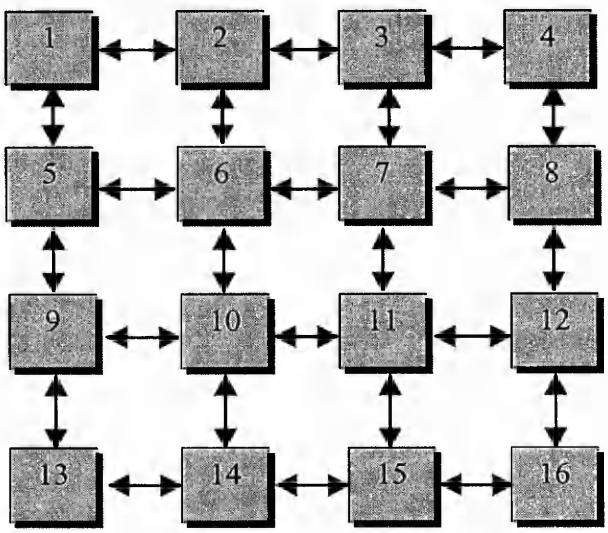


Figure 8: 2-Dimensional 'Mesh' Network Topology

Some multiprocessor systems use a global shared bus communications medium where only one communication can take place at any one time [50]. All nodes can access data on the bus, providing an effective means of transferring information to multiple entities simultaneously. A central arbiter is required to grant access to the bus. Some models, such as the PCI bus [20, 21], utilise a bus-mastering technique, whereby the node wishing to initiate a data transfer must first acquire ownership of the bus. Bus users are polled in turn on a 'round robin' basis to make requests for bus ownership.

Some systems, such as Ethernet [61, 62, 63], attempt to start bus access, only stopping when bus contention is encountered. A Carrier Sense, Multiple Access / Collision Detect (CSMA/CD) protocol is used, whereby on bus contention (where two or more nodes attempt to gain control of the bus simultaneously) both nodes suspend bus access for a period of time before attempting to re-establish control of the bus. All nodes can master the idle bus with the first one to do so gaining control of it and the other nodes must wait until this node has finished the bus transfer, before attempting to access the bus again. This approach wastes bandwidth and leads to low efficiency, which can be a valuable commodity in bus based systems.

A more suitable shared medium based protocol for real time embedded applications is that of Reservation Carrier Sense Multiple Access with Collision Avoidance (RCSMA/CA) [64]. Access contentions are detected in the same way as in CSMA/CD but the contention slot is allocated to a transmitting node instead of allowing the bus to idle. This combines the benefits of the predictable performance token-based protocols, inefficient in light traffic conditions, with CSMA/CD, which is a poor choice for real time systems under heavy traffic conditions.

Bus systems usually suit closely coupled processor nodes. Networks utilising buses as a communications medium usually position nodes that access the bus physically close together as there is often a limited transmission distance. This is due to propagation delays on the arbitration signals wasting valuable bandwidth. Buses often operate in parallel, achieving very high bandwidth. This bandwidth must be shared amongst all the entities that can access the system, resulting in each having an effective bandwidth that is inversely proportional to the number of entities that can access the bus. This often results in bus access becoming the system bottleneck and as such bus networks do not scale well. Bus systems are low cost and offer low latency for closely coupled processors. The scalability and contention issues make buses less suitable for real time embedded multiprocessor communications as access cannot be guaranteed under heavy network traffic conditions. An additional disadvantage of bus based multiprocessor systems is the lack of tolerance to faults due to the communications medium becoming the single point of failure with the probability of failure increasing with the number of entities attached to the bus [45].

Point to point networks offer guaranteed bandwidth and low latency between two communicating entities due to the formation of an exclusive link between them. Point to point connections require no arbitration and cannot have access conflicts as there is only one message channel per physical link. The disadvantage of these is that optimally, every node would require a link to every other node in the network, which is unrealistic as the network size increases, due to extra resources being needed. For example, a network with n nodes requires each node to have $n-1$ communications channels in order to communicate with every other node in the network. This is an extremely inefficient use of communications resources, as any given processor will not be required to send messages to any other processor at any one time. Whilst point to point communications links are impractical for use in larger networks, they have

been used effectively in the 3D Torus [57] topology used by the DEC Alpha Cray T3D supercomputer [56]. A more simple and cost effective method of achieving point to point links between two processor nodes is to utilise a switched network.

'Store-and-forward' communications methodologies reduce the efficiency of intermediate nodes, as they must devote time and resources to receiving, storing and forwarding the message that would normally be devoted to program execution. Multi-hop communications that traverse more than one link, between the message origin and destination, incur massive latency increases compared to messages that only traverse a single link. This is due to the delay increasing proportionally for each link, plus delay incurred at each forwarding node.

Switched networks use message routing devices (routers) to pass messages throughout the network and allow for simultaneous transfer of messages, provided there is no contention for the destination node. No direct route exist between processors, with all communications routed via the router, allowing an entity with a single communications link connected to an n channel router to communicate with $n-1$ other nodes. Adding more routers allows the network to scale exponentially whilst adding only one 'router-hop' or Rhop [23] per router. Routers utilising the full-crossbar architecture [65] allow $n/2$ bi-directional communications transactions to take place simultaneously, provided there is no contention for destinations, as shown in Figure 9. Switched networks scale well, guaranteeing bandwidth irrespective of network size or topology. System complexity increases exponentially as the number of connections increase and latency is worse than that encountered in bus based systems but better than that of the point to point networks. Tight and loosely coupled systems are supported equally well. Routing permits flexible or irregular networks to be formed where network layout is independent of size and application. Advances in silicon technology have enabled switching devices to be developed more easily than ever before and have aided research into network communications, much of which was aimed at switching topologies.

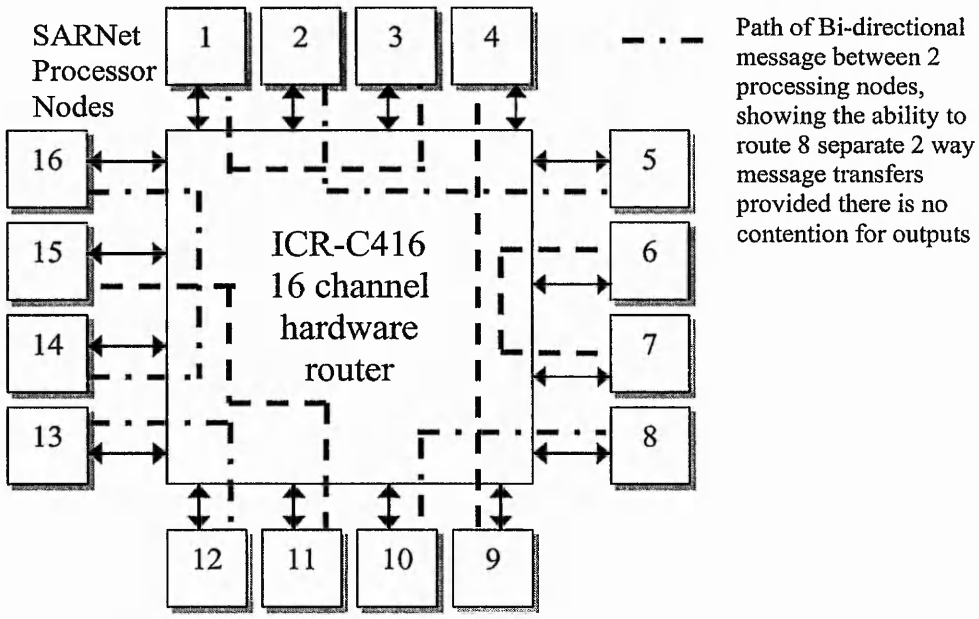


Figure 9: ICR-C416 16 Channel hardware router

2.2.4 Tightly Coupled Vs Loosely Coupled Processor Interfaces

The terms 'tightly-coupled' and 'loosely-coupled' refer to the level of integration between the processor and the interface with the communications network and represents a trade off between system performance and adaptability. Higher bandwidth and lower latency can be achieved by implementing the host system adapter of the network interface onto the same silicon as the processor itself. Processing nodes that utilise such a 'tight' approach include Transputers, iWarp [66, 67] and MDP [68]. Closely coupled communications may seem desirable, particularly in compact embedded applications where space is at a premium, until one takes into consideration the rapid pace of development in the microprocessor market. Advances in the network interface must track those of the processor in order to prevent the interface becoming the performance bottleneck of the system. Enhancements to either the processor or interface may or may not require modifications to the other device, but if both are implemented on the same silicon, the layout for both must be redesigned. A loosely coupled processor interface allows the designer to utilise IP

when upgrading either device. Recent advances in PLD technology have resulted in software based processor core [69] implementation with the remaining logic resources devoted to the design of an easily modifiable tightly coupled processor interface. Closely coupled communications are often suited to parallel bus based interconnections whilst loosely coupled systems are often targeted towards serial links, subject to bandwidth availability.

The success of a tightly coupled processor interface is measured in its efficiency. As a rule, greater performance is achieved through closer integration between the processor and the interface, but at a cost of lack of flexibility. Interfacing to the processor memory bus or its cache controller can integrate the processor and network interfaces more closely. There are other methods of increasing performance without sacrificing flexibility, such as alterations to the interface medium.

2.2.5 I/O Bus Based Interfaces Vs Memory Bus Based Interfaces

A network interface can connect to the microprocessor and its memory via either the system I/O bus or the system memory bus. The I/O bus connection allows the interface, and thus the network, to be connected to any processor with such an I/O bus, affording close tracking of processor technology. This is a particularly favourable solution when interfacing PC's / workstations to the communications network. I/O bus based systems can also be developed relatively inexpensively in comparison to network interfaces utilising proprietary system adapters. The disadvantage of such an interconnection strategy is the competition for I/O bus access with other users of the bus. If the interface is unable to obtain a fixed interval bandwidth from the I/O bus arbiter, the data throughput of the interface will be compromised. Additionally the performance of the I/O bridge, responsible for handling I/O bus transactions, is variable due to wide variations in overhead, bandwidth, and hence latency, between different PC chipsets. Despite such vague implementation parameters, many recent parallel systems, such as Myrinet [70] and SHRIMP [71], use I/O bus based interfaces due to the flexibility obtained from using current, state-of-the-art, low cost, easily upgradeable PC / Workstation processing elements in such systems.

A system utilising the processors memory bus, as its connection between the processor and network interface, forms a more tightly coupled architecture than that of the I/O bus based system, even if the processor and interface are physically distributed on separate IC's. The performance is enhanced for three reasons:

- Messages originate from, or are destined for, memory that is accessed via the memory bus. Overheads are reduced if the message does not require transferring from the I/O bus to the memory bus, thus reducing latency.
- The memory bus normally operates at a higher data rate than the I/O bus, boosting bandwidth.
- The memory bus usually has fewer devices accessing it, thus reducing bus contention.

Memory bus based systems can be viewed as a compromise between the tightly coupled integrated processor systems where the processor and interface occupy the same IC, and the loosely coupled I/O bus based systems, as they afford both performance gains and flexibility. A disadvantage of memory bus based systems is that they must not impede the processors access to the memory bus, as this would affect the processor efficiency. Cycle stealing DMA is often used by such systems to access memory as it utilises unused processor cycles, minimises bus congestion and increases efficiency despite incurring initial overheads. Tailoring individual accesses to the transfer size with burst mode DMA can make further gains in efficiency. The MAGIC network controller, utilised in the FLASH multiprocessor [72], utilised the closeness of communications provided by a memory bus based network interface whilst maintaining flexible support for both message passing and shared memory models, with little performance loss [73].

2.3 Hardware Router System Comparison

2.3.1 ICR-C416 Based Systems

The ICR-C416 is a 16-channel dynamic hardware routing switch developed by the parallel processing research group at The Nottingham Trent University and subsequently marketed by IC Routing Ltd for use in distributed control applications

requiring fast and flexible connections between first generation Transputers [5]. The router switch architecture allows up to 8 simultaneous bi-directional messages, provided there is no resource contention. Each of the 16 links consists of a pair of serial, asynchronous, full duplex lines transferring data at rates of either 10 or 20Mbits per second. As data transfer is asynchronous, no clock information is encoded into the data stream and the receiving node uses an over-sampling (OS) technique to recover the data. Data is transmitted to and from the router in token form, with each token consisting of 11 bits. These are: a logic 1 Start bit, logic 1 ID bit, a data byte and a logic 0 Stop bit. The communication links employ a credit-based stop-and-wait flow control mechanism, requiring the acknowledgement of every token before the next can be transmitted. The 2-bit acknowledgement token consists of a logic 1 Start bit followed by logic 0 ID bit.

The maximum theoretical unidirectional data throughput is 14.55Mbits/s or 1.82MBytes/s at a 20Mbits/s data rate [74]. Bi-directional data requires the acknowledge tokens to be inserted into the bit-stream between data tokens and increases the number of bits that need to be transmitted over the communications link in order to convey a byte of information, referred to hereafter as 'bits per byte', from 11 to 13. In theory, this gives a maximum bi-directional data throughput of 3.08MBytes/s at 20Mbits/s but in practice the system has been shown to take up to 17 bits per byte [5]. This parameter is dependent on factors such as transmission length, network traffic and receiver buffer status. Messages in the ICR-C416 network are divided into 256 byte packets. The maximum message length is 64kBytes.

The packet format of the ICR-C416 system adheres to a Header, Length, Payload format. Headers can be multiple bytes with one routing header for each of the routers that the message passes through and one message header to identify to which message that packet belongs. Routing headers are stripped as the message traverses the router network and the last header byte is identified by an MSB of 0, indicating to the receiver that the next byte contains the length information for that packet. The router network utilises wormhole routing to minimise buffering requirements.

The ICR-C416 could not transmit link status information within the data stream, relying on a control port to monitor faults. This utilised a bi-directional OS link channel separate from the data stream. The control ports enabled a network controller

to configure, monitor and control the devices. A basic mechanism for detecting stalled messages was achievable, whose response to such faults was to issue a global reset. This approach did not scale well due to its design being targeted towards single router networks. The controller required a dedicated link to each router and interface in the network, requiring increased overheads, circuit complexity and a proportionate increase in fault detection and intervention times as the network scaled. The dedicated link was required because the data transmitted along the control link was not packetised and, as such information from one source was not distinguishable from another.

In summary, the ICR-C416 is a simple, flexible, low cost solution for interprocessor communications for irregular embedded networks. It operates at a slower link speed than many comparable networks but its generic format suits many applications. Minimal wiring and low pin counts are considered advantageous in physically distributed networks. When differential transceiver circuits are fitted, the network has been proved to operate with a bit error rate of 3.6×10^{-12} at a data rate of 44Mbits/s over 100m of Cat 5 unshielded twisted pair cable [75].

2.3.2 STC-104 Based Systems

The STC-104 [76, 77] was developed by SGS-Thomson as a routing switch aimed at the second generation T9000 Transputer series. It has 32 bi-directional serial communications channels and each channel consists of two pairs of serial, full duplex links, one pair per direction. The links utilise the Data Strobe (DS) links protocol, doubling the wiring requirements for each channel. The Data and Strobe lines carry data together with an encoded clock signal used by the receiving node to synchronise incoming data. Data rates of up to 100Mbits/s are achievable. Data tokens are 10 bits long, consisting of a parity bit, a logic 0 ID bit and a data byte. Four bit long control tokens consist of a parity bit, logic 1 ID bit and two bits determining the nature of the control token. Theoretically a maximum unidirectional data rate of 80Mbits/s or 10MBytes/s is achievable.

Like the ICR-C416, a credit based stop and wait mechanism is utilised but the number of tokens that can be sent before an acknowledgement token is required is

increased from 1 to 8, permitting a maximum bi-directional bandwidth of 19.05MBytes/s. Messages can be split into packets but there is no limitation on packet size. Messages and packets follow the Header, Payload, Termination token format, with the latter indicating either end of message or packet. Message movement inside the router utilises virtual channels [78] and transfers along these channels require acknowledgement. Unlike the ICR-C416, which possessed little hardware error detection, the STC-104 checks the parity of each token and the DS links provide an inherent disconnection error detection mechanism, as either the data or strobe lines must change state every clock cycle. These oscillations occur even when there is no data to transmit increasing power consumption.

In summary, the STC-104 was one of the highest performance routers available and has managed to remain in demand long after the demise of the T9000, due to the DS links protocol being incorporated into the IEEE-1355 standard [79]. DS links are aimed at systems with distances between nodes of up to 1 metre, increasing to 10 metres if differential drivers are used and 500 metres with fibre optics.

2.3.3 Myrinet Based Systems

Myrinet, developed by Myricom [80, 81], is a switched network originally designed for use within Massively Parallel Processor (MPP) systems. Myrinet built upon the results of two earlier research projects; the Caltech Mosaic multicomputer [82], and the ATOMIC LAN, [83, 84] which comprised Mosaic components. The ATOMIC LAN had some similarities with the ICR-C416 network, most notably, the adoption of the credit based flow control mechanism. Like the other two routers examined, wormhole routing is utilised and devices are available with 4, 8, 16 or 32 bi-directional communication channels. A single channel consists of 9 full duplex pairs of wires, transferring data in parallel at a frequency of 80MHz, or 720Mbits/s per channel. A Myrinet message consists of a series of 9 bit parallel characters, with each character being either an 8 bit data byte or 5 bit control token. No start or stop bits are required, boosting bandwidth efficiency further. This gives a maximum unidirectional data throughput of 640Mbits/s. A full duplex pair of 640Mbits/s channels is referred to as a 1.28Gbits/s link. 10 and 100 Base-T Ethernet use such

figures to determine bandwidth despite the fact that Ethernet channels only transfer data in one direction at any one time [30].

Interprocessor communications in Myrinet systems are asynchronous, with receivers employing a pipeline-synchroniser circuit [85] to recover the data. Cumulative signal skew effects limit the maximum distance between nodes to 25m for cable connections but fibre optics enable the realisation of more physically remote systems. Non-return-to-zero (NRZ) encoding and a sampling window technique enable bit error rates of the order of 10^{-15} over 25m cables to be achieved. The worst case (path formation) latency through an 8 port router is quoted as 550ns [30].

Unlike the ICR-C416 and STC-104 systems, Myrinet utilises a permission based flow control mechanism that does not require acknowledgement of characters before transmission of the next character can begin. This approach allows back-to-back data transmission to be achieved and, if receiver buffer throughput is maintained bi-directional data rates of 160MBytes/s are achievable. The packet format is similar to that used by the STC-104, which is Header, Payload, and Termination (the termination can also contain a CRC checksum). The header can be of variable length and follows the same technique as the ICR-C416 to denote end of header, with the last header character indicated by an MSB of 0 instead of 1. Header stripping techniques are employed at each router. Fault detection and correction exists in the form of a 50ms link time-out period after which a Forward Reset (FRES) token reinitialises the link. Myrinet offers very high bandwidth in comparison to the other systems studied due to the parallel nature of communications, but at a cost of increased wiring and connections (a single 32 port switch has over 900 pins).

2.3.3.1 Myrinet / PCI Host Interface

The Myrinet system is of particular interest as it possesses commercial interface hardware to link PCs / workstations to Myrinet routers. The interface contains a custom designed processor [81], host system interface and memory. The processor can be programmed to handle different communication protocols. The interface significantly reduces the PC's involvement in packet transmission to achieve low-latency communications.

2.3.4 The Reliable Router

The Reliable Router [86] was designed solely for use in high performance parallel systems, as were its contemporaries, the Chaos router [87], and the WARRP [88]. Whilst its target applications are very different to those of the Transputer based networks studies, it is worth noting for its refinements in features and functionality. The Reliable Router utilised 28 transmission lines per bi-directional channel, including 16 data lines, 8 control lines and 4 clock lines. The links were bi-directional to reduce the pin count, but permitting communications in one direction only at any one time. To compensate for this the links were capable of operating at bit rates of up to 3.2 Gbits/s, utilising a streamlined switching technique to maximise throughput and minimise delay along the message path. The Reliable Router was designed for board level or backplane level systems, where the distance between nodes was minimal due to the network being targeted at very high bandwidth parallel processing applications. Plesiochronous data recovery [89] was employed, the transmitting clock was sent with the data, then decoded and used to read the received data.

Unlike the ICR-C416 based system, which thrived on a minimal protocol, the Reliable Router utilised a complex data format required in very high bandwidth systems. Data transfer occurred in parallel, but four consecutive transfers were used to assemble a 'frame' of data. Each frame included 16 bits of data plus 8 control bits, three of which were parity, one for each data byte, one for control information. This may seem excessive, but it should be considered that the bit rates and synchronisation techniques of the Reliable Router were revolutionary at the time. The Reliable Router had four channels plus one control channel, with each physical channel possessing five 16 deep, 75 bit wide buffers, as it could implement up to 5 virtual channels. A 'permission to release' buffering mechanism prevented the overwriting of a buffer until its contents had been successfully transferred ahead. This permitted frame retransmission following failure. Though implemented at a cost of massive buffering increases, fewer resources were required than with the 32 channel STC-104. Virtual channels had also been used up to this point in regular networks to implement deadlock avoidance and adaptive routing mechanisms.

The Reliable Router utilised a Unique Token Protocol (UTP) in conjunction with the flow control mechanism. Data transfer was confirmed on a per-hop basis, removing end-to-end acknowledgements and thus distributing the responsibility for ensuring message arrival amongst the network resources, in addition to providing a scalable transmission media. Resources were only released when the flow control system was certain of the delivery of the flow group, whose integrity was ensured upon receiving a unique token (which followed the packet) acting in a similar manner to a termination token. If a link failed during packet transfer, each part of the message could continue to the destination node, appended with a modified token highlighting the error and requiring routers to store the routing header for the duration a packet was active. Message reconstruction could take place given the severed message, modified token and other packet information, presumably using the software layers of the protocol. This highly complex, flow control based, fault handling mechanism attempts to minimise the overheads of fault recovery by including enough information within the transmission to enable message reconstruction. One can assume that for such an approach to be viable, the frequency of failure was high enough to warrant transmission of such a great number of extraneous non-data bits in each message. Such a premise is similar to that of deadlock avoidance versus deadlock detection and recovery schemes, where the overheads of the latter is preferable when compared with the penalties imposed through the routing restrictions of the former. Whilst the Reliable Router is clearly unsuitable for the target applications of this research, important lessons can be learnt in terms of the recovery-based fault tolerance implementation in the data stream and its merits.

2.4 Interfacing to PCs – The PCI Bus

Most processors need to communicate with peripheral components and the outside world by some means or other and to this end a plethora of communications standards and protocols have been developed, some communicating data serially, others in parallel. This section reviews some of the communications protocols utilised by PCs, focussing mainly on the PCI bus [32, 33].

Serial PC interfaces include the IEEE 1394 (Firewire) standard [90, 91], Serial ATA [92] and Universal Serial Bus (USB) [93, 94]. Parallel interconnection methods

are preferred where bandwidth is crucial, especially when it is recalled that access to these bus based communications media must be shared amongst all of the devices attached to that bus. As modern, graphic oriented PC applications require large data transfers, so the interface to the PC becomes the bottleneck. As processors tend to operate at substantially higher clock frequencies than their associated I/O buses, it makes sense to provide a bus that operates at the same speed as the CPU, which is the essence of the local bus. A local bus moves peripherals from the slower I/O bus and places them closer to the processor's system bus that permits faster data transfer to the processors memory. The PCI local bus is becoming the most popular in modern PCs. It is not directly dependent on the speed or size of the processor bus and offers expansion as processors develop.

Many types of expansion bus have been developed, some have become standardised, and others are targeted at particular applications or particular platforms. With a variety of different buses on offer, the bus selected should be fast, cheap and in common use.

Several bus systems, used in various different processor configurations, have been compared [95], identifying the PCI bus as the most suitable for the three criteria specified above. The ISA bus [96], used in PCs, is cheap and in common use, but suffers in terms of performance and can only be used with *x86* processors. The MCA [97] and EISA [98] buses are fairly fast with data transfer rates of up to 20MBytes/s and 33MBytes/s respectively, but are not widely used. The former is moderately expensive and only used in IBM computers whereas the latter is expensive and can only interface to *x86* processors, in addition to being held back due to backwards compatibility with the ISA bus, limiting its bandwidth to 33MBytes/s. The industrial VME bus standard IEEE 1014-1987 [99] can be used on any platform, has a bandwidth of 20MBytes/s, fairly common but suffers from costs which whilst acceptable for large scale industrial applications, are excessive for most PC users. The PCI bus is ubiquitous, being included in virtually every modern PC, very cheap, very fast and has no limitations in terms of target platforms. Unlike some of the other buses, such as the ISA bus, it is also a recognised standard [33].

When selecting an interface medium to link to the SARNet, during development of the PCI-OSLi, as discussed in section 1.2, it was considered that the shared nature

of the PCI bus was offset by its high bandwidth, relative to the serial router link. In the implementation of the PCI, the 32 bit parallel bus operating at a 33MHz-bus rate gave a throughput of 132MBytes/s. Higher data throughput is achievable by increasing either the width of the PCI bus to 64 bits or the operating speed to 66 MHz, giving total data throughputs of up to 528MBytes/s. The ISA bus throughput of 16MBytes/s, whilst higher than the 3.08MBytes/s achieved by the credit based ICR-C416 protocol at 20Mbits/s was considered too low, given that the bandwidth must be shared amongst all the other devices connected to the ISA bus. Given the proven ability of the ICR-C416 to operate at data rates up to 44Mbits/s, giving a theoretical maximum throughput of 3.38MBytes/s, the higher data rate and enhanced technology of the PCI bus was strongly favoured during development of the PCI-OSLi. The same argument applies when interfacing the PC to the NTR-FTM08 router network via the FT-PCI-OSLi.

The PCI bus has the following characteristics:

- The processor and bus are coupled via a bridge separating the processor and its main memory from all other devices attached to the bus.
- Arbitrary length DMA burst transactions are synchronised to the rising clock edge of the PCI bus clock.
- Address and data buses utilise the same physical connections, and therefore require multiplexing. Only 49 pins are required for the 32-bit bus implementation, reducing connector and chip sizes and pin counts.
- Supports ISA / EISA and MCA buses via an interface to the expansion bus.
- Configurable through software and registers.
- Platform independent specification.

2.4.1 PCI Bus Operation

Data transfer operations on the PCI bus occur between two devices attached to the bus, known as PCI agents, as shown in Figure 10. The agent that requests the transfer is known as the initiator, or master, whereas the recipient is referred to as the target, or slave agent. By definition, a target cannot initiate the transfer. A PCI data transfer has

similarities to a DMA transfer, where data is written directly to or from the system memory. The PCI master initiates the transfer to acquire ownership of the bus as is necessary to access the system memory in a technique termed bus-mastering. The bus-mastering technique allows data transfer between the system memory and the PCI agent without involving the host system, freeing it for other purposes.

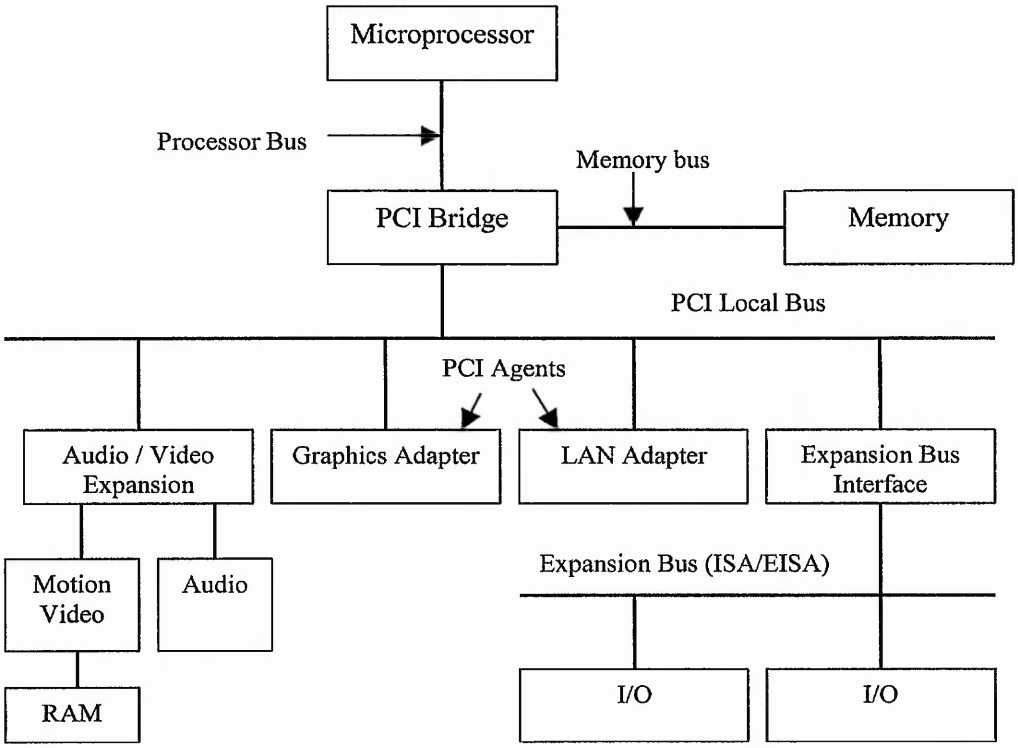


Figure 10: A typical PCI bus arrangement

Arbitration of the PCI bus is performed separately for each access, preventing an initiator from holding up the bus between accesses (which can occur with the EISA / MCA buses). A burst transfer is seen as a single access in arbitration terms. Arbitration can be performed whilst the bus is still running, preventing it from reducing the bus bandwidth. A central arbiter receives requests from each initiator and grants control of the bus. Control must be assumed, in the form of starting a transfer within 16 clock cycles, otherwise access is surrendered and an error flagged. The PCI specification leaves the arbiter implementation to the system designer rather than specifying a particular model.

The PCI bridge is far more intelligent than the bus controllers used in the ISA / EISA and MCA buses and can optimally co-ordinate CPU accesses to the addressed PCI unit. The PCI bridge can effectively function as a fast buffer between the initiator and the target, synchronising those devices, and permitting the bridge to translate a single CPU access into a PCI burst.

3 FAULT TOLERANCE

The specification for the features aimed at improving the tolerance to faults possessed by the FT-PCI-OSLi and FT-SARNIC was determined by implementation of these features in the NTR-FTM08 router. This section of the thesis defines the mechanisms employed by the router to achieve a robust communications medium and their operation. Fault detection and recovery in the ICR-C416, STC-104 and Myrinet routing networks is reviewed to compare and highlight the increased functionality gained through utilisation of the NTR-FTM08 network.

3.1 Overview

Network failure can be classed as either 'hard' (permanent) or 'soft' (transient) [52]. Failures are initially treated as transient, as the higher levels of the system attempt to recover from the fault by trying to repeat the failed action. Failure to recover indicates the presence of a hard fault that must be isolated and bypassed. A fault in wormhole routing networks [22, 34] will result in the network stalling as progress depends on available buffering resources. The inability of a message to make progress directly affects all other messages, as they require the use of the storage resources occupied by the stalled message.

In the event of a fault occurring, one can either attempt to work around the fault or fix it. Much of the research associated with fault tolerance in parallel processing systems has focused on adaptive routing algorithms [100], which enable network traffic to bypass permanently disabled links by finding alternate paths through the network. Such action is possible due to the fixed, regular layout of such network topologies as the 2D-mesh topology, commonly utilised by Transputers. No attempt is made to fix the fault and return the network to its full operational capacity; hence traffic increases on the remaining operational links. Multiple faults may eventually bring the network to the point where it would be more productive to reset it. Adaptive routing algorithms facilitate network operation in partially disabled networks. The NTR-FTM08 router, like the ICR-C416, utilised Group Adaptive Routing [60] to

allow the user to configure network-specific node groupings, permitting multiple paths to avoid failed links.

Irregular networks, such as those employing routers and including the target area of networks in this research, have a myriad of possible network topologies, further complicating the use of adaptive routing algorithms. Adaptive routing algorithms must ensure that deadlock cannot occur as a result of their action as this results in the effects of the fault becoming far more widespread. In such networks a more productive solution would be to restore the network to a fully functional operational status as soon as possible. This requires the detection, isolation and removal of faults by the network constituent parts. Many earlier switched networks ignored the possibility of hardware failure and incurred large overheads due to their reliance on software layers to provide fault tolerance.

Minimal adaptive algorithms, such as Group Adaptive Routing [101], can easily be incorporated into irregular networks. Group Adaptive Routing is used in the ICR-C416 and NTR-FTM08 based routing networks, where there is a limited control over possible alternate routes. Group adaptive routing maximises link bandwidth by spreading network traffic over as many links as possible, allowing the easing of bottlenecks and improving a network's cross-sectional bandwidth [102].

3.2 Fault Detection

The NTR-FTM08 router provided hardware support to detect and isolate faults, reduced software overheads and increased system reliability and robustness. The NTR-FTM08 network ensured that when errors occurred, they were isolated and their effects were prevented from propagating through the network. This section of the thesis identifies the types of faults detected by the NTR-FTM08 and thus defines the fault detection requirements for the FT-SARNIC and FT-PCI-OSLi interfaces.

The previous OS links based ICR-C416 network, with the SARNIC and PCI-OSLi interface, possessed no means of error detection within the token format. Group adaptive routing enabled blocked links to be bypassed but the lack of error detection and isolation meant the network might stall before the fault was isolated and a detour

could be taken. The system could be monitored via the control port of the ICR-C416 but, as only a single control port monitored the system, reaction and intervention times increased proportionally with system size and as such the solution was not scaleable.

The modified OS links based protocol, employed by the NTR-FTM08 routing network and utilised by the FT-SARNIC and FT-PCI-OSLi, provided a means of verifying link functionality and offered protection against several failure scenarios. Instead of a single, centralised, fault monitoring system, the responsibility for fault detection and recovery was shifted from a global solution to a local one, where each communications link was monitored by the two nodes at either end. This allowed for a fully scaleable, decentralised network with improved tolerance to faults.

The enhanced fault detection and recovery solution provided by the NTR-FTM08 network addressed several possible causes of network failure [103, 6], including:

- Buffer overflow
- Retaining Link Confidence
  - Loss of an acknowledge token
  - Disconnected network connection
- Message Delivery Errors
  - Packet arrival out of order
  - Incorrect message length
  - Synchronisation errors
  - Delivery of message to the wrong address
  - Undeliverable messages
- Deadlock detection and avoidance

3.3 Buffer Overflow

When data was received but there were no free resources to accept the data, the buffer overflowed, resulting in either 'spilled' or overwritten data. Such a situation resulted in the loss of part or all of a message, which in certain systems was

undesirable, and in others fatal. Point to point networks could order the retransmission of the message but this was impractical, if not impossible, in multi-router networks, as the node ordering the retransmission was ignorant of the origin of the lost message (beyond the link the node received the message on). Even in point to point networks, ordering a retransmission might not have been possible due to the nature of the error. As there was insufficient space in the receiver buffer for the incoming message, existing data, which was part of another message, was overwritten by the incoming message. As the incoming message replaced the existing data, the receiving node treated it as part of the existing message and might not have been aware there was a new message or that it was erroneous and thus a retransmission would not be requested. The easiest way to prevent buffer overflow was to ensure there was always sufficient buffering resources available at the receiver before transmitting data.

3.4 Flow Control Protocols

3.4.1 Credit Based Flow Control

Credit based flow control is a stop-and-wait data handshaking procedure in which data transmission is prevented until acknowledgement of the receipt of the previous data is received, informing the transmitting node that sufficient buffering resources are available at the receiver. This flow control method was used in OS and DS [79] links and the OS link based protocol used by the ICR-C416 router and SARNIC and PCI-OSLi interfaces. An acknowledgement token was returned for each 'frame' of data received, as Figure 11 demonstrates. For OS and DS links, a frame of data was set at 1 token and 8 tokens respectively. An 11 bit token consists of a logic 1 start bit, an ID bit, 8 data bits and a logic 0 stop bit. The ID bit was logic 1 for data tokens and logic 0 for acknowledge tokens. Acknowledge tokens were just 2 bits long, and consisted of a logic 1 start bit and a logic 0 ID bit. Transmission of subsequent tokens was suspended until the transmitting node has been made aware that the previous token had been recognised by the receiver. Back-to-back data transfer could occur in unidirectional credit based data transfer, as Figure 11 shows, but bi-directional transfers required the interleaving of acknowledge tokens between data tokens, reducing the bi-directional throughput, as shown in Figure 12.

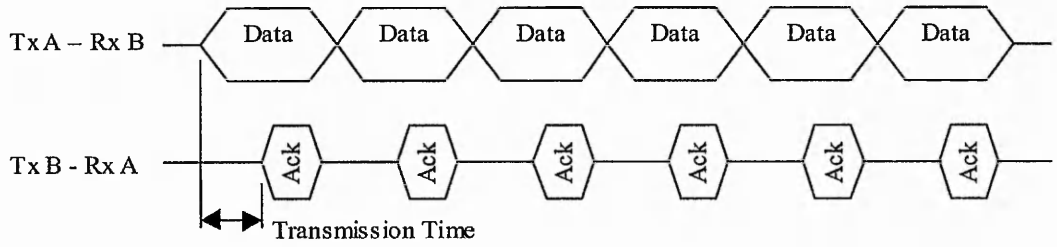


Figure 11: Unidirectional Data Transfer Using Credit Based Flow Control

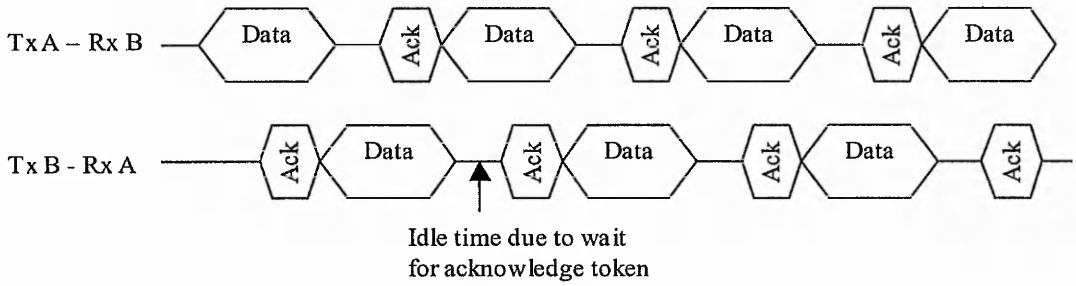


Figure 12: Bi-directional Data Transfer Using Credit Based Flow Control

Thirteen bits were required for every byte of data transferred over the serial ICR-C416 link with the acknowledge token contributing to the duration of the data token. Unless the transmission of an acknowledge token occurred the instant the received token arrived at the receiver, transmission of the next data token was delayed until receipt of the acknowledgement, thus taking in excess of 13 bit periods to transmit a data byte, as discussed in section 2.3.1. Failure of the credit based flow control protocol to achieve 13 'bits per byte' transmission resulted in idle periods on the transmission link and reduced the link efficiency. The minimum 13 bits per byte transmission also assumed zero delay across the transmission medium and, more significantly, the transceiver chips. Figure 12 shows how ICR-C416 bandwidth could be wasted because the transmitter could not send out the next token until arrival of an acknowledge token.

Hence, in practice, the implementation of the OS Links protocol in hardware would not achieve 13 bits per data byte bi-directional transfers. It was documented

that Transputers, utilising the OS Links protocol, could demand up to 17 bits per data byte for bi-directional data transmission [5].

A major shortcoming of the credit based flow control mechanism was the problems incurred with the loss of a control token. As the transmission of subsequent tokens relied on the receipt of an acknowledgement, the transmitter would assume that the receiver buffer was full, whilst the receiver would assume that the transmitter had run out of data to send. The result would be a stalled link, which cannot be detected and whose operation could only be restored following the reset of the link.

3.4.2 Permission Based Flow Control

The issue of link stalling, in addition to the loss of bandwidth incurred in the transmission of acknowledgement tokens, led to the development of an alternative method of controlling communications. This method, also referred to as Stop/Go or Xon/Xoff flow control, permitted back-to-back data transfer, subject to the receiver buffer being able to handle the incoming data stream. It was utilised by the Myrinet [70, 80, 81] message-passing network from Myricom for communications within Massively-Parallel Processors (MPPs) and is also prevalent in point-to-point networks.

Figure 13 demonstrates the flow of data between processors across the NTR-FTM08 communications network, using a permission based flow control protocol to dictate when the communicating entities could transmit data tokens. Initially, a Go (or Xon) token was transmitted when the node was ready to receive data. The Go token was one of the new flow control tokens, denoted by an ID bit set to zero. The other side of the link transmitted data tokens back to back, requiring no acknowledgement before subsequent tokens were transmitted. Once the number of tokens stored in the receivers buffer reached a predetermined figure, the receiving node sent a Stop (or Xoff) token to the other side of the link, to inform the transmitting node to temporarily suspend data transmission. The receiving node could then clear the backlog of tokens stored in its receiver buffer. Once the number of tokens in the receiver buffer fell below a certain level, the receiving node sent a Go token to the transmitting node, informing it to resume transmission.

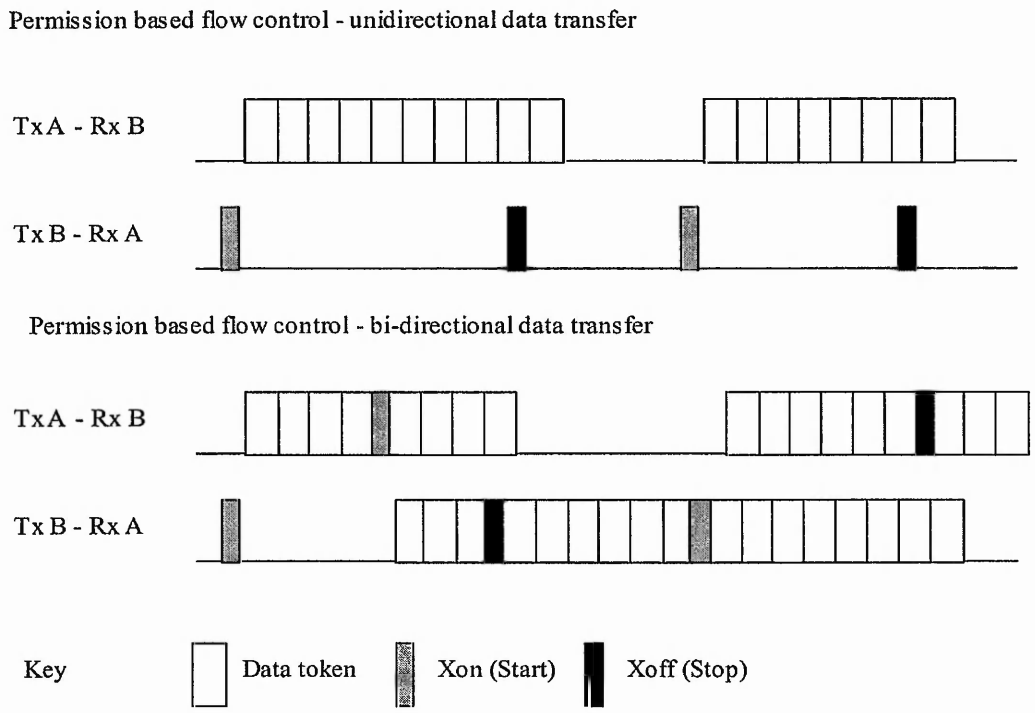


Figure 13: Unidirectional and Bi-directional Permission Based Flow Control

The amount of data throughput across the link was dependent on the ability of the receiver to keep data flowing through its buffer. Theoretically, the receiver could prevent the accumulation of a token in its buffer by passing it through, for transfer to memory, before the next token entered the buffer. Hence, the maximum data rate depended on the rate at which data could be sent down the transmission medium, which was determined by the clock speed. However, in practice the receiver may not be able to transfer the data to memory as fast as it would like. If the receiving node was another router, it might not be able to output the incoming data to its destination as that particular output could be in use. Grouped adaptive routing could reduce the likelihood of this occurring, but not eliminate it completely. If the receiving node was a processor interface, such as the SARNIC, the data may have to wait to be transferred to memory, if the memory was being used. This was because it utilised cycle stealing DMA transfers to access memory where the processor always assumed a higher priority. Other factors that could impede the receiver, in the ejection of tokens from its buffer, included: faults on other links, delivery of messages to the wrong destination, synchronisation and framing errors resulting in corruption of data and undeliverable messages monopolising network resources.

When the link was in the 'Go' mode of transmission, only 11 bits were required to send each data byte, due to the lack of acknowledge tokens. This meant that the maximum amount of data present in each token (Number of data bits / total number of bits) increased from 61.5% to 72.7%. A disadvantage of this protocol was that it was harder to calculate data throughput for any given time as it depended on suspension of transmission, which in turn is dependent on many factors.

The Stop, or Almost Full (AF) [37] and Go, or Almost Empty (AE) [37] values determined the efficiency of the protocol, most notably the difference between these two values. If they were too close together, the link spent more time than necessary with link activity suspended. If they are too far apart, the link was not as responsive as possible to variations in network traffic, wasting the network's bandwidth. For a link to be operating at its most efficient, the receiver should always have some data to process, and should never have to suspend link operation, due to there being too much data to process.

3.5 Retaining Link Confidence

For a communications link to operate successfully, both end nodes must always be aware of the current state of the link. Link activity can be monitored, indeed certain systems, such as IEEE Std. 1355-1995 (DS Links), require constant signal activity in order to operate successfully, as clock recovery data is encoded into the data and strobe lines. The OS links based system utilised by the ICR-C416 based system did not require such activity and in the absence of data to transmit, the line was dormant. In fact there was no way of distinguishing between a dormant link and a faulty link. One possible way of overcoming this was to provide a status check on the link in the absence of data. A control token could be used for such a purpose but by using the Go and Stop flow control tokens further information about the link status could be conveyed. Continuous transmission of link activity signals increased power consumption and noise and was unnecessary. So it was more efficient to transmit such tokens periodically, signalling an error if further activity had not occurred since the last status token, whether in the form of data or another status token. Such tokens were referred to as Heartbeat tokens and were used in the NTR-FTM08 to detect disconnection errors. Following such errors, the link must be reset in order to re-establish the disconnected link. This was done by a handshaking mechanism, whereby

the first node to detect such an error sent a 'Connection Request' token. This was repeated periodically until the connection request token was received, indicating the restoration of the channel between the two nodes.

Heartbeat tokens meant that the loss of a Go token did not lead to the link stalling as retransmission occurred after another time-out period had elapsed. The only loss incurred was in terms of bandwidth. The loss of a Stop token was more serious. Another would be transmitted but the receiver buffer might have filled by the time the time-out had elapsed and the next Stop token sent. Expanding the receiver link interface buffer to be able to accommodate the loss of a single Stop token could require up to 32 more buffer locations *above* the AF level to store the data tokens that could arrive before the next Stop token. There was no guarantee that the second Stop token would not be lost also, so this precaution was not really worth the resources required in implementation.

3.6 Message Delivery Errors

3.6.1 Packet Arrival Out of Order

Long messages could be split into multiple packets to prevent monopolisation of system resources by a single message. In multi-router networks with Group Adaptive Routing, where many possible paths between source and destination existed, uneven network loading could result in a packet reaching the destination before its predecessor. Such a scenario could also occur if an earlier packet was lost or stalled during transit across the network. This would result in the message being reassembled in the wrong order, leading to the data being incorrectly interpreted.

A solution was to ensure that the data arrived in the same order that it was sent; alternatively to number the packets so that they were rearranged into the correct order by the receiving node. Numbering packets could be a problematic solution for three reasons:

- Numbering packets required another header byte, which would need to be distinguishable from routing and message headers. The header contained the message ID, which is identical for all packets in the message.

- An alternative packet numbering strategy was to assume that few packets from any message were actively in transit across the network at any given time, and assign a single bit, set on alternate packets to identify a packet relative to the packets either side of it.

- If an out of order packet was detected, it could not be removed from the network, as it was still required. Therefore it must be stored until the expected packet(s) arrived at the receiver. When the out of order packet was the next in sequence, it could be transferred to memory from the system. However, if a packet were removed from the network due to an error, the next packet in the sequence would not arrive. Hence the receiving node, unaware of this, would store every packet, utilising massive buffering resources.

A lesson from the ICR-C416 network protocol was that 256 bytes was an adequate packet length for small control messages of the type used in the Transputer based control applications it was designed for use in. For larger data communications, of the type favoured by microprocessors such as the SA-110 and the PC, it increased overheads, as multiple packets were required to send relatively small messages. An alternative solution [37] was developed which allowed the user to dictate the maximum packet size and also to allow an entire message to be sent in a single packet. This affords the user greater flexibility in network operation by giving them the option of sending a message in a single burst, reducing overheads, or splitting it to share resources.

3.6.2 Incorrect Message Length

This error occurred when tokens were lost or incorrectly sampled. In the OS links based protocol employed by the ICR-C416 based network, packets had no 'end of packet' delimiter. Once a message channel was set up to sample incoming data tokens, all incoming tokens were assumed to be part of that packet until the counter

reached zero, to indicate end of packet. In the lost token scenario, illustrated in the ICR-C416 network depicted in Figure 14, packet 1 has one missing token. As the last token from packet 1 is received, the count is not zero, so the first few tokens from packet 2 are assumed to belong to packet 1 and are assigned thus. The next tokens from packet 2 are interpreted as packet 2's header and if they are not what is expected, packet 2 stalls at the receiver whilst packet 1 is transferred to memory, with no error being noticed until the data is used.

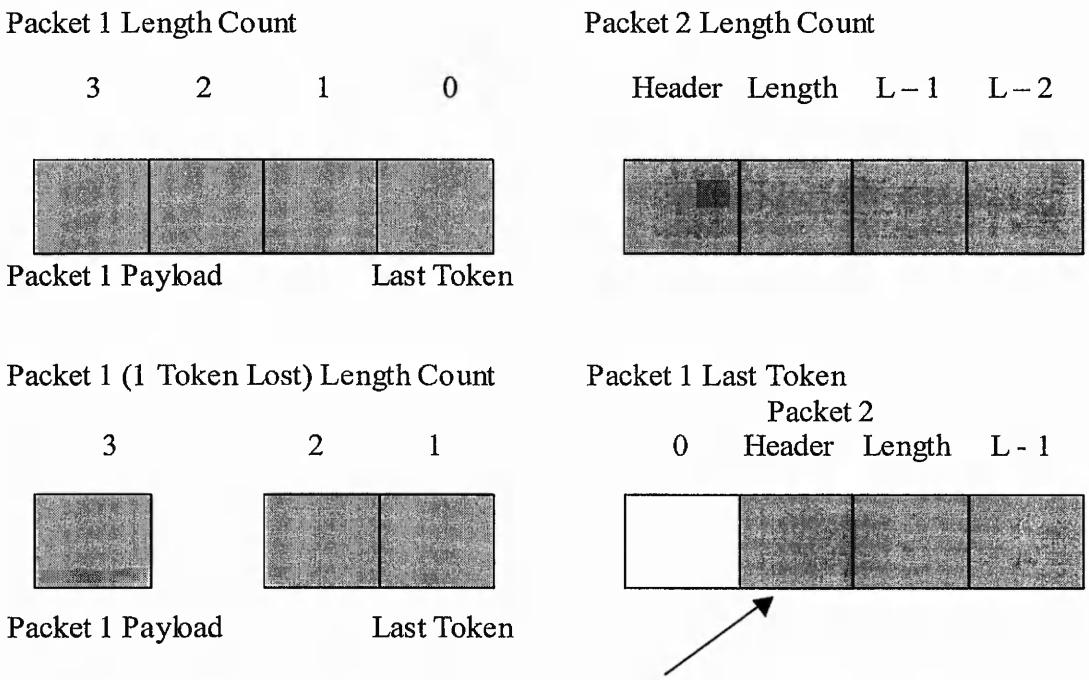


Figure 14: Token Loss Resulting in Incorrect Message Interpretation

There was obviously a need to monitor messages for missing tokens. The addition of an End Of Message termination token in the NTR-FTM08 protocol permitted the message length count to be checked against the occurrence of such a missing token. If the message termination token occurred simultaneously with the message count reaching zero, there was no error. Otherwise, both longer and shorter than expected messages could be detected. The detection of an incorrect message length at a midpoint through the network, before the message reached the destination, required a

means of notifying the receiving node that an error had occurred and that the message data should be treated with caution.

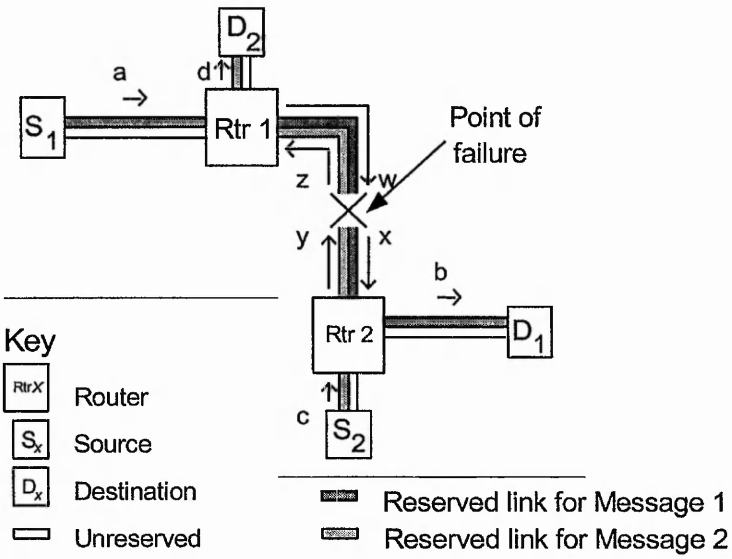


Figure 15: Network Failure in a Multi-router NTR-FTM08 Network

As the NTR-FTM08 router network in Figure 15 shows, the failure at point X affected message 1, travelling from S1 to D1, and message 2, midway between S2 and D2. Once this problem was detected, the headers of the messages had already reached the destination, but due to the use of wormhole routing techniques, the tails of the messages had not left the source node. Message 1 was terminated prior to Router 2 and was flushed from the network, freeing the other links of router 1 for use by other messages. At Router 2, Message 1 was terminated with a 'Bad End of Packet' (BEOP) termination token, informing destination D1 that it was the end of that particular packet. This also freed other links from Router 2 for use by other messages, as the transmission of that packet could be considered complete. The same actions were taken for Message 2: the part of the message held in Router 2 was flushed, as were any remaining tokens to be transmitted by S2. The tokens belonging to Message 2 which were held in the buffers in Router 1 were passed to destination D2 as normal, but were appended with a BEOP token, indicating the message had been prematurely terminated. The user could decide, at a higher level, the action to be taken on receiving a BEOP token, allowing greater flexibility.

This approach allowed the message to be terminated as soon as a fault had been discovered, freeing up network resources for other messages and informing further nodes along the messages path that no further tokens were expected, preventing the lost data scenario described earlier.

3.7 Synchronisation Errors

Framing errors were possible in the ICR-C416 based network due to the asynchronous data transfer and the over-sampling techniques employed in data recovery at the receiver. The over-sampling technique employed by the receiver allowed a skew tolerance of up to 1/3 of a bit per 11-bit token. Cumulative skew in excess of this lead to synchronisation errors of which the ICR-C416 network had no means of detecting or responding to.

The last bit of each token was a logic 0 stop bit, required to allow the sampling circuit to 'recover' between each token. Asynchronous data was recovered from the communications links by taking three samples per bit. Sampling on the leading and trailing edges of the sampling clock effectively doubled the link data rate with respect to the sample clock. The operation of the asynchronous '1.5-times' oversampling technique [60] employed by the serial communications links was dependent on whether or not the start bit of each token was first detected on the leading or trailing edge of the sampling clock. The synchronisation error mechanism checked the incoming serial data line and flagged an error if logic 1 was detected when the stop bit was expected. Such an error detection system was not foolproof as a synchronisation error may be overlooked if the bit preceding the stop bit was incorrectly sampled and was zero.

3.8 Incorrect Message Address / Undeliverable Messages

'Lost' messages could occur due to header corruption or a blocked or damaged link. If the routing header was corrupted to another valid header value, the message would be delivered to the wrong destination. If the routing header was corrupted but did not conform to any valid headers, the message would not be outputted from the router, its progress across the network would cease and it could block the progress of

other messages reliant on the resources that this message was occupying. Undeliverable messages must be removed from the network as quickly as possible to prevent them blocking resources required by other messages and also to impede the formation of deadlock dependency cycles.

The arrival of an unexpected message at an end node in the ICR-C416 network, such as the SARNIC or PCI-OSLi interface, triggered an interrupt requesting a software header check. Unexpected packets could not always be identifiable in software, making this an unreliable and time-consuming process. The user had to respond to the interrupt either by providing the interface with the information required for processing the message, or flushing it from the network. On receiving a message intended for another node, the recipient could not pass it back onto the network for the intended node to collect. It had to retrieve the message, depacketise it to store in memory and then construct a new message for the intended node (after having found out which node the message was intended for, requiring user intervention), and output it back onto the router network. In almost all cases removal of the message from the network was the only viable solution.

Removal of messages from the network required their retransmission, involving initiation by the user via the higher software levels of the system. This was hard to achieve, as the user was unaware of the problem that had occurred.

3.9 Deadlock

Deadlock is a state caused by the cyclic dependency of two or more messages on each another, which results in the network stalling [103]. Deadlock was not of major concern to the FT-PCI-OSLI and FT-SARNIC designs as, being end nodes, they could not form part of a deadlock cycle. Hence the FT-SARNet and the FT-PCI-OSLI formed part of a system with an integrated deadlock detection and recovery strategy.

Figure 16 displays a single cycle deadlock situation, possible in ICR-C416 networks, in which a message can not advance towards its destination until other messages have relinquished resources, which is in turn dependent on the movement of another message and so on.

Research was carried out to assess the effects of this complex problem in irregular switched networks [104, 105]. Three strategies were identified for dealing with deadlock: Prevention, Avoidance and Recovery [106].

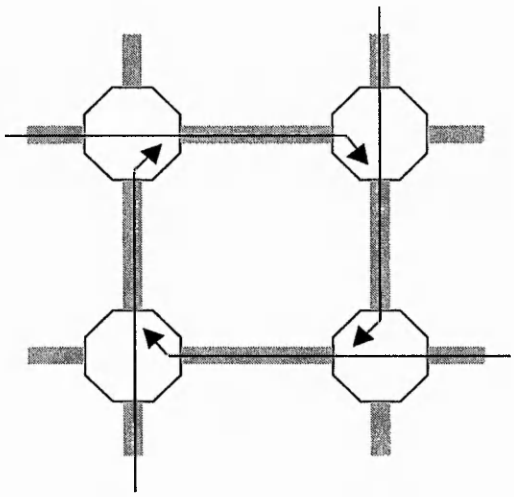


Figure 16: Deadlock in a simple router network

3.9.1 Deadlock Prevention

Deadlock prevention requires the reservation of all required network resources between source and destination before message transmission. This is essentially a step backwards, in terms of network communications, as it reduces the network from packet switched to circuit switched, guaranteeing bandwidth whilst monopolising network resources for the duration of message transmission. The techniques of Pipelined Circuit Switching [107] and Scouting Routing [108] have been used in wormhole routing networks.

All preventative approaches enforced routing restrictions on paths prone to the formation of deadlock cycles, reducing the number of available paths and, therefore throughput. Bit errors on the data link could cause many of these techniques to fail.

3.9.2 Deadlock Avoidance

This approach reduced the probability of deadlock cycle formation by constraining routing algorithms in compliance with certain rules. There were many types of routing algorithms for regular topologies. However irregular topologies had a far wider range of possible network permutations, making network behaviour more unpredictable, limiting irregular topologies to derivatives of the same avoidance technique. Such algorithms were based around a 'tree structure' of valid routes, by which a message traversed the network with routing decisions made on the basis of whether or not it was moving towards or away from the root of the tree [109]. Avoidance strategies could result in non-minimal paths being taken, leading to further research into minimising path lengths whilst retaining functionality [110]. Virtual channels could also be used to prevent deadlock cycles as part of a partially adaptive deadlock evasion solution [111]. Each physical link could have several virtual channels associated with it in descending priority with routing decisions assigned to the next virtual channel in the sequence.

3.9.3 Deadlock Recovery

Recovery systems were shown to have certain performance advantages over some avoidance-based solutions [112] due to the low cost of recovery if executed infrequently. Care must be taken to ensure that the bandwidth lost in resolving deadlock was not greater than the bandwidth lost through the implementation of an avoidance algorithm. The probability of deadlock occurring in a fully adaptive network only becomes unacceptably large as the network approached saturation. This could be reduced by restricting traffic entering the network, reducing deadlock probability, relaxing routing restrictions, enhancing bandwidth and relying on high-confidence recovery based systems when necessary [113].

Most deadlock detection systems relied on time outs to indicate an incomplete data transfer in a specified period [114, 115]. This guaranteed detection, but introduced the need for a time-out value dependent on several variables. These included the topology, size and data throughput of the network, as well as the sizes of the packets that traversed the network. A generic time-out value resulted in false

detections if the time-out value was too low or loss of efficiency as the stalled network awaited attention if too high.

Three methods of deadlock recovery have been identified [116], all of which resulted in message removal in order to escape from cyclic dependency, but each differed with respect to what happened following removal from the deadlock cycle. The methods were:

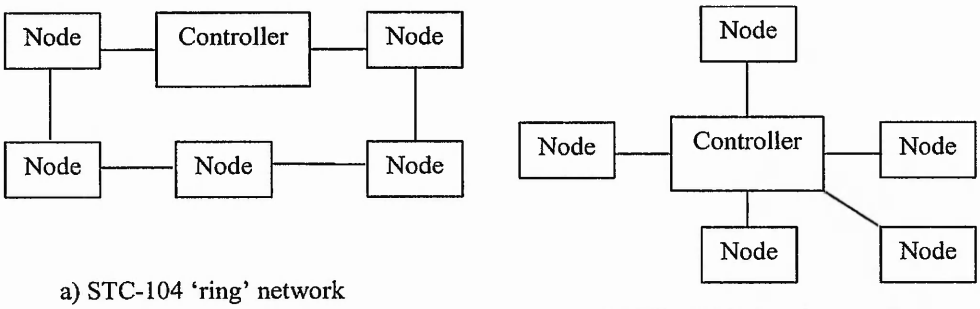
- Deflection – Message moved away from its destination.
- Progression – Message moved towards its destination.
- Regression – Message removed from network and was retransmitted from source.

In summary, the ICR-C416 router employed source routing, exposing it to the formation of cyclic dependencies, and thus required the use of deadlock avoidance strategies, at a cost of network performance. Avoidance mechanisms were the most prevalent in irregular wormhole routing networks, such as the ICR-C416, due to ease of implementation. Networks incorporating recovery [106, 116, 117] and preventative [118] strategies were relatively new. Deadlock was dependent on network topology and the routing algorithm used. Most irregular networks used algorithms that were derived from a single algorithm to take account of the myriad of possible network topologies.

3.10 Tolerance to Faults In Other Studied Systems

3.10.1 Fault Detection and Recovery in the STC-104 Based System

The STC-104 router, like the ICR-C416, utilised source routing and also required deadlock avoidance techniques to prevent the formation of cyclic dependencies. The router also possessed a control port, similar to that employed by the ICR-C416, but with the ability to daisy-chain multiple control ports together, to form a 'ring' network, whereas the ICR-C416 control port could be said to be a 'star' network. Both are shown in Figure 17.



a) STC-104 'ring' network

b) ICR-C416 'star' network

Figure 17: Ring and Star topologies as used to connect the control ports of the STC-104 and ICR-C416. Note; these are fixed, unlike the data connections.

The centralised control port protocol was different to the DS links protocol of the data links and possessed the ability to address each of the devices in the daisy-chain separately. This approach reduced overheads compared to the ICR-C416 approach but required extra wiring. The STC-104 had the ability to reset a link and remove any further tokens to be sent, following an error, if the 'localise error' configuration setting was set. Another configuration setting, 'discard if inactive' sought to delete packets destined for inactive outputs and ensured the network would not stall due to the inability of a message to progress through the network. The STC-104 documentation [76] implied that this function did not take into account any grouped outputs and as such, may have removed the routing availability provided by group adaptive routing. These two configuration settings were the only automatic fault detection and recovery mechanisms in the STC-104 router. All others required a command from the central control port to remove faulty packets.

Like the ICR-C416, the STC-104 network had no means of conveying information concerning the operational status of the network within the token format of the communications protocol. The only means of verifying that a token was correctly received was by receipt of the four-bit acknowledge token after the transmission of eight tokens. In its absence, the presence of a problem was recorded. As with the ICR-C416 network, acknowledgement transmission was delayed until sufficient buffering resources were available to accommodate the next flow group.

Unlike the ICR-C416 protocol, the DS links utilised by the STC-104 ensured that link disconnection could be detected quickly and easily as the protocol demanded either the data or strobe lines to switch every clock cycle.

3.10.2 Fault Detection and Recovery in the Myrinet Based System

The Myrinet network utilised the 9 bit parallel data links to convey control information across the network, employing dedicated control tokens to this end. A forward reset (FRES) token was designed to clear the resources it encountered as it progressed across the network, following detection of a fault. It operated at a lower level than the data sent across the link, allowing its use irrespective of the status of the link flow control mechanism. After sensing a FRES, a node cleared its buffers and transmitted it on to any links that had connections to other nodes that were occupied by that particular message. This had the ability to remove many messages from the network, which could be complex, regressive and highly resource intensive. The first generation Myrinet specification detailed several additional control tokens that were not implemented, possibly due to their complex operation. These tokens included: backward reset (BRES), over run alarm (ORUN), probe (PRB) and probe reply (REPL) [37, 80].

4 DESIGN DISCUSSION

4.1 Introduction

This chapter details alterations made to the protocol and the fault detection and recovery features of the PCI-OSLi and SARNIC designs at a conceptual level. The ideas detailed in chapter 3 are developed to describe the methods used to detect faults. The fault detection and recovery features of the NTR-FTM08 dictated the specification for the FT-PCI-OSLi and FT-SARNIC as the end nodes must interface to the routing elements in the FT-SARNet. Firstly, the design requirements of the FT-SARNIC and FT-PCI-OSLi interfaces were examined. Secondly, the decisions made involving certain network criteria are developed into the basis of a system with improved fault tolerance. Thirdly, the operation of these new features is detailed.

4.2 FT-SARNIC Network Interface

The purpose of the FT-SARNIC device was to facilitate data transfers between the communications network and the SA-110 CPU's SDRAM module. Data from other processors entered the FT-SARNode via the communications links of the FT-SARNIC. The data was stored in the SDRAM until the SA-110 accessed it for manipulation. This procedure of transferring data to the processor via the SDRAM allowed data to be made available for use by the processor prior to it being required. This reduced the possibility of the processor idling, whilst waiting for data held up in heavy traffic conditions.

The FT_SARNIC interface built on the SARNIC whose main achievement was the development of a single chip network interface, designed to relieve the processor of as much of the message transfer overheads as possible. The design consisted of three main parts: a memory bus interface controller, a processor interface controller and a communication link interface controller. The purpose of these was to interconnect the constituent parts of the SARNet processor node: the microprocessor, SDRAM and communications link respectively.

The salient features of the SARNIC [17, 32] were detailed in section 1.2 but the FT-SARNIC was required to possess the following enhancements over the SARNIC interface [119]:

- The performance of the FT-SARNIC had to equal, and preferably exceed that of the SARNIC in terms of data throughput and latency.
- The FT-SARNIC interface required an integrated, decentralised, automatic approach to fault tolerance that built on lessons from previous research. The centralised control port of the SARNIC design had to be removed, being replaced by a protocol that aided the systems fault tolerance transferred across the communications medium, whilst incurring minimal losses in data throughput. These aspects are discussed in greater detail in this chapter.
- The protocol utilised by the communication links of the FT-SARNIC was bound by the operating characteristics of that used by the NTR-FTM08 router.

4.3 FT-PCI-OSLi Network Interface

The purpose of the FT-PCI-OSLi interface was to form an efficient means of transferring data from the serial NTR-FTM08 router network to the 32-bit parallel format required by the memory of a PC.

Incoming data from the router network was to be converted from serial to 9-bit parallel format by the interface, whereupon control tokens (excluding message termination tokens) would be removed from the incoming data stream. The message would pass through a link interface buffer, whose capacity governed the flow control of the communications link. Header and terminator tokens would then be removed, leaving only data tokens, whose ID bits were stripped and the data bytes assembled into 32-bit data words. These will be loaded into a DMA buffer until the buffer was full or the message terminated. A DMA 'burst' transfer would be utilised to transfer the contents of the DMA buffer to the PCs memory.

The cycle stealing DMA transactions employed in the SARNIC consisted of an address transfer, followed by the transfer of a single data word. Burst mode transfer consisted of an address phase, giving the start address for the assigned block of memory to which the data was destined, followed by consecutive data transfers until the transaction was complete. The length of the block governed the amount of time the DMA controller had access to the memory at any one time.

The system requirements of the FT-PCI-OSLi were deemed to be similar in many ways to those of the PCI-OSLi interface. These included [42, 43]:

- The interface should be capable of constant data transmission onto the router network in order to saturate the serial communications links. The interface should always be ready to receive data from the router network in order to avoid suspension of data flow due to a receiver bottleneck. Avoiding suspension of data flow is vital to ensuring an efficient communications medium.
- The data throughput of the interface to the 32-bit PCI bus must be many times that of the serial communications links. Full-duplex bi-directional data flow across the communications link must be possible without monopolising the shared PCI bus. Whilst the serial communications links could support full-duplex bi-directional data transfer, incoming and outgoing data transactions to and from memory were multiplexed onto the bi-directional PCI bus, where data flow occurred in only one direction at a time.
- The hardware and software of the PCI interface that linked to the host system had to be versatile enough to be supported by many different platforms.
- Functions should be implemented on hardware if possible to reduce the load and complexity of the communications overhead on the software. Such hardware, as in the example of packet assembly and de-assembly should be efficient enough to prevent major performance bottlenecks occurring.
- The interface was to be integrated onto a single programmable logic device (PLD), in order to facilitate tight system integration and to meet the stringent PCI timing requirements for set-up and hold times. The interface was mounted on a PCB designed to slot directly into a vacant PCI slot with the track distances

between the edge connector and the PLD minimised for those signals with the most stringent timing requirements.

The FT-PCI-OSLi interface was deemed to require the following enhancements over the original design:

- The performance of the FT-PCI-OSLi must equal, and preferably exceed that of the PCI-OSLi in terms of data throughput and latency.
- The FT-PCI-OSLi interface was required to possess an integrated, decentralised, automatic approach to fault detection and recovery, building on lessons learnt from previous research. The removal of the centralised control port of the PCI-OSLi was necessary. An efficient communications protocol transferring control information across the communications medium was to be implemented.
- The protocol utilised by the communication links of the FT-PCI-OSLi was bound by the operating characteristics of that used by the NTR-FTM08 router.
- An important feature of a design committed to reducing inefficiency, should be the ability to accommodate message information for multiple messages. One of the major disadvantages of both the SARNIC and the PCI-OSLi interfaces was the requirement that an interrupt be called every time a packet or message arrived belonging to a different message than the previous communication. Software would then be required to perform a header check to ascertain to which message the arrival belonged.
- Several additional features to enable the user to devolve task execution powers to the interface.

4.3.1 PCI Bus Performance

In section 2.1, three parameters were identified that were used to assess the performance, in terms of efficiency of a communications system. These were bandwidth, latency and processor overhead. Bandwidth represented the amount of data that could be transferred across the medium, or its throughput. Latency represented the time taken to initiate a transfer, and reduced the throughput from the

theoretical maximum. Processor overhead measured the extent to which message handling impeded the normal operation of the processor.

As the FT-PCI-OSLi and its non-fault tolerant predecessor were both PCI agents operating in the same manner, they incurred similar overheads on the processor due to the operation of the PCI transaction procedure. The higher level software also affected the overhead, but the PCI-OSLi driver software was not updated for use with the FT-PCI-OSLi, making comparison of processor overhead academic.

The latency incurred in initiating a PCI transaction has four components. These are:

- The time elapsed between making a request for a PCI transaction and the granting of ownership of the bus.
- The time taken for the initiator to begin the address phase after being granted ownership of the PCI bus.
- The delay between the address phase and the first data transfer.
- The insertion of wait states when either of the communicating entities is not ready to transfer data.

These total latencies can be expressed as the time taken following assertion of the active low Grant signal (nGNT), to the assertion of the active low Initiator Ready (nIRDY). A PCI transfer consists of a burst of transfers to / from sequential memory locations following the initiation of a PCI transaction. The latency lowers the efficiency from its theoretical maximum as no data is transferred in this period. The efficiency Eff_{PCI} is given by:

$$Eff_{PCI} = \frac{\text{No PCI Transfer Cycles}}{\text{No PCI Transfer Cycles} + \text{Total PCI Latency}}$$

The efficiency of a PCI burst lowers the PCI bus throughput proportionately;

$$Throughput_{PCI} = Eff_{PCI} * \text{PCI Bandwidth}$$

Where PCI Bandwidth = 132MBytes/s for a 32-bit, 33MHz PCI bus

To maintain bandwidth efficiency, latency must be minimised, whilst the number of data transfers that follow the set-up latency should be maximised.

4.4 Areas of Improvement Following Analysis of Previous Research

The realisation of network-wide fault tolerance must not be at the expense of network performance. As was noted in chapter 3, conveying status information across the network, via the data channels, could be more efficient than having a dedicated control link, especially if the transmission of status information was infrequent. In order to maintain an efficient throughput of data, whilst incorporating system integrity information into the data link, communication and protocol overheads must be minimised wherever possible. Earlier research demonstrated that bi-directional data transfers were a source of lost bandwidth due to the credit based flow control mechanism. A more efficient protocol could be advantageous, but a change in the physical link was also considered as a means of increasing data throughput.

4.4.1 Communications Links

The option of communicating tokens in parallel, as in the Myrinet network was rejected. This was due to the expense of multi-core cabling required to minimise skew and the high pin counts incurred, especially in routers. If the total communications link throughput was too high with respect to the PCI throughput, the host system interface would become the performance bottleneck of the system, denying other users access to the bus. The serial data / data strobe (DS) technique used in the IEEE-1355 standard incurred skew problems and doubled the wiring requirements whilst encoding a clock signal within the bit-stream. An encoded clock signal was not deemed necessary as the over-sampling technique employed by the current technologies performed adequately, despite the need for a clock speed 50% faster than the data rate.

The NTR-M04 router utilised an optional second pair of bi-directional serial DS links in an attempt to double link bandwidth. Such an approach differed from previous parallel link implementations, as bits were not sent in parallel. Separate serial data streams permitted two links to share in the burden of transferring a message by alternating where the link tokens were transmitted down. In normal operation, there was a skew of at least four bit periods between tokens arriving at the top and bottom links. The NTR-M04 incurred higher than expected resource usage as each bi-directional link required four transmitter circuits, four receiver circuits and additional control logic, whilst increasing the bandwidth by only 1.85 times [23].

The lower bandwidth offered by a single pair of serial transmission links was acceptable considering the benefits of increased transmission distances and reduced costs. The over-sampling technique was already proven and was capable of operating at speeds up to 44Mbits/s over 100m of Category 5 unshielded twisted pair cable [75].

The DS protocol had one advantage over the OS protocol, due to improvements in the token format, which gained a theoretical bandwidth improvement of up to 14.7% over the ICR-C416 protocol. This improvement was achieved by reducing the data token to 10 bits and implementing a 4-bit control token. Modifying the protocol to send the minimum amount of control tokens per data token would increase bandwidth utilisation further still.

4.4.2 Flow Control

As stated in chapter 3, the credit based flow control protocol employed by ICR-C416 based networks reduced data throughput. This was due to waits for acknowledgement and the appending of each 11-bit data token with a 2-bit acknowledge token for bi-directional communications. In theory this made the data content only 61.5% per token, although in practice it could fall as low as 47.1% if 17 bits were required to transmit a byte of data [5]. This was due to the delay incurred in interleaving data and acknowledge tokens for bi-directional communications as demonstrated in section 3.4.1.

The DS links protocol used by the STC-104 and the NTR-M04 increased the flow group (the number of tokens which can be sent before acknowledgement was required) from one token to eight. This increased the ratio of data bits per acknowledge token, limiting bandwidth losses on bi-directional transfers caused by the injection of acknowledgement tokens into the data stream, but incurred proportionately larger buffering increases. A single path across the STC-104, with its 8 token flow group, required 70 tokens of buffering [120], whereas the corresponding path across the ICR-C416 required only enough buffering to store three data tokens [121]. As DS link flow control tokens were only four bits long, theoretically up to 76.2% of the link bandwidth could be devoted to data transfer.

The NTR-M04 reduced the buffering requirements for an 8 token flow group from 70 to 22 tokens as buffering was minimised due to limited buffering resources in the target technology at the time, limiting the number of ports on the device. The STC-104 utilised ASIC technology and as such was not affected by this problem. NTR-M04 data and flow control token were 11 bits long, theoretically allowing bandwidth utilisation of up to 64.6% for data.

The permission based flow control employed by Myrinet required a buffering capacity of 59 tokens per receiver, as inferred by the first generation specification [122], a figure dependent on link speed and transmission length. Permission based flow control was dependent on the ability of the receiving node to process the contents of the receiver buffer. In an ideal case, where data was processed as soon as it entered the buffer, bandwidth loss was zero. Clause 3 of the flow control dictates the length of time data flow was suspended for when the Stop level was reached. The first generation Myrinet figure of 16 tokens gave a worst case bandwidth loss of 6.25% due to flow control but an average of 8% improvement over the NTR-M04 was noted in behavioural tests [123].

The flow control and fault handling features of the Reliable Router were incorporated into the 75-bit data unit, split into four frames. Each receiver possessed buffering resources capable of holding 16 data units for each of the five virtual channels per physical link. Data was held in the buffer until a successful transfer had

been confirmed, allowing retransmission via the use of the Unique Token Protocol (UTP) discussed in section 2.3.4.

4.4.2.1 Permission Based Flow Control Threshold Level Analysis

As mentioned in section 3.4.2, the efficacy of Permission Based Flow Control (PBFC) depended on the levels of the Almost Full (AF) and Almost Empty (AE) thresholds, as these respectively governed the suspension and resumption of data flow. There were conditions for setting these levels to prevent buffer overflow, buffer underrun and excessive loss of bandwidth due to flow control token transmission. These were:

1. The receiver must possess sufficient buffering to store any incoming data tokens that arrive following the assertion of a Stop and this request taking effect.
2. The receiver must have sufficient buffering to pass data tokens continuously from the link buffer to the DMA message controller, to prevent the receiver buffer emptying before the arrival of the first data token across the link following resumption in data flow.
3. The distance between the AF and AE levels must be sufficient to ensure that the link did not devote unnecessary bandwidth to the transmission of flow control tokens.

The flow control mechanism must adhere to Rule 1, as it was unacceptable to lose data due to buffer overrun. Rules 2 and 3 affect the efficiency, with the former determining whether or not the message handling elements of the design are idle or not due to the link status. Flow control tokens must have a higher priority than data tokens in order to allow the system to operate. If the Stop and Go levels were too close together, the link wastes bandwidth transmitting unnecessary flow control tokens. In the sample circuit of Figure 18, whose transmission time was nominally set to one token, Rule 1 required node B to store four data tokens once the AF level triggered transmission of a Stop token. If node B had less than four tokens in its link interface buffer when data flow resumed, the depacketiser at the buffers output would idle, wasting link bandwidth whilst waiting for the arrival of data token D_5. Rule 3

dictated how soon after the transmission of the Stop token the data transmission could resume, and depended on the rate at which the depacketiser could empty the receiver link interface buffer. The first generation Myrinet specification [80] required 23 tokens for clauses 1 and 2, and 16 tokens for clause 3, for a maximum cable length of 25 metres at 80MBytes/s data rate.

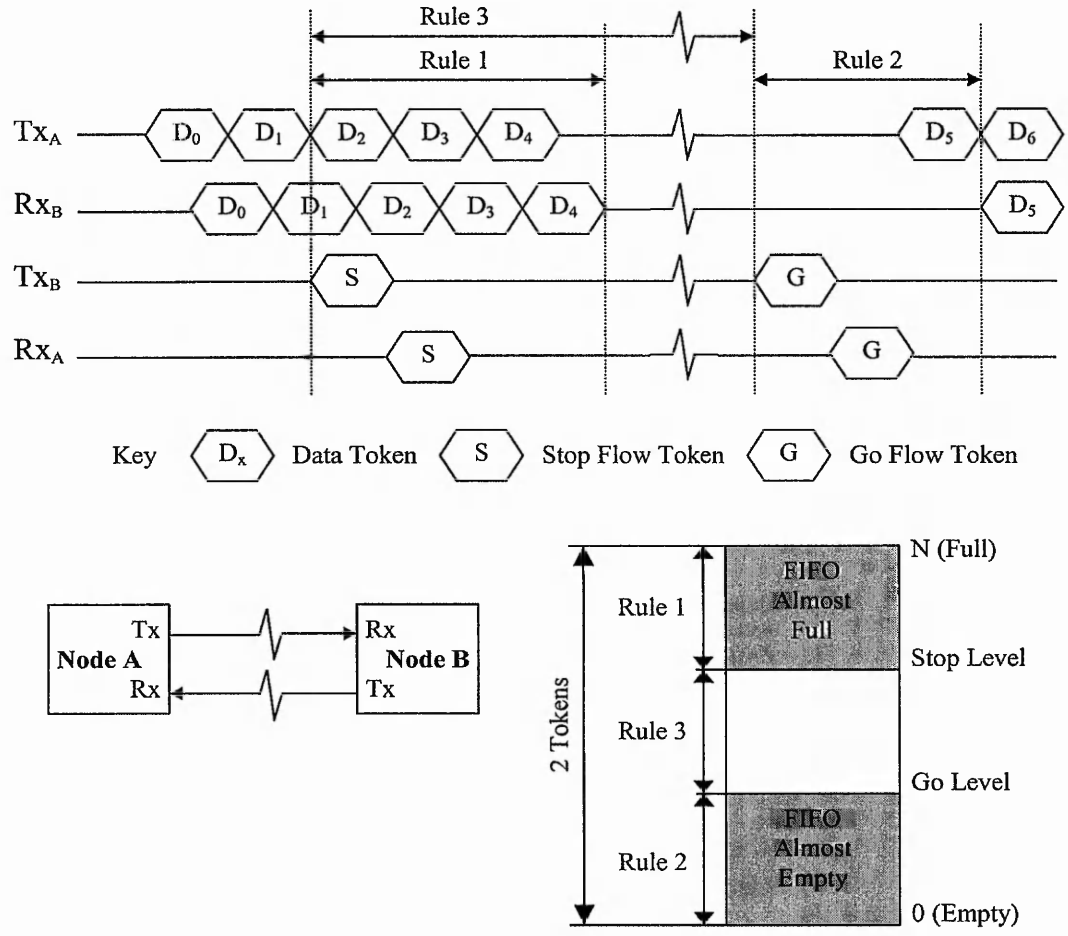


Figure 18: Representation of Permission Based Flow Control and how its rules translate into practical buffer implementation for correct operation

4.4.2.2 Determination of Stop and Go Flow Control Levels

The levels at which the AF and AE levels were set depend on the amount of data that could cross the link from the transmitter to the receiver and back, plus the delay

incurred. In the case of rule 1, this was due to the possibility of back-to-back data tokens doubling the amount of time taken between generation of a Stop token and the token taking effect. Rule 2 stated that it would take a time equivalent to the transmission of two tokens following generation of a Go token before arrival of the first data token. This was reflected in the 'total extra delay' parameter where the transmission times for the flow control token and data token were added to the equation.

$$Buff_{min} = \frac{SigR \cdot \left(\left[\frac{D}{S} \right] \cdot 2 + Dly \right)}{Bpu}$$

Permission Based Flow Control AF / AE Levels Equation

SigR (Signalling Rate) = 50Mbits/s

D (Distance) = 110 metres – These are the two worst case link bandwidth and distance values used for calculations in a 30Mbits/s 100 metres communications link.

S (Propagation Speed) = 2×10^8 metres/second [124].

Bpu (Bits per unit) = 11 (Start, ID, Data Byte, Stop bits).

Dly (Total Extra Delay) = Logic Delay + Flow Control Transmission Time + Data Token Transmission Time

The interface logic delay depended on the amount of time taken to generate the control signal and the time for it to take effect, where both were worst case values. As flow control tokens possessed a higher priority than data tokens, the worst case transmission delay was one whole data token. Calculations for the NTR-FTM08 determined minimum rule 1 and rule 2 values of 6 buffer locations and on the recommendation of Myrinet, rule 3 requires a minimum of 12 buffer locations separating the AF and AE thresholds [125].

4.4.2.3 Flow Control Differential Analysis

The permission based flow control experiments conducted during the development of the NTR-FTM08 included post-synthesis simulations to determine the effect that flow control variations had on multi-router network performance [124]. The two network topologies studied were a four router 2D mesh and an eight router 2D torus network [126]. The receiver buffers had a capacity of 48 tokens with the Stop threshold held at 40 tokens for all tests. The Go threshold was set at 32, 24 and 8 tokens giving differentials between the Stop and Go levels of 8, 16 and 32 tokens respectively. The tests concluded that when offered vs accepted data load graphs were plotted [124], the maximum variance between the different thresholds on the same network was approximately 1%. The difference between the mesh and torus networks was approximately 6%. The threshold differential had much less effects on network performance than network topology, although the tests were not comprehensive.

When accepted data loads were plotted against average normalised packet latencies, more conclusive trends were visible. It was shown that, for a network approaching saturation, a differential of 16 tokens provided the best performance characteristics. The results demonstrated a lower latency for the same data load and a higher saturation level than differentials of 8 and 32 tokens respectively. These results indicated that the optimum Stop threshold level might not be the same as the Stop threshold level determined using equation 1 in section 4.4.2.2. The NTR-FTM08 routers 32 token receiver buffers Almost Full and Almost Empty threshold levels were set to 20 and 10 respectively. Following the above conclusions, the FT-PCI-OSLi and FT-SARNIC interfaces utilised 32 token buffers with AF and AE levels of 24 and 8 respectively, giving a differential of 16.

4.4.3 Control And Message Information

In addition to the transport of data and flow control information across the communications channels, information relating to the functional status of the network must also be transferred to give increased fault tolerance. The centralised fault

monitoring and intervention features offered by the ICR-C416 and STC-104 lacked scalability and thus increased intervention times. If a central monitoring point failed, the entire network could be left with no tolerance to faults at all (referred to as a 'single point failure'). A distributed fault tolerance mechanism allowed scalability and forced the hardware implementation of features to detect and recover from faults. This was because fewer functions could be performed at higher (software) levels without replicating functionality and reducing efficiency. An autonomous low-level (hardware) mechanism that improved tolerance to faults was highly desirable. Distributed fault tolerance provided a generic solution whereas centralised fault handling systems tend to be tailored to the requirements of a particular networks and were therefore less flexible and portable between applications.

The dedicated control link employed by the ICR-C416 and STC-104 networks was an expensive resource, in terms of both wiring and logic usage when compared to the functionality provided. The Myrinet, Reliable Router, NTR-M04 and NTR-FTM08 protocols used the 9^{th} 'type' bit set to zero to convey control information allowing up to 256 separate control tokens. In comparison, the DS links four-bit long control token was fixed, preventing further expansion.

Two disadvantages of the control tokens sharing the data link were the loss of bandwidth due to the transmission of excess control tokens and the difficulty in conveying control information across a failed link. Minimising the frequency of control token transmission could reduce the bandwidth loss of the former. The problem of link failure could be solved by way of an effective link blockage detection and reset mechanism, discussed in section 4.4.5. In this approach the reliability of the data link was carried over to the control link.

The complex Unique Token Protocol (UTP) employed by the Reliable Router utilised a software based message reconstruction mechanism and required the transmission of many other non-data bits in each data unit. In spite of this, the network provided a good example of distributed fault tolerance, where the responsibility for ensuring link integrity was devolved to the nodes at either end of the link. Software overheads were spread across the network with short and constant

intervention times, providing a scalable solution. These advantages made the Reliable Routers fault handling strategy, rather than its protocol worth investigating.

4.4.4 Faulty Packet Removal

The ICR-C416 network possessed no means of removing disconnected messages from the network. The failure to include any tolerance to faults in the token layer of the link meant that, in the case of message truncation demonstrated in section 3.6.2 the receiving node stalled. The non-zero value in the receiver's packet length counter indicated that more data was expected. A count of zero denoted the end of the packet. The error prevented further data from being received and the occupation of the physical channel by that particular message meant that until the resource was relinquished, by way of an operator initiated reset, other messages could not use the resource and the system remained stalled until it was reset. Such a system was deemed acceptable at the time given the extremely low failure rate but could be considered unacceptable in safety critical systems where it is preferable to have an automatic fault detection and recovery system.

The STC-104 possessed such autonomy via the configurable 'localise error' and 'discard if inactive' settings which allowed packet truncation and deletion respectively, at the expense of any group adaptive routing configurations. These features were the minimum for an autonomous fault detection and recovery strategy but were implemented, along with others including group adaptive routing, in the NTR-FTM08. The forward reset token (FRES) used by Myrinet and the NTR-M04 freed up resources for the next message initiated by a time-out through lack of link activity. The FRES had the capability to remove many messages from the network but retransmission was impractical, as the higher levels of the protocol had no knowledge of what was removed.

After the truncation of a packet, the receiving end node, whether the FT-SARNIC or the FT-PCI-OSLi, needed to be aware that a fault had occurred and that the message contents should be treated with caution. The Bad End of Packet (BEOP) token, similar to the Exceptional End of Packet (EOP2 or EOPE) token defined in

IEEE Std. 1355-1995, was appended to the data destined for the end node. The remainder was flushed. Detection of this packet called an interrupt and higher software levels were required to decide whether or not to retain the truncated message.

4.4.5 Link Initialisation Procedure

There was a requirement to define a start-up procedure to ensure that both sides of the communication link were ready before beginning data transfer. Defining which tokens were valid for each state allowed the state machine to presume that receipt of any other tokens constituted an error, returning the state machine to the 'reset' state. Figures 19 and 20 demonstrate the link initialisation procedure utilised by the FT-PCI-OSLi and FT-SARNIC devices. After reset, a short time-out elapsed before transmission of connection request (CONREQ) token moved the state machine into the 'asleep' state. A node receiving such a token also moved into the 'Asleep' state before returning a 'CONREQ' token, indicating its readiness, and moving the state machine into the transient 'Waking' state. On completion of the handshake between the two nodes, the system moved to the 'Awake' state, the only one where data tokens could legally be transferred. The handshake involved both nodes exchanging 'Start', or 'Xon' tokens demonstrating that they were ready for receipt of data and their buffers had sufficient capacity to accommodate the incoming tokens. Failure to complete the handshake within a set time restarted the procedure.

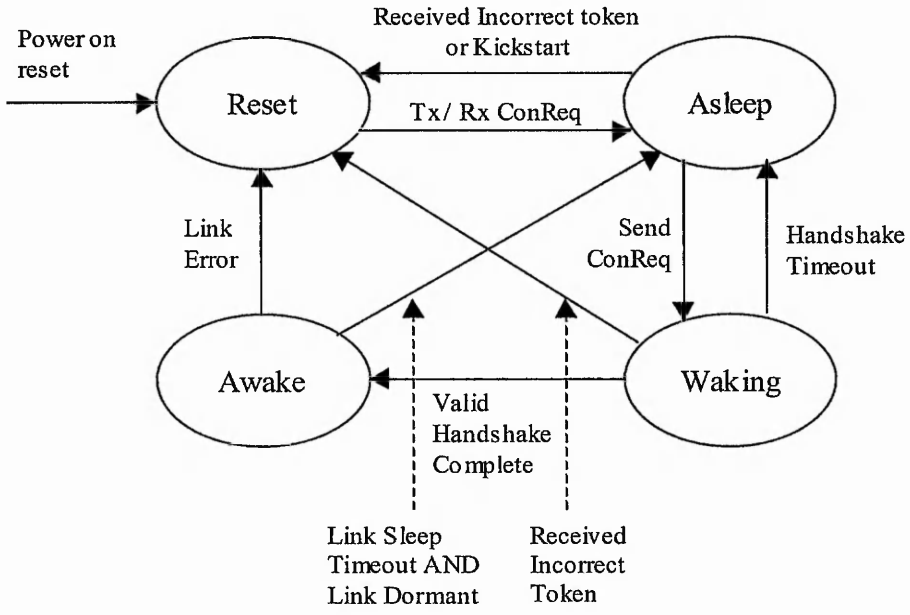


Figure 19: Link Status State Machine Diagram

Receipt of a 'CONREQ' token whilst 'Awake' informed a node that an error had been detected by the other end of the link, returning the state machine to the 'Reset' state. The procedure followed either of the two available paths shown in Figure 20 dependent on whether the node received or sent a connection request first. After a reset, a short timeout elapsed before a node transmitted this token. As two nodes were not reset at exactly the same time, one node would transmit a connection request token before the other one.

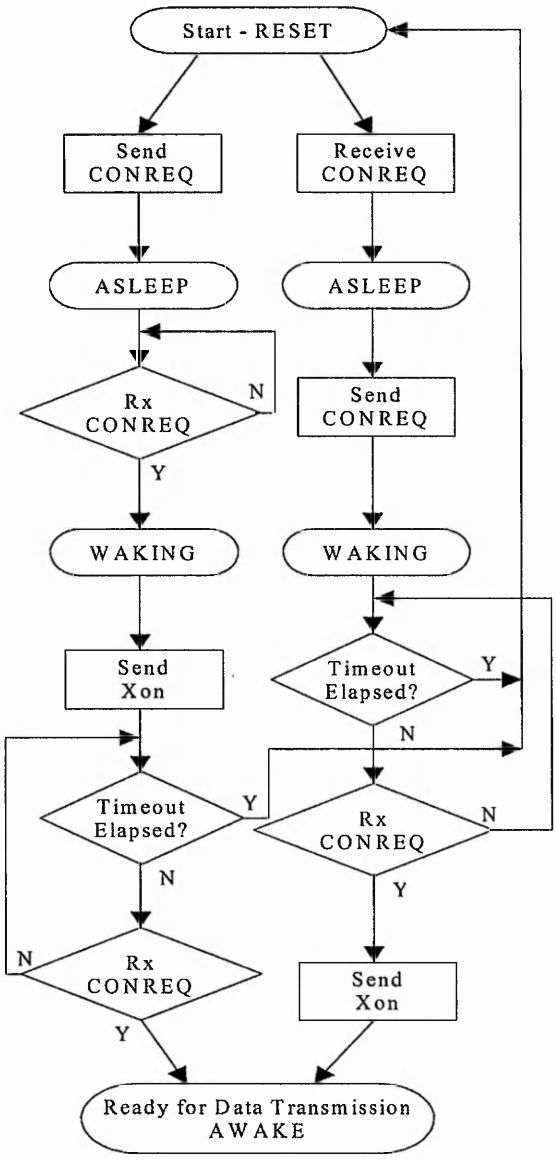


Figure 20: Link Initialisation Flow Diagram

4.4.6 Link Dormancy

Link dormancy was a configurable feature of the NTR-FTM08 protocol that permitted a link to fall asleep after a pre-defined period of link inactivity. It allowed the state machine in section 4.4.5 that determined link status to make a possible state move from the 'Awake' state directly to the 'Asleep' state. The NTR-FTM08 protocol required the transmission of flow control tokens approximately every 15 tokens time, following assertion of 'Heartbeat'. This interval might be too frequent for some

applications and undesirable for others. If both ends of the link were configured as dormant, no flow control tokens were transmitted in the absence of data and the links remained silent. Once asleep, in the event of one of the nodes wishing to transmit a message, writing to the transmitter message length register triggered the kickstart process. Kickstart returned the link state machine to the reset state, from which the initialisation procedure commenced. Link dormancy was asserted in the FT-PCI-OSLi module by setting bit 19 in the Command register (register offset $0F_H$) (see Appendix C for more information).

4.4.7 Virtual Channels

In a multiprocessor network employing routers, different messages were multiplexed onto the same physical channel as shown in Figure 21. In Figure 22, messages are divided into packets in order to avoid domination of the physical channel by any one message.

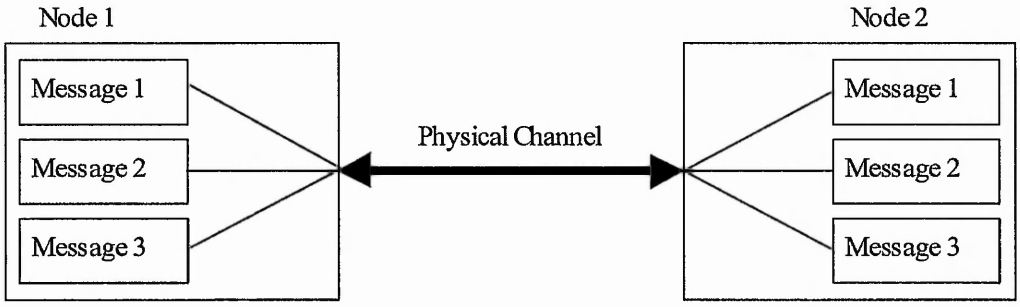


Figure 21: Virtual channels showing multiple messages traversing the same physical link

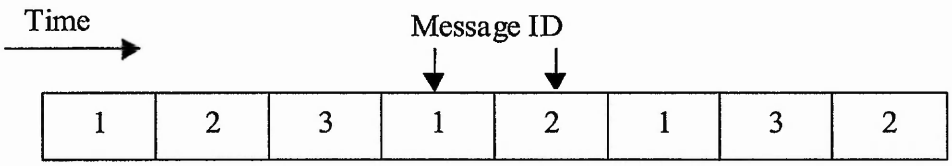


Figure 22: Messages arriving at the receiving node showing multiplexed packets

Each message had a unique header to aid identification. Messages were depacketised and transferred to memory at the receiving node. The processor fetched the data from memory for manipulation when required, as shown in Figure 23. The message length and address in memory of the data were specified by the software programmer and the operating system respectively.

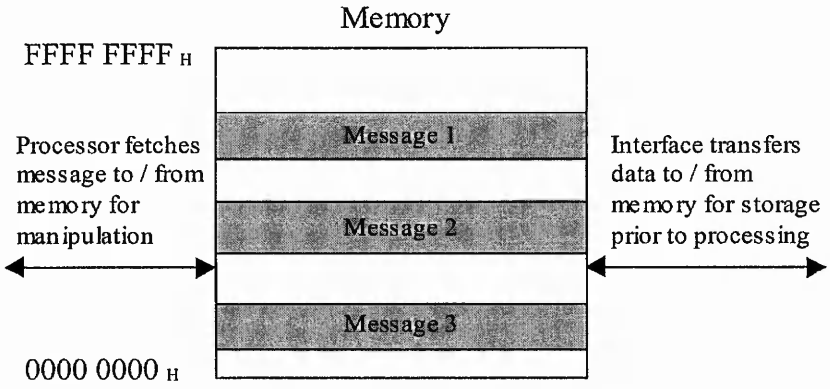


Figure 23: Messages stored in memory 'pots' prior to processing

On receiving notification of a message arrival, the communications interface must first match the received message header against all expected headers, as the receiver does not know in what order messages might arrive. Following a header match, the corresponding address and length must be obtained to identify where the message was to be transferred to and when transfer would be complete. The PCI-OSLi interface stored such information in software, requiring incoming headers to trigger an interrupt, requiring the processor to intervene and search for a matching message header. This approach was expensive in terms of time and resource utilisation as the processor must suspend activities and devote time to searching for a matching header. If an unexpected header arrived, the user had to flush the message in the PCI-OSLi interface before another message could occupy the interfaces' resources again. This allowed messages arriving at the wrong node to be removed from the network but did not permit the user to process the message.

The PCI-OSLi generated an interrupt every time a header arrived that was not identical to the header of the previous message, as only one message ID was stored at any one time. Messages split into packets possessed identical message headers so the

consecutive arrival of packets from the same message did not trigger a new message search. Incoming packets whose ownership alternated between two or more messages, such as those shown in Figure 22 required new message ID searches. The SARNIC receiver had two DMA channels and could thus handle two different messages without requiring an interrupt to be called when a packet from either message arrived. Arrival of a third message header initiated an interrupt and required the removal of one of the existing message ID's from the message allocator. Message removal was performed in software with user assistance required to determine which packet was removed. Message header comparisons were performed in software, necessitating an interrupt so the only advantage gained was when multiplexing two messages onto one link. The FT-SARNIC had the same message handling features, but improved on the PCI-OSLi interface by possessing a second message channel.

4.4.8 Header Storage – CAM

The PCI-OSLi design required an interrupt to be called following arrival of an incoming message to the interface. Servicing the interrupt required the PC to suspend its activities, validate the authenticity of the message ID and supply associated message information. Devolving the message ID checking functionality to the communications network interface permits its execution without involving the processor, hence increasing efficiency. Such an operation required the message information, (message ID, address and length) to be pre-loaded into the FT-PCI-OSLi in anticipation of the arrival of that particular message. This approach suffered from the two main drawbacks of the system used in the PCI-OSLi but both were easily resolved. These were:

- Many messages might be expected at the receiver, yet their order of arrival was unknown. If the message information for a different message resided in the FT-PCI-OSLi, the message IDs would not match, halting data transfer to memory.
- Incoming packets that alternated between two or more messages required the same message information to be reloaded several times.

To avoid these situations, the message information storage facility must possess these respective characteristics:

- It must possess the ability to store multiple message IDs.
- The option of the message IDs being reusable or one-time active must be available to allow messages occupying multiple packets.

In addition, it would be beneficial to the message ID validation procedure if the following features were available:

- The header validation process, which starts on header reception and concludes with the DMA channel enabling, must be performed as quickly as possible.
- The ability must exist for the user to remove or overwrite message IDs if they were no longer needed.
- Functionality must exist to allow the user to 'probe' or view the contents of the storage facility.
- The user should retain the right to pass or flush, either manually or automatically, messages whose headers do not match.
- Possibility of clearing the message information storage facility.
- The user must be allowed to pre-set message address and length, enabling message information to be loaded into the FT-PCI-OSLi with a minimum of write accesses.

Such a system effectively created X virtual channels, where X was the number of message IDs that could be stored in the message storage facility. This provided rapid header verification whilst minimising the need to call a processor interrupt. Network efficiency could be maximised by devolving as much message handling responsibility as possible to the FT-PCI-OSLi.

4.4.9 Message Storage

This section examines what occurred once the authenticity of an incoming message had been verified, and its contents depacketised ready for transfer to the host systems memory. The host system must exercise caution over where the FT-PCI-OSLi interface will transfer data, to ensure memory contents were not overwritten. Several memory 'pots' were assigned, into which messages could be stored before being accessed by the host system, manipulated and stored elsewhere. The concept was to have multiple designated 'pots' for message storage prior to receiving attention from the processor. More than one was necessary as the processor could manipulate information from messages in a different order to that which they were transferred to memory. A message could not be overwritten if it was yet to receive the processor's attention. The message 'pots' permitted the message information (message ID, length and start address) to be active and readable for the entire duration that the message was assigned to that 'pot'. Other messages were prevented from utilising that 'pot' until the processor had handled the message.

To aid message storage, messages were grouped into three classes, determined by length and application. The header storage facility, located in the CAM, was effectively split into three sections, and possessed the ability to store '*X*' class 1 messages, '*Y*' class 2 messages and '*Z*' class 3 messages. Class 1 had 10 locations, class 2 had 4 locations and class 3 had 2 locations. Expansion of class sizes and / or the CAM size was relatively simple, requiring logic duplication. The class lengths were user configurable by addressing registers in the interface and were the *maximum* permitted, not the only message lengths allowed. Class 3 was intended to be a 'catch-all' class with no specified maximum length, and any maximum message length up to the current 1Mbyte maximum for the FT-PCI-OSLi. Class 3 messages were designed for one time use only and as such were removed from the CAM following the match.

Grouping messages into classes according to lengths and assigning each group a designated message storage area in memory reduced message information storage space. Each message address was 30 bits long, due to word accesses making the least significant two bits redundant. Each messages length required 18 bits with the least significant two bits ignored for the same reason. Header storage required 8 bits for

each header byte that was to be sent. For a single header byte, a minimum of 54 bits worth of information, whether stored in memory or registers, was needed for each message. This figure increased proportionately, so to store 16 messages of information required a total of 864 bits. As all messages in a particular class possessed the same maximum message length, only three length registers were required, reducing logic requirements significantly with no real loss in flexibility.

4.5 Digital Systems Implementation Issues

An overview of some of the problems encountered in the implementation of prototype digital designs and some potential solutions is offered in this section. It is not intended to be a definitive discussion on this open-ended subject and focuses on solutions utilised in the development of the FT-SARNIC and FT-PCI-OSLi interfaces.

An important issue in digital circuit design is currently that of the length of the design cycle. Many designs build on previous designs in some way, retaining common features or components. Utilising these reusable aspects of the design, called Intellectual Property (IP), can significantly reduce the design cycle and increase cost effectiveness. The time taken to verify the design can also be reduced as the IP aspects of the design have been proven to function correctly although the design effort required in integrating IP into the design can be significant [127].

Improvements in silicon fabrication techniques have led to higher gate counts being implemented on ever decreasing areas of silicon. Combined with advances in Surface Mount Technology (SMT) [128], devices are decreasing in size, putting a strain on the device I/O. Larger designs require larger devices but can also require increases in the I/O requirements due to the increase in microprocessor bus width from 8 bits (in the 1980's) [129], to 64 bits in contemporary state of the art processors [130]. This creates a pin out bottleneck, as the Printed Circuit Board (PCB) track layout around the device becomes more complex. Serial communications links allow a high number of logical connections to be realised whilst maintaining a relatively low number of physical wire connections. Serial communications technologies have advanced significantly, resulting in much higher speeds than were previously possible

when parallel connections replaced serial. Modern serial based communications systems include IEEE Std. 1394 (FIREWIRE) [90, 91] and Universal Serial Bus (USB) [93, 94]. These communicate via media shared between users, similar to a bus. Performance is guaranteed by limiting the number of connections and transmission distance.

Until relatively recently, the only means of implementing large-scale digital designs involved utilising Application Specific Integrated Circuit (ASIC) technology [131]. ASICs are full custom devices requiring massive investment in terms of time (for laying the design out on silicon) and money (cost of die production). For very high volume production, per unit costs become very low, but for prototyping and small production runs, these initial overheads are prohibitively high. At that time the design of highly complex circuits could only be contemplated on ASICs, making the development of high-speed communications switches an option available to few. Some systems attempted to implement a processor node on a single integrated circuit, [5] permitting a complete high-speed solution but offering design flexibility at a very high cost.

Programmable Logic Devices (PLDs) are arrays of programmable logic that allow the implementation of custom designs tailored specifically to the application. Early PLDs contained few programmable elements and could only be used to implement glue logic. ASICs and PLDs have both benefited from advances in silicon technology such as improved fabrication methods and the reduction of minimum feature size to enable higher gate counts [132]. Higher speed and noise immunity was achieved via the use of new materials. PLDs have also benefited from improved internal architectures. The ability to implement hardware functionality using Look Up Tables (LUTs) [133] and on-chip memory resulted in devices such as the Altera EP20K1500E, capable of implementing 1.5 million programmable gates [134].

The first generation PLDs were one time programmable with the logic functions programmed by 'blowing fuses' within the device [135]. Most current PLDs utilise Electrically Erasable Programmable Read Only Memory (EEPROMs) [136] to program Static Random Access Memory (SRAM) devices [134]. The latter lose their device program when power is removed and so use the former to reload the program

on power up. The ability to reprogram the EEPROM and thus SRAM devices in system made these PLDs particularly suited to prototype and development work where modifications to the design were required.

PLDs have started to eclipse ASICs in terms of the increased flexibility offered, technological and architectural advances, ease of modification and even increased performance in certain functions [137]. Vendors have developed PLD logic cores with a fixed design feature implemented alongside a large amount of programmable logic for the customer to implement an on-chip interface. The PLD is sold or licensed to the customer as IP. IP cores have been developed by Altera to cover application areas such as signal processing, bus interfaces, communications protocols and controllers. It is the introduction of embedded processor cores developed by Altera [69] and Xilinx, that are more relevant to the subject area. Such features allow the opportunity to realise a System-on-a-Chip (SOC) or System-on-a-Programmable-Chip (SOPC) solution [138, 139].

IP cores are usually 'soft' cores, being software based with the function implemented by the compiler during synthesis. The customer purchases source code with which to realise these functions. The soft core will have certain design constraints and parameters defined by the vendor. Information on how the design would be implemented in the PLD was also included in the source code although the synthesis tool dealt with layout. As the vendor encrypted the source code, the core features were fixed and could not be modified by the user to suit the application, allowing for a generic, but inflexible implementation of the core. Some core vendors permitted users to alter the core features by charging additional license fees in return for extra access rights. Another aspect of IP cores was the need to spend a significant amount of time and effort evaluating the core to examine its suitability, although most vendors provide IP cores for evaluation before licensing them.

An alternative approach is to provide 'hard' cores. These were referred to as Embedded Standard Products (EPSs) [140], by one vendor, QuickLogic. Hardware functions and their associated glue logic are replaced with hardwired IP cores and programmable logic on a single programmable device. This resolved layout problems,

which were normally dealt with by the synthesis tool in soft cores. The lack of flexibility with regard to modifying the core still remained.

ASICs still dominate the high volume, high performance end of the IC market, despite a recent shift away from their use in an attempt to cut costs [141]. PLDs were better suited for low volume production and for implementing projects requiring rapid, easily verifiable solutions as the development cycle times were much faster in comparison to ASICs.

4.6 Synthesis

As discussed in the previous section, digital designs could be implemented on ASICs, Gate Arrays or PLDs. The former two technologies incurred expensive initial development costs. This made them highly unsuitable for development purposes but they became more cost effective with volume production. For these, and additional advantages outlined below, PLD technology was selected. These advantages included:

- Flexible designs due to reprogrammable devices.
- Low initial costs made PLDs suitable for prototyping.
- Shorter development cycles relative to ASICs as layout already existed and the compilation software performed logic connections and optimisation tasks.
- Modular designs could be reused as IP.
- PLD designs were portable to ASICs but not vice-versa.

The FT-PCI-OSLi interface was implemented on an Altera EP20K200EQC240-1 programmable logic device. Previous design work performed by the research group had targeted Altera's Flex 10K PLD family. The PCI-OSLi interface utilised 83% of available logic resources and 70% of available memory on a Flex 10K50S device [43], leaving little room for significant alterations to the design. The decision to advance to the newer Apex 20K technology was prompted by several requirements. These included: increased logic resources, increased performance, improved software support and the rapidly advanced PLD market, which resulted in the rapid obsolescence and discontinuation of device families. Like the Flex 10K devices, the

Apex 20K devices are SRAM based, requiring programming on power-up. This suits prototyping as design modifications could be implemented quickly and with no need for component replacement. Timing-based simulation could effectively be performed by way of hardware verification, omitting a stage of the design cycle and reducing design time. The following section discusses how the internal architecture of the Apex 20KE device family is beneficial in achieving maximum performance for a generic, non-optimised design solution.

4.6.1 Target Device Characteristics

The Apex 20KE [134] device family had an internal operating voltage (VccInt) of 1.8V to attain the low power consumption often required in embedded systems and a PCI compatible external I/O voltage (VccIo) of 3.3V. Fast, bi-directional, tri-state I/O pins made it suitable for a PCI interface. The 168 user pins available on the 240 pin device were more than adequate for the number of pins required by the interface and left many more that could be used as debug pins. The 20K200E device had 526,000 system gates that typically translated to 200,000 user gates. These gates form 8,320 Logic Elements (LEs) and 52 Embedded System Blocks (ESBs) comprising 104kbits of programmable memory. If more resources were required, the device could be substituted for the 20K300E PLD as they shared identical packaging and dedicated (non-user configurable) pins. The 20K300E device possessed 50% more LEs than the 20K200E device and a similar increase in ESBs. The device required two Electrically Erasable Programmable ROM devices (EEPROM) to configure the PLD after power-up. The EEPROMs were programmed via a 10-pin JTAG header from the programming PC's parallel port and retained their program until overwritten.

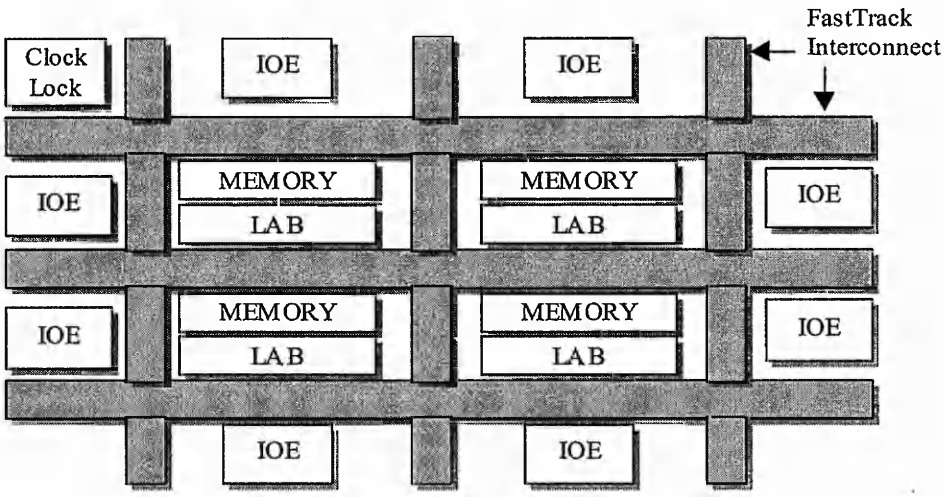


Figure 24: Apex 20KE Device Architcture

4.6.2 Apex 20KE Architecture

The internal architecture of the Apex 20KE devices distributes the design features in blocks throughout the device to reduce delays between logic and memory, as shown in Figure 24. Programmable logic in the PLD has one of four functions:

- **ClockLock** – Clock management circuit. The fastest available (–1) speed grade [134] can support both 32 and 64bit and 33 or 66 MHz PCI timing requirements. Multiple clocks could be used on the same device with clock skew minimised in even the largest of designs.

- **IOE** – I/O Elements supporting a wide range of I/O standards, including PCI. Dedicated 'Fast I/P' pins used dedicated device-wide routing channels to distribute signals across large designs, minimising set-up times to meet the stringent PCI timing requirements on bi-directional I/O.

- **LAB** – Logic array blocks implement registered and combinatorial logic functions based around small, four variable look-up tables. These are discussed in greater detail below.

- **Memory** – Embedded memory within the device allows for reduced area and increased performance. Cascadable memory blocks permitted the implementation of variable areas of memory. Memory was available as dual port RAM, FIFO buffers, RAM, ROM and CAM. Separate ESB blocks allowed the creation of many independent memories in terms of both size and function. For example, the FT-PCI-OSLi device contained two 64 deep by 32-bit wide FIFO buffers, two 32 deep by 9-bit wide buffers and one 16 deep by 8-bit wide CAM.

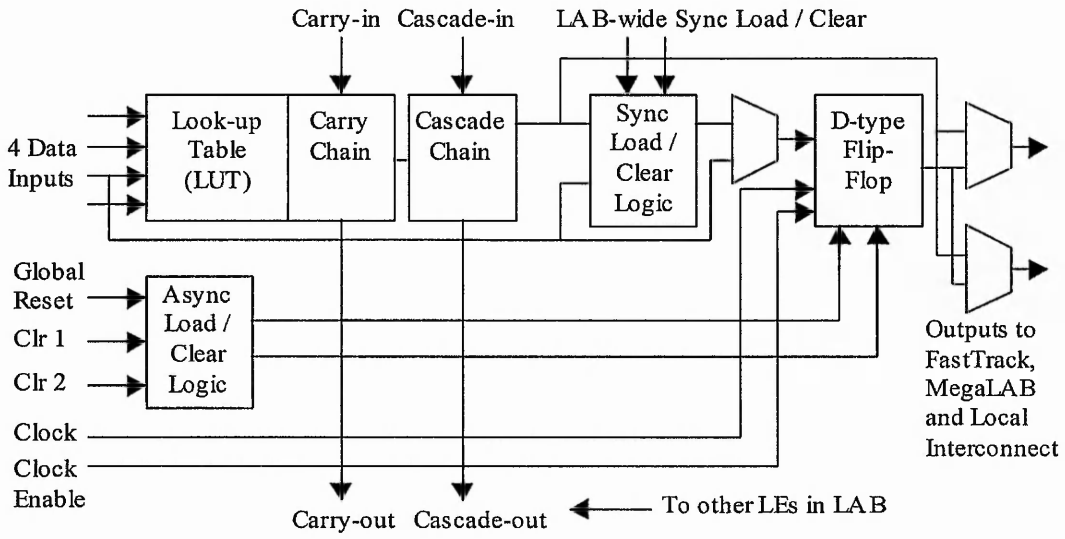


Figure 25: Apex 20K Logic Element

4.6.2.1 Logic Elements

Logic Elements (LEs) were the smallest logic units in the device and could implement any logic function requiring up to four input variables. A fifth input could be utilised by way of the Carry and Cascade functions. The LE inputs fed a four input look-up table with an optional registered output, as Figure 25 demonstrates. Control pins could be any global or local I/O or logic signal. The flip-flop was bypassed when implementing combinatorial logic but a single LE could implement two outputs, one clocked, one not. Additionally, device utilisation can be improved by utilising unused flip-flops to clock signals external to the device, a technique known as register

packing [134]. LEs permitted the formulation of any type of digital circuit whilst still retaining a degree of efficiency.

4.6.2.2 Logic Array Blocks

A Logic Array Block (LAB) was a group of 10 LEs, interconnected by fast local interconnects with LEs in the same and immediately adjacent LABs. These allowed a LE to drive up to 29 other LEs, minimising global interconnect usage. LEs in the same LAB could be connected using carry and cascade chains. Each LABs could be controlled by any combination of the following signals: two separate clocks, two clock enables, and asynchronous and synchronous load and clear signals. Input signals for the LAB entered via the local interconnect either side of the LAB. The LAB output signals could drive either local, row, column or MegaLAB interconnects as shown in Figure 26. LABs permitted the realisation of simultaneously triggered logic with minimised skew, aiding the implementation of parallel design features.

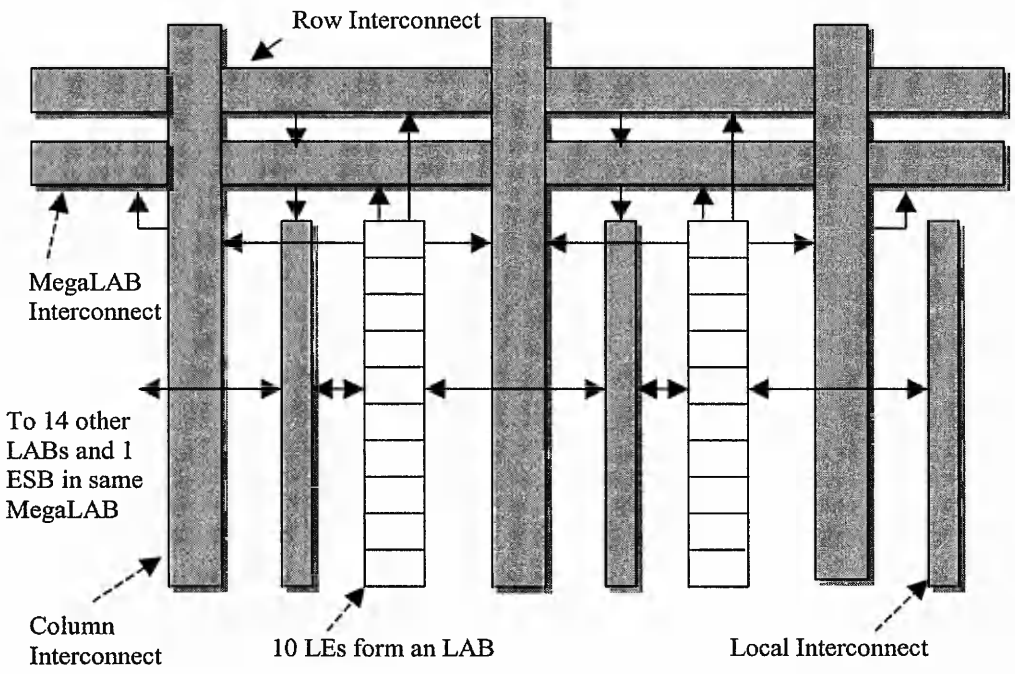


Figure 26: LAB Structure Demonstrating Surrounding Interconnections

4.6.2.3 MegaLAB

In the Apex 20KE devices, a group of 16 LABs and an ESB block formed a MegaLAB, interconnected by a MegaLAB interconnect. Signals external to the MegaLAB entered via FastTrack and local interconnects. Carry and cascade functions could be implemented between any LAB in a MegaLAB. MegaLABs allowed fast signal paths between resources both inside and outside the MegaLAB and could be useful in partitioning designs to gain speed advantages in large designs.

4.6.2.4 FastTrack Interconnect

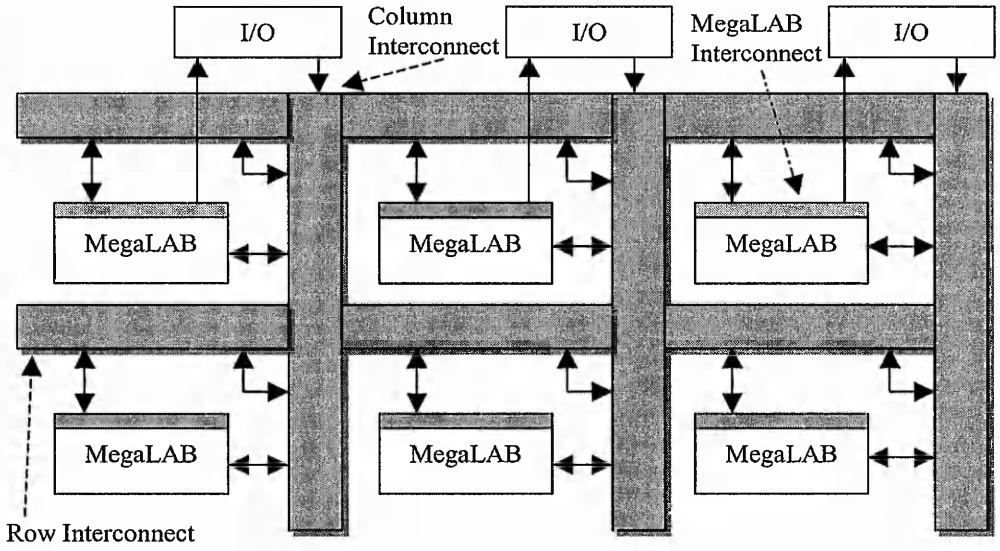


Figure 27: FastTrack Interconnection Grid Structure

FastTrack interconnect was a series of horizontal and vertical (row and column) global routing channels, as shown in Figure 27. LEs with a high fan-out (ie drove a high number of other LEs) could connect directly to the row interconnect by being placed in the last LE of an LAB, or the column interconnect if placed in the LAB nearest the column interconnect. To achieve lower set-up times, column I/O pins can also drive the FastRow interconnect to route signals directly to the local interconnect, bypassing the MegaLAB interconnect.

The architecture of the device was of note when designing a PCI interface, which was subject to strict timing characteristics. This was especially true when implemented on PLDs, which are slower than ASICs, despite other advantages as mentioned in section 4.5. The mapping of the designs onto logic resources was performed by Alteras Quartus II synthesis software [155]. The user could influence the outcome by specifying certain synthesis options. The user could even dictate certain modules and parts of modules to be implemented in certain sections of the PLD, even down to specifying the use of particular LEs in order to obtain higher speeds. The layout and placement of the design was left to software, apart from the pin-out, which was predefined in order to place all PCI I/O along one side of the device. This minimised PCB track layout length and complexity and in turn reduced skew, delay, noise and crosstalk on these signals.

4.6.2.5 Context Addressable Memory

Context Addressable Memory (CAM) [142] was a memory feature first introduced with the Apex 20KE family. CAM was unavailable on the Flex 10K devices used in the implementation of the SARNIC and PCI-OSLi interfaces and the Apex 20K family. CAM could be thought of as the inverse of RAM. When retrieving data from RAM, the address of the desired data was entered with the output being the contents of that address. Retrieving data from a CAM entailed entering the expected data, termed 'pattern', and if the pattern already existed in the CAM, a match was declared with the output being the address of the matched pattern.

CAM achieved very fast search speeds, irrespective of the CAM size, as locations were searched concurrently as opposed to the sequential operation of RAM. Searching the CAM took a single clock cycle with a single ESB capable of implementing a 32-bit wide 32 pattern deep CAM. The CAM implemented in the FT-PCI-OSLi design was 8 bits wide and 16 pattern deep but was expandable if desirable, with most additional work involving the replication of surrounding glue logic. The CAM could support 'don't care' mask bits but this feature was deemed unnecessary, as message

ID bits must be exact. It could be used if message IDs were numbered by masking out the number bits.

4.6.3 High Performance Digital Design Implementation

This section identifies ways in which the internal architecture of the PLD aided the implementation of the FT-PCI-OSLi. The PLDs defined layout gave accurate estimations of the delay paths through the design. Every gate that the signal passed through could be used to form a timing analysis prior to hardware implementation. Post-synthesis simulation was used to obtain performance results for the FT-SARNIC. A timing analysis was used extensively during the development of the FT-PCI-OSLi to ensure that the PCI timing requirements were met before implementation. Certain signals, most notably nIRDY and nTRDY (initiator and target ready respectively) were associated with timing problems due to their fast set-up times and high fan-outs. Pipelining these signals and using high speed interconnects helped reduce the delays to acceptable levels. As designs and their target technologies increase in size, delays across the device increase, leading to signal skew and implementation problems. The internal PLD architecture attempted to minimise such problems through the global clocking which used dedicated clock pins. Clock management circuitry distributed clocks around the design with minimal skew.

The MegaLAB architecture permitted neighbouring design functions to be placed in close proximity physically. The skew incurred through the implementation of combinatorial logic in large designs could cause timing problems due to signals changing at different times. A pipelined architecture was an obvious, but sometimes impractical, solution.

The location of memory segments in the MegaLAB, along with the LABs reflected the need of many circuit elements that utilised memory to have associated glue logic. For example, the Virtual Channel Message Store circuit that surrounded the CAM, detailed in section 5.4. Dividing the memory into multiple smaller blocks allowed for flexibility of implementation and reflected the idea that some designs, like the FT-PCI-OSLi, have many memory requirements.

The PCI bus standard utilised bi-directional I/O reflecting the shared nature of communications and reduced the pin count. The PCI-OSLi design required the bi-directional nIRDY and nTRDY signals to use separate input and output signals and merge the signals externally. This permitted the use of the dedicated fast input pins of the PLD, necessary to meet the timing requirements of this signal. Internal optimisation of these signals, in conjunction with the more advanced Apex 20KE devices, enabled these signals to be implemented as bi-directional I/O, as shown in Figure 28, simplifying PCB layout and reducing noise on these signals.

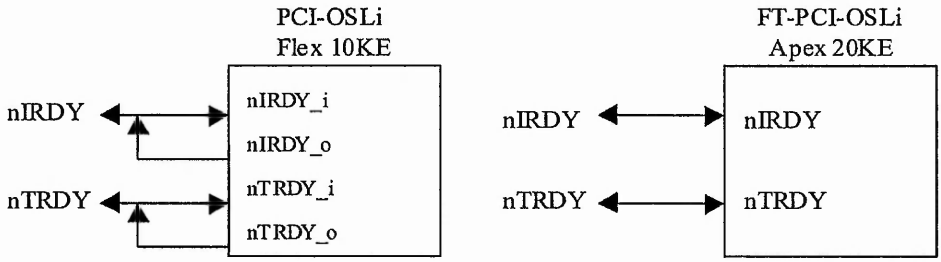


Figure 28: nIRDY and nTRDY signal improvement between the two PCI-OSLi designs

5 DESIGN STRUCTURE

5.1 FT-PCI-OSLi Module Description

The design of the FT-PCI-OSLi interface was divided into three main areas, as shown in Figure 29. The Host System Interface was responsible for co-ordinating transactions with other devices connected to the PCI bus. Message Format Processing Logic was responsible for altering the 32-bit data words to a format more suitable for transmission on the NTR-FTM08 network. The Communications Link Interface was responsible for data insertion and extraction to and from the NTR-FTM08 router network, and the encoding and decoding of tokens respectively. The Host System Interface and Data Flow Layer were linked by two DMA buffers, one for each direction. The 64-word deep buffers accumulated data until there was sufficient to warrant a PCI transfer. The Link Interface buffering formed the boundary between the Data Flow Layer, which was synchronised to the 33MHz PCI clock and the Communications Link Interface. The Communications Link Interface was synchronised to the sample clock, which formed the basis of the 1.5 times oversampling technique utilised, by the asynchronous communications across the NTR-FTM08 network [37]. The 32 token deep Link Interface buffers facilitated the flow of data across the serial communications links. The FT-PCI-OSLi design differed from that of the PCI-OSLi in virtually all modules, with many completely redesigned.

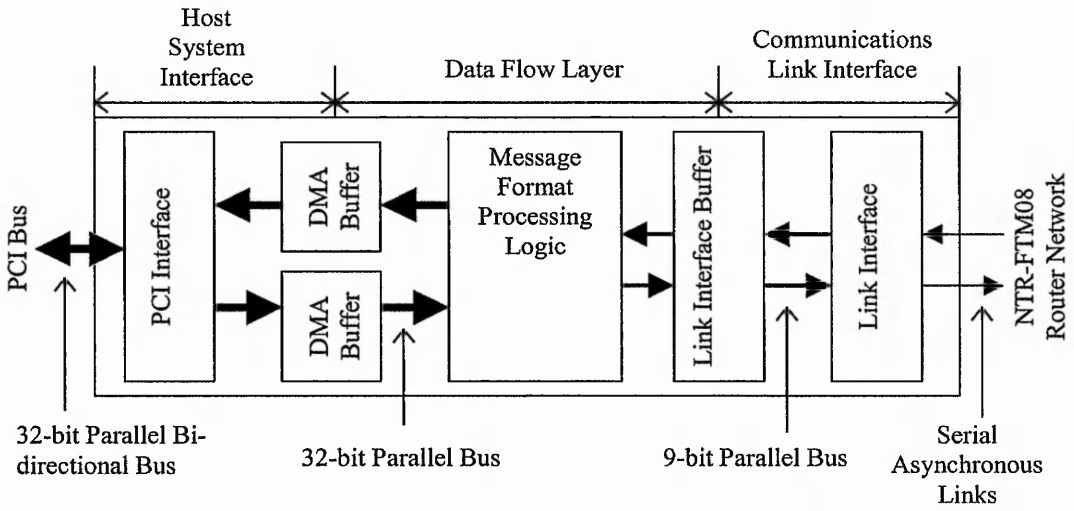


Figure 29: Block Diagram of the FT-PCI-OSLi Interface

5.1.1 PCI Interface

The PCI Interface block was responsible for the initiation of PCI bus transactions involving data sent to and received from the serial communications link. Reception required the PCI interface to monitor the capacity of the receiver channel DMA buffer, initiating a transfer when the number of data words stored reached a certain value. Transmission required the PCI Interface to monitor the PCI bus control signals to identify the initiation of a transfer by another device attached to the bus. These control signals were decoded to determine if the FT-PCI-OSLi was the intended recipient of the transaction, requiring the FT-PCI-OSLi to prepare for data reception.

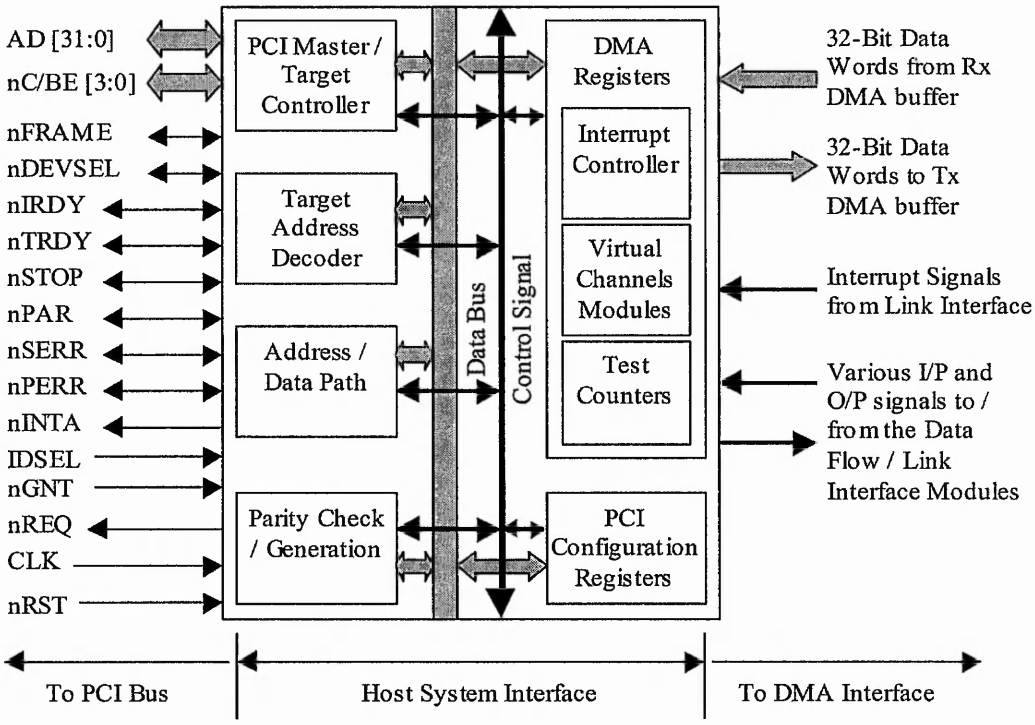


Figure 30: Block Diagram of the PCI Interface and its associated I/O signals

The PCI Interface consisted of six main components, as Figure 30 demonstrates.

- A PCI Master / Target Controller determined the mode of operation for the current transaction.
- The Target Address Decoder determined if PCI transactions initiated by another PCI agent were destined for the FT-PCI-OSLi.

- The Address / Data Path module multiplexed and demultiplexed these two buses onto a single bus and synchronised the PCI data bus signals to meet the strict set-up times of the bus.
- A Parity Check / Generator controller maintained the integrity of communications over the PCI data bus and the four bit Command / Bus Enable (nC/BE) signals.
- PCI Configuration Registers hold information necessary to identify the FT-PCI-OSLi to other users of the PCI bus.
- The DMA Registers housed three main functional blocks: An Interrupt Controller, hardware to implement Virtual Channels and a series of test counters used to aid development.

5.1.1.1 PCI Master / Target Controller

The PCI Master / Target controller module controlled the interface between the FT-PCI-OSLi and the PCI bus. It was responsible for the creation and monitoring of many of the PCI bus signals used in transactions involving the FT-PCI-OSLi. The main difference between the Master / Target controller implemented in the FT-PCI-OSLi and the comparable module in its non-fault tolerant predecessor was the implementation of the master and target state machines. The number of states in the state machines were reduced from 7 to 4, and from 8 to 5 for the master and target state machines respectively. These two state machines operate independently from one another but only one should be active at any one time.

Acquiring PCI Bus Ownership

A request for ownership of the PCI bus is made following assertion of the active low 'nREQ' PCI signal (see Appendix B). Request arbitration is handled by the PCI chipset with priority usually given on a 'round-robin' basis although the actual implementation is left to the PCI chipset designer, being unspecified in PCI specification 2.1. Making the request decrements the count stored in the latency counter. This 8-bit value is stored in bits 8 to 15 of configuration register offset $0C_H$

(see Appendix E) and is decremented every PCI clock cycle whilst a transfer is occurring. When the count reaches zero, the FT-PCI-OSLi must surrender bus ownership and make another request to transfer data if the message has not yet been completed. The latency count is decremented both when the PCI transfer is occurring and whilst it is being set up and can be used to calculate the number of clock cycles taken to acquire ownership and to initiate transfer following the granting of ownership.

Bus Master Mode of Operation

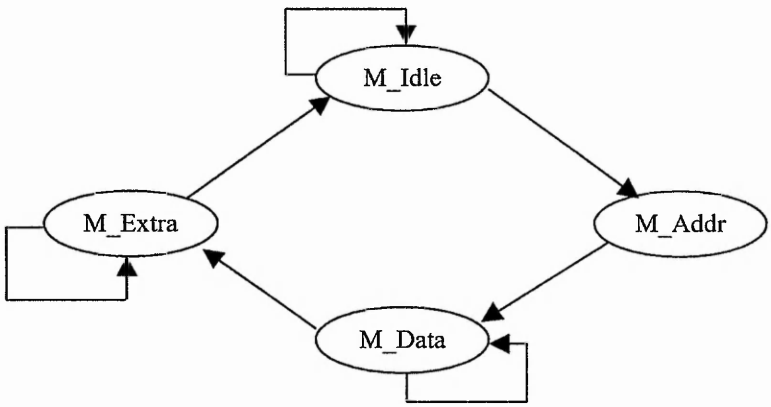


Figure 31: FT-PCI-OSLi Master State Machine

Figure 31 illustrates the operation of the Master State Machine. The first state, entered on reset was 'M_Idle' and it remains in this state until one of two conditions was met. Either there was enough data in the receiver DMA buffer for the FT-PCI-OSLi to initiate a transfer or the PCI Bus Arbiter could decide to allocate the FT-PCI-OSLi the task of 'bus parking' [41, 145]. These conditions de-asserted the 'nREQ' (see Appendix B) signal, indicating that the device had requested ownership of the PCI bus. The second, 'M_Address' state was entered on being granted ownership of the PCI bus, indicated by receipt of the 'nGNT' signal. This state was occupied for a single PCI clock cycle, during which the initial memory address for the transfer was loaded into an internal register as part of the PCI address cycle.

On the following PCI clock cycle, the 'M_Data' state was entered, during which data transactions could occur. This state was left when either:

- Data transfer had been completed – initiated by the 'm_lst1_data' signal from the DMA Registers module indicating that the number of data words transferred equalled the number of expected transfers.
- The FT-PCI-OSLi device was forced to surrender bus ownership. This occurred if the transfer was not completed within a set time, indicated by the expiration of the latency counter.
- The target requested that the bus master halt the transfer through assertion of the 'nSTOP' signal (see Appendix B).

Satisfying any of these conditions moved the state machine into the 'M_Extra' state, again only for a single PCI clock cycle. During this state the FT-PCI-OSLi surrendered bus ownership before returning to the 'M_Idle' state.

Target Mode of Operation

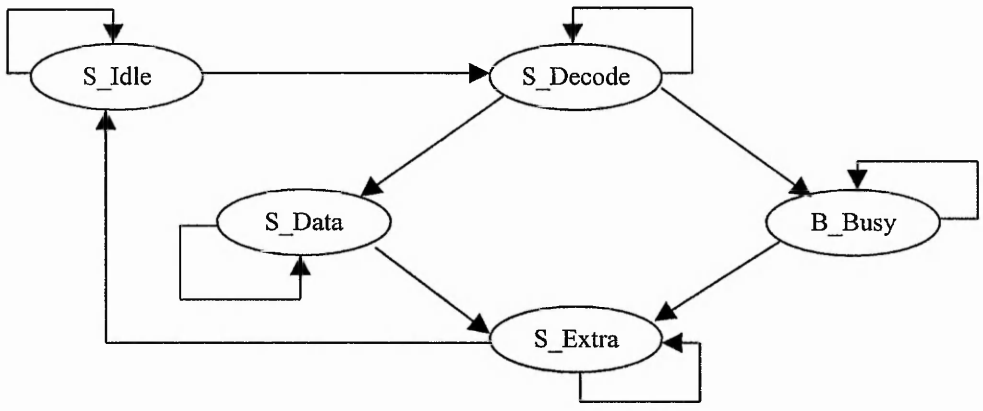


Figure 32: FT-PCI-OSLi Target State Machine

If the FT-PCI-OSLi became the recipient of a PCI transaction, the interface became the 'target', invoking the use of the target state machine, illustrated in Figure 32. This state machine entered the 'S_Idle' state on reset, moving to the 'S_Decode'

decode state on assertion of the 'nFRAME' PCI signal (see Appendix B). Assertion of the 'hit_f' signal from the 'Decode' module or the 'cnfg_cyc' signal indicated that the FT-PCI-OSLi was the intended recipient of the transaction. These signals were decoded from the address bus and 'nC/BE' buses respectively. If the FT-PCI-OSLi was the intended destination for the transaction, the target state machine moved from the 'S_Decode' to the 'S_Data' state.

If another device attached to the PCI bus was the intended recipient then the state machine entered the 'B_Busy' state, indicating bus activity not connected with the device. The FT-PCI-OSLi target state machine remained in either of these states until the de-assertion of the 'nFRAME' signal indicated the end of the data transfer, irrespective of the recipient. This moved the state machine to the 'S_Extra' state, valid for one clock cycle during which time the FT-PCI-OSLi must handover possession of the bus and return to the 'S_Idle' state.

5.1.1.2 Address Decode Module

This module latched the PCI address bus signals to determine if the FT-PCI-OSLi was the target of the next PCI bus transaction. These signals were also decoded to provide pointers to the address of the current data word in the configuration and data storage spaces in memory. Address lines 2 to 7 and 20 to 31 were latched with the former used to determine the read / write address for the next memory access. Bits 20 to 31 decoded a 1MByte area of memory with the 'hit_f' signal indicating an address match with a configuration register address. The Master / Target Controller used this signal to indicate that the FT-PCI-OSLi was the intended recipient of the transaction.

5.1.1.3 Address / Data Path Module

The separate 32-bit address and data buses of the PCI interface were multiplexed in this module and were latched to ensure the stringent timing requirements dictated by the PCI bus specification were adhered to.

5.1.1.4 Parity Generator / Verifier

The 32-bit PCI data bus, AD[31:0] and the 4-bit active low 'nC/BE' bus (see Appendix B) were sampled in this module and an even parity signal was generated for each PCI word outputted onto the bus. The receiving PCI entity checked the parity signal against the received buses. The 'nSERR' or 'nPERR' parity error signals denoted errors detected in the address or data phases of the PCI transfer respectively.

5.1.1.5 PCI Configuration Registers

This module contained sixteen 32-bits wide registers used for storing information denoting the identity and characteristics of the FT-PCI-OSLi, used in setting up the PCI bus and determining which devices were attached to it. Appendix E lists all registers used in this design.

5.1.1.6 DMA Registers

This module contained thirty-two word-length user registers, mapped into the PC memory, as listed in Appendix C. These registers were used for several different functions. These included the display of control and status signals to assist development and the writing of message information to the registers to initiate transfers. Interrupt sources were configured and controlled via the Interrupt Enable Register and Interrupt Pending Register respectively.

The FT-PCI-OSLi user registers were accessed via free software called the PCIWave Exerciser from PLD Applications [156]. It permitted the user to read and write data directly to the host system memory without the need of a device driver. Such a task needed performing with care to avoid corrupting the memory contents. A program searched through the devices attached to the PCI bus, checking the device ID against that of the FT-PCI-OSLi. When found, the program allocated several available

memory segments of 65,536 double words (262,144 bytes) for use by the FT-PCI-OSLi as data sources and receptacles.

The DMA Registers module also contained the Virtual Channels sub-modules that are discussed in greater detail in section 5.4.4.

5.1.1.7 Interrupt Controller

Located within the DMA Registers module, the Interrupt Controller was responsible for the generation of the active low 'nINTA' PCI signal. The host system serviced requests following the assertion of any of the multiple interrupt sources when that interrupt source was enabled in the Interrupt Enable Register. All interrupt sources requesting interrupts set corresponding bits in the Interrupt Pending Register. Only enabled interrupts resulted in the generation of 'nINTA' and the interrupt being serviced.

5.1.2 Data Flow Layer and Communications Link Interface

This section gives a detailed breakdown of data path through the data flow modules of the FT-PCI-OSLi design, from the serial communications network to the DMA FIFOs prior to transfer over the PCI bus. Figure 33 demonstrates the flow of data, and its format, through this part of the FT-PCI-OSLi interface.

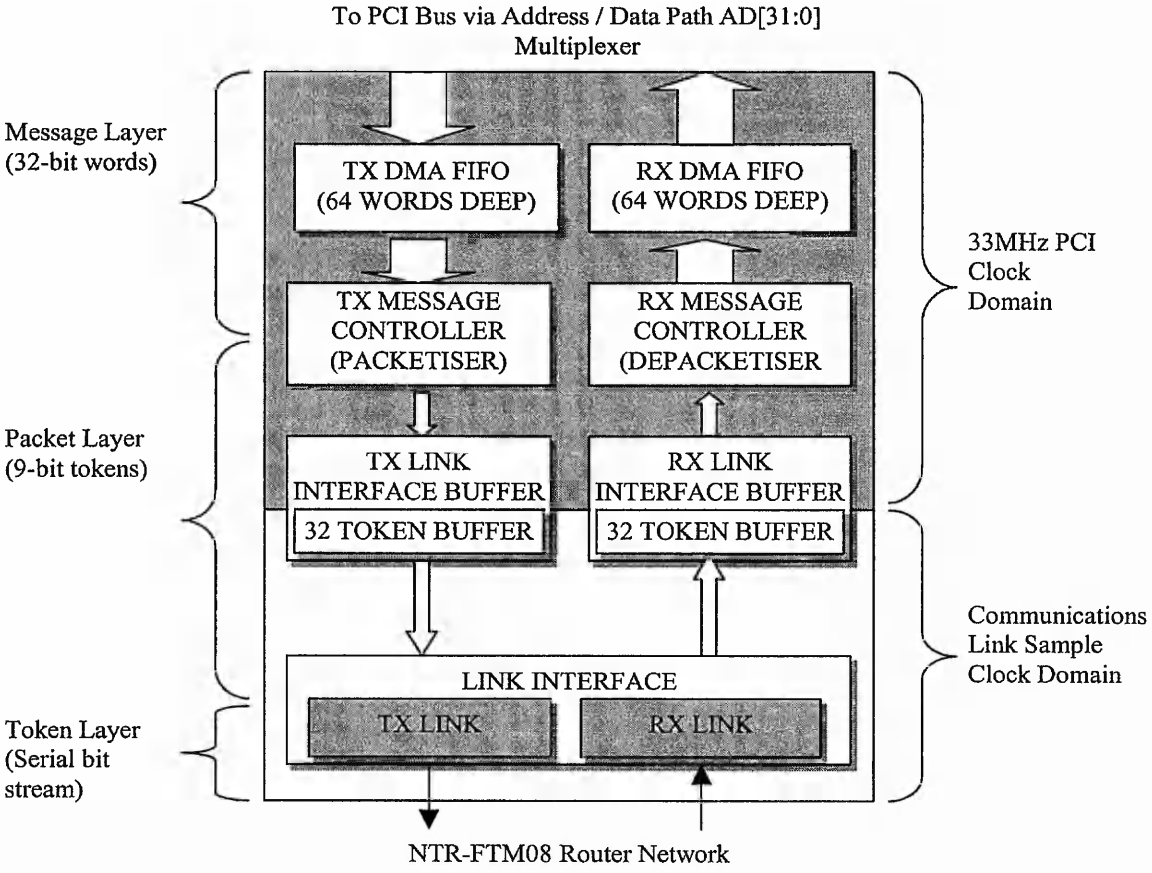


Figure 33: Block Diagram of the Data Flow and Link Interface sections of the FT-PCI-OSLi interface

5.1.2.1 Link Interface

The link interface converted 9-bit parallel tokens into a serial bit stream for transmission onto the router network and vice versa. Only the data byte and type bit were transferred to and from the link interface and its buffers. Start and stop bits were inserted into the bit stream immediately before transmission and removed immediately after reception. Control tokens were inserted into and removed from the data stream at the link interface with only termination tokens progressing to the link interface buffers. Stop and Go flow control tokens took a higher priority than data tokens and were inserted and extracted from the bit-stream when required. The link interface determined which state the link was in, as detailed in section 4.4.5, and regulated the link initialisation and connection request procedure.

The disconnection detection mechanism was located within the link interface module. It utilised two signals, namely 'Heartbeat' and 'CheckPulse' to generate and verify link activity respectively. Heartbeat tokens were generated once every 255 sample clock cycles, approximately equal to the time taken to transmit 15 tokens. In the absence of any data to transmit, a heartbeat token (introduced in section 3.5) was sent to allow the receiver to verify that the link was still operational. Periodic token transmission reduced signal activity when compared to protocols such as IEEE Std. 1355-1995, which required constant logic transitions on the physical channels. Assertion of the Heartbeat signal resulted in the transmission of the flow control token relative to the current link status. This allowed regular link status verification and safeguarded against the loss of control tokens. The CheckPulse signal cleared a flag which was set on receipt of any token, whether data or control. If the flag had not been set when the CheckPulse signal was next asserted a disconnection error was flagged. This was because a flow control token should have been received through assertion of the Heartbeat signal, even in the absence of data. Triggering any error detection mechanism moved the state machine that governed link status into the reset state. This invalidated any further tokens received before the link had been correctly initialised by way of the connection request procedure (see section 4.4.5).

5.1.2.2 Link Interface Buffering

The link interface buffers formed the boundary between the sampling clock controlled link interface and the PCI clock controlled message interface. The buffers were 32 tokens deep, 9 bits wide and operated in a first-in-first-out (FIFO) manner. Type (or ID) bits passed through the FIFOs as termination tokens were added and removed at the message interface part of the design. Data entering and leaving the transmitter buffer was synchronised to the PCI and sample clocks respectively. 'Full' and 'Empty' signals left the buffers to notify other modules of data saturation and starvation respectively. The receiver buffer created two additional level pointers, termed 'Almost-full' and 'Almost-empty', set to trigger when the number of tokens in the buffer was in excess of 24 and less than 8 respectively. The buffers were

implemented in the embedded memory arrays of the PLD as dual-port RAM to allow simultaneous read and write accesses.

5.1.2.3 Transmitter Message Control

This Transmitter Message Controller co-ordinated the transfer of data from the transmitter's DMA buffer to the link interface buffer and the intermediate message formatting, as shown in Figure 33. The data was packetised in hardware to reduce software overheads. Header and termination tokens were added at this stage. Data words were split into four data bytes and a logic one ID bit was added to form the ninth bit of the token passed to the link interface buffer. Up to four header tokens could precede the start of the data stream with up to three of these being optional routing headers. These could be stripped as the message traversed the router network. The final and mandatory header was the message ID. All header tokens possessed a logic one type bit and were denoted by their position in the received data stream. The message length information was passed to this module from the DMA registers module. As data was outputted from the transmitter DMA buffer, the message length counter was decremented until a count of zero denoted the end of message. At this point, an EOM token, with a logic zero ID bit was appended to the data stream.

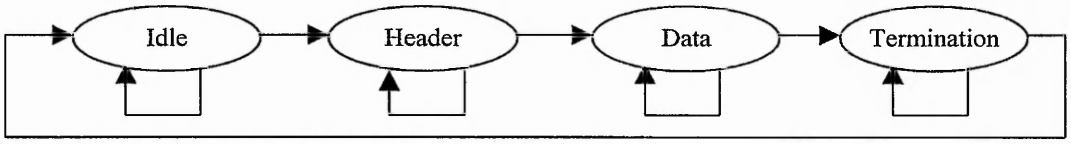


Figure 34: FT-PCI-OSLi Transmitter Link Controller State Machine

The transmitter message controller possessed two state machines. Their functions were to control data transfers to the transmitter DMA buffer and the transmitter link interface buffer. The transmitter link interface state machine had four states; 'Idle', 'Header', 'Data' and 'Termination'. These reflected the type of tokens that were being transferred to the link interface buffer at that time. This state machine is shown in

Figure 34. The state machine moved from 'Idle' to the 'Header' stage when the command to start packet transfer was received from the higher levels of the design. This occurred by writing to the Transmitter Length Register (Offset $0D_H$) in the DMA Registers module. In the 'Header' state, header tokens were loaded onto the link interface FIFO from the register, where they were stored after being fetched from memory. Once the last header token had been loaded into the link interface buffer, the state machine moved into the 'Data' state, loading data from the DMA buffer. After the last data token had been packetised and written to the link interface buffer, the 'Termination' state was entered. An EOM token was loaded into the buffer and the state machine returned to the 'Idle' state to await the next message.

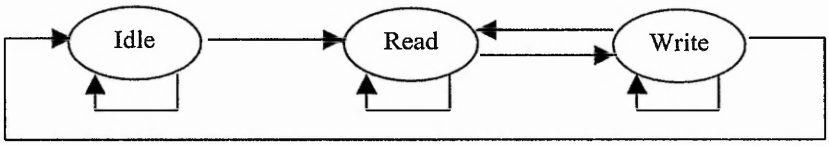


Figure 35: FT-PCI-OSLi Transmitter Message Controller DMA State Machine

The transmitter DMA state machine shown in Figure 35 had three states; 'Idle', 'Read' and 'Write'. It only operated when the transmitter link interface state machine was in the 'Data' state. On entering this state, the DMA state machine advanced from 'Idle' to 'Read'. This transferred a 32-bit data word from the DMA FIFO for packetisation, before moving to the 'Write' state. The 'Write' state appended a logic 1 ID bit to each of the four data bytes that made up the 32-bit data word and transferred each 9-bit token to the link interface buffer in turn. The state machine moved back to the 'Read' state upon reading the fourth byte in the word into the link interface buffer. The process was repeated until the transmitter DMA buffer was empty. The empty transmitter DMA buffer indicated that the packetisation of the data transferred in that PCI transaction was complete, moving the state machine to 'Idle' to prepare the transmitter DMA channel for another PCI transaction.

5.1.2.4 Receiver Message Control

The Receiver Message Controller facilitated the transfer of data from the receiver link interface buffer to the receiver DMA buffer via the depacketiser. Data was converted from the 9-bit token format of the link interface buffer to the 32-bit data words required for PCI transactions.

The type bit of tokens passed from the link interface FIFO to the receiver DMA FIFO was examined in this module. All other control tokens had been removed from the data stream by this stage with only the EOP, EOM and BEOP tokens remaining in addition to the data tokens. On receiving acknowledgement from the header storage module that the received message ID was valid (or expected and therefore the length and address for transfer to memory was also known), an associated message length value was transferred from the header storage module. The length was decremented as data was outputted from the receiver link interface FIFO. The time at which the message length counter reached zero, relative to the detection of the termination token in the depacketiser, determined the mode of termination, of which there were three:

- If the message count did not equal zero when the termination token was read then the message was terminated earlier than anticipated – Early termination.
- If the message count equalled zero when the termination token was read then the message was terminated when expected – Normal termination.
- If the message count reached zero and the termination token was not present then the message was longer than expected – Late termination.

Early and normal termination resulted in the message being transferred to memory in its entirety. The setting of bit 2 in the interrupt pending register was used to denote early termination. Late termination indicated that the message was longer than expected, and as such there could be insufficient resources allocated to that message in memory. The expected number of data bytes were transferred to memory and the remainder flushed from the receiver. A late termination error was flagged using bit 3 of the interrupt pending register. Header tokens were compared with the contents of

the CAM to determine whether or not a header match was achieved. Termination tokens were removed upon identification so that only data was transferred to memory.

Like the transmitter message controller, the receiver message controller had two state machines. One controlled the passage of data from the link interface buffer, shown in Figure 36, the other, shown in Figure 37 controlled the writing of data to the DMA buffer.

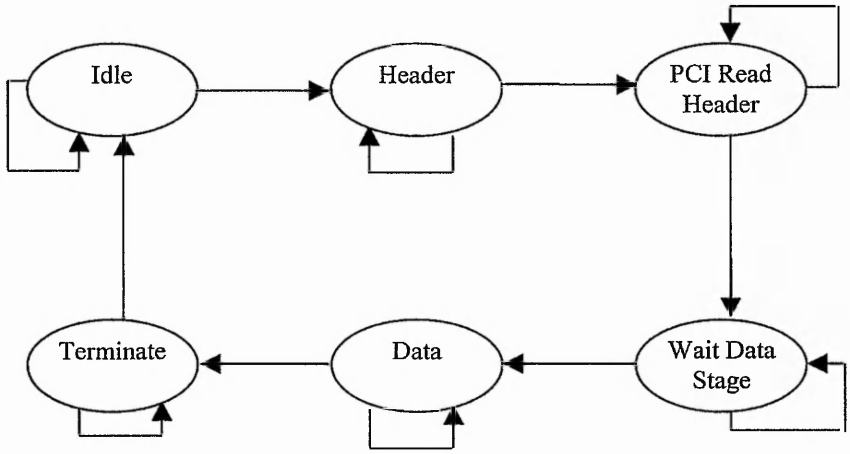


Figure 36: FT-PCI-OSLi Receiver Link Controller State Machine

The Receiver Link Controller State Machine resided in the 'Idle' state until the loading of the first received token into the link interface buffer. Its position in the incoming bit-stream indicated that it should be the header of the message, moving the state machine into the 'Header' stage. Once complete, the received header was compared with the contents of the CAM to determine whether the header was expected. Whilst this occurred the state machine was in the 'PCI Read Header' state, advancing to the 'Wait Data Stage' state when the header had been validated. This state synchronised the two state machines preventing the passage of data until the DMA buffer was ready to receive it. When the receiver message controller DMA state machine, shown in Figure 37, reached the 'DMA Sync' state, the link interface state machine advanced to the 'Data' stage. It remained there until a termination token was read from the output of the link interface buffer. The state machine then moved into the 'Terminate' state, returning to the 'Idle' state when the termination token had been removed from the data stream.

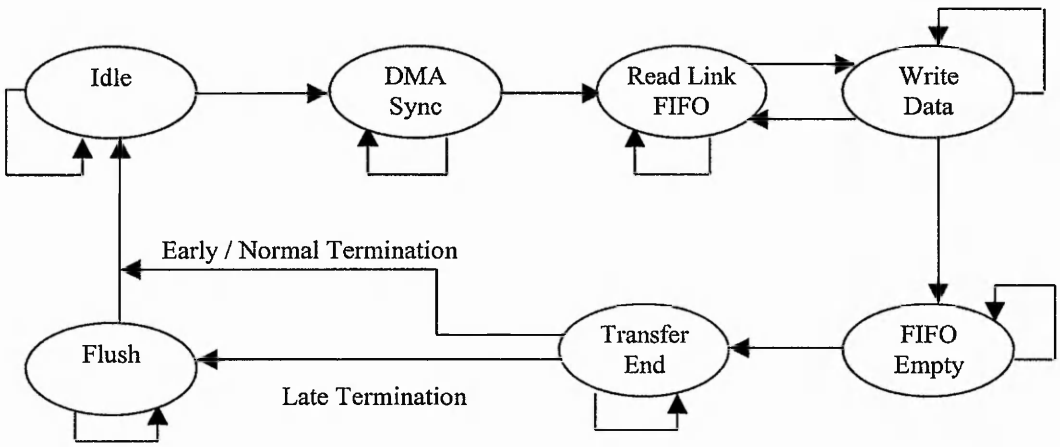


Figure 37: FT-PCI-OSLi Receiver Message Controller DMA State Machine

The Receiver Message Controller DMA State Machine initially resided in the 'Idle' state. It moved to the 'DMA Sync' stage when the received header had been validated by the contents of the CAM. This state was equivalent to the 'Wait Data Stage' in the link interface state machine in Figure 36. When both states were active simultaneously the DMA state machine advanced to the 'Read Link FIFO' state. In this state data tokens were transferred from the link interface buffer to the depacketiser. After four bytes had been transferred from the Link Interface Buffer, a 32-bit word was ready for transfer to the DMA buffer, forcing the state machine into the 'Write Data' state. Once the data word had been transferred to the DMA buffer the state machine returned to the 'Read Link FIFO'. This procedure was repeated until the DMA buffer was full, advancing the state machine to the 'FIFO Empty' state. Whilst in this state the DMA buffers contents were transferred to memory. Once completed the 'Transfer End' state was entered whilst the mode of termination was determined. If normal or early termination occurred, the state machine returned to the 'Idle' state. Late termination moved the state machine to the 'Flush' state removing the excess data from the link interface buffer before returning the state machine to the 'Idle' state.

5.1.2.5 DMA Buffer

The DMA buffers were temporary data stores for data that had been read from or written to the PCI bus. There was one buffer per direction operating on a first-in-first-out (FIFO) principle. The DMA transmitter buffer stored data previously fetched from the host system memory, via the PCI bus, prior to packetisation and transmission onto the serial router network. PCI data bursts allowed data to be loaded onto the buffer at a rate of one 32-bit double word per PCI clock cycle. Data exited the buffer at one quarter of the rate it entered, being packetised at a rate of one token per PCI clock cycle. Following a DMA burst the link interface was required to catch up, as it effectively became a bottleneck in the system. Despite this, the serial link interface was the true bottleneck as it took 11 sample clock cycles to transmit a data token onto the router network.

The receiver DMA buffer provided temporary data storage for data coming from the link interface, storage was necessary to accumulate sufficient data to make an efficient DMA transfer to the host system memory, via the PCI bus. A request was made when the receiver DMA buffer was almost full or when the end of the message was reached. If message transfer was not complete following the emptying of the buffer, subsequent requests to master the bus were made at a rate dependent on link throughput. The 64 word deep DMA buffer used in the non-fault tolerant PCI-OSLi interface was implemented in the FT-PCI-OSLi.

5.1.2.6 DMA Controller

Data flow within the FT-PCI-OSLi interface could occur in both directions simultaneously in every part of the design except one. The 32-bit bi-directional PCI bus could only transfer data in one direction at any one time, requiring the multiplexing of the transmitter and receiver DMA channels. The DMA controller arbitrated between these two functions to ensure that no one resource could monopolise access to the bus. Figure 38 illustrates the state machine of the DMA controller. The controller initiated a transfer when either:

- The transmitter DMA buffer was almost empty, in order to request a transfer from memory or,
- The receiver DMA buffer was almost full, in order to request a transfer to memory.

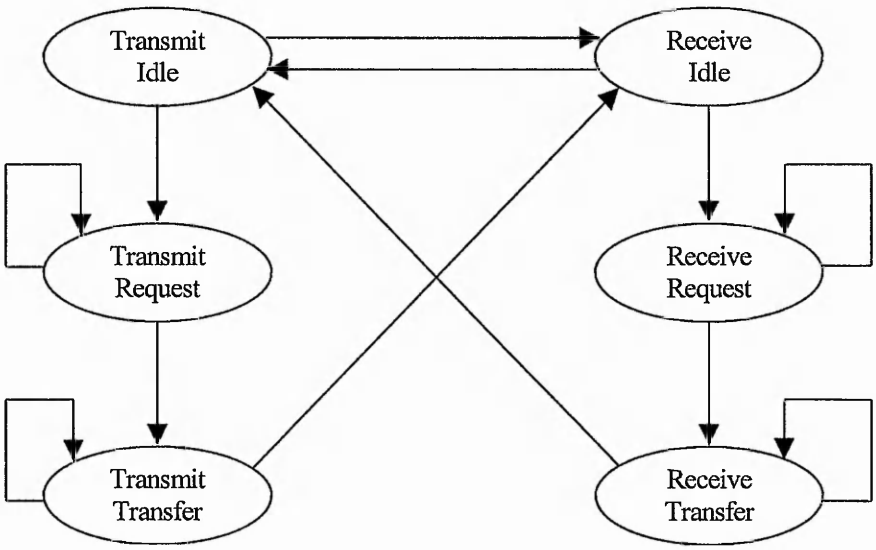


Figure 38: FT-PCI-OSLi DMA Controller State Machine

On the leading edge of every PCI clock cycle the DMA controller checked both DMA channels to see if either were ready to transfer data. Control over which channel took priority toggled between the transmitter and receiver DMA channels every clock cycle. If a channel was ready to transfer data, the DMA controller made a request to master the PCI bus and initiated a transfer, moving the state machine into the request state for the appropriate channel. On being granted bus access, the corresponding transfer state was entered and on completion of the transfer, the state machine moved to the idle state associated with the other type of transfer. For instance if a transmission transfer had just completed, the state machine would move to the 'Receiver Idle' state. This ensures fair arbitration by giving a pending receive transaction a chance to execute.

5.1.3 Virtual Channel Message Store

Although part of the DMA Registers submodule, the VCMS (Virtual Channel Message Store) is dealt with separately as it is a major novel part of the FT-PCI-OSLi. The VCMS (Virtual Channel Message Store) enabled the pre-loading of expected message headers and their associated address and length information into the FT-PCI-OSLi interface. Pre-loading multiple message IDs allowed the interface to handle the scenarios noted in chapter 3 that were observed in the ICR-C416 based network. These included: Message arrival out of order, alternating incoming messages and minimising the need for user intervention following message arrivals from the embedded network. The VCMS had the capacity to hold sixteen byte-length message IDs. As mentioned in section 4.4.9, messages were assigned to one of three classes, based on maximum length. The class 1 and 2 length values were user definable. These stores had capacity for ten and four message IDs respectively. Class 3 possessed two locations, and no defined maximum length, other than that of the maximum message size. This class was designed for one-off messages whose message IDs were removed from the CAM after use. Class 1 and 2 headers remained in the CAM for future use, only being removed if that particular (or all) message IDs were deleted. These procedures occurred through the user accessing the Command Register (offset $0F_H$).

The Command Register was used to input user commands into the FT-PCI-OSLi. Its contents are detailed in Appendix C. The Status Register was used to display information from FT-PCI-OSLi relating to the last messge read or write transaction to or from the interface in addition to displaying the contents of the CAM. Its contents are detailed in Appendix C.

5.1.3.1 VCMS Operation

Initially maximum message lengths for classes 1 and 2 were written to registers (additional details in Appendix C) which enabled the message length comparator necessary to determine message class. Message IDs and lengths were written to the appropriate registers. This process compared the expected message length with the class 1 and 2 preset values. The result of this comparison determined which class the

message belonged to and its position in the CAM. Each location in the CAM had a mask bit associated with it indicating if that particular CAM location contained a header or not. Each class had an priority encoder to fill the CAM locations in order. Header deletion cleared the mask bit for that location, forcing the next expected header write to fill that CAM location as it had a higher priority than subsequent locations, as Figure 39 shows. The order in the CAM is irrelevant but the priority encoder ensured that the status of each class of the CAM (full or empty) was always known and appropriate action could be taken.

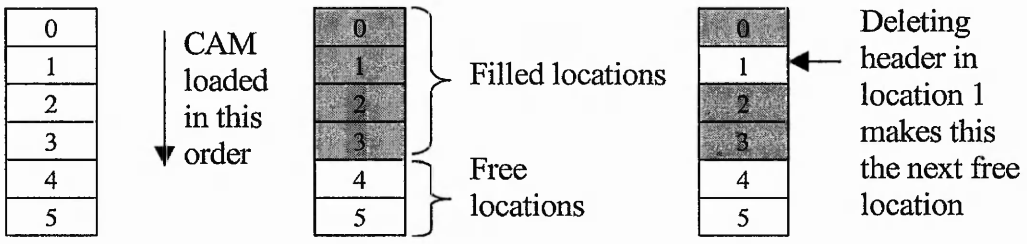


Figure 39: CAM Priority Loading Principle

If all locations associated with that particular class have been filled, no write could occurr and bit 30 in the Status Register was set to denote a failed CAM load. If this was the case, the message could be written to one of the two class 3 locations, if free.

The procedure for handling incoming message headers received from the routing network was different to the handling of expected message headers. The former classified messages according to the header pattern whilst the latter classified messages according to length. Incoming message IDs were loaded into the 'Pattern' input of the CAM but as the write signal was not asserted, the CAM compared the pattern with its contents, returning the address of a match, if one existed. The match address was decoded to determine the class of the received message. Its associated length and address parameters were loaded into registers in the DMA Registers module. Class 3 messages were used only once and the mask bit for that message was cleared at this point. A match signal was generated and used to start the depacketisation procedure. Figures 40 and 41 show the procedures for writing and reading message information to and from the VCMS.

Expected Message Information Load Procedure

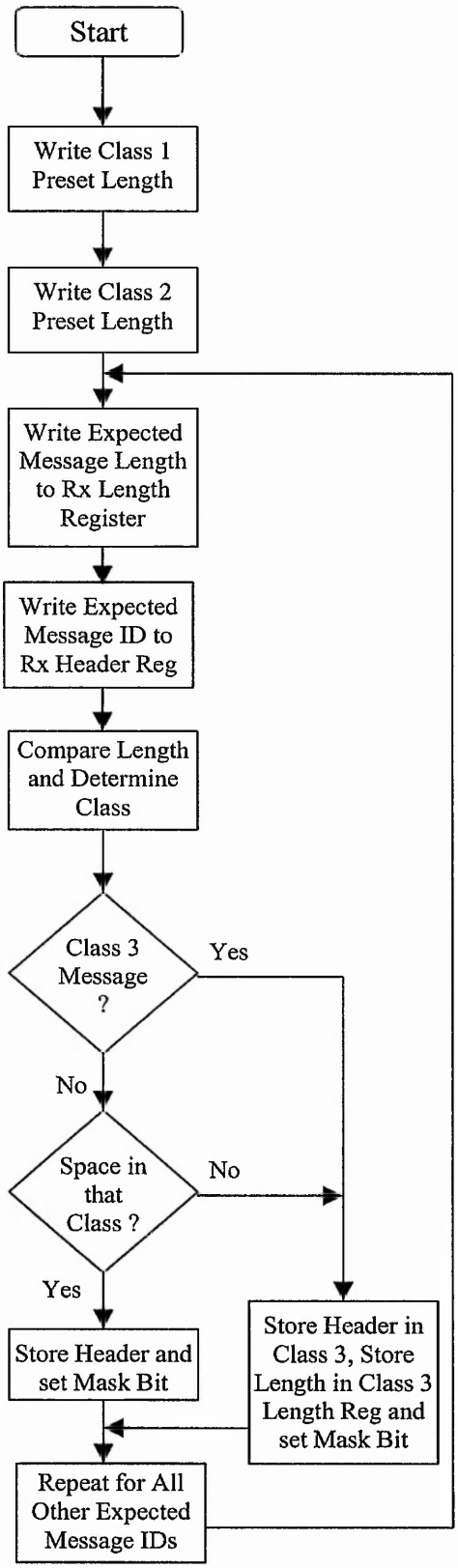


Figure 40: FT-PCI-OSLi Expected Message Information Load Procedure

Incoming Received Message Header Verification Procedure

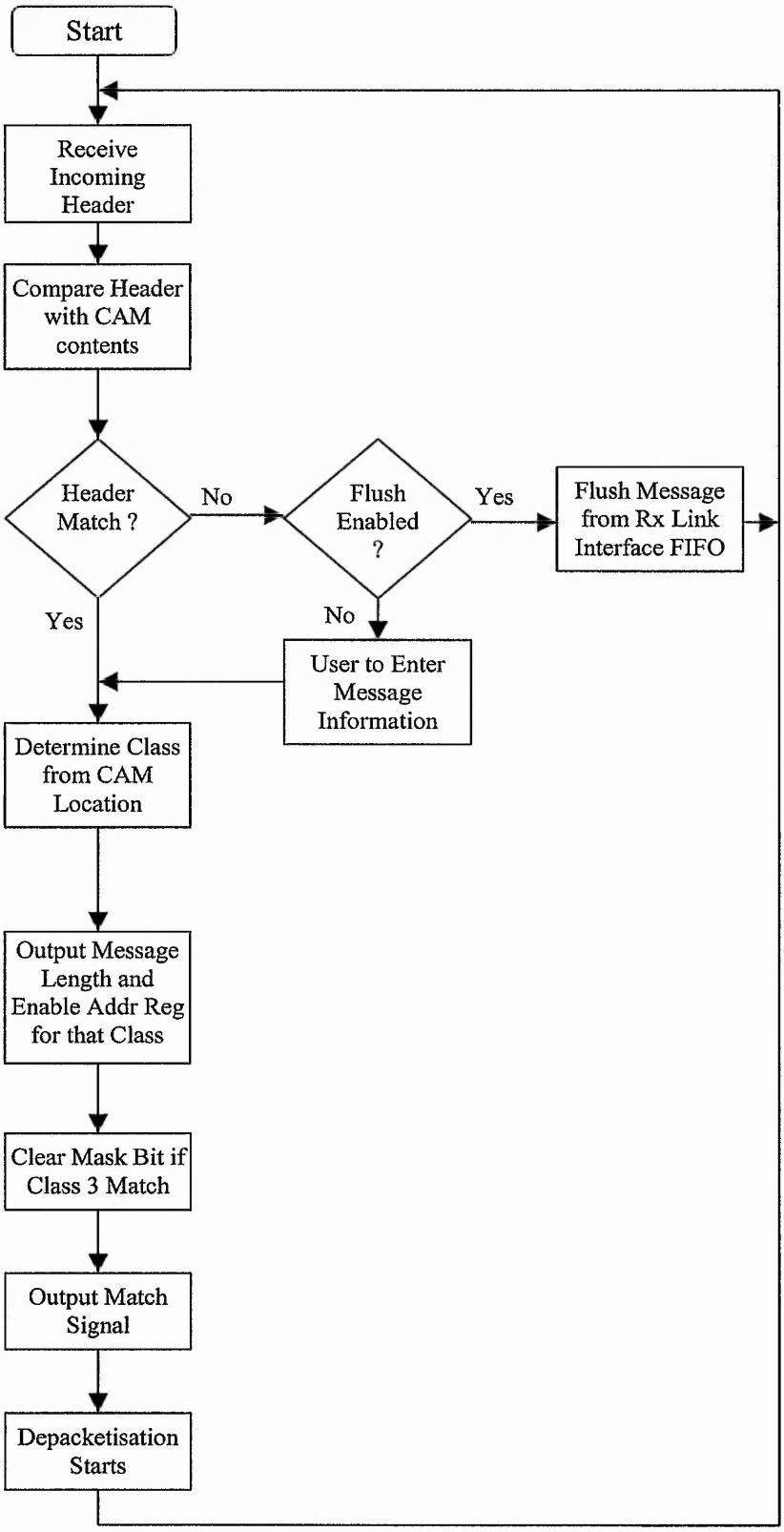


Figure 41: FT-PCI-OSLi Incoming Message Header Verification Procedure

5.1.3.2 Virtual Channel Message Store Modular Breakdown

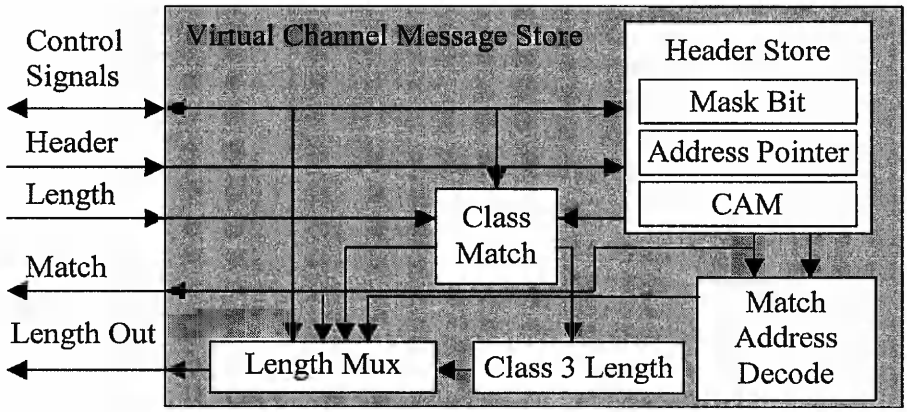


Figure 42: Block Diagram of the Virtual Channel Message Store Submodules

The VCMS was the top-level module in this part of the design, housing all the lower level modules, as shown in Figure 42 and latched the flags that make up the Status Register.

Class 1 and 2 maximum message lengths were stored in registers located in the Class Match module. Expected message length values were compared and message class determined on this basis. Length comparison could not occur until class 1 and 2 values had been specified, as writes to these registers are required to enable the comparator. Message length comparison occurred concurrently for all locations in all three classes.

In addition to the CAM, which stored the message IDs, the Header Store module also housed synchronisation logic to control CAM access. CAM comparison was constantly active so there existed a requirement to ensure that comparison only occurred on receipt of a header from the communications link. Logic was required to prevent a header match from starting the depacketisation process until the interface was ready. This was when the transfer of the previous message in that class to memory had been acknowledged by the clearing of the appropriate 'next message' flag in the Status Register. This module was also responsible for the removal of

message IDs from the CAM. This could be performed manually, via user commands or automatically (class 3 header removal following a match).

The Address Pointer module selected the next free location for that particular class, if available, for writes to the CAM. Signals from the Class Match module indicated the class of the message and this module indicated if there was a free location for that particular class.

The Mask Bit module was located within the Address Pointer module, it contained one flag per CAM location. The flag was set on a CAM write, and cleared following a deletion of that location. The flag also cleared following receipt of a message header from the communications link that matched a header held in the class 3 section of the CAM.

The CAM possessed 16 locations, each of which held a byte length message ID. Message IDs were presented at the 'Pattern' input of the CAM. Assertion of the write signal determined whether or not the header was written to the CAM or compared with the contents of the CAM. The CAM address signals, generated in the Address Pointer module dictated the location in which the header was stored. In the case of header deletion, the address pointer gives the location whose contents were to be removed. Write and delete signals were asserted for two clock cycles. If the message ID at the 'Pattern' input to the module matched the contents of the CAM, the 'Match' output was asserted. At this point the 'Match Address' bus indicated the location of the match, from which the message class was deduced. The CAM was not set up to allow multiple matches. Although this option was available, it was deemed unnecessary as message IDs could be reused (with the exception of class 3 headers) and a message ID associated with two or more classes defeated the object of message classes. If the same message ID was written to multiple CAM locations, only the most recent was valid. The internal operation of the CAM is detailed more thoroughly in section 4.6.2.5.

The Match Address Decode module decoded the address of the matching message ID to decide which class the message belonged to and which maximum message length should be used. A two bit select signal was generated whose value indicated

message class: 00_B indicated a class 1 message, 01_B indicated a class 2 message, 10_B indicated a class 3 message (lower location), and 11_B indicated a class 3 message (upper location). Class 3 had two selects as the two messages could be of different maximum lengths.

The Class Three Length Module stored up to two class 3 message lengths following a message ID write to the CAM whose length did not match either class 1 or 2 criteria. Class 3 message lengths could only be overwritten following a class 3 header match or deletion of one of the class 3 messages.

The Length Multiplexer module used the two bit select lines generated in the Match Address Decode module to output the maximum message length. This was passed to the receiver length register in the DMA Registers module and used for determining the end of the DMA transfer to memory.

5.2 FT-SARNIC Module Description

This section gives an overview of the submodules of the FT-SARNIC design and their functions.

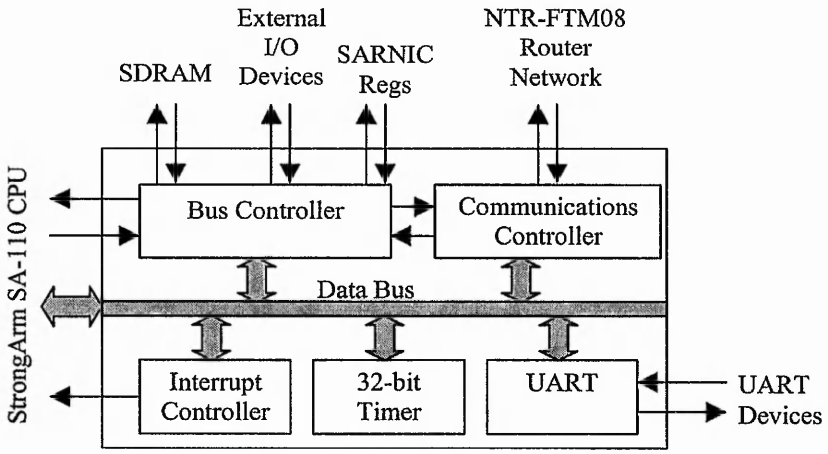


Figure 43: Block diagram of the top-level modules of the FT-SARNIC design

The FT-SARNIC design illustrated in Figure 43 consisted of five main components which are described in the following subsections in more detail.

The FT-SARNIC design built on that of the SARNIC, adding to it the hardware fault detection and recovery features described in section 4.4. The implementation of these was largely confined to the Communications Controller, leaving the remainder of the interface design largely unchanged, except for the removal of the ICR-C416 control port as it was replaced by the ability to transport link status information across the communications links.

5.2.1 Bus Controller

The Bus Controller co-ordinated access to the data bus by the CPU, the addressable memory mapped devices and the FT-SARNICs DMA channels. There were three addressable memory mapped devices each controlled by its own separate module, located within the Bus Controller, as illustrated in Figure 44. The SDRAM controller allowed the addressing of up to 32Mbytes of SDRAM. The External I/O interface allowed up to four devices 256Mbytes of addressable memory locations each. The Internal Registers interface provided access to 256 bytes of the FT-SARNICs user configurable registers. The Arbiter Core was the final component in the Bus Controller. It arbitrated between access requests from the CPU and DMA channels and selected which memory device should be activated. Requests from the CPU always have priority over DMA requests in order to prevent network communications affecting the processors performance.

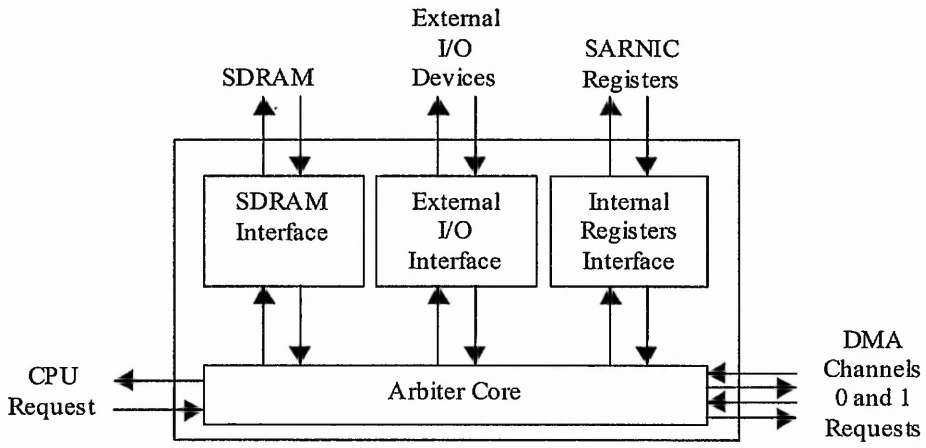


Figure 44: FT-SARNIC Bus Controller submodule block diagram

5.2.1.1 Arbiter Core

The Arbiter Core received requests to access the memory from the StrongArm SA-110 microprocessor and the two DMA channels of the communications controller. Transfer requests were latched, pipelined, queued and serviced in turn. The bus arbiter state machine in Figure 45 demonstrates the order in which they were serviced. When the memory bus was idle, CPU requests were given priority. Bus access alternated between the CPU and either of the two DMA channels when requests were pending. This prevented a single resource monopolising bus access, providing fair access and permitting high DMA data throughput without affecting the CPU performance. The memory address was decoded to select the target memory device: SDRAM, External I/O, the FT-SARNICs internal registers or either of the transmitter DMA channels. The selected memory device interface was activated to commence the bus transaction once several wait states (the number dependant on the memory device activated) had elapsed. These were required to stall the request source until the data bus was ready for the commencement of data transmission.

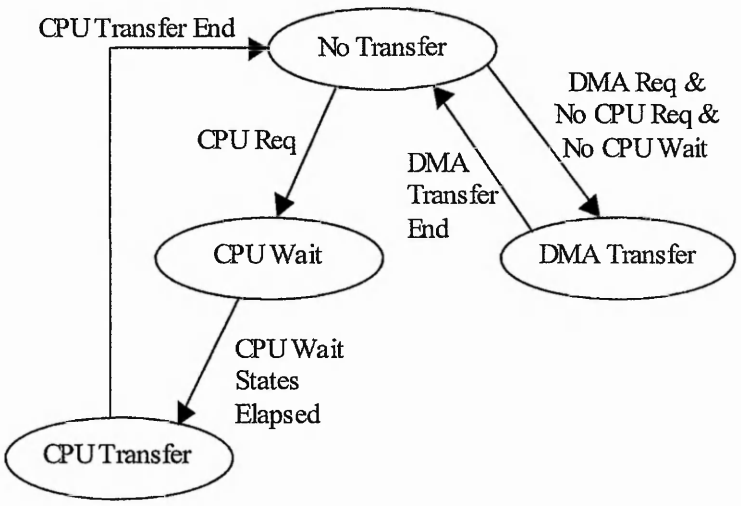


Figure 45: FT-SARNIC Bus Arbiter State Machine Diagram

5.2.1.2 SDRAM Interface

The SDRAM interface was responsible for the generation of the SDRAM start-up sequence, the physical row and column address signals and the SDRAM control signals. The interface could address up to 32Mbytes of memory, having 4 chip select lines, 11 row address lines, 9 column address lines and a bank address line. The memory must be capable of operating at variable speeds. This was because the FT-SARNIC derived its core clock frequency from the user-configurable StrongArm SA-110 processor clock. Allowing user-configurable SDRAM timing parameters tailored the SDRAMs timing to the chosen clock speed, maximising efficiency. Additionally, different SDRAM families and manufacturers might have different timing specifications. The user-addressable SDRAM timing register affords the possibility of optimising the SDRAM timing to a specific SDRAM model. The SDRAM refresh rate was fixed at the standard rate of 4096 cycles every 64ms, as used by most SDRAMs [146, 147]. As the refresh rate was fixed, it was generated from the communication controllers 30MHz sample clock instead of the variable CPU clock.

The four-state SDRAM access state machine, illustrated in Figure 46 determined the amount of cycles spent in each state. The state machine moved from the 'Idle'

state to 'Row Address' upon receiving a request where the row address of the SDRAM was decoded from the memory access address. The column address was decoded in the 'Column Address' state and the appropriate read or write command was generated. SDRAM data writes occurred in the same clock cycle but read operations took several clock cycles before the data was available. Read operations followed a delay defined by the SDRAMs CAS (Column Address Strobe) latency. The final 'Pre-charge' state cleared the SDRAM address lines and prepared the memory for the next access.

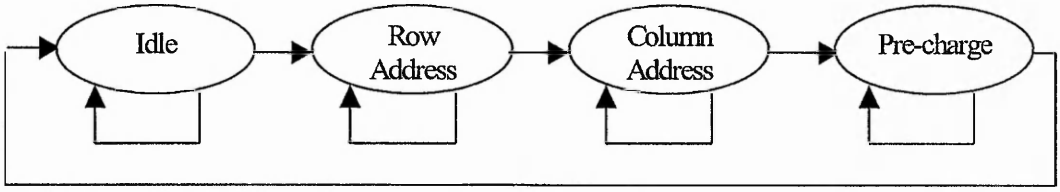


Figure 46: FT-SARNIC SDRAM Interface State Machine

The SDRAM operation could support 2, 4 or 8 word bursts, requiring the column states to be specified only once with the burst commencing from that location. Most SA-110 CPU accesses took the form of 8 word cache line fills suited to the bursts. Shorter accesses, from a single word to four word bursts could also occur.

The advantage of the 8 word cache line fill was the SDRAM interface could tell exactly when the operation would end, as it was the longest allowable access. Unfortunately it exceeded the memory bus access window. Operation of the shorter cache line fill bursts was awkward, as the interface must wait for the de-assertion of the microprocessors memory request signal indicating the end of these transfers. The SDRAM interface avoided this by limiting burst lengths to one word and issuing back-to-back read / write commands [148].

5.2.1.3 External I/O Interface

The External I/O Interface provided control signals to perform read and write operations on up to four low speed peripherals. Each of the I/O devices had up to

256Mbytes of addressable locations with address lines 28 and 29 used as device select lines. The I/O control register provided software configurable timing parameters to determine: the duration of each I/O access, which cycles in each I/O access were read or write cycles and the frequency of the I/O interface control. These parameters allowed the user to tailor the I/O interface signals to the attached device.

5.2.1.4 Internal Registers Interface

This module created the control signals for the FT-SARNIC internal registers, covered in greater depth in previous research [149]. Address lines *a2* to *a7* were decoded to give access for up to 64 different internal word-length registers used in the operation of the FT-SARNIC.

5.2.2 Communications Controller

The largest part of the FT-SARNIC interface was the Communications Controller, responsible for the passage of all messages to and from the interconnection network. The Communications Controller module of the FT-SARNIC housed the fault detection and recovery features of the interface. It represented the majority of the novel design work on the FT-SARNIC that resulted from the research. Unlike the SARNIC interface, which possessed two communications links, the FT-SARNIC had only one bi-directional communications link due to limited available logic resources. The FT-SARNIC, like its non-fault tolerant predecessor, implemented four DMA message channels, allowing two full-duplex, bi-directional virtual channels. A hardware Message Allocator switch routed messages to and from the appropriate DMA channel. Figure 47 illustrates the block diagram of the Communications Controller submodule.

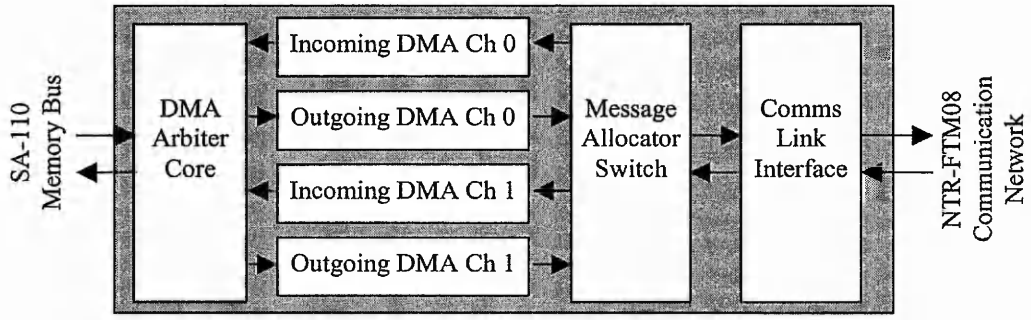


Figure 47: FT-SARNIC Communications Controller Block Diagram

5.2.2.1 DMA Channels

Four DMA channels, two in either direction, supported message passing operations between the Communications Link Interface and the SDRAM via the memory bus of the SA-110 processor. FT-SARNIC memory accesses utilised a cycle stealing mechanism, transferring one data word at a time during periods when the processor was not accessing the memory. As mentioned in section 5.5.1.1, CPU accesses were given priority over DMA accesses to memory. The pipelined DMA Arbiter Core reduced memory bus switching inefficiencies. The two DMA message channels in each direction could be multiplexed onto the Communications Link Interface to allow hardware virtual channel support. This gave more efficient switching and higher bandwidth utilisation as the communications link could switch to a second message immediately after completion of the first.

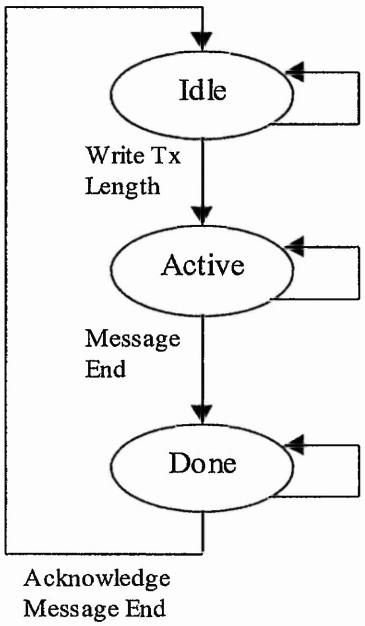


Figure 48: FT-SARNIC DMA Channels State Machine Flow Diagram

All DMA message channels operated on the same basis, with each channel having its own state machine, illustrated in Figure 48. This has with three states; 'Idle', 'Active' and 'Done'. Message channels were 'Idle' until supplied with message channel information (Header, Address and Length parameters). The write to the Length Register must be the last of these three as it was responsible for moving the state machine into the 'Active' state, permitting no further register modifications. When 'Active', a DMA transaction occurred subject to sufficient empty resources at the destination of the transfer, whether it was the memory bus or the communications link. At the end of the message transfer, the message channel terminated and moved to the 'Done' state. No further use could be made of the DMA channel until the CPU had acknowledged completion of the message transfer. This was achieved by returning the state machine to 'Idle' and readying it for another transfer. DMA message channel 0 possessed additional functionality to enable a 'boot from link' option, upon which the incoming data bytes were transferred to the memory area that the SA-110 CPU boots from.

5.2.2.2 DMA Arbiter Core

This module was responsible for arbitrating between the four DMA channels. It also made requests to the bus controller module to gain access to the memory bus. The four channels were sampled every clock cycle when the processor was not accessing memory and any requests were latched. The receiver channels had a higher priority than the transmitter channels to attempt to keep data flowing through the interface, reducing the chances of having to suspend transmission across the link. If more than one-channel requested a transfer, the latched requests were serviced in turn following completion of the highest priority transaction.

5.2.2.3 Message Allocator Switch

The message allocator was responsible for matching the message IDs of messages in DMA channels with those in communications channels, and routing data between them accordingly. Hardware message channel allocation reduced the overheads incurred in virtual message channel support, reducing the need for processor intervention. Assigning both message channels to the communication link effectively implemented hardware virtual channel transmission as both DMA channels multiplexed messages onto the same physical channel. This reduced switching inefficiencies as one message channel could be set up whilst the other was in operation. It had the additional benefit of inserting a short message in between packets of a long message in cases where multi-packet transmission was used.

Hardware virtual channel reception was possible by routing two incoming messages onto the communications link to each of the two DMA receiver channels. Support for more than two incoming virtual channels required software intervention, as one of the DMA channels must be freed in order to service the new message.

5.2.2.4 Communications Links Interface

This section of the design converted outgoing data from 32-bit parallel words to a serial bit stream and vice-versa for incoming data, as illustrated in Figure 49. It was responsible for packetising and depacketising messages and inserting and extracting headers and termination tokens into the data stream. The Link Interface submodule handled the flow of information across the link and the implementation of the hardware fault detection and recovery features. Booting features were implemented in this section of the design, with the option of booting from an internal ROM or from the NTR-FTM08 router network.

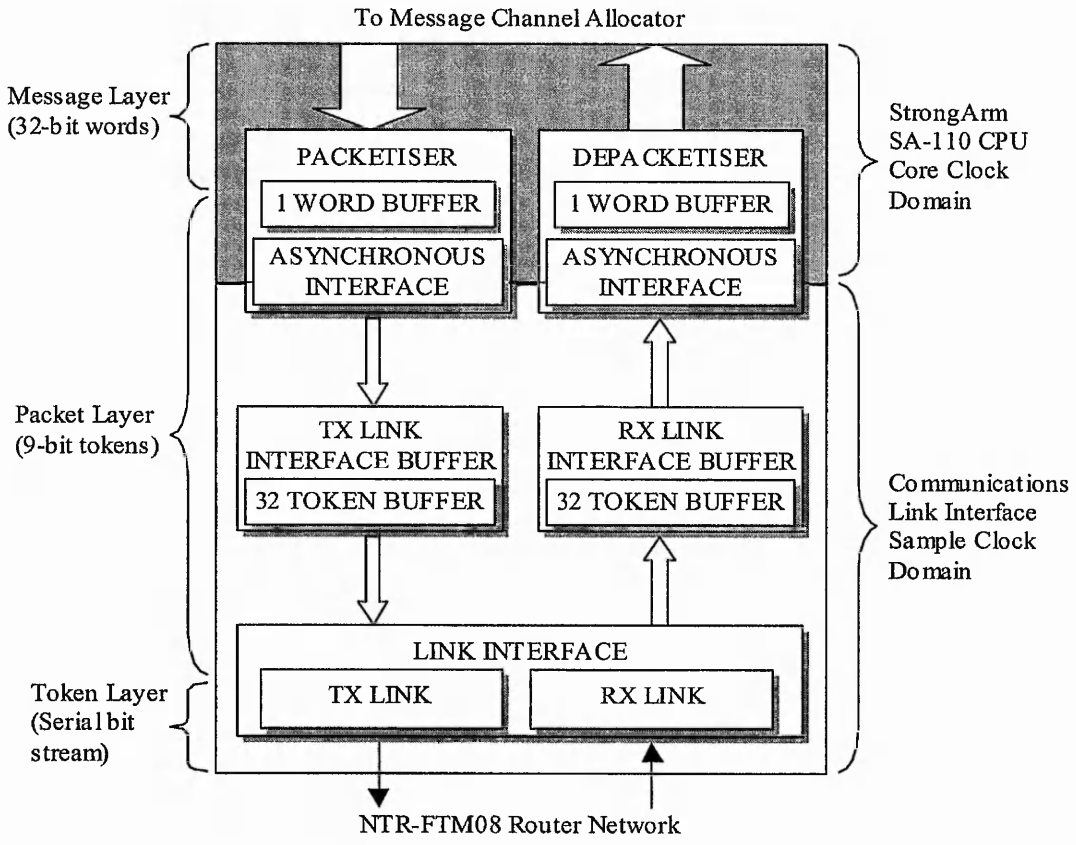


Figure 49: FT-SARNIC Communications Link Interface Module Block Diagram

Transmission Procedure

As data progressed through the communications link it was converted from one format to another to allow for certain operations and procedures to be performed on the data. Data entered the communications link from the DMA channels in the form of 32-bit words, each of which was subsequently split into four data bytes. Before these were outputted from the Packetiser submodule, a header prefixed each message. This could be of user-configurable variable length between 0 and 6 tokens to allow up to five routing IDs and a message ID. The routing header tokens were limited to five due to the small to medium size of target systems. Once the last word of the message had been packetised and outputted from the module, a termination token was appended to the data stream. A ninth 'Type' bit was added to differentiate between header and control tokens, appending a logic zero to termination and control tokens, logic one to all others. The nine-bit parallel tokens crossed the Asynchronous Interface boundary between the core and sample clock domains. The former was supplied by the SA-110 CPU's bus clock whilst the latter was driven by a 30MHz external oscillator. The tokens were stored in the 32 token deep Transmitter Link Interface Buffer, being outputted onto the Link Interface when logic resources were free. Logic one start and logic zero stop bits were added at this stage and the 11-bit token was serialised for transmission onto the router network. Flow control tokens were added at this point.

Reception Procedure

Serial data was retrieved from the router network and converted into an 11-bit parallel token. The start and stop bits were removed, leaving a 9-bit token whose type bit was tested to determine the presence or otherwise of flow control tokens. Flow control tokens were removed and acted upon. The remaining tokens were header, data or termination tokens, and were transferred to the Receiver Link Interface Buffer. The tokens were passed to the depacketiser when there was sufficient space in the one word deep DMA buffer. Headers were removed in the depacketiser and were identified by their position in the incoming data stream. They were checked with the message IDs held in the Message Allocator Switch to see if they had been allocated a

DMA channel. Termination tokens were identified by their logic zero type bits and were removed and the message terminated.

5.2.2.5 Link Interface

In addition to the procedures described above, the link interface determined which state the link was in, as detailed in section 4.4.5 and regulated the link initialisation and connection request procedure. Section 5.3.1 described the operation of the link activity verification procedure implemented in the FT-SARNIC, being the same as that of the FT-PCI-OSLi.

5.2.2.6 Link Interface Buffering

The transmitter link interface buffer must be large enough to ensure that a data drought would not occur between successive DMA transactions. The following equation [17] gave the worst case delay between successive DMA transactions to be the same amount of time taken to transfer 3.15 tokens across the communications link. This value was rounded up to the next token, giving a four token deep transmitter link interface buffer.

$$T_{MAX} = \frac{(4 \times (4 + \text{DMASize}))}{\text{Mem Bus Clk}} + \frac{(4 \times (\text{Byte Length}))}{\text{Mem Bus Clk}} \times \frac{\text{Data Rate}}{\text{Token Length}}$$

Maximum time between DMA operations, in terms of link byte time

The equation uses the following parameters;

- Mem Bus Clk (Memory Bus Clock, or SA-110 CPU clock) = 36.9MHz
- DMA Size – 1 word per transfer
- Token Length – 11 bits, not 13 as was the case in the SARNIC

- Data Rate – 20Mbits/sec being 2/3 of the 30MHz oscillator frequency

The receiver link interface buffer was 32 tokens deep, 9 bits wide and operated in a First-In-First-Out (FIFO) manner. Full and empty signals left the buffers to notify other modules of data saturation and starvation respectively. The receiver buffer created two additional level pointers, termed 'Almost-Full' and 'Almost-Empty'. These were set to trigger when the number of tokens in the buffer was in excess of 24 and less than 8 respectively. In the FT-SARNIC interface, the buffer was implemented in the embedded array blocks (EAB) of the PLD to reduce logic requirements. Due to the internal architecture of the Flex 10KE and the MaxPlus2 version 10.0 synthesis software, the buffer occupied two EABs, in a 9 x 16 configuration. This was an inefficient use of the memory resources as only 144 bits of a possible 2048 were utilised in each EAB. This was an additional incentive to upgrade to the Apex 20KE and Quartus technology utilised in the implementation of the FT-PCI-OSLi.

5.2.2.7 Packetiser

The Packetiser formatted data ready for transmission as tokens by splitting 32-bit words into bytes and adding a ninth (ID) bit and header and termination tokens. The header arrived from the 'DMA Transmitter Header' register [149], of which there were two (upper and lower). This allowed for a header up to six bytes long to be used whose length was determined by the user by writing to the most significant three bits in the upper register. Unlike the ICR-C416 protocol, the most significant bit of the header did not determine the presence of the last header byte. The packet length was not limited to 256 bytes, being user configurable at any value up to the 64kbytes message length.

The Packetiser state machine had four states, demonstrated in Figure 50. It moved from 'Idle' to 'Header Enable' upon activation of a DMA message channel or moved straight to the 'Data Request' state if the header length was set to zero bytes (an option which might favour certain applications). Once all header tokens had been transferred to the link interface buffer, the 'Data Request' state was entered. This

advanced to 'Data Enable' upon receiving a data word from the DMA channel. At the end of each word the state machine returned to 'Data Request' to load another data word. It returned to 'Idle' and loaded the termination token into the link interface buffer after the packetisation of the last data word.

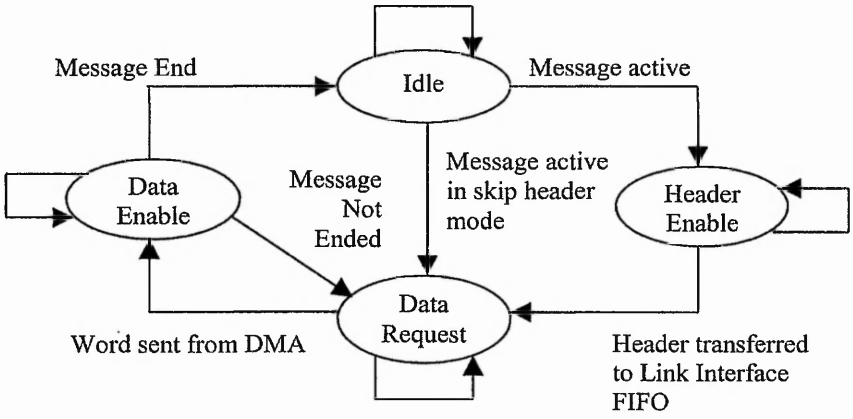


Figure 50: FT-SARNIC Packetiser State Machine Diagram

5.2.2.8 Depacketiser

The FT-SARNICs Depacketiser requested DMA transactions following receipt of an incoming message. The four-state Depacketiser state machine, illustrated in Figure 51, defaulted to the 'Header Enable' state. The first bytes to enter the submodule were labelled as header bytes by their position at the start of the message. Once the expected number of header bytes was read from the FIFO and loaded into the receiver header register the state machine advanced to the 'Header Ready' state. In this state the received message ID was compared with the message IDs in the Message Allocator Switch. On finding a match, the Depacketiser received a header acknowledge signal indicating that there was a DMA message channel ready to receive the data. This moved the state machine into the 'Data Enable' state in which data tokens were outputted from the Receiver Link Interface Buffer. The ID bit was tested to identify termination tokens then stripped. The remaining data byte formed part of a 32-bit word transferred to the DMA channel, freeing the one word deep DMA buffer for the next word. After reading four tokens from the FIFO the state

machine advanced to the 'Data Ready' state and transferred the data word to the DMA channel.

A successful DMA transfer freed up resources for another transfer, returning the state machine to the 'Data Enable' state to assemble another data word. The state machine moved from 'Data Ready' to 'Header Enable' when a termination token was detected or when flushing the remainder of the message following a fault. As with the FT-PCI-OSLi's 'Receiver Message Controller' module, three modes of termination existed: Early, Normal and Late, as discussed in section 5.3.4. Both the packetiser and depacketiser modules each contained a one word DMA buffer. This was the first stage of the link buffer pipeline and was used to hold the data word during assembly and deassembly.

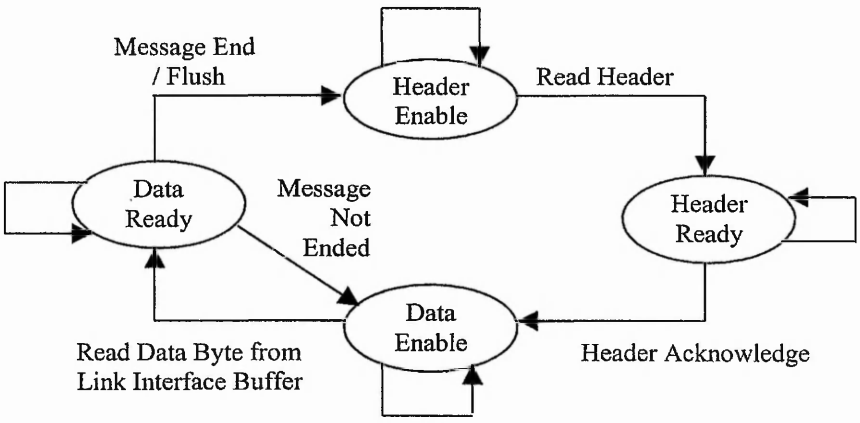


Figure 51: FT-SARNIC Depacketiser State Machine Diagram

5.2.3 Interrupt Controller

This module generated interrupt requests for the SA-110 CPU from sources such as the DMA message channels, timer and four additional external interrupts for the external peripherals addressed by the External I/O Controller. Two levels of SA-110 interrupts were handled. FIQ and IRQ were the fast and standard priority interrupts respectively. Interrupt sources could activate either interrupt dependent on the

interrupt register status. Enabling one interrupt type disabled the other type for that source.

Each set of interrupt registers had an Internal Enable Register (IER), write only Interrupt Enable Set Register (IESR), write-only Interrupt Enable Clear Register (IECR) and read-only Interrupt Status Register (ISR) associated with it. The FIQ and IRQ lines shared an Interrupt Raw Status Register (IRSR) for each source.

5.2.4 Timer

The timer function was carried over from the SARNIC interface to give the debugger and software developer additional support. A 32-bit wraparound counter incremented every 1μs, giving a 1MHz real-time timer, used in a similar way to the OCCAM timer function [24]. A special relation operator function determined whether or not a timer value was supplied before or after the current timer value, as shown in Figure 52. The most significant bit determined whether or not the timer event occurred in the past or the future as a comparison result of logic 1 for this bit indicated a negative number. Timing events separated by up to 35 minutes 47 seconds could be handled. The heartbeat and checkpulse signals, used to verify the presence or otherwise of link activity, were generated in this module.

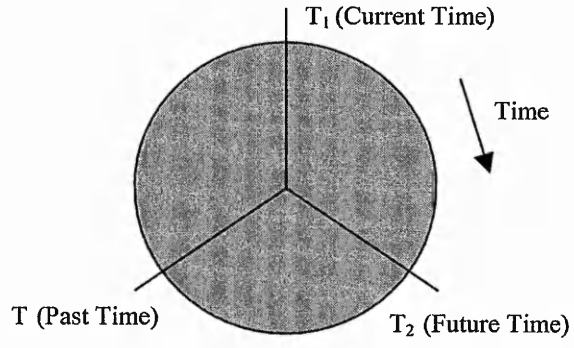


Figure 52: FT-SARNIC Timer with Past and Future Time References

5.2.5 UART Communication Port

A Universal Asynchronous Receiver / Transmitter interface was implemented in the FT-SARNIC design, carried over from the non-fault tolerant SARNIC interface. It provided a low speed serial link to a PC, via the COM port, allowing the display of information on a monitor and input from a keyboard, assisting debugging. The UART functionality was reduced to minimise resource usage, omitting error detection mechanisms and buffering. The UART operates at a fixed baud rate of either 9600, 19200, 38400 or 57600 bps.

6 RESULTS

6.1 FT-PCI-OSLi Hardware Test Results

This section of the thesis reports on the hardware tests performed on the FT-PCI-OSLi interface. Similar tests were undertaken using the PCI-OSLi interface, implemented on an identical APEX 20K200EQC240-1 to aid comparison between the two designs. The test results were also compared to the theoretical performance characteristics of the FT-PCI-OSLi.

The FT-PCI-OSLi interface was implemented on a custom made, multi-layer, PCB slotted into a PCs empty PCI slot. A 'loopback' test method was utilised, as shown in Figure 53. This form of test involved fetching the data from the PCs memory and processing it ready for transmission onto the serial communications network. The message was transmitted through the differential driver circuitry and across a short length of cable (approximately 5cm long) before returning back to the interface board and the FT-PCI-OSLi via the differential transceivers. The FT-PCI-OSLi reformatted the data, assembling it into 32-bit data words and accumulated sufficient words to initiate a PCI transaction, which passed the message to the PCs memory.

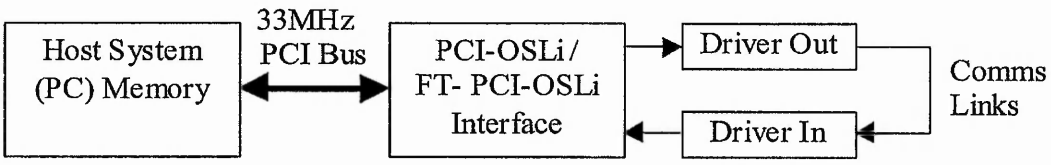


Figure 53: Loopback Test Block Diagram for FT-PCI-OSLi

This method of hardware testing assessed the bi-directional communication capabilities of the interface, requiring the interleaving of transmission and reception PCI bus block transfers between all other transactions on the PCI bus. The PCI bus operated at 33MHz for all tests. The link interface clock, derived from an external crystal oscillator was altered between tests from 30MHz to 40, 48 and 64MHz to give link data rates of 20, 26.67, 32 and 42Mbits/s respectively. Only the results of tests performed at the maximum data rate (42Mbits/s) are detailed in this section.

Performing similar tests on the FT-PCI-OSLi and PCI-OSLi designs on identical hardware allowed many differences regarding implementation to be eliminated and allowed a focus on the comparison of the two interface designs and respective protocols. Neither interface design was optimised in any way for implementation to the target technology. Both PLDs used identical pin-outs and synthesis parameters with respect to meeting I/O timing requirements and speed / area trade off.

An advantage of the loopback tests was that the FT-PCI-OSLi / PCI-OSLi interface board was the only hardware required, other than a PC with a 33MHz PCI bus. The interface board was fitted with differential transceiver drivers for all input and output communications channels. Data transmission behaviour over these channels has already been investigated and documented [123]. An advantage of testing the interfaces was, being end nodes, they were not subject to routing considerations encountered by the NTR-FTM08 (such as the effects of network topologies, message addressing methods and adaptive routing algorithms) and were not affected by deadlock. The interface outputted data when the recipient was ready for it and accepted data subject to sufficient buffer space.

6.1.1 Hardware Test Parameters and Criteria

The loopback tests used the PCIWave software, introduced in section 5.2.6, to allow the user to load the DMA submodule necessary to initiate message transfer. Due to the lack of software drivers for the FT-PCI-OSLi, the only way in which a message could be transferred at this stage of development was manually via user initiation. Several counters in the DMA Registers module were set up in order to monitor the time taken to perform certain parts of the message transfer. Several criteria were specified in order to denote key stages in the processing of the message:

- Message transmission was said to have started following a write of the message length to the Transmitter Message Length Register (offset $0D_H$). This action was also deemed to have started the DMA transmission as the request for PCI bus ownership was initiated by this command.

- Data transmission began after the transition on the serial communications link of the first bit of the header token. It ended following the transmission of the stop bit of the last data token of the outbound message.
- Data reception began once the first start bit of the header token arrived on the serial communications input line. Reception ended when the last data byte was transferred from the link interface buffer to the depacketiser for assembling into a 32-bit word.
- Message reception began following the transfer of the first data byte from the link interface buffer to the receiver DMA buffer via the depacketiser and ended when the message had been transferred to memory in its entirety.

The hardware tests operated on the principle that:

- Message tokens were transmitted back-to-back, with interleaved acknowledge tokens in the case of the PCI-OSLi. Subject to sufficient space being available in the receiver link interface buffer, the receiver was always ready to receive data, allowing the transmitter to send a continuous stream of message tokens.
- Each message had a single byte header and a byte length termination or length byte in the case of the FT-PCI-OSLi and PCI-OSLi respectively. No routing header bytes were sent.
- The FT-PCI-OSLi message was sent as a continuous transmission whereas the PCI-OSLi split its message into 256 token packets, each prefixed by the two bytes mentioned above. As there was only one message, subsequent packets utilised the message channel set up from the transfer of the previous packet, eliminating the need to assign a message channel more than once for the message duration. Message headers must be matched for message channel allocation to occur.

6.2 PCI Access Efficiency

The calculations in section 4.3.1 provide a means of measuring the performance of a PCI access burst by comparing the latency incurred in initiating a PCI transfer with the amount of data transferred. The PCI access procedure begins with the initiating device requesting ownership of the PCI bus by asserting the active low 'nREQ' signal (see Appendix B). Assertion of the active low 'nGNT' signals the granting of bus

ownership to one of the devices attached to the PCI bus. The decoded states of signals on the 'AD' and 'nC/BE' buses indicates which of the attached PCI devices has been granted ownership of the bus.

The initiator indicates that it is ready to begin data transfer by asserting the active low Initiator Ready 'nIRDY' signal. Between the assertion of the 'nGNT' and 'nIRDY' signals, the PCI access is in operation but no data is transferred across the bus, hence this is the transfer initiation latency. The transfer of data across the PCI bus begins when both the initiator and target are ready to transfer data, indicated by the assertion of 'nIRDY' and 'nTRDY' respectively. A 32-bit double-word is transferred across the bus every PCI clock cycle until the transaction is suspended. This occurs when either the 'nIRDY' or 'nTRDY' signals are de-asserted, after a time period specified by the latency counter of the PCI agents. The latency counter was introduced in section 5.2.1.1 and was initially set to 64. An explanation of the functions of these, and all other PCI signals can be found in Appendix B.

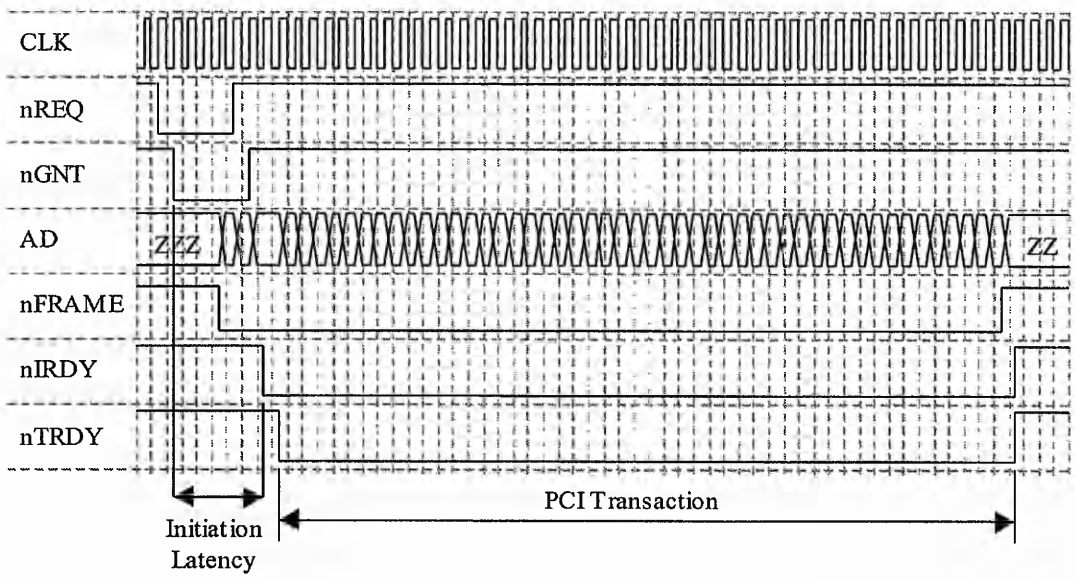


Figure 54: An example of waveforms demonstrating PCI transaction initiation latency

Figure 54 demonstrates a sample of some of the waveforms during a typical PCI memory-read operation. The user counters in the DMA registers submodule of the interface were configured to measure the initiation latency and PCI transaction length as shown in the above diagram, in terms of PCI clock cycles. The latency incurred in

setting up the transaction, following ownership of the bus being granted, lasted for 3 and 5 PCI clock cycles for the FT-PCI-OSLi and the PCI-OSLi interfaces respectively.

Figure 55 demonstrates the efficiency of PCI accesses, in terms of the amount of time that data transfers occur as a percentage of the total transaction time. Messages with payloads under 228 bytes could be transferred in a single PCI transaction using the FT-PCI-OSLi and PCI-OSLi interfaces. Messages longer than this value required multiple transfers. The test measured the duration of the longest PCI access in the case of multiple accesses. As these values were less than the total DMA buffer capacity of 64 double words (256 bytes), it was the initial value of the latency counter mentioned in section 5.2.1.1 that limited the length of the PCI transactions, instead of the DMA buffer capacity. The latency counter was initialised following a request to acquire bus ownership. It included the time taken to gain bus ownership (including access arbitration, if necessary), set up the transfer and transfer the data, hence, less than 64 double-words were transferred.

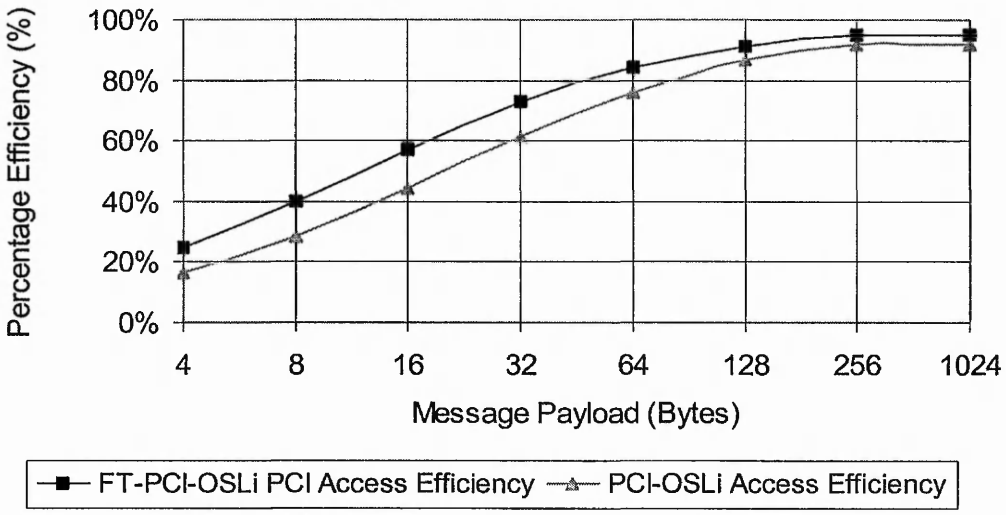


Figure 55: Efficiency of PCI bus accesses for the FT-PCI-OSLi and PCI-OSLi devices in terms of latency as a percentage of overall PCI transaction duration

As can be seen from Figure 55, the efficiency was lowest for small transfers. When only a single double word was transferred (message length 4 bytes), the

initiation latency was three and five times greater than the actual data transfer time for the FT-PCI-OSLi and PCI-OSLi interfaces respectively.

Once the payload exceeds the level required to fill the transmitter DMA buffer, the efficiency characteristic remains constant. The maximum efficiency is reached when the buffer is filled in its entirety and the remainder of the message must be loaded in subsequent bursts, the frequency of which depends on the rate at which the buffer can be emptied. Maximum efficiency, in terms of PCI burst length is reached for payloads above the transmitter DMA buffer capacity and remains constant as payload increases. For this reason, the payload graph does not exceed the level at which this occurs in order to emphasise the increase at lower message lengths.

6.3 Bi-directional Data Transfer Tests

6.3.1 Bi-directional Data Transfer Duration

Figure 56 shows the message duration for messages of varying payloads transmitted and received by the FT-PCI-OSLi and the PCI-OSLi. At a message size of 64kbytes, the PCI-OSLi took 46.6% longer to complete its message transfer than the FT-PCI-OSLi, due to the limitations of the credit based flow control mechanism. The increase in message duration is linear, although at different rates of increase due to the different flow control mechanisms. Message lengths up to 64kBytes are observed, as this was the maximum allowable message length in the SARNet system. The characteristic would continue for longer messages, up to the maximum allowable message length of 1Mbyte in the FT-PCI-OSLi.

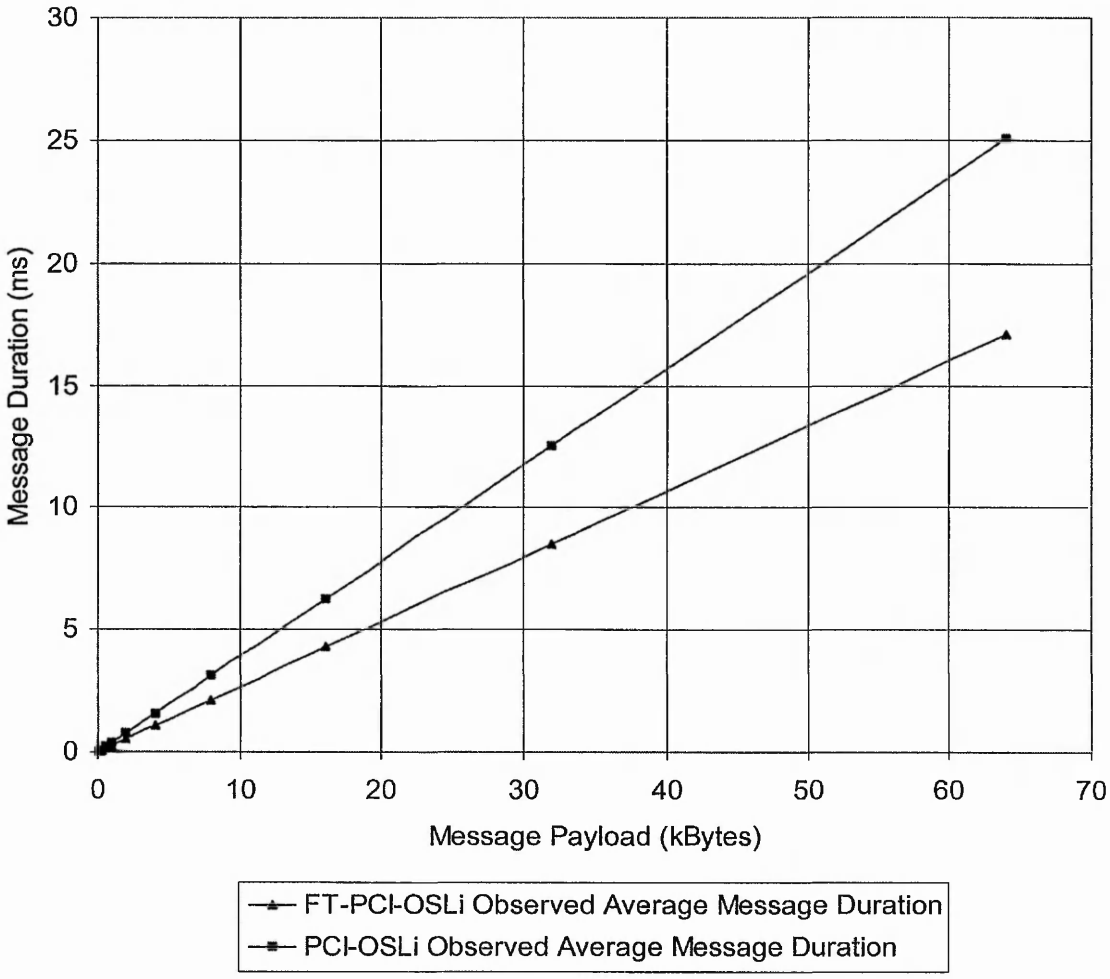


Figure 56: Message duration results for FT-PCI-OSLi hardware tests at 42Mbits/sec link rate

Figure 57 is more informative in terms of assessing the FT-PCI-OSLi performance as it shows more clearly the differences in message duration for payloads under 256 bytes compared to the theoretical message duration of the FT-PCI-OSLi. Increases in message size resulted in linear increases in message duration with the gradient of the PCI-OSLi characteristic increasing at a faster rate than that of the FT-PCI-OSLi, due to the back-to-back data transfer capability of the latter device. The message duration for the FT-PCI-OSLi for payloads of four bytes was almost double the theoretical maximum due to the initiation latency. The FT-PCI-OSLi took three PCI clock cycles to set up the PCI transaction following the granting of bus ownership, but required a single PCI clock cycle to transfer the message across the bus which as Figure 55 showed, gave a PCI access efficiency of 25%.

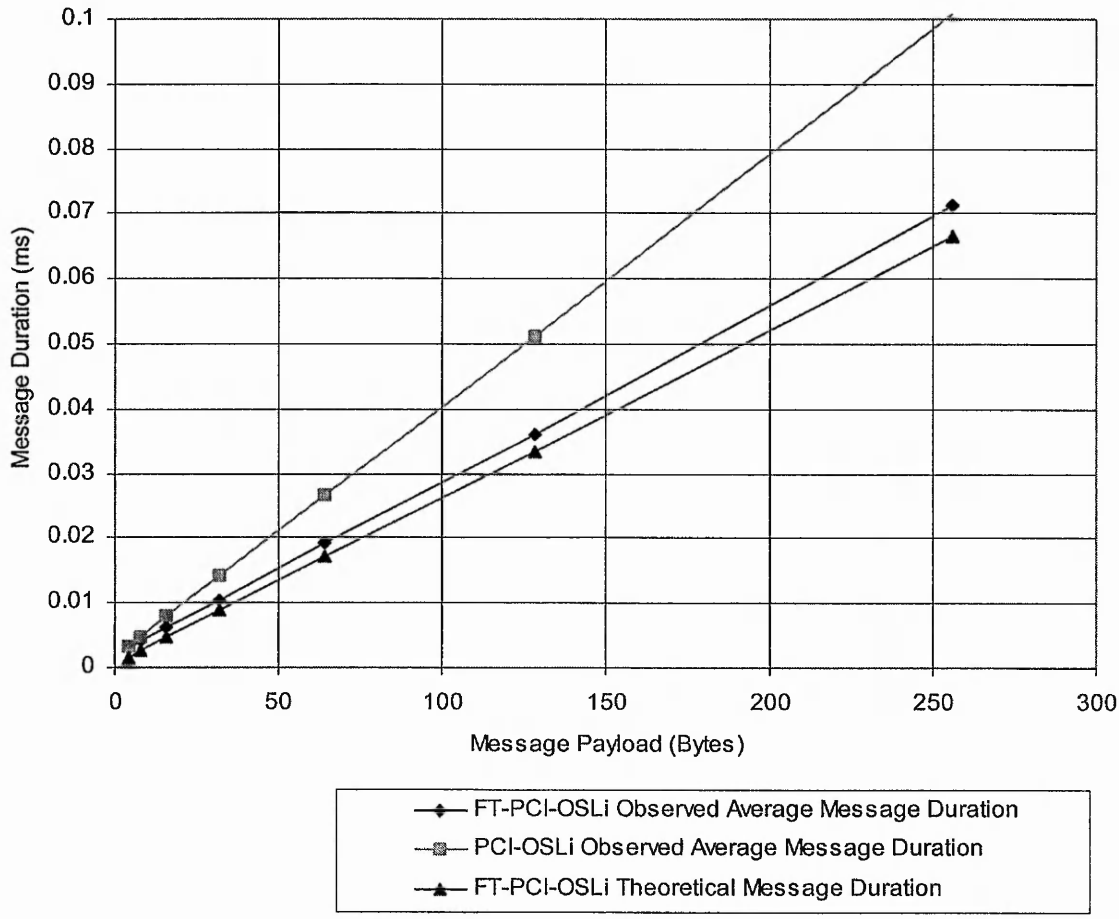


Figure 57: Message duration results for FT-PCI-OSLi hardware tests at lower message lengths at 42Mbits/sec link rate

The difference between the theoretical (meaning 'simplified model', in this case) and observed characteristics was due to the theoretical (simplified model) maximum transfer duration only taking into account the length of time taken to transmit the message across the transmission medium. It ignored factors such as message formatting delays incurred as the message passed through the FT-PCI-OSLi, although the latency incurred in initiating the PCI transaction, (both to transmit and receive the data), was taken into account. As message length increased, the FT-PCI-OSLi message duration increased at an almost identical rate to the theoretical (simplified model) message duration, suggesting that the discrepancy between the two was due to initial start-up latencies that became insignificant as the message size increased. The slight divergence between the two is due to delays such as those incurred by the

transceiver circuitry. The linear increases in message length suggest that the bulk of the message duration was due to the transmission of the serial data across the communications link. Although the link interface circuitry was driven at almost twice the rate as the rest of the FT-PCI-OSLi interface, the serial nature of the communications link made this the bottleneck.

Figure 58 shows the normalised message duration times for the FT-PCI-OSLi and PCI-OSLi interfaces with respect to the theoretical (simplified model) message durations for these devices respectively. In calculating the theoretical minimum message duration it is assumed that the message transit time between the link interface serial output and input is negligible. As both designs used identical transceiver circuitry, there will be a slight increase in the message duration due to the passage of the signal through the driver circuits, relative to the theoretical values.

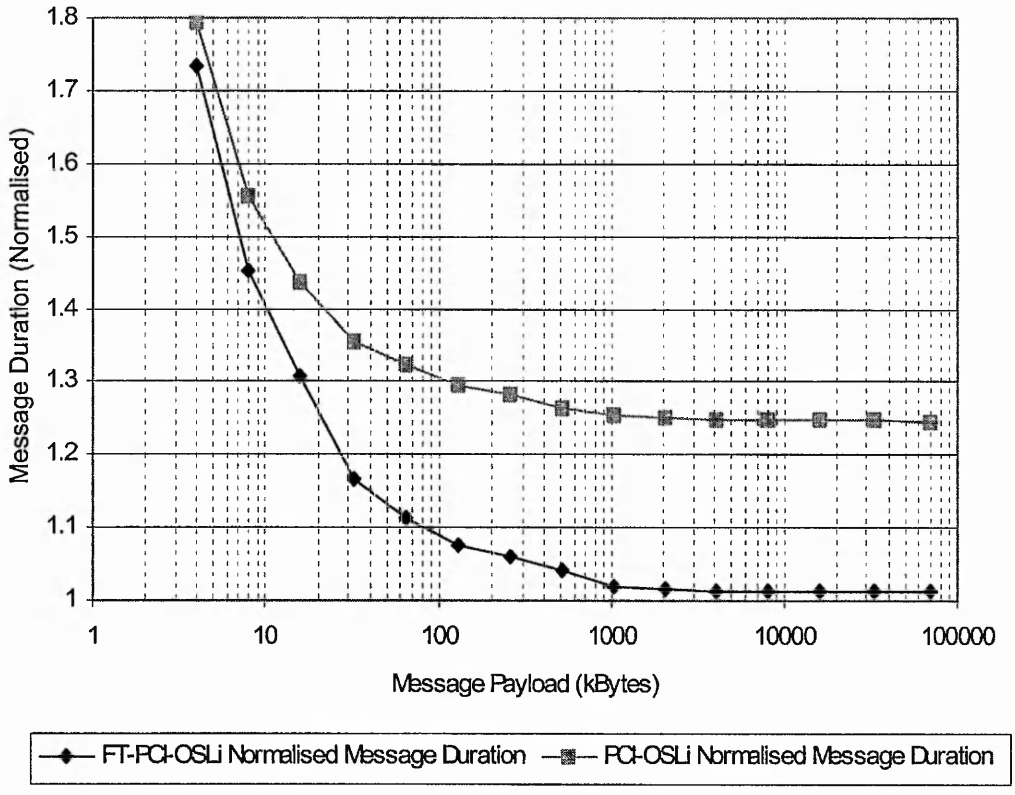


Figure 58: Normalised message duration for the FT-PCI-OSLi and PCI-OSLi interfaces at 42 Mbits/s data rates

The theoretical calculations used here take into account the PCI latency by adding two sets of PCI initiation latency (set at 3 and 5 PCI clock cycles duration for the FT-PCI-OSLi and the PCI-OSLi characteristics respectively) to the theoretical message duration times.

At the minimum payload length of four bytes, the FT-PCI-OSLi took 1.73 times longer than the theoretical minimum message duration. This is due to message overheads and the passage of the message through the FT-PCI-OSLi circuitry being large in comparison to the time taken to transmit the message. The same transfer on the PCI-OSLi takes 1.79 times the theoretical minimum message length. This figure includes the time taken to send an acknowledge token in receipt of the last data byte as the message begins the process of transferring the data to memory on arrival of this data byte. Calculation of the theoretical minimum message time for the PCI-OSLi also included the transmission of acknowledge tokens, except for the last, for the reason above. It was assumed in these calculations that acknowledge tokens were interleaved between data tokens and that back-to-back transmission occurred.

As message sizes increased, the ratio between the observed message duration and the theoretical minimum message duration decreased rapidly as parameters not taken into account in the theoretical calculations became less significant compared to the total message duration. When the message size exceeded 4kbytes, both performance characteristics leveled out with the FT-PCI-OSLi taking under 2% longer to complete message transfer than the minimum theoretical duration. The PCI-OSLi took approximately 25% longer than the minimum theoretical duration. This loss of bandwidth was most likely due to the need to have received an acknowledge token before transmission of the next data token could begin, in addition to the increased overheads incurred through the PCI-OSLi due to its less efficient PCI interface. It has already been documented that the credit based flow control mechanism, utilised by the PCI-OSLi, can occupy up to 17 bits per token [5], four tokens more than the minimum 13 used to send a data token and acknowledge token, giving an increase of 30%.

The normalised message duration characteristic for the FT-PCI-OSLi and PCI-OSLi interfaces demonstrates the efficacy of permission based flow control with respect to credit based flow control.

6.3.2 Bi-directional Data Bandwidth Utilisation

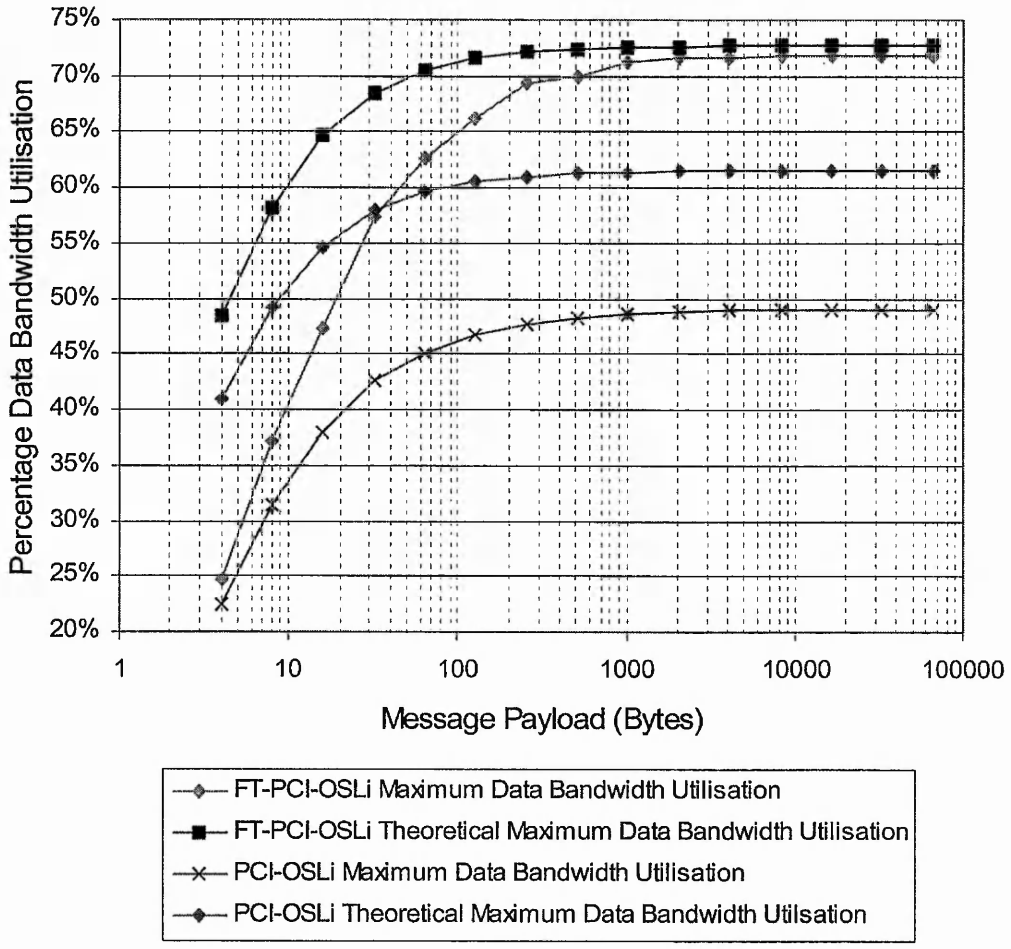


Figure 59: FT-PCI-OSLi and PCI-OSLi percentage data bandwidth utilisation at 42Mbits/s data rate

The percentage of the communications link maximum message bandwidth that was devoted to transferring data is displayed in Figure 59. The theoretical percentage data bandwidth utilisation characteristic for the FT-PCI-OSLi and PCI-OSLi are displayed alongside the results recorded from the hardware tests. The theoretical characteristics take into account the inclusion of non-data tokens in the message, such as header, length, termination and acknowledge tokens. This aberration between the theoretical and observed measurements was due to inefficiencies in message

transmission across the communications link and the overheads incurred in initiating message transmission and reception.

Minimum message sizes of four bytes utilised approximately 50% of the available data bandwidth in both devices making, such transfers inefficient in terms of the amount of data transferred across the network as a proportion of the total number of message bits sent. The FT-PCI-OSLi characteristic rose at a steeper gradient than the PCI-OSLi curve because of the latter incurring two extra non-data bits per token due to the presence of acknowledge tokens. All four characteristics plateau once the message size exceeded 1kByte. The difference between the FT-PCI-OSLi's theoretical and observed data bandwidth utilisation was 0.9% whereas the difference between these values for the PCI-OSLi was 12.54%. Whilst the former value was largely due to the overheads incurred in setting up the message transfer, the latter was due to the bandwidth lost in failing to achieve the 13 bits-per-token throughput. The credit based flow control utilised by the PCI-OSLi prevents the data throughput of the device from reaching that of the FT-PCI-OSLi.

6.3.3 DMA Message Transmission

Figure 60 demonstrates the observed throughput characteristic of data transferred from memory, across the PCI bus to the DMA Transmitter Buffer, with respect to the message size. The DMA transmission was said to have started following the write of the transmitter message length. This action asserts the active low 'nREQ' signal (see Appendix B), which initiates the request for ownership of the PCI bus. The latency count, set to 64 in this case, decrements towards zero following this action. The actual DMA transfer began following the assertion of the 'nGNT' (see Appendix B) signal granting ownership to the PCI bus but starting the transfer from the message initiation meant that the overhead involved in acquiring the PCI bus was also taken into account. The DMA transmission ends when the last double word of data was transferred into the DMA buffer.

The DMA transmission throughput determines the frequency of PCI data access bursts initiated by the FT-PCI-OSLi interface when transferring data to the PCI interface. This is important as the PCI bus is a shared resource.

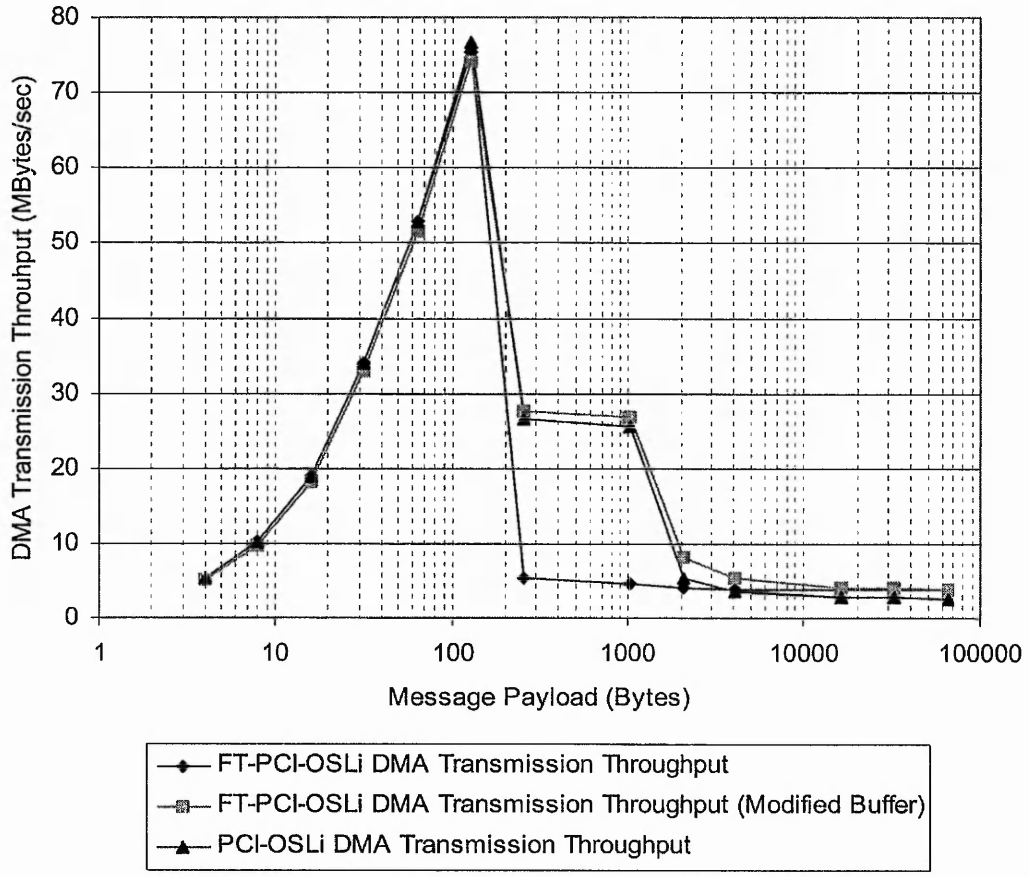


Figure 60: DMA transmission throughput for the FT-PCI-OSLi and PCI-OSLi interface devices including FT-PCI-OSLi with modified buffer capacities

The DMA transmission throughput characteristic shown in Figure 61 can be split into three parts. Area 1 is the part of the graph where the throughput is determined by the capacity of the transmitter DMA buffer. The characteristic of area 2 is governed by the ability of the link interface buffer to store data prior to transmission onto the serial communications link (The more data that can be stored in the link interface buffer, the less frequently PCI access bursts will need to be made). In area 3 the DMA and link interface buffers are both full so the serial communications link is the performance bottleneck and the data rate and data bandwidth utilisation will be the

limiting factors. As all buffering is full, the rate at which data can be outputted onto the communications link will govern the rate at which the buffers can empty sufficiently to initiate another PCI transfer.

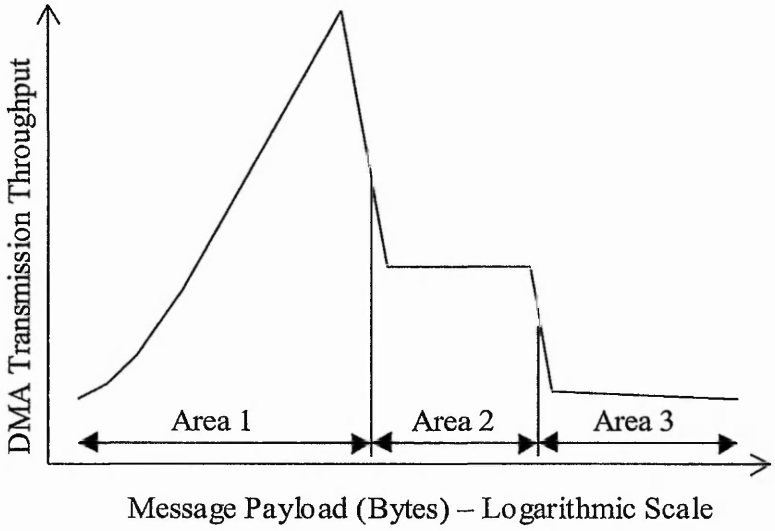


Figure 61: FT-PCI-OSLi (with modified link interface buffer capacity) DMA transmission throughput graph split into three areas

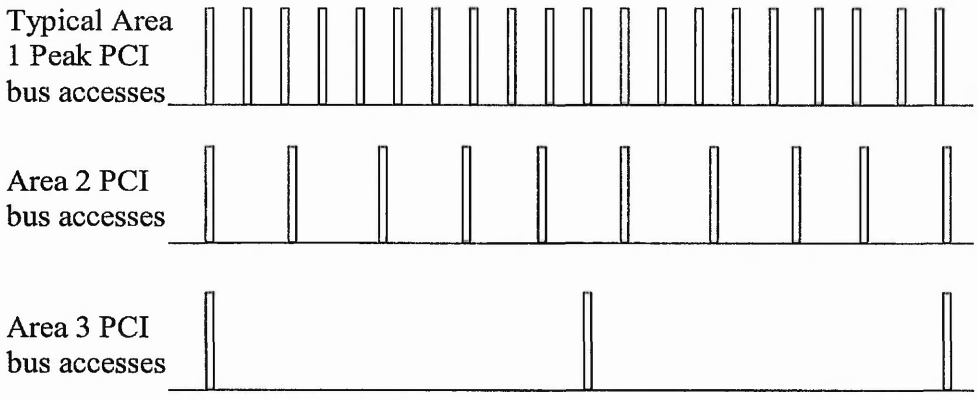


Figure 62: Diagram displaying frequency of PCI accesses in each of the three areas of the DMA transmission throughput characteristic

The boundary between areas 1 and 2 is determined by the DMA buffer capacity and the boundary between areas 2 and 3 is determined by the capacity of the link interface buffer. The DMA buffer capacity is effectively altered by modifying the latency count of the device, limiting the number of data bursts per PCI transaction.

Figure 62 demonstrates the relative frequency of the PCI bus accesses in each of the three areas in order to maintain the maximum possible data throughput over the serial communications link. The relevance to this is due to the shared nature of the PCI bus and the need for other users of the bus to be able to access it. Short messages that can be transferred across the bus in a single burst have the potential to monopolise bus access. Hence it is important that the PCI bus arbitration mechanism prevents this occurring.

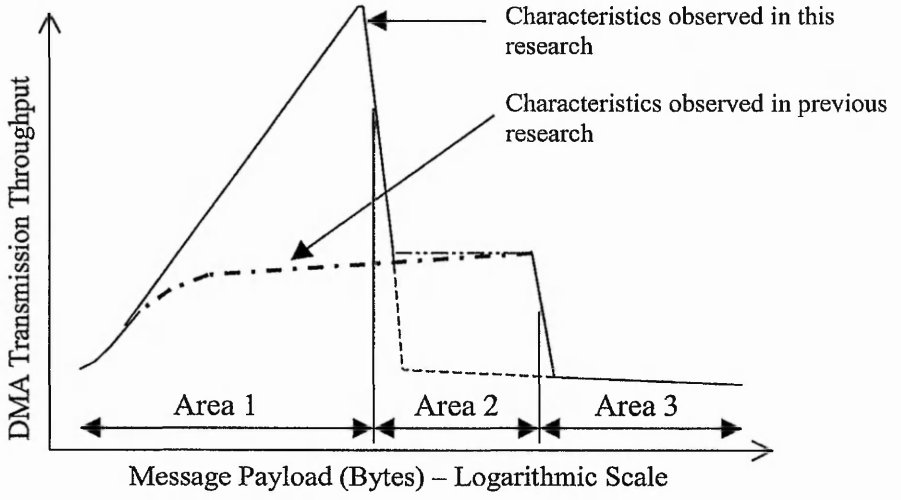


Figure 63: DMA transmission throughput characteristics diagram compared with characteristics made in previous research during development of the PCI-OSLi

A comparison of PCI-OSLi DMA transmission throughput characteristics in this and previous research are shown in Figure 63. These show earlier research with noticeably different characteristics to those observed in Figure 61. Previously there was no peak in area 1 of the graph and the DMA throughput characteristic ramped upwards to the plateau of area 2, giving an apparent maximum DMA throughput of approximately 25MBytes/s, much lower than the maximum 92.8MBytes/s noted during this research (see Figure 64). This was due to the PCI-OSLi interface development tests declaring the end of the DMA transmission process to be the

passing of the last data byte from the link interface buffer to the link interface. Measuring the end of the DMA transmission at this point took into account the packetiser outputting data at a quarter of the rate it entered and missed the peak observed in area 1, showing instead a much more conservative throughput figure. This did not give a true measurement of DMA throughput and consequently, tests undertaken in this research for both PCI-OSLi and FT-PCI-OSLi terminated DMA transmission when the last data byte entered the DMA transmitter buffer.

Figure 60 also displays a third characteristic: the performance of the FT-PCI-OSLi when the transmitter and receiver link interface buffer sizes were altered from the 32 token depth specified in section 4.4.2.3 to the respective 1kByte and 2kByte depths used in the PCI-OSLi interface. This 'modified buffer' FT-PCI-OSLi design was subjected to all tests used to assess the FT-PCI-OSLi performance but the characteristics for most tests were very similar to that of the FT-PCI-OSLi interface. Where the results matched those of the FT-PCI-OSLi, they were omitted from the result graphs in this chapter for the sake of clarity.

In area 1 of Figure 60, the three interfaces exhibited a similar performance characteristic due to the similar nature of PCI bus access between the FT-PCI-OSLi and PCI-OSLi. The PCI transmission throughput was 5.28MBytes/s for messages four bytes long, rising sharply to a peak around message lengths of 128 bytes. The PCI throughput at the peak was 92.84MBytes/s for the FT-PCI-OSLi and 88.93MBytes/s for the FT-PCI-OSLi interface with the increased link interface buffering and 90.86MBytes/s for the PCI-OSLi. The differences in throughput are relatively small with a maximum variation of 3.5% between these characteristics. This variation was caused by a difference in the DMA transmission counter values of one between each of the three values. The DMA transmission counter value measured the time taken between the start and end of the message transfer across the PCI bus. The value used was the average taken after five repetitions of each test. The counter was synchronised from the 33MHz PCI bus clock and a single bit difference made a difference in the DMA transmission throughput. Figure 60 demonstrated that a 4-byte message took 6.25 PCI clock cycles to be transferred to the FT-PCI-OSLi DMA transmitter buffer whereas a 128-byte message took 56 PCI clock cycles.

In area 2 of Figure 60 the size of the transmitter link interface buffer dictated the DMA transmission throughput, as several further PCI data bursts could occur before this resource was filled. The throughput was much lower than in area 1 due to the need for the data to pass through the transmitter message controller, responsible for converting data from 32-bit double words to 8-bit bytes, taking four PCI clock cycles to process a data word. In area 2 of the graph the FT-PCI-OSLi reached a maximum throughput of 4.51MBytes/s before the transmitters 32 token link interface buffer was filled. The PCI-OSLi and modified buffer FT-PCI-OSLi interfaces had 1kByte transmitter link interface buffers, giving DMA transmission throughputs of 25.48MBytes/s and 26.86MBytes/s respectively. Figure 60 demonstrated that a 1kByte message took 7,492 PCI clock cycles to be transferred in its entirety to the DMA transmitter buffer.

The communications link throughput limited the DMA transmission throughput in area 3 of the graph, as all buffering resources were filled and further transactions must wait for these to empty via the transmission of data. The DMA transmission throughput for this section of the graph reduced significantly due to the lower data rate of the serial communications link. The throughput was 3.837MBytes/s for the FT-PCI-OSLi, 3.898MBytes/s for the modified buffer FT-PCI-OSLi and 2.656MBytes/s for the PCI-OSLi when all three interfaces transmitted 64kByte messages. This corresponds to the link interface data throughput for messages of this length as shown in Figure 60, the data throughputs across the communications link being 30.65Mbits/s for the FT-PCI-OSLi (both implementations) and 20.91Mbits/s for the PCI-OSLi.

Figure 64 shows the DMA transmission throughput for the FT-PCI-OSLi at a variety of latency count values. The DMA transmitter buffer size cannot be altered without significant design changes but the latency timer, dictating the length of each PCI burst can be altered. As subsequent bursts can only be made once the transmitter DMA buffer is almost empty, this action effectively reduces the buffer size to that dictated by the latency count value. In reality, this level is reduced further due to the latency counter starting decrementing when a request for bus ownership is made, taking into account bus acquisition and set-up overheads. Thus a latency count value of 64 resulted in only a burst of 55 double-words of data on average. As the FT-PCI-OSLi took three PCI clock cycles to set-up a PCI transaction, following the granting of bus ownership, it took an average of 6 PCI clock cycles to gain bus ownership.

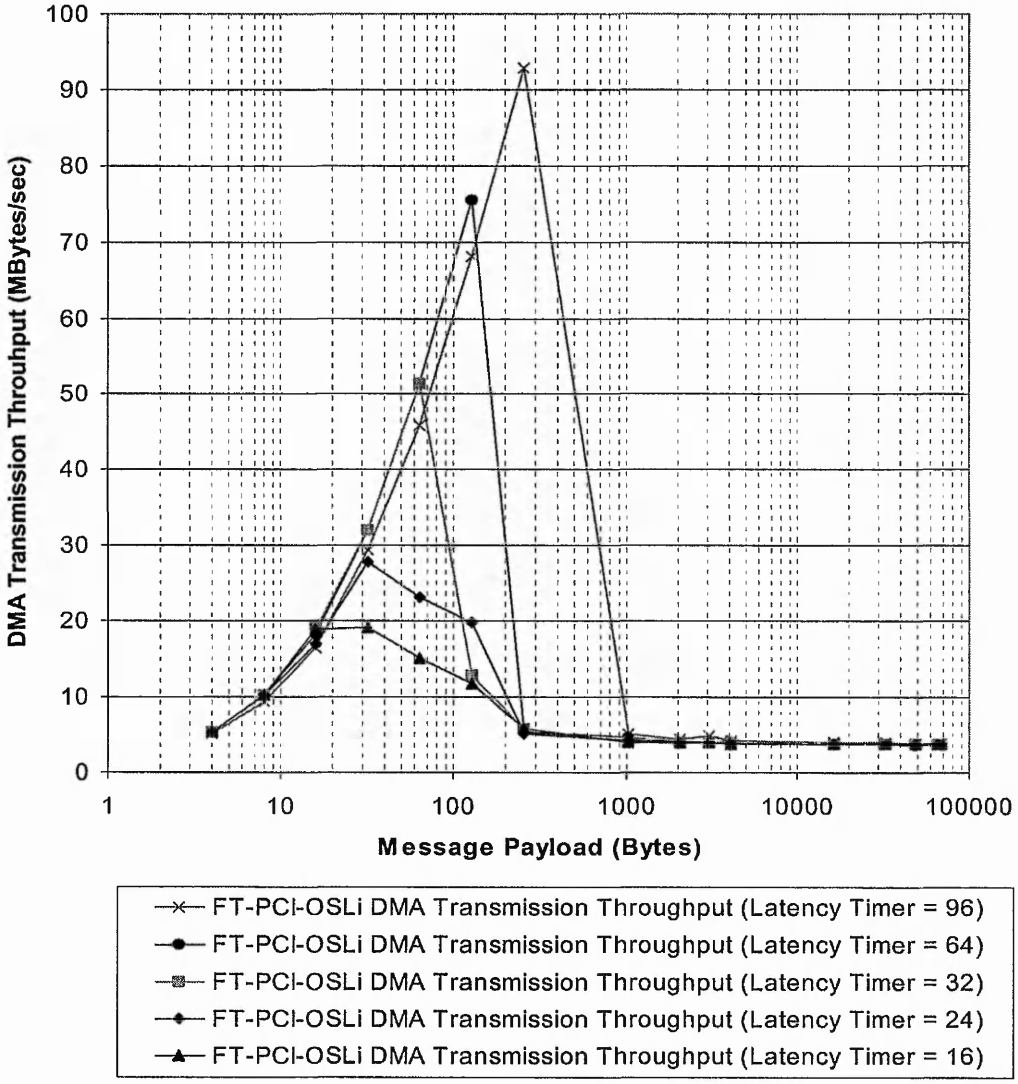


Figure 64: FT-PCI-OSLi DMA transmission throughput for varied DMA transmitter buffer capacities

Figure 64 shows the peak area 1 throughput to be 92.8MBytes/s for a latency count value of 96, filling the entire DMA buffer. The throughput increases linearly in proportion with the latency count value once bus acquisition and set-up overheads are taken into consideration. The plot for a latency count value of 96 peaks for message lengths of 256 bytes whilst the peak occurs at message lengths of 128 bytes and 64 bytes for latency counts of 64 and 32 respectively. The plots for the FT-PCI-OSLi with latency count values of 16 and 24 peak at message lengths of 32 bytes. These

characteristics at latency counts of 16 and 24 are different shapes to the others due to the relatively large overheads incurred in the acquisition of the PCI bus affecting the amount of data that can be transferred in a single burst. Once the message length exceeds the DMA buffer capacity, DMA transmission throughput falls with all characteristics being equal for payloads in excess of 1kByte.

Figure 64 demonstrates the effect that transmitter DMA buffer capacity has on area 1 of the FT-PCI-OSLi DMA transmission throughput characteristic. The graph demonstrates the 32-token capacity link interface buffer is small enough to make the throughput in area 2 virtually identical to that of area 3. The thirty two tokens required to fill the link interface buffer can be transferred across the PCI bus in eight clock cycles. Further data can only enter the link interface buffer following the outputting of data onto the serial link.

Figure 65 shows the effect that altering the transmitter DMA buffer capacity has on the area 1 characteristic of the modified buffer FT-PCI-OSLi (with a 1kByte transmitter DMA buffer capacity). These show noticeably different characteristics compared to Figure 64; for example the area 1 throughput for a latency count of 96 peaks at 88.93MBytes/s, 3.87MBytes/s below the comparable result in Figure 64. The characteristics for the FT-PCI-OSLi with the alternate buffer capacities, with latency count values of 64 and 32, peak at similar values to those in Figure 64. The characteristics for the latency count values of 16 and 24 are noticeably different as instead of peaking at message lengths of 32 and falling, as shown in Figure 64, they continue rising, at a reduced rate, before peaking at message payloads of 128 bytes at 31.29MBytes/s and 35.2MBytes/s respectively.

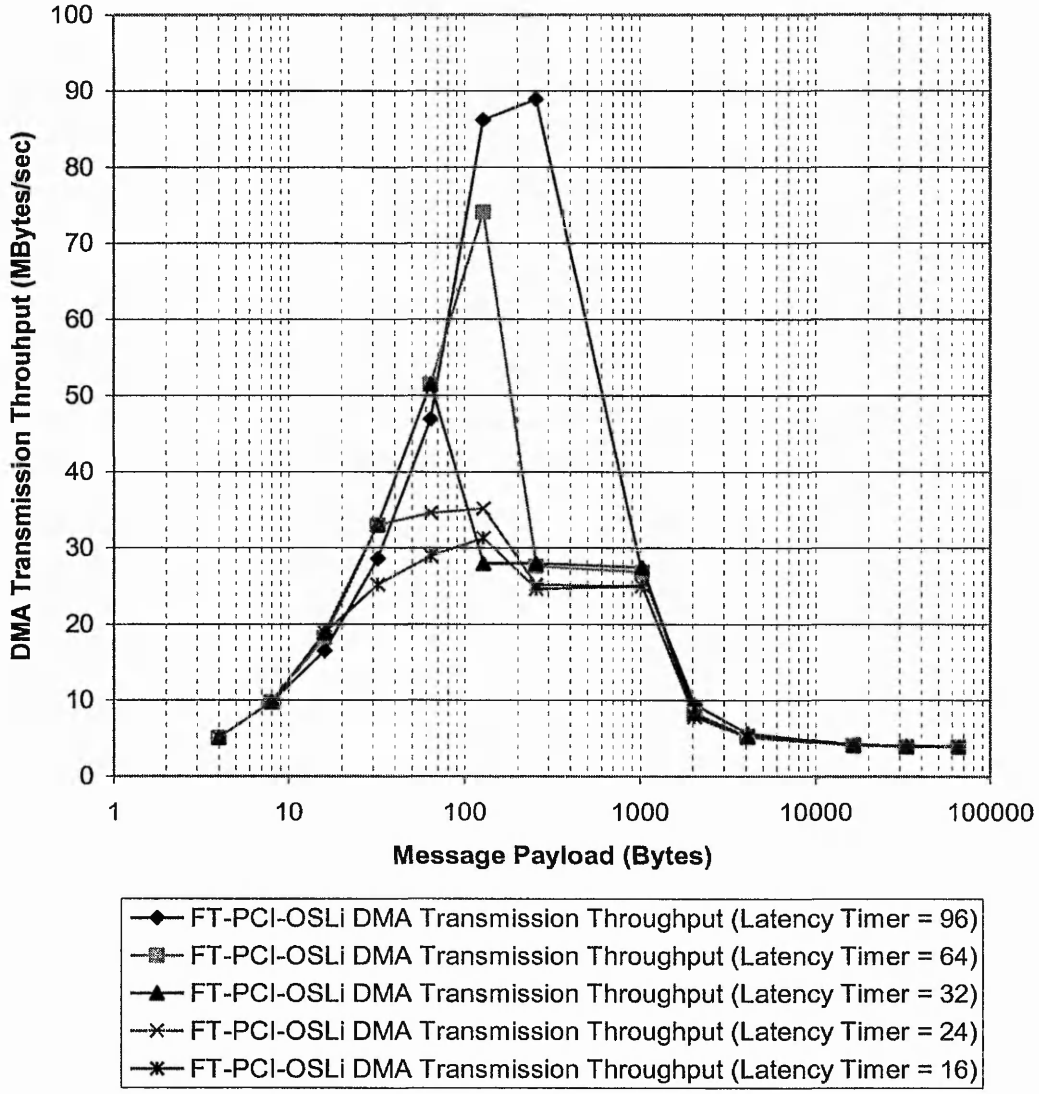


Figure 65: DMA transmission throughput for the FT-PCI-OSLi with 1kByte deep link interface buffer for varied DMA transmitter buffer capacities

With the exception of the characteristic for the FT-PCI-OSLi with a latency count of 96, all other characteristics plateau at around 25MBytes/s for message lengths between 256 and 1024 bytes – this being area 2 of the DMA transmitter throughput characteristic. The plot for the latency count of 96 does not level like the others as the latency count value is large enough to fill the DMA transmitter buffer to capacity for message sizes of 256 bytes, providing only an area 1 peak.

The latency count value of 96 is high enough to fill the DMA transmission buffer completely and also partially fill the link interface buffer. This is due to the streamed

buffering employed by the interface, which passes data straight through the DMA buffer to the link interface buffer. For this reason, the throughput for message payloads of 256 tokens is greater than that of 128 tokens as the PCI transaction is stopped by the DMA buffer filling to its 256 token capacity. This does not occur for the other characteristics in Figure 65 as the PCI transactions are stopped in these cases by the latency timer reaching zero before the DMA buffer reaches capacity. This is intentional as the reduced latency counter value is limiting the degree to which the DMA buffer can fill in order to observe the effect of different DMA buffer sizes on DMA throughput.

At message payloads of 1kByte, the throughput of this characteristic is affected by the link interface buffer capacity and so the throughput at this point falls to the same level as the other characteristics in the graph.

Altering the capacity of the transmitter DMA buffer affects the position and height of the peak in area 1. Altering the link interface buffer capacity was proven to affect the height of the characteristic in area 2 of the DMA transmission throughput, with an increased buffering capacity giving greater throughput, at a cost of increased buffering resources. This increase was not proportionate, therefore a trade off must be made according to the systems requirements. Increasing the FT-PCI-OSLi buffer capacity to 32 times its original size resulted in only a six-fold increase in the transmitter DMA throughput, in this section of the characteristic, as shown in Figure 65.

The communications link data rate affected the height of the area 3 characteristic, but as both the FT-PCI-OSLi and the PCI-OSLi operated on a 42Mbits/s bit rate, the differences in these characteristics were wholly due to the different flow control mechanisms utilised by these two designs. Figure 66 summarised the effects that could be achieved via alteration of the FT-PCI-OSLi parameters.

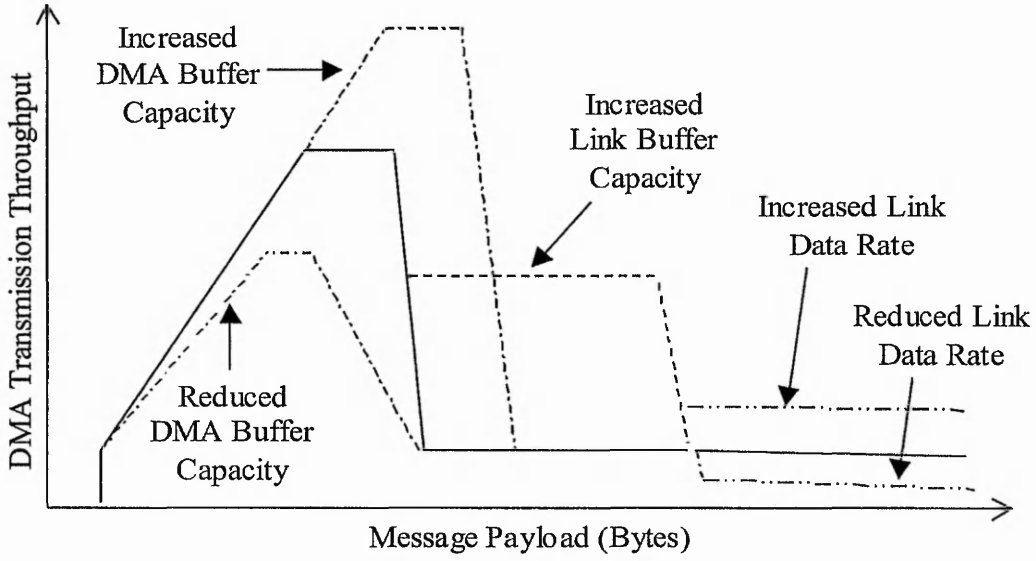


Figure 66: Summary of the effects on DMA transmission throughput caused by alterations to the FT-PCI-OSLi DMA and link interface buffer capacities

6.3.4 DMA Message Reception

Figure 67 shows the DMA reception throughput for the FT-PCI-OSLi compared to that of the PCI-OSLi and a FT-PCI-OSLi interface with a 2kByte deep receiver link interface buffer at a 42Mbits/s link data rate. PCI transactions were only initiated when the DMA buffer was full or the message had ended. The former event depended on the speed at which the message could be received from the communications network. At lower message lengths the overhead incurred in setting up the PCI bus transaction limited the throughput.

The DMA reception throughput for payloads of four bytes was severely limited by the proportionately high PCI bus acquisition overheads, due to the inclusion of the latency incurred in acquiring ownership of the bus in these measurements. Throughput increased with payload, reaching a plateau of 3.813MBytes/s and 2.613MBytes/s for the FT-PCI-OSLi and PCI-OSLi devices respectively. The plateau started at about 1kByte, being approximately equal to the communications link data throughput for payloads of this size upwards. The modified buffer FT-PCI-OSLi

interface (with the 2kByte capacity link interface receiver buffer) had an almost identical characteristic to the FT-PCI-OSLi interface but a slight increase in throughput between payloads of 256 and 2048 bytes which was due to the increased link interface buffer capacity. All three characteristics were steepest for message payloads below 64 bytes, when the DMA buffer was filling. All these characteristics had levelled out once message size exceeded 2048 bytes, filling both the DMA and link interface buffers.

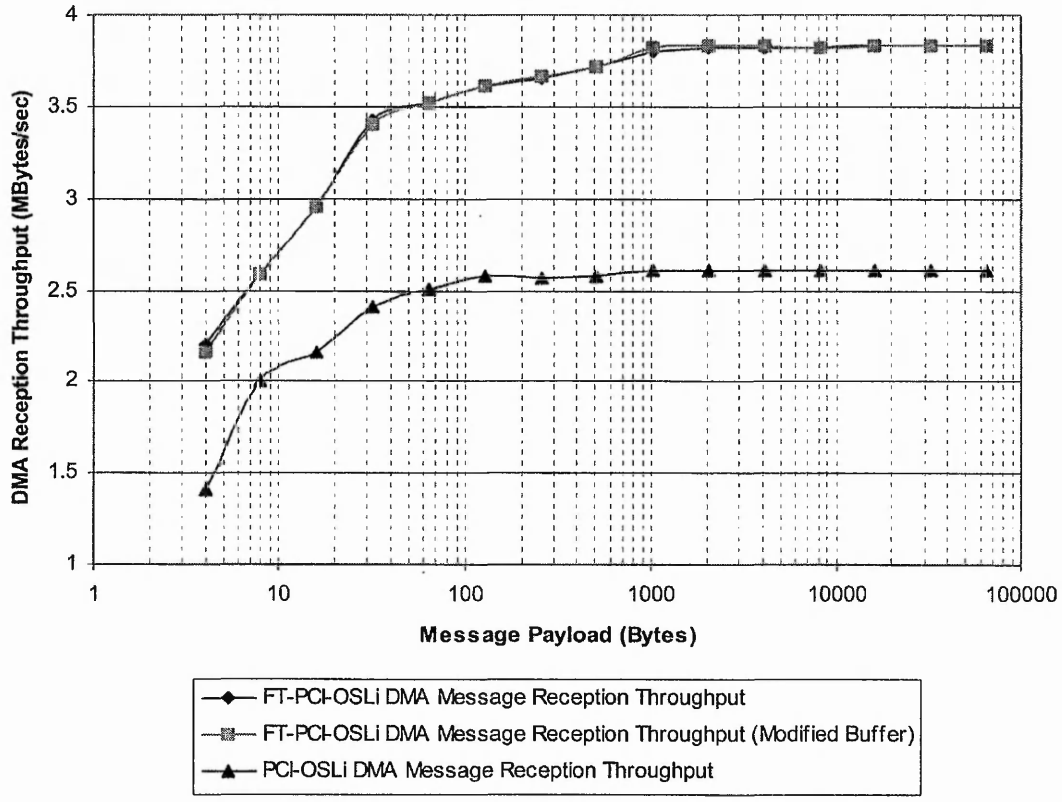


Figure 67: DMA reception throughput during bi-directional data transfer for the FT-PCI-OSLi, PCI-OSLi and modified buffer FT-PCI-OSLi interfaces

The difference in the magnitude of the FT-PCI-OSLi and PCI-OSLi graphs was due to the amount of data that could be transmitted across the communications link in a given time. Both interfaces operated at the same link data rate but the permission based flow control of the FT-PCI-OSLi permitted much more data to be sent as it reduced the amount of excess non-data tokens. Increasing the sample clock rate

increased the amount of data that could be transmitted across the link, and hence the receiver DMA throughput.

6.4 Fault Detection and Recovery Hardware Tests

In addition to the performance tests, a series of hardware tests were undertaken using the FT-PCI-OSLi, operating at a 64MHz sample clock rate, to verify correct operation of a variety of the features used in fault detection and recovery. The tests were performed using the loopback operation described previously and utilised debug counters and flags located in the DMA Registers module, visible using the PCIWave [156] software, to indicate the status of the interface.

6.4.1 Incorrect Message Length Hardware Test

This test consisted of a bi-directional message transfer where the message lengths that were loaded into the Transmitter and Receiver Message Length Registers (register offsets $0E_H$ and $0D_H$ respectively) were deliberately different. The purpose of this was to check the FT-PCI-OSLi could detect a shorter or longer than expected message and trigger an early or late message termination respectively (denoted by the assertion of bits 2 and 3 respectively in the Interrupt Pending Register). The number of data words transferred to memory equalled the smaller of the two message lengths, with any remaining data tokens in the link interface buffer being flushed.

6.4.2 Incorrect Message Header Hardware Test

This test was identical to the previous test but different message IDs were written to the Byte-length Receiver and Transmitter Header Registers (register offsets $0A_H$ and $0B_H$ respectively). The transmitted header was not held in any of the CAM locations, thus on arrival at the receiver no match was made. The message sat in the receiver link interface buffer, filling it to the point where data flow across the communications link was suspended. The test then had two possible actions, which were:

- The user entered the correct message header into the CAM (equal to that of the incoming header). The test verified that the message was successfully transferred into memory in its entirety.
- Setting bit 29 in the Receiver Command Register (see Appendix C) to flush the message from the link interface buffer. The test verified that on clearing the contents of the receiver link interface buffer, flow of data across the communications link resumed and the message was flushed.

6.4.3 Disconnected Link Hardware Test

This test verified the operation of the 'link activity time out' detection mechanism and subsequent attempts to reinitialise a valid path across the communications channel. The test consisted of a bi-directional loopback message transfer that was terminated midway through the test by driving the serial communications link output to ground, preventing the passage of any tokens across the channel. Once this occurred, a timer was started and incremented until the receiver detected the presence of a fault, through lack of link activity. The link was set to disconnect after transmission of the fourth token (including the header token). Tests showed that the link was declared faulty after a time period equating to 60 tokens, following link disconnection. This was due to the FT-PCI-OSLi hardware being set up to trigger a disconnected link following an adjustable time-out equivalent to the transmission of four heartbeat tokens without reception.

The use of permission based flow control meant that, in event of network failure, the transmitting node kept outputting data when the network failed. Doing this effectively removed the message from the system and prevented the indefinite stalling characteristic of credit based protocols. The receiving node did not append a 'Bad End of Packet' token to the message as it was not forwarded to any other nodes and was destined for memory. The FT-PCI-OSLi must indicate to the software that the message was prematurely terminated. The steps have not yet been taken to implement software functionality to deal with premature message termination. Hence, currently there is no means of informing the transmitting node that the message was terminated unsuccessfully at this (hardware) level of the network.

6.4.4 Flow Control Hardware Test

This test verified the operation of the permission based flow control protocol. Data was prevented from passing from the receiver link interface buffer to the depacketiser until the buffer had filled to the point that traffic across the communications link was suspended. Data was then allowed to leave the receiver link interface buffer, for processing, emptying it so that data flow could resume and complete the message transfer. The message payload was 128 data bytes. The test was conducted in the same loopback bi-directional communications manner as before but an additional state machine was incorporated into the FT-PCI-OSLi design to control the passage of data out of the receiver link interface buffer. The state machine possessed six states, which were entered in order during execution of the message transfer.

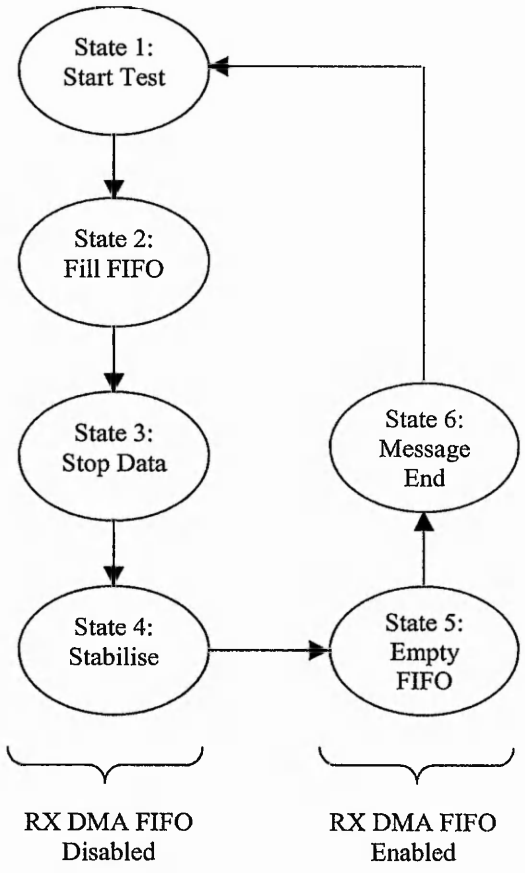


Figure 68: State machine showing the six states used in the flow control test

1. Start Test – entered on power on reset. Left when the first (header) byte was written to the receiver link interface buffer (whose output was disabled in this state to allow data to accumulate).
2. Fill FIFO – entered when first byte was written to the receiver link interface buffer. Left when buffer contents reached 24 tokens, triggering the assertion of the Almost Full flag. The receiver link interface buffer output was disabled in this state.
3. Stop Data – entered on assertion of the Almost Full flag. Exited when the transmitter has received and successfully decoded the command to halt data flow. The receiver link interface buffer output was disabled in this state.
4. Stabilise – entered following the suspension of data flow. Entering this state enabled an 8-bit counter, incremented every PCI clock cycle. The counter incremented until full, before leaving the state. This state allowed the traffic flow to stop with checks made to ensure that no data tokens were active on the link at this stage. The receiver link interface buffer output was disabled in this state.
5. Empty FIFO – entered when the 8-bit counter reached capacity, enabling the receiver link interface buffer output and allowing the data to progress out of the buffer to the depacketiser. Left when the amount of data tokens in the buffer fell to 8 tokens, triggering the Almost Empty flag to resume the flow of link traffic. The receiver link interface buffer output was enabled in this state.
6. Message End – state signifying the message end was entered following assertion of Almost Empty. Left when transfer of the message to the depacketiser had been completed. The receiver link interface buffer output is enabled in this state.

The state machine returned to state 1 for the next test. The results showed that the FT-PCI-OSLi took a time equal to 24.131 token periods to complete state 2, filling the receiver link interface buffer. It took 1.6527 token periods to initiate, transmit, decode and act upon the command to halt data flow via the generation of a Stop flow token (state 3). It took 28.28 PCI clock cycles to enable the buffer output, match the message ID with one held in the CAM and read 18 tokens into the depacketiser (state 5). Once data flow resumed, it took 110.345 token periods to complete the message (state 6).

The test demonstrated the successful operation of the permission based flow control mechanism. Data flow across the communications link was suspended and

resumed after a short delay. Data was prevented from leaving the receiver link interface buffer until it had filled to the Almost Full level and the 8 bit counter triggered by this event had incremented to capacity (which occurred approximately 23 token periods later). A maximum of 26 tokens were held in the receiver link interface buffer. Therefore following triggering of the Almost Full flag, when 24 tokens were held in the buffer, two more tokens were transmitted in the time taken to initiate, transmit and act upon a Stop token.

The loopback test sent the output signals through the differential line driver circuitry in the same manner as would occur if communicating with another interface. However the length of cable between the serial output and input was minimal, being only a few centimetres long. The target network of the FT-PCI-OSLi was aimed at physically distributed processor nodes, separated by distances of up to 100 metres. Stopping the flow of link traffic across a communications distance of this length would take longer. Therefore more data tokens would be received in the interval between requesting the suspension of data flow and the receipt of the last token. The receiver buffer must have sufficient resources to accommodate these extra tokens. The time taken for an electric signal to traverse a length of wire is given by the formula:

$$t_{WIRE} = \frac{\text{Cable Length}}{\text{Speed of Signal}}$$

If the cable length was 100 metres and the speed of the signal across the wire could be approximated to 66% of the speed of light, or 2 x 10^8 metres/second [124], the transmission time is 500ns but as the signal must effectively travel there and back, the total signal transmission time is 1μs (plus the processing time taken from state 3 of the test). At a 42Mbits/s data rate, a single bit lasts for 23.43ns. 42.67 bits can be transmitted across the link, equating to 3.878 tokens if back-to-back transmission takes place. Therefore an extra four tokens could be received by the FT-PCI-OSLi following suspension of link traffic, increasing the receiver link interface buffer occupancy to 30 tokens, but averting buffer overflow.

6.4.5 Link Dormancy Hardware Test

The link dormancy test used a sequence of flags in the DMA Registers module, visible via the PCIWave software, to verify that when the link dormancy option was selected that the Link Interface state machine returned to the Asleep state after initialisation. Link dormancy was enabled via the assertion of bit 19 in the Receiver Command Register (offset $0F_H$). A write to the Transmitter Length Register (offset $0D_H$) when in this state issued a KickStart command (see section 4.4.6), returning the state machine to the Reset state and causing the re-initialisation of the communications channel between the two nodes. Upon completion of the message transfer a count was started. If no other link activity occurred before the counter timed out, the link returned to the Asleep state and further message transfers initiated link re-initialisation.

The test proved that the link entered dormancy after an interval equal to the time taken to generate three checkpulse signals (approximately equal to 93 tokens). Exiting a dormant state and resuming link activity via link re-initialisation was performed successfully and without affecting message transfer, as it was completed in part whilst the message progresses through the transmitter, imposing minimal additional overhead on the message. Overhead was not an issue, due to link dormancy being used in situations where there is very little link traffic, making loss of throughput a relatively minor concern.

6.5 Resource Usage

The FT-PCI-OSLi used 3109 LEs (37% of the total available on the Apex 20K200E device) and 6720 memory bits (6% of the total available), grouped in 83 embedded memory blocks. The maximum PCI clock frequency was specified at 74.88MHz.

In contrast the non-fault tolerant PCI-OSLi design, implemented originally on a Flex 10K50S device occupied 2392 LEs (83% of the total available for that particular device), 28kbits of available embedded memory (70% of available resources for the Flex 10K device). Timing analyses stipulated a maximum PCI clock frequency of

51.54MHz for the PCI-OSLi for this hardware set-up. A new device was required to implement the FT-PCI-OSLi due to the tight fit of the previous design and the significant logic increases.

For the hardware tests, the PCI-OSLi design was implemented on an identical Apex 20K200E device to the FT-PCI-OSLi. This enabled a more accurate comparison between the two designs. This was because all performance advantages gained from the use of a faster and improved PLD architecture and high speed PCB design were cancelled out as both designs benefited from these features. Neither design was optimised for the target technology. When implemented on the Apex 20K200E device, the PCI-OSLi utilised 2929 LEs (35% of the total available) and 28kbits of memory implemented in 80 memory segments (26% of the total available for the target device). The maximum PCI clock rate of the FT-PCI-OSLi was specified at 72.08MHz.

Comparisons between resource utilisation in modules of the two designs were hard to make as the internal module hierarchy was altered significantly. Parts of the FT-PCI-OSLi design benefited from logic reductions, in particular the consolidation of multiple counters into one, in the packetiser and depacketiser modules of the design. The FT-PCI-OSLi design used more logic resources due to the increased functionality in the design, although fewer memory resources were used due to the minimal buffering requirements.

Resource usage in the DMA Registers module leapt from 752 LEs in the PCI-OSLi design to 1620 LEs, as shown in the table in Figure 69. This was due to the Virtual Channels section of the design being situated in this module, adding 371 LEs. Also situated in the DMA Registers were several 32-bit counters, used for test purposes and the debug signals visible via debug software. The DMA FIFO Controller module resource usage rose from 160 LEs to 223 LEs. This was due to logic alterations to improve the set-up and hold times required for interfacing to the PCI interface. Resource usage in the packetiser module of the FT-PCI-OSLi dropped to 74 LEs from 131 LEs, due largely to the reduction in message length counters and a simplification of the state machines in this module. A similar approach led to a reduction in the depacketiser resource usage from 290 LEs to 162 LEs. The transmitter and receiver DMA buffers hold 64 of 32-bit data words, as before, but

each buffer utilised 33 LEs for glue logic, as opposed to 41 LEs in the PCI-OSLi design.

Main Module	Sub Modules	Logic Elements Utilised	Memory Blocks Used	Regs Used	Highest PCI Clock Freq (MHz)	Highest Link Clock Freq (MHz)
Top Level	**FT-PCI-OSLi**	**3109**	**83**	**1531**	**74.88**	**See Note**
PCI Interface	**Master Control**	**620**	**0**	**198**	**146.46**	**N.A.**
	Master / Target Controller	71	0	50	216.59	N.A.
	Data Path	366	0	32	290.02	N.A.
	Target Decode	31	0	24	181.81	N.A.
	Parity Checker	70	0	46	260.48	N.A.
	Configuration Registers	79	0	46	182.68	N.A.
DMA Registers	**DMA Registers**	**1620**	**1**	**749**	**87.00**	**N.A.**
	Virtual Channel Store	371	1	145	85.86	N.A.
	Interrupt Controller	3	0	1	140.84	N.A.
DMA Interface	**Link Interface**	**869**	**82**	**584**	**91.18**	**N.A.**
	DMA FIFO Controller	223	0	155	174.92	N.A.
	DMA Transmit Buffer	33	32	21	179.28	N.A.
	DMA Receive Buffer	33	32	21	179.28	N.A.
Message Interface	Packetiser	74	0	14	252.78	N.A.
	Depacketiser	162	0	104	115.73	N.A.
Link Interface	Transmit Link Interface FIFO	101	9	87	290.02	252.27
	Receive Link Interface FIFO	100	9	87	242.78	290.02
	Transmitter	54	0	34	N.A.	See Note
	Receiver	43	0	31	N.A.	139.9

Figure 69: Modular Resource Usage in the FT-PCI-OSLi Interface

Note: The Quartus II Timing Analyzer did not recognise the 1.5 times oversampling method utilised in the transmitter as producing a valid clock signal, claiming the circuit was not operational due to the clock skew exceeding the data delay. A timing analysis of this module performed on the MaxPlus2 version 10.0 fitted to a Flex 10K device revealed a maximum sampling clock frequency of 129.87MHz.

The Link Interface Transmitter module saw an increase in resource usage from 31 to 43 and from 34 to 54 LEs. These increases were due to the protocol alterations required to achieve improvements in fault tolerance. These alterations included the generation and decoding of control tokens and logic to prevent them from progressing from the receiver module. The Link Interface buffers were reduced significantly in size from 1kBytes and 2kBytes for the PCI-OSLi transmitter and receiver buffers respectively, to the 32 deep 9-bit wide FT-PCI-OSLi buffers. The number of logic elements used as glue logic for the receiver buffer halved from 203 to 100 LEs. The FT-PCI-OSLi utilised 101 LEs for the transmitter link interface buffer whilst the corresponding non-fault tolerant module utilised 186 LEs. For additional hardware tests, link interface buffer sizes were altered to 256 deep 9-bit wide buffers.

Architectural alterations to the PCI interface resulted in most modules in this part of the design experiencing reductions in resource usage. The Address / Data Path module experienced a large increase, from 160 to 366 LEs, cancelling out most of the reductions. Movement of functions to the DMA Registers and DMA FIFO Controller modules reduced the resource usage of some of the PCI interface modules.

6.6 Power Consumption

Embedded systems are often used in portable / hand held electronic applications due to their compact nature. Such applications are frequently battery powered for convenience, requiring minimal power consumption. It was therefore very important to be able to assess the power consumption of a large PLD, which in SoC applications may represent the main drain of energy.

The PLDs pre-defined internal architecture allowed the synthesis tool to identify which resources were utilised following the fitting of a design to a device. This permitted Altera's proprietary Quartus II software to make a highly accurate assessment of the power consumption of the device. Such calculations were possible in ASICs but only after the time consuming process of design layout. Altera's electronic literature possessed a design specific power calculator [143] for the Apex

20KE device family that was used to determine the power consumption of the FT-PCI-OSLi interface.

The approximate total power consumption values for the device were:

Icc_{INT} = 109.5mA, Icc_{IO} = 3.19mA, P_{INT} = 179.11, P_{IO} = 10.53mW.

These values were only approximate as estimations were made in certain calculations and average values were used in others. When operating at an ambient temperature of 20 degrees centigrade without a heatsink, it was decided that airflow around the device would be adequate to ensure that no thermal design issues arose.

6.7 FT-SARNIC Post-synthesis Simulation

This section of the results chapter presents the performance of the FT-SARNIC based on post-synthesis simulation using Altera's MaxPlus2 simulation and synthesis software. Similar tests were performed with the SARNIC design fitted to the same FLEX 10K50VRC240-3 device in order to enable comparisons to be made between the two interface designs. The simulation results were also compared to the FT-SARNICs theoretical performance characteristics to identify how closely the design conformed to the ideal.

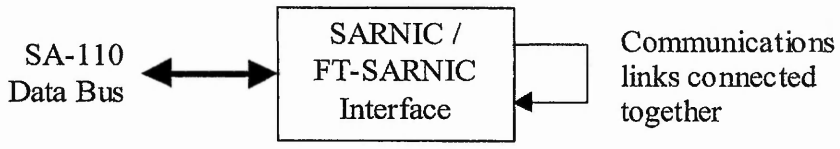


Figure 70: Loopback Test Block Diagram for FT-SARNIC

The simulations were performed using a 'loopback' method involving fetching of data from the SA-110's SDRAM to the FT-SARNIC via the data bus. The data was formatted ready for transmission onto the serial communications link, and was then routed back into the device as Figure 70 demonstrates.

Such tests utilised bi-directional link bandwidth, requiring fair arbitration of DMA accesses as the interface transmitted and received data simultaneously. It assumed that the interface could always access the SDRAM via the SA-110's bus, which might not always be the case. Previous research [150] demonstrated that the processor overhead incurred through the utilisation of 4 DMA channels at a 20Mbits/s link rate reduced the computational performance of the SA-110 by 3.34%. The FT-SARNIC utilised a virtually identical bus access mechanism as the SARNIC so it was presumed that the communications overhead would be similar. The post-synthesis loopback simulations were performed at data rates of 20Mbits/s and 39Mbits/s but only two DMA channels were utilised, (one in each direction) as all packets in the communication belonged to the same message. The loopback test simulated bi-directional traffic flow on the communications links, requiring the interleaving of data and control tokens, exposing inefficiencies incurred in doing this. Simulation of both devices did not take into account the delay incurred by external transceivers on the communications link as the serial communications output signal was routed back to the input before leaving the PLD, in order for the post-synthesis simulation to work.

6.7.1 Bi-directional Data Transfer Duration

Figure 71 shows the linear increase in message duration observed during post-synthesis simulation of the FT-SARNIC design. The FT-SARNIC characteristic demonstrated noticeably larger savings in message duration as the payload increased, in comparison to the SARNIC characteristic. This was due to each SARNIC token requiring the receipt of an acknowledgement token, increasing the number of bits per token to 13. In addition, the SARNIC protocol required a new packet to be set up every 256 data tokens, requiring three additional tokens and the incursion of extra overheads in the process of packet initialisation. The ability of the FT-SARNIC to transmit large messages as single packets eliminated such overheads and as such the ramp of the message duration characteristic increases at a lower rate than that of the SARNIC.

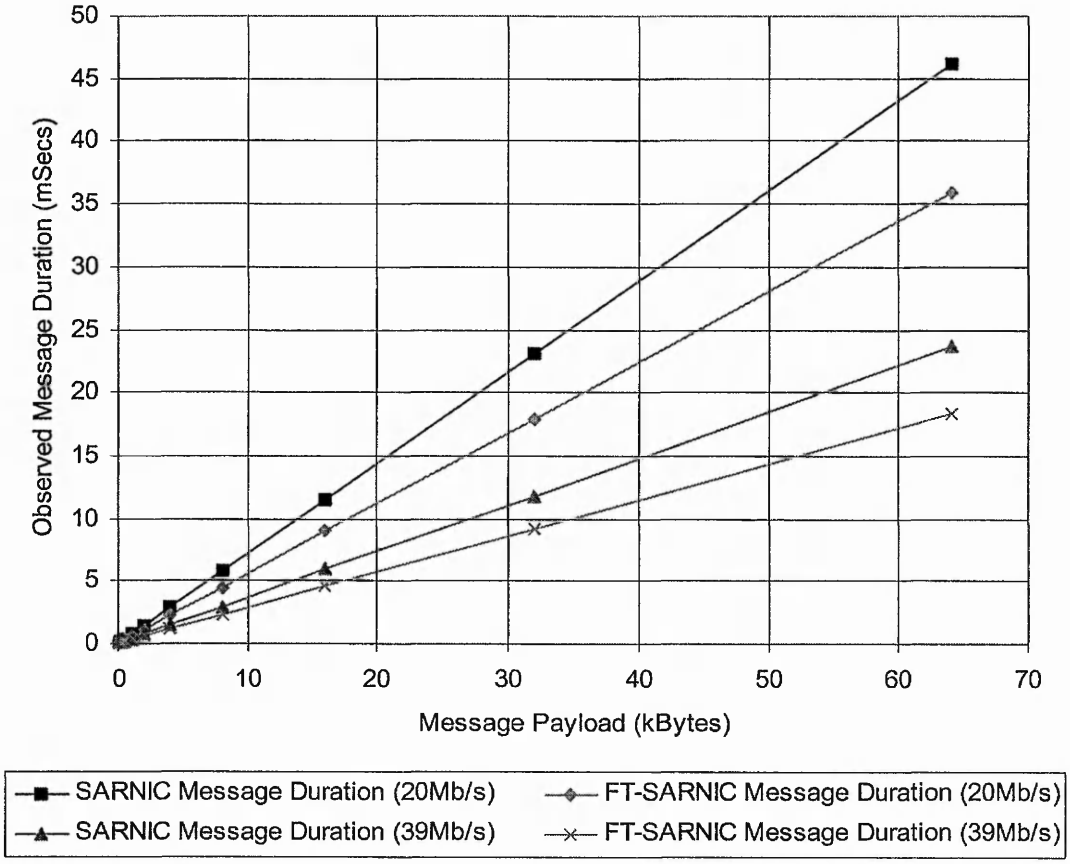


Figure 71: Message duration results for FT-SARNIC post-synthesis simulation

Figure 72 magnified the lower region of Figure 71 to display the message duration versus payload characteristic for message sizes up to 64 bytes at data rates of 20Mbits/s and 39Mbits/s. At the minimum message length of 4 bytes, the SARNIC took 376ns longer to complete message transmission as two extra bits per token were transmitted. An overhead of three tokens per packet (up to 256-bytes) was necessary comprising two header tokens and a length token. So the 4-byte message required the SARNIC to send 12 extra bits (6 acknowledge tokens), occupying 25.5ns apiece at a 39Mbits/s data rate, assuming back-to-back data and acknowledge token interleaving. The last acknowledge token could be discounted as the receiver began processing the message whilst this acknowledge token was still being transmitted. These extra bits took a total of 306ns to transmit, assuming no extra overheads, allowing the FT-SARNIC to complete its message transfer 70ns faster than the SARNIC.

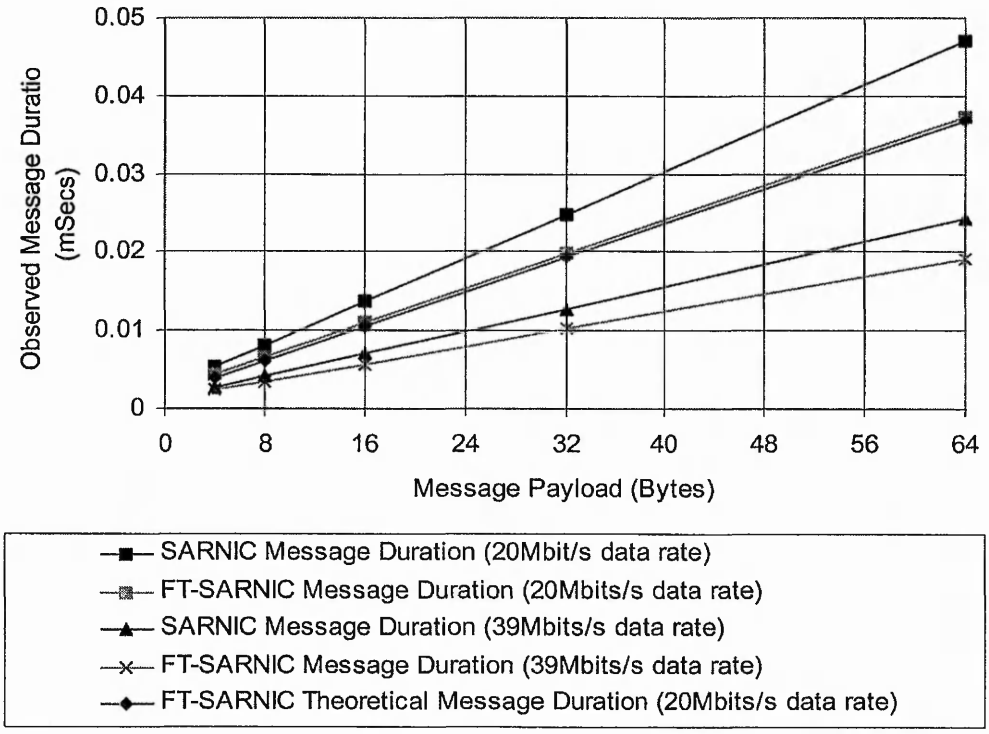


Figure 72: Message duration results for FT-SARNIC post-synthesis simulations at lower message lengths

As message size increased, the latency incurred in initiating and concluding a message becomes relatively insignificant relative to the message duration. A greater proportion of non-data tokens meant the SARNIC message duration versus payload characteristics increased at a faster rate than that of the FT-SARNIC.

6.7.2 Bi-directional Data Bandwidth Utilisation

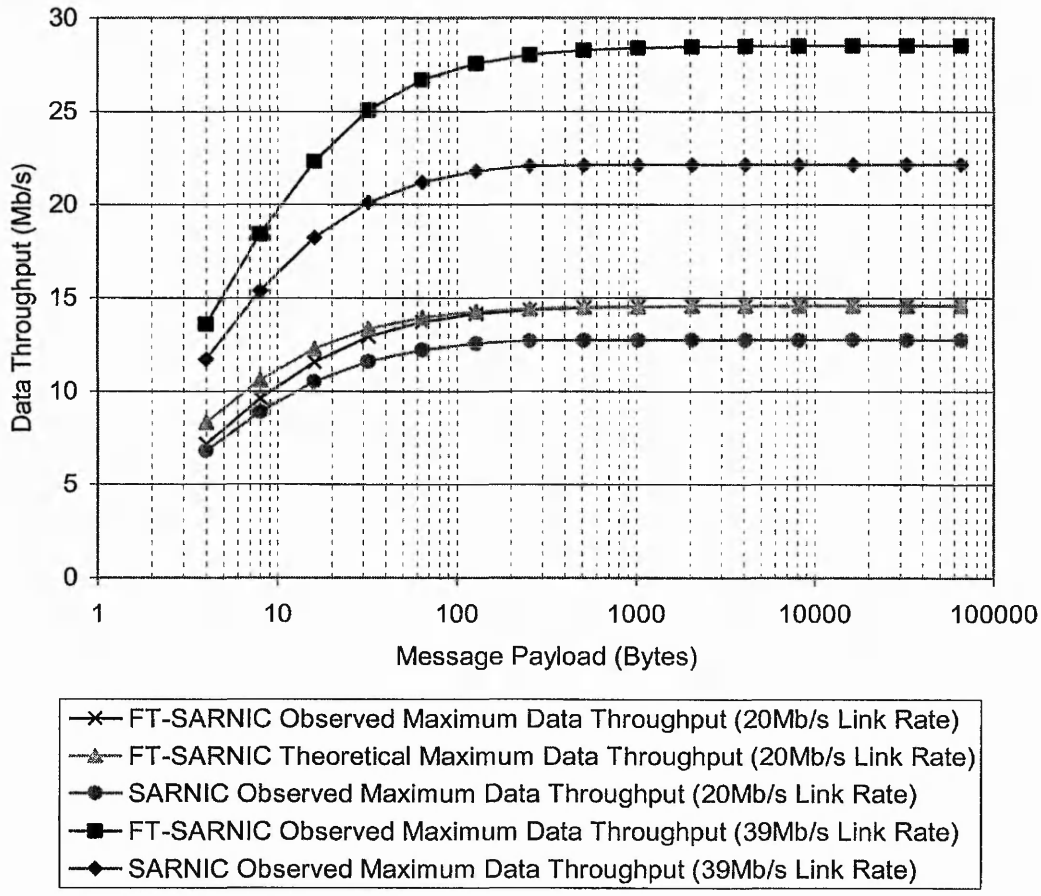


Figure 73: Data throughput results for the FT-SARNIC and SARNIC interfaces

Figure 73 shows the data throughput of the SARNIC and FT-SARNIC interfaces plotted against message payload for data rates of 20Mbits/s and 39Mbits/s. The theoretical performance characteristic of the FT-SARNIC at a data rate of 20Mbits/s was also plotted for comparison.

Figure 73 demonstrated the extent to which message overheads limit throughput as data rates increased. It also demonstrates the effect that permission based flow control has on data throughput. All three plots began to plateau once message size exceeded the 256 token packet boundary imposed on the SARNIC design. The performance characteristics up to this point were much steeper for the 39Mbits/s characteristics than at 20Mbits/s, for both FT-SARNIC and SARNIC devices.

The FT-SARNIC data throughput was much greater than that of the SARNIC as it did not need to send acknowledgement tokens or multiple packets, freeing more bandwidth for data. At a 20Mbits/s data rate, the FT-SARNICs throughput was over 14% higher than the SARNIC for the maximum message length (64kBytes). This figure increased to over 28% when the interfaces operated at a 39Mbits/s link rate.

The FT-SARNIC performed poorly at lower message lengths in comparison to its theoretical performance, with the 20Mbits/s characteristic being closer to that of the SARNICs response at this link rate. The theoretical performance characteristics only took account of the speed at which data could be sent across the communications medium, given the network protocols, excluding message set-up and reception delays. As message size increased, these overheads reduced in significance relative to the overall message duration and the FT-SARNICs performance characteristic conformed more closely to the theoretical prediction, almost reaching parity at the maximum message length. Differences in bandwidth utilisation for messages above 256 bytes are almost entirely due to the different flow control mechanisms.

The data throughput characteristic measured the proportion of data bits in the entire message, comprising of: header, length, termination and acknowledge tokens. Each message had a 2-byte long header and a byte length termination or length token for the FT-SARNIC and SARNIC interfaces respectively. Thus, the data content of a 4-byte message was 32 bits out of a total of 77 and 91 bits for the FT-SARNIC and SARNIC interfaces respectively, making a large difference to data throughput.

6.8 Summary

Section 6.2 compared the tests performed on the hardware implementation of the FT-PCI-OSLi with those of the PCI-OSLi and the FT-PCI-OSLi's theoretical performance characteristics. These results demonstrate the increased efficiency in terms of the PCI set-up latency of the FT-PCI-OSLi when compared to the PCI-OSLi. The results also show the reduction in message duration due to the use of permission based flow control. The DMA transmission and reception results in section 6.3 demonstrate the effect that the transmitter DMA and link interface buffer capacities

have on the DMA transmission throughput. This section also demonstrates the effect to which the data rate of the communications link affects the receiver DMA throughput. The results in section 6.4 demonstrated the correct operation of the hardware fault tolerance features, showing the ability of the FT-PCI-OSLi to detect and recover from faults.

Section 6.5 (and Appendix G) calculate the estimated power consumption for the FT-PCI-OSLi dependent on the number of gates and I/O pins utilised by the design. Section 6.6 displays the resource usage and maximum theoretical clock speeds for each individual design unit in the FT-PCI-OSLi interface. It compares the resource usage in different parts of the design with that of the PCI-OSLi interface.

Section 6.7 compared the post-synthesis simulation of the FT-SARNIC with those of the SARNIC and the FT-SARNICs theoretical performance characteristics. The results demonstrated the superior data throughput of the FT-SARNIC relative to the SARNIC due to the adoption of the permission based flow control. The deviation in the FT-SARNICs observed performance relative to its theoretical performance at lower message lengths was due to the overheads incurred in setting up messages being relatively large with respect to the time taken to transmit the message. Figure 73 indicated the difference between the theoretical and observed characteristic became negligible for message lengths above 256 tokens. These results compared the implementation of the designs by eliminating external delays that would be cancelled out via the implementation of the devices on identical hardware, but would also reduce the performance characteristics slightly.

7 DISCUSSION

Network interfaces are responsible for the conversion of data from one format to another. The FT-PCI-OSLi and FT-SARNIC convert data from the serial format of the communications link to 32-bit parallel words for transfer to PC memory or StrongArm SA-110 microprocessor memory, respectively. The research resulted in the implementation of two processor interfaces with improved fault tolerance that form building blocks in an embedded parallel processing system. These processor interfaces are used to implement a distributed fault detection and recovery mechanism in addition to improving performance.

Much of the previous work into improving fault tolerance in parallel systems concentrated on ensuring networks could operate in a degraded state, bypassing disabled links via the use of adaptive routing algorithms. Such a strategy suits regular network topologies, but not irregular switched networks. This method increases traffic on the remaining links leading to access contention and the formation of bottlenecks. Eventually network throughput and efficiency will be lowered to the point where a system wide reset is required to restore effective network operation. Faults between network end nodes, such as the FT-PCI-OSLi and FT-SARNIC and routers isolate the interface from the rest of the network, unless duplicate communications links are utilised.

This section of the thesis discusses the ways in which the FT-SARNet embedded parallel system and its end nodes, the FT-PCI-OSLi and the FT-SARNIC improve upon their non-fault tolerant predecessors.

7.1 Target Networks

The target applications for the FT-SARNet embedded parallel network are typically control based. Multiple FT-SARNIC processor nodes perform computational, data processing tasks with results and other information passed to the user via a PC and the FT-PCI-OSLi. Real time applications will require a regular flow of input data for monitoring and manipulation at the appropriate sampling interval.

Bandwidth will limit the amount of data that can be sent per sample and the sample interval must be frequent enough for the system to be responsive to change. These requirements are suited to the cycle-stealing DMA transfer methods used by the FT-SARNIC. Transfers of data to and from the FT-PCI-OSLi must be performed as rapidly as possible to prevent other bus users being denied frequent bus access.

7.2 FT-PCI-OSLi Performance

As the results in chapter 6 demonstrate, the FT-PCI-OSLi achieved a throughput of over 92.8MBytes/s when the message size equalled the 256 byte DMA buffer capacity. This fell to a throughput determined by the communications link data rate for payloads above this level. Competition from other PCI agents for bus ownership reduced throughput further due to access arbitration delaying the granting of bus ownership to the FT-PCI-OSLi.

Whilst attempting to push the DMA throughput towards the maximum throughput of the PCI bus might seem desirable, in reality it could result in wasted design effort. This is due to the shared nature of the bus preventing any one resource monopolising access in a practical situation. Additionally, the majority of FT-SARNet communications would involve the FT-SARNIC, which would not be able to achieve such high DMA throughput levels without preventing processor operation. Focussing on improvements to the DMA throughput of the FT-PCI-OSLi could result in wasted bandwidth as the FT-SARNIC becomes the system bottleneck.

If 92MBytes/s of the available PCI bandwidth of 132MBytes/s were devoted to data transfer from memory to the FT-PCI-OSLi, a maximum of 40MBytes/s would be available for all other PCI agents, severely constraining other PC resources.

Such imbalances in data throughput would require data to be ready for use long before it was necessary, due to the time taken to transfer the message. This is unacceptable in systems aimed at real-time embedded applications. Figure 62 in section 6.3.3 showed the effect that message size could have on the requests for PCI bus ownership, were it possible to transmit messages of a size equal to the transmitter

DMA buffer back-to-back. The peak data throughputs displayed in Figures 64 and 65 in section 6.3.3 would be hard to achieve, due to the PCI arbiters' fair access algorithms. These algorithms ensure high priority resources would not monopolise bus access [151]. The PCI specification 2.1 [41] specifies criteria for the operation of a 'fair access' arbitration mechanism [151]. It leaves the implementation of this to the PCI chipset designer, producing variations between systems [144].

The addition of extra links or the substantial increase in the communications link data rate could result in the FT-PCI-OSLi monopolising PCI bus access to the detriment of other bus users. As an example, the first generation Myrinet/PCI host interface [81] linked an 8-bit parallel Myrinet link to a 32-bit 33MHz PCI bus for use in connecting processing nodes in MPPs. The Myrinet throughput of 1.28Gbits/s (per direction) was significantly larger than that of the 1.056Gbits/s of the PCI bus. In such a situation, the PCI bus was the performance bottleneck, irrespective of its use, by other bus users. The Myrinet link never reached saturation whilst the PCI bus could not devote its full bandwidth to the interface as it must be shared. Later generation Myrinet/PCI host interfaces utilised 64-bit 66MHz PCI buses allowing 2Gbits/s (per direction) links to the PCI bus, whose maximum throughput was now 4.224Gbits/s. Although the PCI bandwidth was increased to over twice that of the Myrinet bandwidth, the Myrinet communications channels might still not saturate due to PCI bus access by other users.

Despite the shared nature of the PCI resource, its difficulty in predicting and guaranteeing bandwidth and the possibility of data starvation on the communications link, the PCI bus was still favoured for the following reasons:

- It was an industry-wide specification permitting a generic processor independent I/O bus based connection to many different platforms.
- A maximum data throughput of 132MBytes/s was considered high enough for many applications, including real-time parallel, embedded networks such as the FT-SARNet, with scope for upgrades.
- The burst mode operation of PCI transactions allowed rapid data transfer across the shared resource, increasing the frequency of accesses.

The total PCI latency incurred in initiating transfers was hard to predict due to its dependency on many variables, including other FT-PCI-OSLi bus accesses and accesses by other PCI agents. The arbitration mechanism employed by the PCI chipset was shown to affect the ease and regularity with which PCI agents could be granted ownership of the bus [43]. Hardware testing of the FT-PCI-OSLi was performed on a single PC with both FT-PCI-OSLi and PCI-OSLi devices tested using identical hardware to determine their respective access latencies.

During the development of the PCI-OSLi [43], tests were performed on two different PCs, yielding different results. It follows therefore that the FT-PCI-OSLi performance would vary when used with different hardware and configurations. Altering the latency timer (section 5.2.1.1) of the FT-PCI-OSLi adjusted the number of double words transferred across the PCI bus per transaction, effectively altering the DMA buffer size and thus the efficiency of each burst, as shown in section 6.3.3. The ability to alter burst sizes, coupled with the PCI bus access uncertainties, meant that the FT-PCI-OSLi required adequate buffering to ensure data starvation did not occur on the communications links. The relatively slow communications link data rate ensured that data starvation would be unlikely, although multiple smaller accesses increased the bus acquisition and set-up overheads.

The maximum observed DMA transmission throughput of the FT-PCI-OSLi was lower than the theoretical maximum of 132MBytes/s due to several latencies incurred in initiating a transfer; these included:

- Acquiring ownership of the bus – 9 PCI clock cycles,
- Initiating a PCI transaction once granted bus ownership – 3 PCI clock cycles,
- Delay incurred in the PC fetching the first data word from memory and placing it on the PCI bus (Delay unknown but the use of the 'multiple memory read' command located in the Master / Target Controller (section 5.2.1), meant that the fetching subsequent words was not subject to a delay, unlike the 'memory read' command).

The bandwidth limitations of the FT-PCI-OSLi were not crucial in the target network, as a throughput of 132MBytes/s would be unattainable for any one device in the shared PCI bus. The latencies were of consequence for shorter message lengths where the delays were very large relative to the message transfer time across the PCI bus.

The DMA transmission results for the FT-PCI-OSLi observed the utilisation of the shared resource in a way that was previously overlooked. The large peak in area one of the DMA transmission throughput graph was not detected in previous research [144] during the development of the host system interface of the PCI-OSLi. This was due to a different definition being used in determining the end of the DMA transfer. The research conducted into the development of the FT-PCI-OSLi used a more accurate definition of DMA transfer termination. As a result of this, an advance was made in the understanding of the operation of the PCI bus and its interface to the host system interface of the FT-PCI-OSLi.

7.3 FT-SARNIC Performance

In a small scale FT-SARNet the majority of communications will be between FT-SARNIC end nodes and NTR-FTM08 routers. There will be a proportionate increase in communications between routers as the network scales. The performance of inter-router communications has been documented in previous research [37]. The ability of the end node to transmit and receive data depended on the rate at which it could access the node's memory and the rate at which data could be outputted onto the communications link. The minimum DMA access interval of the FT-SARNIC was equal to that of the SARNIC. This was due to the Bus Controller design remaining unchanged with the main differences between the designs located in the Communications Controller.

The results in chapter 6 demonstrate a significant improvement in communications link throughput due to the adoption of permission based flow control. This led to a shorter message duration for a given payload, thus increasing the frequency of memory accesses. Unlike the FT-PCI-OSLi the maximum interval

between FT-SARNIC memory accesses could be calculated (section 5.5.2.6) as the FT-SARNIC and the CPU were the only devices that utilised the memory bus. It should be noted that the FT-SARNIC possessed two DMA channels per direction but only a single bi-directional communications channel. This effectively implemented two hardware virtual channels.

The post-synthesis simulations of the FT-SARNIC and SARNIC designs utilised a single DMA channel per direction to simulate bi-directional communications. This used only two of the four available DMA channels as only a single bi-directional communications link was implemented due to logic constraints. Targeting the design to a larger device and implementing the second communications link would benefit the data throughput of the FT-SARNIC. This had the potential to double the amount of data that could be transmitted and received by the FT-SARNIC as the communications links could operate concurrently.

The SARNIC design implemented two communications channels due to the reduced functionality of the communications controller, but the FT-SARNIC omitted the second channel in favour of the ROM module. This was required to initialise the interface for testing as the use of loopback tests prevented the booting from an external device as the communications links were driven and received by the FT-SARNIC.

The SARNIC operated two bi-directional channels at 20Mbits/s link rate with only a 3.34% drop in processor performance [151], but the FT-SARNIC exhibited a 14.4% increase in bi-directional throughput at this link rate, compared to the SARNIC. This would increase the frequency of memory accesses. At a 39Mbits/s link rate, the difference in throughputs increased to 28.75% for larger message sizes (section 6.7.2), which would increase DMA activity further. Increased DMA activity would reduce processor performance as memory bus access alternates between the CPU and DMA when accesses were pending (section 5.5.1.1). The extent to which this occurred would be worth investigating, but requires new hardware.

7.4 Buffering Considerations

The FT-SARNIC utilised a cycle stealing DMA approach to data transfer, transferring a single word of data at a time. Such an approach required a single stage of buffering large enough to hold enough data for processing, during the time between two FT-SARNIC memory accesses, to prevent data starvation. As the FT-SARNIC utilised the same host system adapter as the SARNIC, the same 'fair chance' (see section 5.5.1.1) arbitration mechanism was used, interspersing DMA accesses between CPU accesses. The FT-SARNIC interfaced directly to the memory bus of the SA-110, being the only external component in competition for this resource, thus guaranteeing bandwidth at fixed intervals.

The packetiser and depacketiser of the FT-SARNIC possessed a one word deep DMA buffer in each direction, which held a data word whilst it was packetised and depacketised to and from byte format, respectively. However, the link interface buffer held sufficient data to keep the communications link saturated between DMA transfers. The Transmitter Link Interface Buffer used in the SARNIC design was four tokens deep in order to hide the interval between memory accesses. The credit based flow control mechanism utilised by the SARNIC meant that a single token would be sufficient to prevent data starvation, were the time interval between DMA accesses short enough. The permission based flow control adopted by the FT-SARNIC required the Receiver Link Interface Buffer to be 32 tokens deep, for reasons discussed in section 4.4.2.3. This was due to the ability of the FT-SARNIC to transmit data tokens back-to-back. Increased buffering was also required due to the increased throughput of the FT-SARNIC caused by the adoption of the new protocol.

The block transfer approach of the FT-PCI-OSLi only initiated a PCI transaction when the DMA buffer was full or the message was completed, whichever occurred first. Messages longer than the 64 double word capacity of the DMA required subsequent bursts to transfer the message following the emptying of the resource.

A two stage buffering strategy was required with the first stage necessary to maintain access efficiency and maximise throughput across the PCI bus. The second stage provided buffering to store data to pass on to the communications link. This

prevented data starvation and provided data for processing by the receiver during periods where data transfer was halted across the communications link. Figure 65 in section 6.3.3 showed how the capacity of the link interface buffer affected the characteristic of area 2 of the DMA transmission throughput graph.

Increasing the link interface buffer size to the levels used in the PCI-OSLi increased throughput in area 2 of the graph to 25MBytes/s. This gave a six-fold throughput increase for messages of 1kByte, but at the cost of increased buffer size: 32 times the size used in the FT-PCI-OSLi. Short messages, that could be transferred across the PCI bus in a single PCI burst, could monopolise the bus if sent back-to-back. Longer messages must be emptied from the DMA transmitter buffer before subsequent PCI bursts could refill it. This action was dependent on the communications link data rate. The ability to drive the communications link into saturation at a 42Mbits/s link rate will depend on the length of the messages being transmitted. The PCI chipsets arbitration mechanism could prevent saturation if shorter message lengths are used, due to PCI bus acquisition overheads.

If the FT-PCI-OSLi utilised a single buffering stage, similar to that of the FT-SARNIC, the PCI bus bandwidth would be utilised inefficiently due to the relatively large overheads incurred in initiating a PCI transfer for a 4 byte message. The FT-PCI-OSLi must acquire ownership of the shared PCI bus. Conversely the FT-SARNIC utilised idle memory bus cycles, with only the SA-110 CPU in contention for access to this resource. Similarly, transferring data from the FT-SARNIC to memory in a burst would be possible, but awkward. This was due to the difficulties in terminating the transfer without incurring inefficient bus usage, as mentioned in section 5.5.1.2. Eight double words of data could be transferred in a single cache line fill but PCI bursts of this length are relatively inefficient due to overheads reducing PCI bus throughput to 10.15MBytes/s. Performing data transfers of 64 double word bursts for the FT-SARNIC in a manner similar to that of the FT-PCI-OSLi would interfere with the SA-110 access to the memory bus, and as such cannot be considered a viable option.

7.5 Data Streaming

Both the FT-PCI-OSLi and the FT-SARNIC employed streamed data transmission: formatting data and outputting it onto the serial communications link as soon as it was transferred from the host systems memory. Streamed data transmission was favoured as it required minimal buffering and did not impose any limits on the length of transactions. An other advantage of streamed data transmission was the ability to disguise message-formatting delays by the injection of subsequent data tokens into the DMA buffer. An inconsistency of transfers across the host system interface could result in data starvation in a streamed network as the link interface of the FT-PCI-OSLi waits on data to transfer. In reception, the inability of the PCI agent to gain access to the bus would result in the DMA Receiver Buffer not emptying fast enough resulting in the need to suspend data flow across the communications link. Both data starvation and network back pressure result in inefficient communications link bandwidth utilisation.

An alternative strategy, as adopted by the Myrinet/PCI host system interface, was to transfer an entire packet across the PCI bus to a temporary data storage buffer. Transfer of the message from the buffer to the communications network began once the entire packet was held in this buffer. Buffered transmission will impose a limit on either the buffer capacity or the packet length, with consequent resource or overhead implications. Once a buffered transmission was stored in memory, it could be outputted onto the communications link, via the message processing logic, in back-to-back transmissions. This resulted in more efficient use of the communications link bandwidth. Buffered transmission can be advantageous in I/O bus based systems, where the presence of several competing entities make it impossible to guarantee bandwidth. The Myrinet/PCI host interface [81] communications link data rate, of 1.28Gbits/s, was already much higher than that of the 32 bit 33MHz PCI bus (1.056Gbits/s). This resulted in data flow problems in the Myrinet / PCI system irrespective of accesses by other PCI agents. For this reason, buffered transmission was required.

The streamed transmission SHRIMP [71] interface, whilst having a peak transfer bandwidth of 200MBytes/s, only passed data to other users via its communications

links at a rate of 33MBytes/s. This was due to the performance bottleneck of the EISA bus [98]. Streamed data transmission allowed the EISA bus to become saturated but utilised little of the available interface bandwidth [71].

The communications link throughput of the FT-PCI-OSLi was sufficiently low with respect to the PCI bus throughput. Data starvation and network back pressure was unlikely due to the FT-PCI-OSLi serial format. The addition of other communications channels would increase the total communications link bandwidth to levels that could cause data starvation / network back pressure. This is discussed as a potential avenue of further research in section 8.2.2.

7.6 Interface Coupling

The FT-SARNIC interfaced to the SA-110s memory bus, creating a high performance, processor specific interface, tailored to that particular processor. Its tightly coupled host system interface increased communications efficiency due to its close proximity to the processor, but reduced the design generality. Migrating the FT-SARNIC design for use with a different microprocessor could require substantial design alterations, due to the interface design being optimised for the SA-110 timing requirements. However, the modular hierarchy of the FT-SARNIC would permit the alteration of either the processor interface, or the network interface without altering the other design unit.

The FT-PCI-OSLi, being an I/O bus based interface, sacrificed efficiency and therefore performance for the ability to provide a processor independent interface. The FT-PCI-OSLi must share the PCI bus bandwidth with the other PCI agents attached to the bus. The FT-SARNIC connected to the SA-110s memory bus and therefore only competed with the CPU for bus access. The FT-SARNIC could only transfer data during processor idle cycles to prevent communications interfering with the processor's memory accesses and therefore computational abilities. The FT-PCI-OSLi could request ownership of the PCI bus at any time but the amount of bus access granted depended on many variables, such the number of devices accessing the

bus, the activity of these devices and the arbitration mechanism utilised by the PCI bus arbiter.

The PCI bus standard, being a recognised transfer protocol, had a defined bus acquisition sequence in order to set up a PCI transaction between two agents. This procedure incurred significant overheads onto small transactions, making the PCI bus better suited to the transfer of large amounts of data. The FT-PCI-OSLi offered the FT-SARNet an interface to a far wider range of processors than previously available.

7.7 Virtual Channels

The FT-PCI-OSLi design built significantly on the previous PCI-OSLi interface by providing a hardware virtual channel capability to deal with the arrival of both expected and unexpected out of order message arrivals. Expected message IDs were loaded into the CAM prior to the message arrival. Headers of messages recovered from the serial communications link were compared with the contents of the CAM, a match being generated if the message was expected. CAM searches operated concurrently, making for fast comparisons, irrespective of CAM size, and providing an easily expandable solution.

The use of virtual channels eliminated the need for processor intervention when a valid message arrived at the receiver ahead of the expected message. Message arrival ahead of time triggered an interrupt and required the removal of the expected message information from the receivers' message information buffers. This was then replaced with the received message information. The virtual channel functionality of the FT-PCI-OSLi demultiplexed a single DMA channel to one of three message 'class' locations in memory. This method was found to be more effective than the twin DMA channels utilised by the FT-SARNIC.

The FT-SARNIC had two receiver DMA channels, to which packets belonging to one of two messages could be routed to, before being transferred to the SA-110s SDRAM. The FT-SARNIC could handle incoming packets that alternated between two messages. It required intervention when a third message was received, as

software must be used to decide which message must be replaced with the third message. This could be performed more efficiently if a 'least recently used' algorithm was implemented. The implementation of the virtual channel functionality of the FT-SARNIC was performed using logic elements due to the memory constraints of the Flex 10K50 PLD. Available resources limited the number of virtual channels to two per direction. Updating the target tPLD technology for implementation of the FT-SARNIC to the Apex 20K device family [134] would provide more logic resources in addition to increased operating speeds.

7.8 Modified Message Router Protocol

The ICR-C416 based message routing protocol used in the PCI-OSLi and SARNIC devices was abandoned in favour of the improved protocol used by the NTR-FTM08. The new protocol permitted the transmission of information relevant to the operational status of the communications links over the data link. This removed the need for the inefficient and unscalable control link used by the SARNIC and PCI-OSLi in the ICR-C416 based network. The ICR-C416 protocol suffered in event of link failure due to the inability of a node to detect the difference between an idle link and a faulty link. In event of link failure the nodes at either end could stall indefinitely, as discussed in chapter 3. Even if a mechanism existed to detect a stalled link, the node detecting this could not communicate the information to the other end of the link.

The periodic transmission to reaffirm link activity was similar to that used by DS links [79], but idle tokens were transmitted at intervals to reduce switching (see section 3.5). This provided both sides of a communications link with a fault detection mechanism, and confirmed link status. Further power savings, desirable in portable embedded applications, were available via the link dormancy mode of operation. The ability to shut down unused links was a marked improvement over the simple ICR-C416 based protocol. The FT-SARNIC was able to distinguish dormant links from stalled or faulty links, and return them to an active status with a fast and simple start up sequence.

The removal of the 256-byte maximum packet size enabled more efficient use of the communications link bandwidth and signalled an expansion in target applications from short, control messages to include longer communications. This was reflective of the increasingly compact nature of modern multiprocessor networks as well as the increased performance requirements of embedded systems.

Altering the communications link's flow control mechanism from credit based to permission based increased the data throughput of the FT-SARNIC and FT-PCI-OSLi interfaces due to the removal of acknowledge tokens. This action permitted back-to-back bi-directional data flow. It freed two bits per token for the transmission of data and utilised bandwidth previously wasted due to the need to interleave data and acknowledge tokens, as outlined theoretically in section 3.4.1 and confirmed in practice in chapter 6. The new flow control mechanism benefited the fault detection and recovery strategy as the loss of tokens no longer caused the link to stall with no means of resetting. Lack of link activity was used to indicate link failure. Receipt of a 'connection request' token was used to reset the receiving node's link interface state machine.

The addition of three different delimiter tokens to denote 'End of Message', 'End of Packet' and 'Exceptional End of Packet', aided the process of active packet recovery by providing the receiving node with information on the received message. The latter token informed the node that an error was detected during the transmission of the message and resulted in premature termination.

The distribution of fault detection and recovery features throughout the network permitted a more scalable fault detection and recovery solution with faster response times and the responsibility for each link devolved to the nodes at each end of it. Any unexpected behaviour on a link, which did not conform to the expected link traffic for that particular link state, would reset the link interface state machine. This action forced a node to transmit a connection request token, the receipt of which reset the state machine for the node at the other end of the communications link.

It should be noted that the hardware tests all utilised bi-directional communications, due to the nature of the loopback tests. These tests showed the

performance of credit based flow control, utilised by the SARNIC and PCI-OSLi, to be inferior to that of permission based flow control used by the FT-SARNIC and FT-PCI-OSLi. Uni-directional credit based transfers, which were more likely to occur to and from end nodes in a control-based network, could achieve 13 bits per byte (section 2.3.1), and thus achieve back-to-back data transmission. Communications between routers were more likely to be bi-directional as more messages will be active in the central branches of the network. Applications utilising uni-directional credit based communications will still be subject to bi-directional communications constraints, reducing their performance in the areas of the system with the heaviest workload.

7.9 Proprietary Vs Custom Prototype PCI Interfaces

With several commercially available PCI interfaces available, the benefits and costs of designing a custom interface in house must be considered when a proprietary one could be bought. A System on Chip (SOC) solution was desirable, eliminating many interfaces as additional functionality would be required to link the device to the communications link. This was not a problem as PCI cores were obtainable for PLDs, implemented in embedded memory, leaving on-chip programmable logic for the implementation of a communications link interface. These programmable interfaces, such as the Altera PCI Master/Target MegaCore [152] and Xilinx PCI LogicCORE [153] might seem attractive options initially but when design flexibility was considered, PCI IP cores lose much of their appeal. The interface developer buys the IP but does not necessarily buy the right or ability to modify it. The developer could buy a licence to customise the core, at a cost, in some cases, but otherwise no modifications were possible. The ability to modify the interface, tailoring it to the application, can make a significant difference to its operation. The Altera PCI MegaCore provides a single DMA channel, with a buffer capacity of 16 double words. Figure 60 showed that the DMA transmission throughput for messages of this size was 19.2MBytes/s. This would become the peak throughput for area 1 of the DMA transmission throughput graph. Figure 55 showed that the efficiency of FT-PCI-OSLi PCI accesses would become 84.2%, reduced from 94.7% for a DMA buffer capacity of 16 double words. A PCI interface utilising this core would be significantly

less efficient than the PCI interface of the FT-PCI-OSLi. The implementation of a single DMA channel would require sharing the DMA buffer. This would delay transfers until the buffer was emptied and would reduce efficiency, removing the performance gains obtained from full-duplex internal communications in the FT-PCI-OSLi.

Most PLD based PCI cores provide only the master / target controllers for the PCI interface, without DMA support as standard, this important feature costing extra. The user may benefit with the ability to design custom DMA channels, tailored to the communications link, at a cost of extra design effort. When the time and effort spent evaluating different IP, and integrating the IP with the custom designed DMA interface, is taken into account it may prove easier to design the entire interface. Many problems could be encountered when attempting to interface third party IP to other designs. The increased design time incurred with a fully customised interface was less important as time-to-market is less crucial in research, than in the commercial electronics field.

Other PCI IP interfaces, such as that available from PLD Applications [154], provide the logic to implement DMA channels but the responsibility for DMA buffer implementation falls to the developer. Internal implementation of these buffers requires the availability of large amounts of on-chip buffering whilst external implementation would represent a major design challenge in order to meet the stringent PCI timing requirements required to achieve zero wait-state burst reads and writes.

The ability to modify the PCI interface of the FT-PCI-OSLi was a major advantage compared to proprietary IP cores, as the source code was readily available and could be tailored and updated for optimum transfer over the PCI bus. Features not available in proprietary IP could be implemented as desired and unnecessary features omitted.

The use of other PCI interfaces, such as the Myrinet/PCI host interface may seem attractive, due to its bandwidth of 1.28Gbits/s. This solution would require the design of an interface to link the Myrinet link with the serial communications channel

utilised by the NTR-FTM08, incurring an extra stage of message formatting and latency. Additionally, the serial communications link would become the performance bottleneck of the interface and the bandwidth figure of 1.28Gbits/s would never be attainable.

A fully flexible, easily modifiable, licence free source code solution, already tailored to the format of the communications link, was readily available. Were the project to start at the time of writing, with the advances that have been made in available PCI interfaces, a different conclusion may have been made. Hence, during development of the PCI-OSLi the option of using a third party PCI interface was considered and rejected due to the limited choices and features of the available IP. Since then the boom in off-the-shelf solutions has resulted in the increase in the specifications, performance and available features of PCI IP, coupled with reduced costs. The PCI-OSLi itself could be considered IP, with many advantages compared to a proprietary PCI IP interface.

7.10 66MHz PCI Bus Operation

The data throughput of the PCI bus could be doubled to 264MBytes/s by increasing the clock rate of the bus from 33MHz to 66MHz. Post-synthesis timing analysis of the FT-PCI-OSLi revealed the maximum PCI clock frequency to be 74.88MHz. This makes 66MHz PCI operation theoretically possible, until the PCI timing requirements are considered. The PCI clock period is 15ns at this frequency, which cannot meet the PCI timing requirements. Certain PCI signals, most notably the initiator and target ready signals 'nIRDY' and 'nTRDY' (see Appendix B), must meet strict timing requirements in order to synchronise PCI transactions. These signals had a maximum set-up time of 7ns and a maximum clock-to-output delay of 11ns [41]. The FT-PCI-OSLi design had a set-up time (t_{su}) of 5.091ns and 5.404ns for the 'nIRDY' and 'nTRDY' signals respectively. The clock-to-output time (t_{co}) of the FT-PCI-OSLi was 10.593ns and 11.909ns for the 'nIRDY' and 'nTRDY' signals respectively. The time of the clock period when the signals could be reliably sampled was 14.316ns and 12.687 for the 'nIRDY' and 'nTRDY' signals respectively with the 33MHz PCI bus. The time when the signals could not be sampled, was 15.684, and

17.313, for the 'nIRDY' and 'nTRDY' signals respectively. These times exceed the 15ns clock period for the 66MHz PCI bus. Therefore, to achieve 66MHz PCI operation, the t_{SU} and t_{CO} times must be significantly reduced.

Faster designs can be realised through effective targeting of design effort towards specific areas. As the devices used to implement the design increase in performance with advances in PLD technology, designs will benefit from speed increases. It could be argued that the FT-PCI-OSLi and FT-SARNIC interfaces are at a disadvantage in comparison to commercial interfaces as the latter are implemented on faster ASIC technology. Although as section 4.5 pointed out, the difference in performance between the two technologies is reducing [137].

8 CONCLUSIONS AND FURTHER WORK

8.1 Conclusions

This thesis has documented research into multiprocessor systems with a view to enhancing fault tolerance, which led to the development of two network interface devices. These were designed to form building blocks in a router based serial communications network with increased fault tolerance. The FT-SARNet network was targeted at real-time distributed embedded multiprocessor applications. The interface devices could be utilised to produce a novel decentralised fault handling communications network linking PCs and StrongArm processors. The system would allow RISC and general-purpose processors to operate as processor nodes in the same network, increasing system flexibility and applications. Interprocessor bi-directional data throughput was increased compared to previous non-fault tolerant devices due to flow control modifications. The addition of hardware fault tolerance features provided the embedded network with the ability to detect, isolate and recover from several fault scenarios.

Interprocessor communications in this embedded multiprocessor network utilised custom NTR-FTM08 8-channel off-the-shelf, hardware message routers. One interface was designed to connect StrongArm SA-110 processors to the router network using a protocol that facilitated the implementation of improved fault tolerance features. This was named the FT-SARNIC. The functionality and performance of the FT-SARNIC was verified via post-synthesis simulation and a synthesised design was produced ready for hardware implementation. A second interface, called the FT-PCI-OSLi, was designed to link a general purpose PC, via the PCI bus, to the network and was implemented in hardware. Both interfaces built on previous non-fault tolerant prototype designs, making significant alterations to accommodate the features aimed at enhancing the fault detection and recovery abilities of the NTR-FTM08 routing protocol.

Nodes in the FT-SARNet were required to exchange information relating to the operational status of the link in order to provide distributed fault tolerance. This required different design features to the previous SARNIC and PCI-OSLi interfaces, as outlined below:

- Credit based flow control was abandoned in favour of a permission-based mechanism. This eliminated the acknowledgement token system, made better use of the link bandwidth and reduced the chance of stalled links due to loss of acknowledgements.
- Back-to-back data transmission was achievable, subject to the availability of receiver buffering resources. Stop and Go flow control tokens utilised the data path to inform the transmitting node of the link's operational status and prevent buffer overflow.
- Idle tokens were transmitted periodically, in the absence of link activity, to reaffirm the integrity of the data path. Stop and Go flow control tokens were used to provide validation of link status.
- Inactive links could be configured as dormant, in order to reduce power consumption, re-activating upon the command to transmit a message. Link dormancy provided a means of distinguishing between idle and disabled links that was previously impossible.
- The addition of a handshaking start-up procedure, to ensure that both end nodes on a link were ready to transmit data, provided a means of resetting individual links on detection of irregular link activity.
- Altering the message format, by eliminating the need to packetise data in multiples of 256-bytes, reduced the message overheads. It also signalled an expansion in applications from shorter control-style messages to encompass PC based applications that could require longer data communications.
- Altering the message format, from header-length-payload to header-payload-terminator, enabled the truncation of messages affected by link failure. This provided the receiving node with a warning that the message had ended prematurely and a reset had occurred on that link, terminating that particular communication.

The FT-PCI-OSLi design built on the previous non-fault tolerant device, with the new and modified submodules of the design simulated in software to verify functionality, before being incorporated into the design. Hardware testing and debugging of the interface was aided by the in-system re-programmable SRAM based PLD. This device could be programmed by an EEPROM allowing hardware

modifications and experimentation without the need for investment in additional hardware. The FT-PCI-OSLi design was implemented on an Altera Apex 20K200E PLD mounted on a custom interface PCB containing transmitter and receiver differential line drivers for the communications links. The operation of the FT-PCI-OSLi communications channels was tested at several data rates, up to a maximum sample clock frequency of 64MHz, giving a 42Mbits/s data rate. A PCI-OSLi interface was implemented on identical hardware for similar testing, eliminating many differences in implementation and enabling comparisons to be made between the two designs.

The FT-PCI-OSLi improved on its non-fault tolerant predecessor by implementing hardware virtual channels via the use of Context Addressable Memory (CAM). The CAM was used to store up to sixteen expected message IDs at any one time, allowing pre-loading in anticipation of their use. This reduced the need for processor intervention to replace message IDs in the DMA channels in event of the arrival of a new message. The use of CAM provided concurrent search capabilities and a scalable solution whilst minimising logic usage via embedded memory implementation.

The key conclusions obtained from the hardware tests of the FT-PCI-OSLi are presented below:

- The FT-PCI-OSLi outperformed the PCI-OSLi in terms of efficient use of communications link bandwidth, due mainly to the adoption of the new flow control protocol.
- The efficiency of PCI bursts was improved upon, reducing the PCI bus set-up latency from five clock cycles to three.
- The transmission throughput of the PCI interface reached a maximum of 92.8MBytes/s when the message size is equal to that of the DMA transmission buffer. This was a significant improvement in the understanding of the host system interface of the FT-PCI-OSLi and PCI-OSLi devices as previous research during the development of the PCI-OSLi did not observe the behaviour of the interface to the same degree of accuracy. This was due to a different measure of DMA transmission duration being used, leading to a less accurate representation of the DMA transmission characteristic. The previous incorrect maximum DMA

throughput observed during development of the PCI-OSLi was only 25Mbytes/s. The PCI throughput of the FT-PCI-OSLi has not increased in comparison to that of the PCI-OSLi, but the more accurate measurement enabled its characteristics to be observed more accurately than was previously performed, demonstrating the Area 1 peak that was overlooked during development of the PCI-OSLi.

- The DMA transmission throughput was always sufficiently high to saturate the communications link, even when back-to-back transmission was used. This indicated that the communications channel was the system bottleneck. It is possibile that the arbitration mechanism of the PCI chipset could prevent the communications channel from saturating at smaller message lengths. This would require verification but could not be taken as definitive as the arbitration mechanism could vary between systems, as documented in section 7.2.
- A correlation was found between the DMA and link interface buffer capacities and the DMA transmission throughput.
- The fault detection and recovery strategy showed the FT-PCI-OSLi could prevent buffer overflow and stalled links due to lost acknowledge tokens. This was an improvement on the ICR-C416 based router network utilised by the PCI-OSLi and SARNIC. The FT-PCI-OSLi could detect and recover from faults such as disconnected network connection, packet arrival out of order, incorrect message length, synchronisation errors and the delivery of messages to the wrong address.

Post-synthesis simulation of the FT-SARNIC, with similar tests performed on a post-synthesis simulation of the SARNIC revealed similar improvements in the utilisation of the communications link bandwidth, again due to the alterations to the link protocol. The host system interface of the FT-SARNIC was left unchanged, leaving the communications between the FT-SARNICs DMA channels and the SDRAM of the SA-110 operating in an identical manner to that of the SARNIC.

The FT-SARNet improved on the previous non-fault tolerant SARNet network in the following ways with respect to the parameters used to gauge network performance described in section 2.1:

- **Bandwidth**: The communications network gained significantly due to the adoption of permission based flow control. The design of the host system interface

of the FT-PCI-OSLi remained unchanged but a greater understanding of its operation was obtained.

- **Latency**: The initialisation latency incurred between acquiring ownership of the PCI interface and commencing a transfer was reduced by 40% due to hardware refinements to the master/target controller in the host system interface of the FT-PCI-OSLi.
- **Processor Overhead**: The FT-SARNIC and FT-PCI-OSLi interfaces reduced software involvement from their respective processors when handling faults. The FT-PCI-OSLi reduced the need for processor intervention, following out of order message arrival, due to the implementation of hardware virtual channels.
- **Scalability**: The control port monitoring mechanism of the ICR-C416 based network was replaced by a fully scalable fault detection and recovery strategy. A solution valid for networks of any size could be realised by devolving fault detection and recovery features so that responsibility for fault tolerance over a commmunications link lay with the nodes at either end of the link.
- **Reliability**: Enhanced fault tolerance functionality permitted the detection of a wider range of network faults. Hardware implementation reduced software overheads. An automatic recover strategy enabled a rapid response, of particular use in remote systems.

The strength of a message processing interface utilising a shared bandwidth resource such as the FT-PCI-OSLi or the FT-SARNIC is its ability to saturate the dedicated communications link whilst utilising minimal bandwidth of the shared resource. Access to the shared resource must be guaranteed, despite its dependence on many variables. Real time systems require message delivery within a short enough time for the information to still be useful. Access contentions to shared resources delay this process. The communications link was the performance bottleneck of the FT-PCI-OSLi design, as expected. This was preferable to the shared resource being the bottleneck, as the communications link throughput would not hinder bus access by other users.

A possible description of fault tolerance in the NTR-FTM08 embedded routing network could be the ability to enable faults to occur without impairing the network performance to a significant extent. This is possible due to features such as group

adaptive routing, which permit faulty links to be bypassed. The FT-PCI-OSLi and FT-SARNIC network interfaces are end nodes of the routing network, and as such are either the source or destination of a message, and thus cannot be bypassed. Both interfaces possess a single bi-directional communications channel, which if faulty isolates the processor node from the network. Such situations require the link failure to be detected and fixed rapidly. The PCI-OSLi and SARNIC devices did not possess any means of doing this, relying on software and thus imposing large overheads, in implementing any form of fault detection and recovery. A robust network must possess the ability to detect link failure and automatically recover from it.

8.2 Further Work

8.2.1 Realisation of the FT-SARNet

The research concluded with the design and synthesis of the FT-SARNIC interface and the hardware implementation of the FT-PCI-OSLi. The FT-PCI-OSLi was tested using the loopback tests, effectively sending messages to itself. To enable the construction of an embedded FT-SARNet network one must integrate the processor nodes (comprising a processor, interface and memory) into an NTR-FTM08 router network. This would permit research to assess the effectiveness of the network as a whole and identify the weak links of the FT-SARNet as scope for further improvements. The FT-SARNIC design is ready for hardware implementation and the other two building blocks of the FT-SARNet, the FT-PCI-OSLi and the NTR-FTM08, have been successfully implemented in hardware.

The main focus of effort required in construction of the FT-SARNet is in the software levels of the design, as software designed for use with the SARNIC and PCI-OSLi requires significant modification for use with the improved fault tolerance devices. In particular, work is required to implement support for the features and actions that must be taken to enhance fault tolerance. The software supporting the network interfaces must be minimised in order to reduce operating system intervention, thus minimising overheads and maximising the computational abilities of the parallel network.

Hardware realisation of the FT-SARNet would permit research into permission based throughput in multi-router networks for various different network topologies. Such tests would advance the understanding of the efficacy of permission based flow control. They would show how network topology could influence traffic patterns, which could be used to minimise the necessity to suspend traffic flow across links, thus increasing the efficiency of the system as a whole.

The testing of the FT-SARNIC implemented in hardware would result in performance characteristics that were very similar to those achieved in the post-synthesis simulations documented in chapter 6. This is due to the accuracy of the proprietary simulation and synthesis tools used in the development of the designs. Detailed knowledge of the internal architecture of the PLD permits a highly accurate analysis of the design to be obtained as the compiler can consider the delay through each gate in the design. Any differences between the simulated and observed behaviour of the device would mainly be due to timing differences in external signals (such as those from the microprocessor or SDRAM).

8.2.2 Additional Communications Channels

The addition of more communication channels to the FT-PCI-OSLi interface is possible without monopolising the PCI bus. At 20Mbits/s data rates a modified PCI-OSLi design used in a Transputer based network has been shown to be capable of driving four bi-directional communications channels operating in saturation (eight DMA channels in total). This was achieved without preventing access to the bus, due to the PCI bandwidth being far greater than that of the communications links. At 42Mbits/s data rates the maximum transmitter DMA throughput exceeds 92MBytes/s. In area 3 of Figure 64 (section 6.3.3), the throughput was limited to 3.8MBytes/s, due to the link data rate. A total of 24 bi-directional DMA channels, 12 in each direction, could theoretically be supported.

The addition of extra channels requires the duplication of the link interface and data flow parts of the design. Development of the NTR-FTM08 router revealed that the duplication of the message formatting parts of the design can lead to large increases in resource usage unless functionality can be shared amongst devices.

Sharing functionality is rarely possible if communications links operate concurrently. The state machine in the DMA controller also needs modifying in order to arbitrate not only between the direction of DMA transfer but also which of the channels is transmitting or receiving. Each channel will require a DMA channel in either direction in order to accumulate data for transfer to / from memory. These channels cannot be shared as bi-directional data transfers can take place concurrently on all communications channels, filling and emptying the receiver and transmitter DMA buffers onto the communications channels. The more channels that are added, the greater the likelihood of a DMA channel having to wait for completion of a previous transaction to access the bus to transfer its contents to memory.

It should be noted that the replication of features in a design increases the fan-out of signals, which can lead to timing problems, as was noted in the development of the FT-PCI-OSLi. The ability to modify source code enables such problems to be solved; whereas the use of off-the-shelf components, whilst possessing sufficient resources to enable expansion, may not permit a solution due to timing issues that cannot be resolved due to the inability of the user to modify the design.

8.2.3 Interface Adaptation for use with Alternative Processors

The rapid advancement of processor technology shortens the amount of time that the modern processor is utilised before being upgraded. In order for the embedded network to track these processor developments, the network requires frequent updating of processor support. Easily adaptable IP is a necessity in such networks as it enables as much of the original design to be retained, whilst inserting a proven module to fulfil the remaining requirements.

The modular nature of the FT-SARNIC and FT-PCI-OSLi interfaces permits updating of the host interface without requiring alteration to the network interface. However, modifying the FT-SARNIC for use with another processor would require significant design effort due to the host system interface being optimised to the StrongArm SA-110.

The FT-PCI-OSLi is designed to interface to any processor with a 32-bit 33MHz PCI bus, irrespective of the processor model and operational parameters. As general purpose processor specifications advance, so too will the implementation of the PCI bus, from 32-bit, 33MHz to 64-bit, 66MHz. The issues regarding the interfacing of the FT-SARNet to a 66MHz PCI bus were discussed in section 7.1.8. It was concluded that this would currently require the set-up and clock-to-output times of the PCI signals of the FT-PCI-OSLi to be significantly reduced in order to meet the timing requirements. Further advances in PLD technology result in the introduction of device families with smaller feature sizes, increased gate counts and higher switching speeds. Migrating the design to a newer PLD technology would reduce or even eliminate the amount of optimisation required to meet the timing requirements. Implementing a 64-bit, 33MHz FT-PCI-OSLi interface requires only logic duplication and a widening of the bus. This PCI bus implementation would double the maximum theoretical PCI bus bandwidth to 264MBytes/s whilst requiring no additional optimisation of the interface. The FT-PCI-OSLi PCB would require redesigning, with a 64-bit PCI extension to the PCI edge connector and connections made from the relevant signals on this to the PLD.

8.2.4 Enhanced Virtual Channel Capabilities for the FT-SARNIC Interface

The CAM based Virtual Channel Message Store utilised in the FT-PCI-OSLi could be implemented in the FT-SARNIC to eliminate the message channel allocation problems encountered by the latter when handling three or more different incoming messages. The twin DMA channels and logic intensive message channel allocator could be greatly simplified, considerably reducing logic usage in the Communications Controller.

8.2.5 FT-SARNIC Asynchronous Interface

The data paths of the FT-SARNIC interface utilised an asynchronous interface in both directions of data flow between the message formatting modules and the link interface buffers, as Figure 49 in section 5.5.2.4 shows. The asynchronous interface was responsible for synchronising control signals for the link interface buffer read and

write operations (that were generated using the sample clock) to the core clock. The asynchronous interface was housed in the packetiser and depacketiser modules for the transmitter and receiver channels, respectively. Implementation of this circuitry was complex, requiring large amounts of logic, which could be removed if the link interface buffers were implemented using dual-port RAM. This would permit the control signals to the write side of the link interface buffer to be synchronised to the processors core clock whilst the read side of the link interface buffer is synchronised to the sample clock. This effectively moves the clock domain boundary to the link interface buffer and has the benefit of requiring no extra logic, as the clock synchronisation is performed within the software-based dual-port RAM Megafunction.

Dual-port RAM was not available when the SARNIC was under development, as the Flex 10KA family did not support this feature. The Flex 10KE and Apex 20KE device families can implement such functions.

8.2.6 System On a Programmable Chip Solution

Further development could include the implementation of a complete SARNode on a single programmable device to achieve a fully integrated system on a chip solution (as noted in section 1.1). This would comprise of an ARM based processor core, an FT-SARNIC, an NTR-FTM08 and an FT-PCI-OSLi interface, subject to available logic resources. The Excalibur device [69] is one of the first solutions in attempting to achieve a balance between the advantages of speed and gate density of dedicated hardware, and the flexibility of programmable logic. The Excalibur PLD range feature an 200MHz ARM 922T based hard core CPU which occupies a small area of the PLD design, leaving the remainder for implementation of up to 38,400 LEs and 320k memory bits. The FT-PCI-OSLi utilised less than 8% of this total, and the FT-SARNIC used even less. Such a device is feasible, although the resources required by the router, which are substantial due to the high level of replication, may require a reduction from 8 to 6 channels, with current technology.

The Excalibur based ARM processor [69] utilises an AMBA (Advanced Microcontroller Bus Architecture) bus to interface to external components, raising the

possibility of an SOC solution comprising a general purpose embedded processor and the FT-PCI-OSLi, with many of the host system interface features of the latter aided by the architecture of the Excalibur device. The AMBA bus possesses three buffers: address, read data and write data in each direction. The FT-PCI-OSLi had to multiplex these buffers together onto a bi-directional bus whereas the AMBA bus reads data directly from the buffers onto the bus. This avoids the bus multiplexing issues encountered during development of the FT-PCI-OSLi, reducing timing constraints and permitting set-up times to be met. AMBA compliant high performance buses enable the separation of communications from computational activities, one of the key principles of interprocessor communications.

The Excalibur devices would enable a FT-SARNode SOC solution to be achievable through the elimination of much of the interface functionality currently implemented using programmable logic. Functions such as the interrupt controller, UART, timer and debug logic are implemented in the embedded hardware processor 'stripe' [69], freeing the logic elements used to implement these functions for other uses. Additionally, an internal SDRAM controller can address up to 512MBytes of memory at speeds up to 266MHz and an expansion bus can address 32MBytes of memory in up to 4 external devices. Implementing an FT-SARNode on an Excalibur device would permit the majority of the SARNode functionality to be implemented on the embedded 'stripe', leaving the PLD area free for the implementation of the communications controller and bus controller modules.

Publications

S.Triger, B.C.O'Neill & S.Clark *"Multiprocessor Communications for Embedded System Applications"* Postgraduate Research into Electronics and Photonics 2001 (PREP), Keele University, April 2001, pp 65 - 66.

S.Triger, B.C.O'Neill & S.Clark *"Adapted OS link / DS link protocols for use in Multiprocessor Routing Networks"* Communication Processing Architectures 2001 (CPA), University of Bristol, September 2001, pp 37 - 48.

References

1 Valiant, L.G., "General Purpose Parallel Architectures", Technical report TR-07-89, Aiken Computation Laboratory, Harvard University, Prentice-Hall, April 1989.

2 Fox, G.C., Johnson, M.A., Lyzenga, G.A., Otto, S.W., Salmon, J.K., Walker, D.W., "Solving Problems on Concurrent Processors: Volume 1 General Techniques & Regular Problems", USA, Prentice-Hall International Inc., 1988, pp. 17 – 38.

3 Almasi, G.S., Gottlieb, A., "Highly Parallel Computing", The Benjamin/Cummings Publishing Company Inc., ISBN 0-8053-0177-1, pp 45-47, 1997.

4 Patterson, D., "Reduced Instruction Set Computers", Communications of the ACM, Vol. 28, No. 1, January 1985, pp. 9-21.

5 The Transputer Handbook, INMOS Ltd (now part of SGS Thomson), 2nd Edition, Trowbridge, 1989.

6 Hotchkiss, R., Wong, K.L., O'Neill, B.C., Coulson, G.C., Clark, S.,Thomas, P.D., "The Building Blocks for a Parallel Network Incorporating the StrongArm Microprocessor", The 1998 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'98), Las Vegas, July 1998, pp. 1863-1870.

7 Dario, P., Carrozza, M.C., Marcacci, M., D'Attanasio, S., Magnami, B., Tonet, O., Megali, G. "A Novel Mechatronic Tool for Computer-assisted Arthroscopy", IEEE Transactions on Information Technology in Biomedicine", Vol. 4, No. 1, March 2000, pp. 15-29.

8 Silva, L.M., Silva, J.G., "Global Checkpoints for Distributed Programs"; Proceedings 11th Symposium on Reliable Distributed Systems, Houston, TX, USA, 1992, pp 155 – 182.

9 IC Routing Ltd, "16 Port Dynamic Routing Switch for Transputer Link", Data Sheet, Version 1.3, 1996, pp. 1 – 3.

10 Hinton, J., Pinder, A. "Transputer Hardware and System Design", Prentice Hall International, (UK) Ltd., 1993, ISBN 0-13-953001-0 (pbk), pp. 1 - 17, pp. 142 – 154.

11 East, I "Parallel Processing with Communicating Process Architecture", UCL Press, London, 1995, pp 1 – 24, 41 – 83.

12 Welch, P.H., "The Role and Future of Occam", "Transputer Applications - Progress & Prospects", IOS Press, 1992, pp. 152-179.

13 Hoare, C.A.R. "Communicating Sequential Processes" Hemel Hempstead : Prentice Hall International, 1985.

14 Stone, H.S., "High Performance Computer Architectures"; Addison Wesley; 1987

15 Gelernter, D., "A DAG based Algorithm for Prevention of Store and Forward Deadlock in Packet Networks", IEEE Transactions on Computers, Vol. C30, No. 10, 1981, pp 709-714.

16 Joerg, C. F., Henry, D. S., "A Tightly-Coupled Processor-Network Interface", Computation Structures Group Memo 342, Massachusetts Institute of Technology, March 1992.

17 Wong, K. L., "A Message Controller for Distributed Processing Systems", PhD Thesis, The Nottingham Trent University, UK, June 2000.

18 O'Neill, B. C., et al. 'Design and exploitation of a 26 000 gate message routing device' Fifth EURCHIP Workshop on VLSI Design Training, Dresden, Germany, Oct 1994, pp 290 – 303.

19 Ellis, J.W., O'Neill, B. C., Clark, S. "A router design for T800 compatible Transputer arrays", Transputer Applications Vol 1(2), 1993, pp 12-18.

20 O'Neill, B. C., Wong, K. L., Coulson, G. C., Clark, S., Thomas, P. D., Cook, B.M., Keele University, Walker, C. P. H., 4Links., "A Low-cost High-performance Multicast Routing Switch Chip for Communication Networks", European Multimedia, Microprocessor Systems and Electronic Commerce Conference, Florence, ISBN 90-5199-385-4, Nov 1997, pp. 836 – 843.

21 "Clipbox Fact Sheet", Quantel. http://www.quantel.com (last checked 14 February 2002)

22 McKinley P.K., Xu, H. Esfahanian, A.H., Ni, L.M., "Unicast-Based Multicast Communication in Wormhole-Routed Networks", IEEE Transactions on Parallel and Distributed Systems, Vol 5, No 12, December 1994, pp 1252 – 1264.

23 Coulson, G.C. "An ASIC Implementation of a Multicast Message Routing Switch for Interprocessor Communications", PhD Thesis. The Nottingham Trent University, UK, September 1998, pp. 23, 68-70, 122-124.

24 De Carlini, U., Villano, U., "Transputers and Parallel Architectures - Message-Passing Distributed Systems", UK, Ellis Horwood, 1991, pp. 148 - 204.

25 Hinton, J., Pinder, A. "Transputer Hardware and System Design", Prentice Hall International, (UK) Ltd., 1993, ISBN 0-13-953001-0 (pbk), pp. 195 – 212.

26 Bakoglu, H.B., Grohoski, G.F., Montoye, R.K., "The IBM RISC System/6000 processor: hardware overview", IBM Journal of Research and Development, Vol. 34, No, 1, January 1990, pp 12 - 22.

27 Digital Equipment Corporation "Digital Semiconductor SA-110 Microprocessor - Technical Reference Manual", Maynard, Massachusetts, October 1996.

28 Mukherjee, S. S., Hill, M. D., "The Impact of Data Transfer and Buffering Alternatives on Network Interface Design", Fourth International Symposium on High-performance Computer Architecture (HPCA), February 1998.

29 Digital Semiconductors SA-110 Microprocessor Technical Reference Manual, EC-QPWLC-TE, DEC Ltd, Maynard, Massachusetts, USA, 1996.

30 Irwin, G.W., Fleming, P.J., "Real-Time Control Applications of Transputers", "Transputer Applications - Progress & Prospects", IOS Press, 1992, pp. 26-41.

31 O'Neill, B. C. et al,'An Interface Device to Support a Distributed Parallel System for the StrongARM Microprocessor', High Performance Computer Networks 98, Amsterdam, April 1998, pp 1047- 1050.

32 Wong, K. L. et al, "Interfacing StrongArm Microprocessors in a Parallel Network", Postgraduate Research in Electronics, Photonics & Related Fields (PREP'99), January 1999, pp. 382-385.

33 Gerla, M., Palnati, P., Walton, S., "Multicasting in Myrinet – A high speed, wormhole-routing network", IEEE GLOBECOM 1996. Communications: The Key to Global Prosperity; IEEE. Vol. 2, ISBN 0 7803 3336 5; New York, USA; pp 1064 – 1068.

34 Ni, L.M., McKinley, P.K., "A Survey of Wormhole Routing Techniques in Direct Networks", IEEE Computer, Vol. 26, No 2, February 1993, pp 62 – 76.

35 Dally, W.J., Aoki, H., "Deadlock-free Adaptive Routing in Multicomputer Networks Using Virtual Channels", IEEE Trans. Parallel and Distributed Systems; Vol. 4, No. 4, April 1993.

36 Duato, J. "A New Theory of Deadlock-free Adaptive Routing in Wormhole Networks", IEEE Transactions on Parallel and Distributed Systems, Vol. 4, No 12; December 1993, pp 1320 – 1331.

37 Hotchkiss, R. "A Fault Tolerant Multicast Message Routing Switch for Interprocessor Communications", PhD Thesis, The Nottingham Trent University, UK; Sept 2000.

38 Ingenieurburo Ingo Mohnen "BBK-PCI User's Manual", Version 1.1, May 1998, Address: Ingenieurburo Ingo Mohnen, Rottstrasse 33, 52068, Aachen, Germany.

39 Ingenieurburo Ingo Mohnen "BBK-PCI Light User's Manual", Version 3.2, September 1998, Address: Ingenieurburo Ingo Mohnen, Rottstrasse 33, 52068, Aachen, Germany.

40 Shanley, T., Anderson, D., "PCI System Architecture", 3rd Edition, Addison-Wesley Publishing Company, 1995, ISBN 0-201-40993-3.

41 "PCI Local Bus Specification, Revision 2.1s", PCI Special Interest Group, 1995.

42 Ng, J.H., O'Neill, B.C., Clark, S., "A PC Interface Board for Parallel ARM Processor Network", PREP 2000, IEE UK., ISBN 0 86341 3218, pp 469-474, April 2000.

43 Ng, J.H., "Message Routing Interface for Multiprocessor Networks", PhD Thesis, The Nottingham Trent University, UK, June 2000.

44 Liew, E.W.K., O'Neill, B.C., Wong, K.L., Clark, S., Thomas, P.D., Canr, R., "A Proposal for an Operating System for a Multi-processor StrongARM System", Concurrent Systems Engineering, ISSN 1383-7575, 1999, Vol. 57, pp 37-47.

45 Jones, A.M., Davies, N.J., Firth, M.A., Wright, C.J., "The Network Designers Handbook", IOS Press, Amsterdam, 1997, ISBN 90-5199-380-3, pp. 41 - 45.

46 Wong, K.L., "A Message Controller for Distributed Processing Systems", PhD Thesis, The Nottingham Trent University, UK, April 2000, pp. 17.

47 Ni, L.M., McKinley, P.K., "A Survey of Wormhole Routing Techniques in Direct Networks", IEEE Computer, February 1993, Vol. 26, pp. 62-76.

48 Kermani, P., Kleinrock, L., "Virtual Cut-Through: A New Computer Communication Switching Technique", Computer Networks, Vol. 3, No. 4, September 1979, pp. 267-286.

49 Bhoedjang, R., Ruhl, T., Bal, H.E., "User-Level Network Interface Protocols", IEEE Computer, November 1998, Vol. 31(11), pp. 23-39.

50 Welsh, M., Basu, A., Eicken, T. von, "ATM and Fast Ethernet Network Interfaces for User-level Communication", Proceedings of High-Performance Computer Architectures (HPCA) 3, San Antonio, TX, February 1997.

51 Coulouris, G., Dollimire, J., Kindberg, T., "Distributed Systems - Concepts and Design", 2nd Ed, Addison-Wesley, Wokingham, UK, 1994, ISBN 0-2016-2433-8, pp. 1-124.

52 Liebowitz, B.H., Carson, J.H., " Multiple Processor Systems for Real-Time Applications", London, Prentice-Hall, 1985, pp. 50-61, 104-123, 159-216.

53 Morrow, R., "Bluetooth: Operation and Use (Telecommunications)", McGraw-Hill, ISBN 0-0713-8779-X, 2002.

54 The Bluetooth SIG, "Bluetooth Specification Version 1.0 A", Nov 1999, pp. 67-69.

55 Wong, K.L., "A Message Controller for Distributed Processing Systems", PhD Thesis, The Nottingham Trent University, UK, April 2000, pp. 67.

56 Cray Research Inc, "The Cray T3D System Architecture Overview", Cray Research Inc, Technical Document HR-04033, 1993.

57 Jones, A.M., Davies, N.J., Firth, M.A., Wright, C.J., "PACT The Network Designers Handbook", IOS Press, Amsterdam, 1997, ISBN 90-5199-380-3, pp. 51 - 211.

58 M.Zhu, D A Thornley, J Pech, B Martin, N H Madsen, R Heeley, S Haas & R W Dobinson, C R Anderson, 'Realisation and Performance of IEEE 1355 DS and HS Link Based, High Speed, low Latency Packet Switching Networks', IEEE Transactions on Nuclear Science, Vol. 45, No 4, August 1998. pp. 1849 – 1853.

59 A Paskins, 'The IEEE 1394 Bus', The Institute of Electrical Engineers Half Day Colloquium on New High Capacity Digital Media and Their Applications, 1997. Pp. 4/1 – 4/6.

60 Hotchkiss, R. "A Fault Tolerant Multicast Message Routing Switch for Interprocessor Communications", PhD Thesis, The Nottingham Trent University, UK; Sept 2000, pp. 14.

61 Minoli, D., Alles, A., "LAN, ATM and LAN Emulation Technologies", Artech House Inc, ISBN 0-89006-916-6, 1996, pp 6-25.

62 Gigabit Ethernet Alliance, "Gigabit Ethernet: Accelerating the Standard for Speed", White Paper, 1999. Address: 10 Gigabit Ethernet Alliance, 1300 Bristol Street North, Suite 160, Newport Beach, CA 92660, USA.

63 Metcalfe, R.M., Boggs, D.R., "Ethernet: Distributed Packet Switching for Local Computer Networks", Communications of the ACM, Vol. 19, No. 5, July 1976, pp. 395-404.

64 Koopman, P.J., Upender, B.P., "Communication Protocols for Embedded Systems", Embedded Systems Programming, Vol. 7, No. 11, Nov 1994, pp 46 – 58.

65 Dally, W.J., Song, P., "Design of a Self-timed VLSI Multicomputer Communication Controller", Proc. of the International Conference on Computer Design (ICCD-87), 1987, pp. 230 - 234.

66 Hord, R.M., "Parallel Supercomputing in MIMD Architectures", CRC Press, 1993, pp. 61-81.

67 Borkar, S., Cohn, R., Cox, G., Gleason, S., Gross, T., Kung, H.T, Lam, M., Moore, B., Peterson, C., Pieper, J., Rankin, L., Tseng, P.S., Sutton, J., Urbanski, J., Webb, J., "iWarp: An Integrated Solution to High-speed Parallel Computing", Proceedings of Supercomputing'88, IEEE Computer Society Press, ISBN 0-8186-0882-X, 1988, pp. 330-339.

68 Dally, W.J., Fiske, J.A.S, Keen, J.S., Lethin, R.A., Noakes, M.D., Nuth, P.R., Davison, R.E., Fyler, G.A., "The Message-Driven Processor: A Multicomputer Processing Node with Efficient Mechanisms", IEEE-Micro, April 1992, pp. 23-39.

69 "Excalibur Device Overview Data Sheet", Altera Coproration; Document Part No: DS-EXCARM-2.0

70 Boden, N.J., Cohen, D., Felderman, R.E., Kulawik, A.E., Seitz, C.L., Seizovic, J.N., Su, W.K., "Myrinet - A Gigabit-per-Second Local-Area Network", IEEE-Micro, February 1995, Vol. 15, No. 1, pp. 29-36.

71 Blumrich, M.A., Li, K., Alpert, R., Dubnicki, C., Felten, E.W., "Virtual Memory Mapped Network Interface for the SHRIMP Multicomputer", Proceedings of the 21st Annual International Symposium on Computer Architecture, April 1994, pp. 142-153.

72 Kuskin, J., Ofelt, D., Heinrich, M., Heinlein, J., Simoni, R., Gharachorloo, K., Chapain, J., Nakahira, D., Baxter, J., Horowitz, M., Gupta, A., Rosenblum, M., Hennessy, J., "The Stanford FLASH Multiprocessor", Proceedings of the 21st International Symposium on Computer Architecture, Chicago, IL, April 1994, pp. 302-313.

73 Heinrich, M., Kuskin, J., Ofelt, D., Heinlein, J., Baxter, J., Pal Singh, J., Simoni, R., Charachorloo, K., Nakahira, D., Horowitz, M., Gupta, A., Rosenblum, M., Hennessy, J., "The Performance Impact of Flexibility in the Stanford FLASH Multiprocessor", Proceedings of the6th International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, October, 1994, pp. 274-285.

74 Wong, K. L., "A Message Controller for Distributed Processing Systems", PhD Thesis, The Nottingham Trent University, UK, June 2000, pp. 27.

75 Wong, K. L., "A Message Controller for Distributed Processing Systems", PhD Thesis, The Nottingham Trent University, UK, June 2000, pp. 70-79.

76 SGS-Thomson Micorelectronics "C104 Asynchronous Packet Switch Engineering Data", Ref 42-1470-06, April 1995.

77 Simpson, M., Thompson, P.W., "DS Links and C104 Routers" in "Networks, Routers and Transputer", May, M.D., Thompson, P.W., Welch, P.H. (eds), IOS Press, 1993.

78 Harrison, S., Brown, C., "Dynamic Creation of Virtual Links within T9000 Networks", Proceedings of The 19th World Occam and Transputer User Group (WoTUG), IOS Press, ISBN 90-5199-261-0, April 1996, pp. 11-20.

79 Institute of Electrical and Electronic Engineers Inc, "IEEE Standard for Heterogeneous Interconnects (HIC) Low-cost, Low-latency, Scaleable Serial Interconnect for Parallel System Construction)", IEEE Std 1355-1995, SH94378; IEEE, NY, USA; June 1996

80 Myricom Inc, "Myrinet Link Specification", Archived specification available from: Myricom Inc, 325 N.Santa Anita Ave, Arcadia, CA 91006, USA.

81 Myricom Inc., "LANai 4", Draft document, Myricom Inc., February 1999. Address: Myricom Inc, 325, N.Santa Anita Ave, Arcadia, CA 91006, U.S.A.

82 Seitz, C.L., Boden, N.J., Seizovic, J., Su, W.K., "The Design of the Caltech Mosaic C Multicomputer", Proceedings of the University of Washington Syposium on Integrated Systems", MIT Press, 1993, pp. 1-22.

83 Felderman, R., DeSchon, A., Cohen, D., Finn, G., "ATOMIC: A High Speed Local Communication Architecture", Journal of High Speed Networks, Vol. 3, No. 1 919940, pp. 1-29.

84 Finn, G.G., "An Integration of Network Communication with Workstation Architecture", Computer Communication Review", October 1991.

85 Seizovic, J.N., "Pipeline Synchronisation", Proceedings of the International Symposium on Advanced Research in Asynchronous Circuits and Systems", IEEE Computer Society Press, Nov 1994.

86 Dally, W.J., Dennison, L.R., Harris, D., Kan, K., Wanthopolous, T. "Architecture and Implementation of the Reliable Router", Proceedings of Hot Interconnects II, Stanford, USA, August 1994, pp. 122-133.

87 Bolding, K., Yost, W., "Design of a Router for Fault-tolerant Networks", Proceedings of the 1994 Computer Routing and Communications Workshop, May 1994, pp. 226-240.

88 Pinkston, T.M., Choi, Y., Raksapatcharawong, M., "Architecture and Optoelectronic Implementation of the WARRP Router", Proceedings of Hot Interconnects V, Palo Alto, CA, USA, 1997.

89 Dennison, L.R., Dally, W.J., Xanthopoulos, D., "Low Latency Plesiochronous Data Retiming", 1995 Conference on Advanced Research inVLSI, Chapel Hill, NC, USA, March 1995.

90 Wicklegren, I.J., " The facts about FireWire serial communication bus"; IEEE Spectrum, Vol 34, No 4; April 1997; pp 19 - 25.

91 A Paskins, 'The IEEE 1394 Bus', The Institute of Electrical Engineers Half Day Colloquium on New High Capacity Digital Media and Their Applications, 1997. pp. 4/1 – 4/6.

92 Serial ATA Workgroup, "Serial ATA: High Speed Serialized AT Attachment", Revision 1.0, Serial ATA Workgroup, August 2001, pp. 11-12.

93 Quinnel, R.A., "USB: a neat package with a few loose ends"; EDN, Vol 41, No 22; October 1996; pp 38 - 46, 48, 50, 52.

94 USB Implementers Forum, "Universal Serial Bus Specification", Revision 2, USB-IF, pp 1-14, April 2000. Address: USB Implementers Forum, Inc., 5440 SW Westgate Drive Suite 217, Portland, OR 97221, USA.

95 Walker, C.P.H., "Hardware for Transputing Without Transputers", Proc. 19th World Occam and Transputer User Group (WoTUG-19), IOS Press, 1996, pp. 1 - 10.

96 Shanley, T., Anderson, D., "ISA System Architecture", Third Edition, Addison-Wesley, ISBN 0-201-40996-8, 1995, pp. 335-364.

97 Messmer, H.P., "The Indispensible PC Hardware Book" 2nd Ed. Addison-Wesley, ISBN 0-201-87697-3, 1995, pp.494 - 498.

98 Messmer, H.P., "The Indispensible PC Hardware Book" 2nd Ed. Addison-Wesley, ISBN 0-201-87697-3, 1995, pp.477 - 493.

99 Institute of Electrical and Electronic Engineers Inc, "IEEE Standard for a Versatile Backplane Bus: VMEbus)", IEEE Std 1014-1987, Available from IEEE Customer Services, 445 Hoes Lane, PO Box 1331, Piscataway, NJ, 08855-1331, USA.

100 Jones, A.M., Davies, N.J., Firth, M.A., Wright, C.J., "The Network Designers Handbook", IOS Press, Amsterdam, 1997, ISBN 90-5199-380-3, pp. 17 - 18.

101 Lee, H.J., Song, B.Y., "Performance of multiple links over single link in STC-104 networks", Proceedings, 1997 International Conference on Parallel and Distributed Systems, (CAT No 97TB100215), Los Alamitos, CA, USA, 1997, pp. 196 - 202.

102 Nguyen, T.D., Snyder, L., "Performance Analysis of a Minimal Adaptive Router", Proceedings of the 1994 Parallel computer routing and communication workshop, Seattle, USA, May 1996, pp. 31 - 44.

103 Hotchkiss, R. "A Fault Tolerant Multicast Message Routing Switch for Interprocessor Communications", PhD Thesis, The Nottingham Trent University, UK; Sept 2000.

104 Warnakulasuriya, S., Pinkston, T.M., "Characterization of deadlocks in interconnection networks", Proceedings, 11th Int. Parallel processing symposium, IEEE Computer Soc. Press 1997, Los Alamitos, CA, USA, 1997, pp. 80 - 86.

105 Folkestad, A., Roche, C., "Deadlock probability in unrestricted wormhole routing networks", IEEE Int Conference on Communications, Vol. 3, Montreal, Canada, 1997, pp. 1401 - 1405.

106 Lopez, P., Martinez, J.M., Duato, J., "A Very Efficient Distributed Deadlock Detection Mechanism for Wormhole Networks", Proceedings, 4th Int Symposium on High-Performance Computer Architecture. IEEE Computer Soc. (1998), pp. 57 - 66.

107 Gaughan, P.T., Yalamanchili, S., "Pipelined circuit switching: A fault-tolerant variant of wormhole routing", Proceedings, IEEE symposium on Parallel and distributed processing, ISBN 0 8186 3200 3, 1992, pp. 148 - 155.

108 Dao, B.V., Duato, J., Yalamanchili, S., "Configurable flow control mechanisms for fault-tolerant routing", Proceedings, 22nd Annual Int. Symposium on Computer architecture, 1995, pp. 220 - 229.

109 Flich, J., Malumbres, M.P., Lopez, P., Duato, J., "Performance evaluation of a new routing strategy for irregular networks with source routing", Proceedings, Int Conference on supercomputing, ACM Press, ISBN 1 58113 270 0, May 2000, pp.34 - 43.

110 Dally, W.J., Aoki, H., "Deadlock-free adaptive routing in multicomputer networks using virtual channels", IEEE Transactions, Parallel and Distributed Systems, Vol. 4, No. 4, April 1993.

111 Pinkston, T.M., Warnakulasuriya, S., "On Deadlocks in Interconnection Networks", Computer Architecture News. Vol. 25, No. 2, 1997, pp. 38 - 49.

112 Martinez, J.M., Lopez, P., Duato, J., Pinkston, T.M., "Software-Based Deadlock Recovery Techniques for True Fully Adaptive Routing in Wormhole Networks", Proceedings, 1997 Int. Conference on Parallel Processing, 1997, pp.182 - 189.

113 Anjan,K.V., Pinkston, T.M., "An efficient, fully-adaptive deadlock recovery scheme: DISHA", Proceedings, 22nd Int. Symposium on computer architecture, ISBN 0 89791 698 0, New York, USA, June 1995, pp. 201 - 210.

114 VITA Standards Organisation, "Myrinet-on-VME. Protocol Specification Draft Standard", VITA 26-199x, Draft 1.1, August 1998, pp. 21.

115 Pinkston, T.M., "Flexible and Efficient Routing Based on Progressive Deadlock Recovery", IEEE Transactions on computers, Vol. 48, No. 7, 1999, pp 649 - 669.

116 Petrini, F., Vanneschi, M., "Performance Anaysis of Minimal Adaptive Wormhole Routing with Time-dependent Deadlock Recovery", Proceedings, 11th Int. Parallel Processing Symposium, Geneva, Switzerland, April 1997, pp. 587 - 595.

117 Duato, J., Yalamanchili, S., Caminero, M.C., Love, D., Quiles, F.J., "MMR: A high-performance multimedia router - architecture and design trade-offs", Proceedings, 5th Symposium on High-performance computer architecture, 1999, Los Alamitos, CA, USA, pp. 300 - 309.

118 Wong, K. L., "A Message Controller for Distributed Processing Systems", PhD Thesis, The Nottingham Trent University, UK, June 2000, pp. 70 - 78.

119 Triger, S., O'Neill, B.C., Clark, S., "Adapted OS / DS Link Protocols for use in Multiprocessor Routing Networks", Communicating Processors Architectures 2001, Bristol UK, September 2001, pp. 37 - 48.

120 Jones, A.M., Davies, N.J., Firth, M.A., Wright, C.J., "The Network Designers Handbook", IOS Press, Amsterdam, 1997, ISBN 90-5199-380-3, pp. 260.

121 Ellis, J.W., "A Hardware Routing Device for Tranputer Arrays", PhD Thesis, The Nottingham Trent University, October 1995, pp. 38 - 71.

122 http://www.myri.com/open-specs/link-history/index.html

123 Hotchkiss, R. "A Fault Tolerant Multicast Message Routing Switch for Interprocessor Communications", PhD Thesis, The Nottingham Trent University, UK; Sept 2000, pp. 114 - 119.

124 Stallings, W., "Data and Computer Communications", 4th Ed, Macmillan Publishing Comapny, ISBN 0-13-326828-4, 1994, pp. 278.

125 Hotchkiss, R. "A Fault Tolerant Multicast Message Routing Switch for Interprocessor Communications", PhD Thesis, The Nottingham Trent University, UK; Sept 2000, pp. 79.

126 Hotchkiss, R. "A Fault Tolerant Multicast Message Routing Switch for Interprocessor Communications", PhD Thesis, The Nottingham Trent University, UK; Sept 2000, Appendix B1 - B4.

127 Ball, R. "IP and trendy"; Electronics Weekly, No 1915; 23rd June 1999; pp 18.

128 Horowitz, P., Hill, W., "The Art of Electronics", Cambridge University Press, 1990, ISBN 0-521-37095-7, pp 850.

129 Heffer, D.E., King, G.A., Keith, D.C., "Basic Principles and Practice of Microprocessors", Arnold, 1986, ISBN 0-7131-3569-7, pp 149 - 153.

130 Furber, S,. "ARM System-on-chip architecture", Addison-Wesley, Great Britain, 2000, ISBN 0-201-67519-6, pp 263 - 266.

131 Messmer, H.P., "The Indispensable PC Hardware Book", Addison Wesley, 1995, ISBN 0-201-87697-3, pp 1223.

132 "News and Views", Altera Corporation, First Quarter 2002, pp 1 & 4

133 Altera Corporation, Data Book pp 35 - 36. Altera Corporation 101 Innovation Drive, San Jose, California 95134, USA.

134 "APEX 20K Data Sheet", Altera Coproration; Document Part No: A-DS-APEX20K-04.3

135 Horowitz, P., Hill, W., "The Art of Electronics", Cambridge University Press, 1990, ISBN 0-521-37095-7, pp 527 - 530.

136 Altera Corporation "Configuration Devices for SRAM-based LUT Devices Data Sheet", Document Part No: A-DS-EPROM-12.1

137 Altera Corporation "FLEX Devices as Alternatives to ASSPs & ASICs", Technical Brief, Version 1.0, Document Part No. M-TB-003-01, Altera Corporation, February 1996.

138 Xilinx, Inc., "IP Solutions: System-level Designs for FPGAs", Version 4.0, 15 May 2001, Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124-3400, USA.

139 Sanquini, A,. "Cover Story: Riding the New Wave of FPGA System-on-Chip", The Quarterly Journal for Xilinx Programmable Logic Users, XCELL 39, pp 17-19, First Quarter 2001.

140 QuickLogic Corp., "QuckLogic's 2001 Data Book", QuickLogic Corp., pp 1-9 to 1-14, 2001. QuickLogic.Corp, Orleans Drive, Sunnyvale, CA 94089-1138, USA

141 Edwards, C., "Design moves to FPGAs", Electronics Weekly, No 1047, 25 June 2001, pp 30 - 31.

142 Altera Corporation "Implementing High-speed Search Applications with Altera CAM", Application Note 119, Version 2.1, Document Part No. A-AN-119-2.1, Altera Corporation, July 2001.

143 http://www.altera.com/products/devices/apex/utilities/apx-20kec_calc/apx-20kec_power_index.html

144 Ng, J.H., "Message Routing Interface for Multiprocessor Networks", PhD Thesis, The Nottingham Trent University, UK, June 2000. pp. 113 - 117.

145 Shanley, T., Anderson, D., "PCI System Architecture", 3rd Edition, Addison-Wesley Publishing Company, 1995, ISBN 0-201-40993-3, pp. 82 - 84.

146 "168 pin unbuffered COB-DIMM Modules HYS64/72V2200GCU-10, HYS64/72V4220GCU-10", Siemens Semiconductor Group, February 1998.

147 "2, 4 MEG x 64 SDRAM DIMMs MT8LSDT264A, MT16LSDT464A", Micron Technology Inc., June 1998.

148 Novak, M., "Use a CPLD to Implement an SDRAM Controller", EDN Access, June 1997.

149 Wong, K. L., "A Message Controller for Distributed Processing Systems", PhD Thesis, The Nottingham Trent University, UK, June 2000, Appendix A, pp. 153 - 167.

150 Wong, K. L., "A Message Controller for Distributed Processing Systems", PhD Thesis, The Nottingham Trent University, UK, June 2000, pp. 129.

151 Shanley, T., Anderson, D., "PCI System Architecture", 3rd Edition, Addison-Wesley Publishing Company, 1995, ISBN 0-201-40993-3, pp. 77 - 81.

152 Altera Corporation, "PCI Master/Target MegaCore Function with DMA Data Sheet", Version 3.02, Document Part No. A-DS-PCII-03.02, November 1999, Altera Corporation 101 Innovation Drive, San Jose, California 95134, USA.

153 Xilinx Inc. "The Real-PCI Xilinx PCI Data Book", Xilinx Inc., 1999. Address: Xilinx Inc, 2100 Logic Drive, San Jose, CA 95124, U.S.A.

154 PLD Applications, "PCI Core User's Guide, Version 5.1.4", PLD Applications, 14 March 2001. Address: PLD Applications, 32, ZAC de Bompertuis, Avenue d'Armenie, 13120 Gardanne, France.

155 Altera Corporation "Using Quartus II Verilog HDL & VHDL Integrated Synthesis", Application Note 238, Version 1.1, Document Part No. A-AN-238-1.1, Altera Corporation, Sept 2002.

156 Utility previously available from PLD Applications web site at http://www.plda.com. Currently unavailable from this source.

Appendix A: FT-PCI-OSLi Interface Hardware Test Results

All Tests performed with 33MHz PCI Clock and 64MHz Sample Clock

Figure 55 : PCI Bus Access Efficiency

Payload (Bytes)	FT-PCI-OSLi Latency Cnt	FT-PCI-OSLi Access Cnt	PCI-OSLi Latency Cnt	PCI-OSLi Access Cnt	FT-PCI-OSLi Efficiency	PCI-OSLi Efficiency
4	3	1	5	1	0.25	0.1666
8	3	2	5	2	0.4	0.2857
16	3	4	5	4	0.5714	0.4444
32	3	8	5	8	0.7272	0.6154
64	3	16	5	16	0.8421	0.7619
128	3	32	5	32	0.9142	0.8648
256	3	54	5	57	0.9473	0.9193
512	3	55	5	58	0.9482	0.9206
1024	3	55	5	58	0.9482	0.9206

* Note: Cnt = Count value for that respective counter

Figure 56 : FT-PCI-OSLi Message Duration

	FT-PCI-OSLi	FT-PCI-OSLi (ALT-FF)	PCI-OSLi
Message Payload (Bytes)	Observed Average Message Duration (seconds)	Observed Average Message Duration (seconds)	Observed Average Message Duration (seconds)
4	3.030E-06	3.030E-06	3.333E-06
8	4.030E-06	4.030E-06	4.788E-06
16	6.333E-06	6.333E-06	7.909E-06
32	1.045E-05	1.045E-05	1.406E-05
64	1.915E-05	1.915E-05	2.664E-05
128	3.621E-05	3.621E-05	5.127E-05
256	7.118E-05	7.118E-05	1.008E-04
512	1.381E-04	1.381E-04	1.986E-04
1024	2.694E-04	2.694E-04	3.947E-04
2048	5.367E-04	5.367E-04	7.865E-04
4096	1.071E-03	1.071E-03	1.570E-03
8192	2.141E-03	2.141E-03	3.139E-03
16384	4.279E-03	4.279E-03	6.272E-03
32768	8.555E-03	8.555E-03	1.254E-02
65536	1.711E-02	1.711E-02	2.508E-02

Figure 57 : FT-PCI-OSLi Message Duration at Lower Message Payloads

	FT-PCI-OSLi	PCI-OSLi	FT-PCI-OSLi
Message Payload (Bytes)	Observed Average Message Duration (seconds)	Observed Average Message Duration (seconds)	Theoretical Message Duration @ 42.67Mb/s (secs)
4	3.030E-06	3.333E-06	1.547E-06
8	4.030E-06	4.788E-06	2.578E-06
16	6.333E-06	7.909E-06	4.641E-06
32	1.045E-05	1.406E-05	8.766E-06
64	1.915E-05	2.664E-05	1.702E-05
128	3.621E-05	5.127E-05	3.352E-05
256	7.118E-05	1.008E-04	6.652E-05

Figure 58 : FT-PCI-OSLi Normalised Message Duration

	FT-PCI-OSLi	FT-PCI-OSLi	FT-PCI-OSLi	PCI-OSLi	PCI-OSLi	PCI-OSLi
Message Payload (Bytes)	Observed Average Message Duration (seconds)	Theoretical Message Duration @ 42.67Mb/s (seconds)	Normalised Message Duration @ 42.67Mb/s	Observed Average Message Duration (seconds)	Theoretical Message Duration @ 42.67Mb/s (seconds)	Normalised Message Duration @ 42.67Mb/s
4	3.03E-06	1.54E-06	1.734	3.33E-06	1.52E-06	1.795
8	4.03E-06	2.57E-06	1.450	4.78E-06	2.74E-06	1.556
16	6.33E-06	4.64E-06	1.308	7.90E-06	5.18E-06	1.434
32	1.04E-05	8.76E-06	1.166	1.40E-05	1.00E-05	1.353
64	1.91E-05	1.70E-05	1.112	2.66E-05	1.98E-05	1.322
128	3.62E-05	3.35E-05	1.074	5.12E-05	3.93E-05	1.293
256	7.11E-05	6.65E-05	1.066	1.00E-04	7.83E-05	1.282
512	1.38E-04	1.32E-04	1.041	1.98E-04	1.56E-04	1.263
1024	2.69E-04	2.64E-04	1.017	3.94E-04	3.14E-04	1.255
2048	5.36E-04	5.28E-04	1.015	7.86E-04	6.28E-04	1.250
4096	1.07E-03	1.05E-03	1.013	1.57E-03	1.25E-03	1.248
8192	2.14E-03	2.13E-03	1.013	3.13E-03	2.51E-03	1.247
16384	4.27E-03	4.22E-03	1.012	6.27E-03	5.03E-03	1.246
32768	8.55E-03	8.44E-03	1.012	1.25E-02	1.00E-02	1.246
65536	1.71E-02	1.69E-02	1.012	2.51E-02	2.01E-02	1.246

Figure 59 : FT-PCI-OSLi Percentge Data Bandwidth Utilisation

	PCI-OSLi	FT-PCI-OSLi	FT-PCI-OSLi	PCI-OSLi
Message Payload (Bytes)	Observed Percentage Maximum Effective Data Bandwidth	Observed Percentage Maximum Effective Data Bandwidth	Theoretical Percentage Maximum Effective Data Bandwidth	Theoretical Percentage Maximum Effective Data Bandwidth
4	22.50%	24.75%	48.48%	41.03%
8	31.33%	37.22%	58.18%	49.23%
16	37.93%	47.37%	64.65%	54.70%
32	42.67%	57.39%	68.45%	57.92%
64	45.05%	62.66%	70.52%	59.67%
128	46.81%	66.28%	71.61%	60.59%
256	47.61%	69.47%	72.16%	61.06%
512	48.34%	69.89%	72.44%	61.30%
1024	48.65%	71.27%	72.59%	61.42%
2048	48.82%	71.55%	72.66%	61.48%
4096	48.92%	71.69%	72.69%	61.51%
8192	48.93%	71.74%	72.71%	61.52%
16384	48.98%	71.80%	72.72%	61.53%
32768	48.99%	71.82%	72.72%	61.53%
65536	49.00%	71.83%	72.73%	61.54%

Figure 60 : FT-PCI-OSLi DMA Transmission Throughput

	FT-PCI-OSLi	FT-PCI-OSLi	FT-PCI-OSLi (Alt)	FT-PCI-OSLi (Alt)	PCI-OSLi	PCI-OSLi
Message Payload (Bytes)	DMA Message Transmit Duration (seconds)	DMA Message Transmit Throughput (Bits/sec)	DMA Message Transmit Duration (seconds)	DMA Message Transmit Throughput (Bits/sec)	DMA Message Transmit Duration (seconds)	DMA Message Transmit Throughput (Bits/sec)
4	7.576E-07	5.280E+06	7.879E-07	5.077E+06	7.576E-07	5.280E+06
8	7.879E-07	1.015E+07	8.182E-07	9.778E+06	7.879E-07	1.015E+07
16	8.485E-07	1.886E+07	8.788E-07	1.821E+07	8.485E-07	1.886E+07
32	9.394E-07	3.406E+07	9.697E-07	3.300E+07	9.394E-07	3.406E+07
64	1.212E-06	5.280E+07	1.242E-06	5.151E+07	1.212E-06	5.280E+07
128	1.697E-06	7.543E+07	1.727E-06	7.411E+07	1.667E-06	7.680E+07
256	4.791E-05	5.343E+06	9.273E-06	2.761E+07	9.606E-06	2.665E+07
1024	2.270E-04	4.510E+06	3.812E-05	2.686E+07	4.018E-05	2.548E+07
2048	5.058E-04	4.049E+06	2.491E-04	8.221E+06	3.789E-04	5.405E+06
4096	1.043E-03	3.928E+06	7.727E-04	5.301E+06	1.159E-03	3.534E+06
16384	4.250E-03	3.855E+06	3.992E-03	4.104E+06	5.861E-03	2.795E+06
32768	8.464E-03	3.872E+06	8.265E-03	3.965E+06	1.213E-02	2.701E+06
65536	1.708E-02	3.837E+06	1.681E-02	3.898E+06	2.467E-02	2.656E+06

Figure 64 : FT-PCI-OSLi DMA Transmission Throughput For Alternate Latency Counter Values

Lat Cnt in WORDS	FT-PCI-OSLi (Lat Cnt = 96)	FT-PCI-OSLi (Lat Cnt = 64)	FT-PCI-OSLi (Lat Cnt = 32)	FT-PCI-OSLi (Lat Cnt = 24)	FT-PCI-OSLi (Lat Cnt = 16)
Message Payload (Bytes)	DMA Message Transmission Throughput (Bits/sec)	DMA Message Transmission Throughput (Bits/sec)	DMA Message Transmission Throughput (Bits/sec)	DMA Message Transmission Throughput (Bits/sec)	DMA Message Transmission Throughput (Bits/sec)
4	5.077E+06	5.280E+06	5.280E+06	5.280E+06	5.280E+06
8	9.429E+06	1.015E+07	1.015E+07	1.015E+07	1.015E+07
16	1.650E+07	1.821E+07	1.886E+07	1.703E+07	1.886E+07
32	2.933E+07	3.200E+07	3.200E+07	2.779E+07	1.920E+07
64	4.591E+07	5.151E+07	5.151E+07	2.321E+07	1.509E+07
128	6.813E+07	7.543E+07	9.103E+06	1.983E+07	1.170E+07
256	9.284E+07	5.320E+06	5.747E+06	5.029E+06	5.767E+06
1024	5.032E+06	4.571E+06	4.080E+06	4.180E+06	4.037E+06
2048	4.475E+06	4.032E+06	4.040E+06	3.948E+06	3.941E+06
4096	4.150E+06	3.923E+06	3.942E+06	3.891E+06	3.891E+06
16384	3.941E+06	3.854E+06	3.864E+06	3.850E+06	3.847E+06
32768	3.906E+06	3.843E+06	3.845E+06	3.838E+06	3.837E+06
65536	3.891E+06	3.837E+06	3.839E+06	3.835E+06	3.835E+06

Figure 65 : DMA Transmission Throughput Characteristics For Alternate Latency Counter Values For Modified Link Interface Buffer FT-PCI-OSLi

Lat Cnt in WORDS	FT-PCI-OSLi (Lat Cnt = 96)	FT-PCI-OSLi (Lat Cnt = 64)	FT-PCI-OSLi (Lat Cnt = 32)	FT-PCI-OSLi (Lat Cnt = 24)	FT-PCI-OSLi (Lat Cnt = 16)
Message Payload (Bytes)	DMA Message Transmission Throughput (Bits/sec)	DMA Message Transmission Throughput (Bits/sec)	DMA Message Transmission Throughput (Bits/sec)	DMA Message Transmission Throughput (Bits/sec)	DMA Message Transmission Throughput (Bits/sec)
4	5.077E+06	5.077E+06	5.077E+06	5.077E+06	5.077E+06
8	9.778E+06	9.778E+06	9.778E+06	9.778E+06	9.778E+06
16	1.650E+07	1.821E+07	1.886E+07	1.886E+07	1.886E+07
32	2.854E+07	3.300E+07	3.300E+07	3.300E+07	2.514E+07
64	4.693E+07	5.151E+07	5.151E+07	3.462E+07	2.893E+07
128	8.620E+07	7.411E+07	2.797E+07	3.520E+07	3.129E+07
256	8.893E+07	2.761E+07	2.797E+07	2.514E+07	2.463E+07
1024	2.671E+07	2.686E+07	2.747E+07	2.501E+07	2.501E+07
2048	9.436E+06	8.221E+06	8.238E+06	7.969E+06	7.770E+06
4096	5.611E+06	5.301E+06	5.203E+06	5.223E+06	5.174E+06
16384	4.198E+06	4.104E+06	4.111E+06	4.097E+06	4.101E+06
32768	4.031E+06	3.965E+06	3.968E+06	3.960E+06	3.960E+06
65536	3.952E+06	3.898E+06	3.898E+06	3.895E+06	3.895E+06

* Note: Lat Cnt = Initial Value held in the Latency Counter on generation of PCI transaction request

Figure 67 : FT-PCI-OSLi DMA Reception Throughput

	FT-PCI-OSLi	FT-PCI-OSLi	FT-PCI-OSLi (Alt)	FT-PCI-OSLi (Alt)	PCI-OSLi	PCI-OSLi
Message Payload (Bytes)	DMA Reception Duration (seconds)	DMA Message Reception Throughput (Bits/sec)	DMA Reception Duration (seconds)	DMA Message Reception Throughput (Bits/sec)	DMA Reception Duration (seconds)	DMA Message Reception Throughput (Bits/sec)
4	1.818E-06	2.200E+06	1.848E-06	2.164E+06	2.848E-06	1.404E+06
8	3.091E-06	2.588E+06	3.091E-06	2.588E+06	4.000E-06	2.000E+06
16	5.424E-06	2.950E+06	5.424E-06	2.950E+06	7.424E-06	2.155E+06
32	9.333E-06	3.429E+06	9.394E-06	3.406E+06	1.327E-05	2.411E+06
64	1.821E-05	3.514E+06	1.821E-05	3.514E+06	2.552E-05	2.508E+06
128	3.539E-05	3.616E+06	3.539E-05	3.616E+06	4.970E-05	2.576E+06
256	7.009E-05	3.652E+06	6.982E-05	3.667E+06	9.988E-05	2.563E+06
512	1.377E-04	3.717E+06	1.379E-04	3.713E+06	1.988E-04	2.576E+06
1024	2.692E-04	3.803E+06	2.676E-04	3.827E+06	3.928E-04	2.607E+06
2048	5.355E-04	3.825E+06	5.343E-04	3.833E+06	7.842E-04	2.611E+06
4096	1.072E-03	3.821E+06	1.069E-03	3.832E+06	1.569E-03	2.611E+06
8192	2.144E-03	3.822E+06	2.142E-03	3.832E+06	3.138E-03	2.611E+06
16384	4.277E-03	3.830E+06	4.276E-03	3.832E+06	6.271E-03	2.613E+06
32768	8.554E-03	3.831E+06	8.553E-03	3.831E+06	1.254E-02	2.613E+06
65536	1.711E-02	3.831E+06	1.711E-02	3.831E+06	2.508E-02	2.613E+06

Figure 71 : FT-SARNIC Post-synthesis Simulation Message Duration

Message Payload (Bytes)	SARNIC Message Duration @ 20Mb/s (secs)	FT-SARNIC Message Duration @ 20Mb/s (secs)	SARNIC Message Duration @ 39Mb/s (secs)	FT-SARNIC Message Duration @ 39Mb/s (secs)
4	5.284E-06	4.467E-06	2.728E-06	2.352E-06
8	8.075E-06	6.642E-06	4.152E-06	3.473E-06
16	1.366E-05	1.105E-05	7.016E-06	5.729E-06
32	2.481E-05	1.981E-05	1.273E-05	1.021E-05
64	4.712E-05	3.733E-05	2.415E-05	1.918E-05
128	9.173E-05	7.239E-05	4.700E-05	3.714E-05
256	1.810E-04	1.425E-04	9.270E-05	7.304E-05
512	3.616E-04	2.828E-04	1.852E-04	1.448E-04
1024	7.227E-04	5.632E-04	3.700E-04	2.885E-04
2048	1.445E-03	1.124E-03	7.399E-04	5.757E-04
4096	2.890E-03	2.246E-03	1.480E-03	1.150E-03
8192	5.779E-03	4.490E-03	2.959E-03	2.299E-03
16384	1.154E-02	8.977E-03	5.918E-03	4.597E-03
32768	2.311E-02	1.795E-02	1.184E-02	9.193E-03
65536	4.623E-02	3.590E-02	2.367E-02	1.838E-02

Figure 72 : FT-SARNIC Message Duration at Lower Message Payloads

Message Payload (Bytes)	SARNIC Message Duration @ 20Mb/s (secs)	FT-SARNIC Message Duration @ 20Mb/s (secs)	SARNIC Message Duration @ 39Mb/s (secs)	FT-SARNIC Message Duration @ 39Mb/s (secs)	FT-S Theoretical Message Duration @ 20Mb/s	FT-S Theoretical Message Duration @ 39Mb/s
4	5.284E-06	4.467E-06	2.728E-06	2.352E-06	3.850E-06	1.974E-06
8	8.075E-06	6.642E-06	4.152E-06	3.473E-06	6.050E-06	3.102E-06
16	1.366E-05	1.105E-05	7.016E-06	5.729E-06	1.045E-05	5.359E-06
32	2.481E-05	1.981E-05	1.273E-05	1.021E-05	1.925E-05	9.871E-06
64	4.712E-05	3.733E-05	2.415E-05	1.918E-05	3.685E-05	1.890E-05

Figure 73 : FT-SARNIC Bi-directional Data Throughput

Payload (Bytes)	SARNIC Observed Maximum Data Throughput (20Mb/s) (Bits/sec)	FT-SARNIC Observed Maximum Data Throughput (20Mb/s) (Bits/sec)	FT-SARNIC Theoretical Maximum Data Throughput (20Mb/s) (Bits/sec)	FT-SARNIC Observed Maximum Data Throughput (39Mb/s) (Bits/sec)	SARNIC Observed Maximum Data Throughput (39Mb/s) (Bits/sec)
4	6.814E+06	7.164E+06	8.345E+06	1.361E+07	1.173E+07
8	8.916E+06	9.636E+06	1.062E+07	1.843E+07	1.542E+07
16	1.054E+07	1.158E+07	1.230E+07	2.234E+07	1.825E+07
32	1.161E+07	1.292E+07	1.335E+07	2.508E+07	2.011E+07
64	1.222E+07	1.372E+07	1.395E+07	2.669E+07	2.120E+07
128	1.256E+07	1.415E+07	1.427E+07	2.757E+07	2.179E+07
256	1.273E+07	1.437E+07	1.443E+07	2.804E+07	2.209E+07
512	1.274E+07	1.449E+07	1.452E+07	2.828E+07	2.212E+07
1024	1.275E+07	1.454E+07	1.456E+07	2.840E+07	2.214E+07
2048	1.276E+07	1.457E+07	1.458E+07	2.846E+07	2.214E+07
4096	1.276E+07	1.459E+07	1.459E+07	2.849E+07	2.215E+07
8192	1.276E+07	1.460E+07	1.460E+07	2.851E+07	2.215E+07
16384	1.278E+07	1.460E+07	1.460E+07	2.851E+07	2.215E+07
32768	1.276E+07	1.460E+07	1.460E+07	2.852E+07	2.215E+07
65536	1.276E+07	1.460E+07	1.460E+07	2.852E+07	2.215E+07

Appendix B: PCI Signal Descriptions for the FT-PCI-OSLi Interface

In the following table, the type definitions are from the viewpoint of the PCI Interface of the FT-PCI-OSLi. The words 'Master' and 'Target' mean the PCI Interface works as a PCI master and target respectively. Other terms in the type definitions are described below:

- Input – a standard input-only signal
- Output – a standard active driver
- Tri-state – a bi-directional signal, becoming high-impedance when disabled.
- Sustained Tri-state – an active low tri-state signal that is driven by only one PCI device at a time. The device driving the signal low must drive it high for at least one clock cycle before releasing the signal (go to tri-state). No other device is allowed to drive the signal sooner than one clock after it has been released. The signal is pulled-up to sustain the inactive state until another device drives it.
- Open-drain – a signal that can be shared by multiple devices through wire-OR circuitry. The signal is pulled-up to sustain the inactive state until another device drives it.

The polarity in the table represents the condition when the signal is considered active. '0' and '1' means standard logic '0' and '1' respectively while 'N.A.' means not applicable.

Name	Type	Polarity	Description
nC/BE[3..0]	Tri-state Master: Output Target: Input	N.A. for command; '0' for byte-enable	Command/Byte Enable bus (4-bit). The command and byte-enable signals are time-multiplexed onto the bus. During the address phase and data phases, this bus indicates command and byte-enable signals respectively.
AD[31..0]	Tri-state	N.A.	Address/Data bus (32-bit). Each PCI data transfer consists of an address phase followed by one or more data phases. The address and data are time-multiplexed onto the bus during an address phase and data phases, respectively.
nFRAME	Sustained Tri-state Master: Output Target: Input	'0'	Cycle Frame. The signal is driven by the PCI Interface to indicate the beginning and duration of a bus operation. nFRAME is first asserted during the address phase, at the same time when command and address are presented on their respective busses. nFRAME will be asserted for all data phases except for the last.
nDEVSEL	Sustained Tri-state Master: Input Target: Output	'0'	Device Select. The PCI Interface will assert this signal when acting as a target and has decoded its own address the AD[31:0].
nIRDY	Sustained Tri-state Master: Output Target: Input	'0'	Initiator Ready. When asserted, the signal indicates that the PCI Interface, acting as a master now, is ready to complete a data phase.
nTRDY	Sustained Tri-state Master: Input Target: Output	'0'	Target Ready. The signal indicates that PCI Interface, now acting as a target, is ready to complete a data phase. The data on the AD[31:0] bus is valid when both nIRDY and nTRDY are asserted during the data phase.
nSTOP	Sustained Tri-state	'0'	Stop. This signal is asserted by the PCI Interface,

Name	Type	Polarity	Description
	Master: Input Target: Output		now a target, during a data phase to terminate a data transfer.
PAR	Tri-State	N.A.	Even Parity. The signal is generated in a way that the number of 1's across the AD[31:0], the nC/BE[3:0] and the PAR is maintained at even number.
nSERR	Open-drain	'0'	System Error. The PCI Interface uses this signal to indicate parity error during an address phase.
nPERR	Sustained Tri-state	'0'	Parity Error. This signal is asserted when the PCI Interface detects a mismatch between its internal generated parity bit and the PAR signal during for a given data phase.
nINTA	Open-drain	'0'	Interrupt A. This signal can be asserted asynchronous to the CLK. The PCI Interface uses the signal to make interrupt requests to its host system.
IDSEL	Input	'1'	Initialisation Device Select. This signal is used as a chip-select line during configuration-read or configuration-write operations.
nGNT	Input	'0'	Grant. The PCI Interface has control of the bus when this signal is active.
nREQ	Output	'0'	Request. The PCI Interface will assert this signal when it wants to initiate a transfer.
CLK	Input	N.A.	PCI clock. It runs at 33 MHz, providing reference for all other PCI Interface signals except of the reset (nRST) and the interrupt (nINTA).
nRST	Input	'0'	Reset. This signal can be asserted asynchronously to the PCI clock. When active, the PCI Interface is initialized, all PCI output signals are driven into tri-state and open-drain signals float.

Appendix C: Registers of the FT-PCI-OSLi

The following gives descriptions for the FT-PCI-OSLi memory-mapped, 32-bit registers. The base of the register mapped to a memory region is termed as BASE, with a value of 0 in hexadecimal (expressed as 0x00).

Receiver Control/Status Register (Location: BASE)

Bit	State after reset	Description
0	'0'	LABEL: dma_wr_nrd (read-only) This bit indicates the direction of current PCI transfer. It is meaningless when DMA is currently not active (bit 1). '0' - Device is reading data from memory '1' - Device is writing data to memory
1	'0'	LABEL: dma_active (read-only) This bit indicates if the device is actively transferring data utilising DMA. '0' - DMA is not active '1' - DMA is active
2	'0'	Unused. Hardwired to '0'.
3	'0'	LABEL: rx_dma_en (read/write) This bit controls if the receiver is enabled for DMA transfer. '0' - DMA is disabled for the receiver '1' - DMA for the receiver is enabled
4	'1'	LABEL: rx_less_data_det_en (read/write) This bit enables the device to detect if the data received is less than expected. '0' - disable less data detection '1' - enable less data detection
5	'1'	LABEL: rx_more_data_det_en (read/write) This bit enables the device to detect if the data received is more than expected. '0' - disable more data detection '1' - enable more data detection
6	'0'	LABEL: autoflush_enable (read/write) This bit is used to flush the extra bytes of a packet stored in the RX Link Interface FIFO automatically when the payload of that packet is more than

Bit	State after reset	Description
		expected. '0' - disable auto-flush '1' - enable auto-flush
7	'0'	LABEL: manual_flush_rxff (write-only) This bit is to manually flush the extra bytes of a packet stored in the RX Link Interface FIFO when data received is more than expected. This bit will reset itself. This bit is neglected if bit 6 is enabled. '0' - no action '1' - flush the FIFO
8	'0'	LABEL: clear_more_data_err (write-only) The user can set this bit to clear the more data error after it has been reported. This bit will reset itself. '0' - no action '1' - clear more data error
9	'0'	LABEL: manual_rx_threshold (write-only) This bit is only used for debugging purpose. When it is set and the receiving channel has been enabled for DMA transfer, the receiving channel will start requesting for bus ownership. All data in the DMA FIFO will be transferred into the system memory later. This bit will revert to '0' automatically. '0' - no action '1' - assert the threshold
[31:10]	'0'	All bits are hardwired to '0'

Transmitter Control/Status Register (Location: BASE + 0x04)

Bit	State after reset	Description
0	'0'	LABEL: dma_wr_nrd (read-only) This bit indicates the direction of current PCI transfer. It is meaningless when DMA is currently not active (bit 1). '0' - Device is reading data from memory '1' - Device is writing data to memory

Bit	State after reset	Description
1	'0'	LABEL: m_access (read-only) This bit indicates if the device is actively transferring data utilising DMA. '0' - DMA is not active '1' - DMA is active
2	'0'	Unused. Hardwired to '0'.
3	'0'	LABEL: tx_dma_en (read/write) This bit controls if the transmitter is enabled for DMA transfer. '0' - DMA is disabled for the transmitter '1' - DMA for the transmitter is enabled
4	'0'	LABEL: pci_rd_cmmd_sel (read/write) This bit allows the user to choose between MEMORY_READ and MEMORY_READ_MULTIPLE commands when reading data from the system memory. This bit is utilised to find out the performance difference of the two commands. '0' - using MEMORY_READ '1' - using MEMORY_READ_MULTIPLE
[31:5]	'0'	All bits are hardwired to '0'

Receiver Link Interface FIFO and DMA FIFO Status Register (Location: BASE + 0x08)

Bit	State after reset	Description
[5:0]	'0'	LABEL: RX DMA FIFO used entries (read-only) These bits represent how many spaces of the FIFO are currently used.
[7:6]	'0'	All bits are hardwired to '0'
[8]	'1'	LABEL: RX DMA FIFO empty (read-only) '0' - The FIFO is not empty '1' - The FIFO is empty
[9]	'0'	LABEL: RX DMA FIFO full (read-only) '0' - The FIFO is not full

Bit	State after reset	Description
		'1' - The FIFO is full
[15:10]	'0'	All bits are hardwired to '0'
[26:16]	'0'	LABEL: RX Link Interface FIFO used entries (read-only) These bits represent how many spaces of the FIFO are currently used.
[27]	'0'	Unused. Hardwired to '0'.
[28]	'1'	LABEL: RX Link Interface FIFO empty (read-only) '0' - The FIFO is not empty '1' - The FIFO is empty
[29]	'0'	LABEL: RX Link Interface FIFO full (read-only) '0' - The FIFO is not full '1' - The FIFO is full
[31:30]	'0'	All bits are hardwired to '0'

Transmitter Link Interface FIFO and DMA FIFO Status Register (Location: BASE + 0x0C)

Bit	State after reset	Description
[5:0]	'0'	LABEL: TX DMA FIFO used entries (read-only) These bits represent how many spaces of the FIFO are currently used.
[7:6]	'0'	All bits are hardwired to '0'
[8]	'1'	LABEL: TX DMA FIFO empty (read-only) '0' - The FIFO is not empty '1' - The FIFO is empty
[9]	'0'	LABEL: TX DMA FIFO full (read-only) '0' - The FIFO is not full '1' - The FIFO is full
[15:10]	'0'	All bits are hardwired to '0'
[25:16]	'0'	LABEL: TX Link Interface FIFO used entries (read-only) These bits represent how many spaces of the FIFO are currently used.

Bit	State after reset	Description
[27:26]	'0'	All bits are hardwired to '0'
[28]	'1'	LABEL: TX Link Interface FIFO empty (read-only) '0' - The FIFO is not empty '1' - The FIFO is empty
[29]	'0'	LABEL: TX Link Interface FIFO full (read-only) '0' - The FIFO is not full '1' - The FIFO is full
31:30]	'0'	All bits are hardwired to '0'

Reset Register (Location: BASE + 0x10)

Bit	State after reset	Description
0	'0'	LABEL: system_rst (write-only) Setting this bit resets the FT-PCI-OSLi device to its initial state. The bit clears itself automatically. '0' - no action '1' - reset the device
1	'0'	LABEL: rx_rst (write-only) Setting this bit resets the FT-PCI-OSLi transmitter to its initial state. The bit clears itself automatically. '0' - no action '1' - reset the transmitter
2	'0'	LABEL: tx_rst (write-only) Setting this bit resets the FT-PCI-OSLi receiver to its initial state. The bit resets itself automatically. '0' - no action '1' - reset the receiver
3	'0'	LABEL: router_rst (write-only) Setting this bit resets the NTR-FTM08 to its initial state. The bit clears itself automatically. '0' - no action

Bit	State after reset	Description
		'1' - reset ICR C416
4	'0'	LABEL: rxtmr_rst (write-only) Setting this bit resets the FT-PCI-OSLi receiver hardware timer to its initial state. The bit clears itself automatically. '0' - no action '1' - reset the receiver hardware timer
5	'0'	LABEL: txtmr_rst (write-only) Setting this bit resets the FT-PCI-OSLi transmitter hardware timer to its initial state. The bit clears itself automatically. '0' - no action '1' – reset the transmitter hardware timer
[31:6]	'0'	All bits are hardwired to '0'

FT-PCI-OSLi Hardware Timer Register (Location: BASE + 0x14)

Bit	State after reset	Description
[15:0]	'0'	LABEL: tx_tmr (read-only) When the timer is started, its value increases by 1 for every PCI clock. This means it has a resolution of 1/(PCI clock) = 30 ns for 33 MHz clock. This timer is only used for measuring hardware performance.
[31:16]	'0'	LABEL: rx_tmr (read-only) When the timer is started, its value increases by 1 for every PCI clock. This means it has a resolution of 1/(PCI clock) = 30 ns for 33 MHz clock. This timer is only used for measuring hardware performance.

Interrupt Enable Register (Location: BASE + 0x18)

Bit	State after reset	Description
0	'0'	LABEL: rx_new_hdr_inten0 (read/write) If this bit is enabled, an interrupt will be generated when a new packet is

Bit	State after reset	Description
		received. This is done by comparing the received header with the previous value. In case of the payload of the received packet is less or more than expected, the subsequent packet received will be treated as a new one, regardless of its message header. '0' - disable '1' - enable
1	'0'	LABEL: rx_mssg_end_inten0 (read/write) If this bit is enabled, an interrupt will be generated when a whole message has been received and transferred into the memory. '0' - disable '1' - enable
2	'0'	LABEL: rx_less_data_inten0 (read/write) If this bit is enabled and bit 4 in Receiver Control/Status Register is set, an interrupt will be generated when less data was received. '0' - disable '1' - enable
3	'0'	LABEL: rx_more_data_inten0 (read/write) If this bit is enabled and bit 5 in Receiver Control/Status Register is set, an interrupt will be generated when more data was received. '0' - disable '1' - enable
4	'0'	LABEL: txff_afull_inten0 (read/write) If this bit is enabled, an interrupt will be generated when the TX Link Interface FIFO is nearly full (less than 64 bytes of space available). '0' - disable '1' - enable
5	'0'	LABEL: tx_mssg_end_inten0 (read/write) If this bit is enabled, an interrupt will be generated when a message has been completely transmitted to the TX Link Interface FIFO. '0' - disable '1' - enable
6	'0'	Unused. Hardwired to '0'.
7	'0'	LABEL: rx_new_hdr_inten1 (read/write) (RESERVED for future use)

Bit	State after reset	Description
		If this bit is enabled, an interrupt will be generated when a new packet is received. This is done by comparing the received header with the previous value. In case of the payload of the received packet is less or more than expected, the subsequent packet received will be treated as a new one, regardless of its message header. '0' - disable '1' – enable (Reserved for use with second link interface channel – currently duplicate of the channel 0 equivalent of this signal)
8	'0'	LABEL: rx_mssg_end_inten1 (read/write) (RESERVED for future use) If this bit is enabled, an interrupt will be generated when a whole message has been received and transferred into the memory. '0' - disable '1' – enable (Reserved for use with second link interface channel – currently duplicate of the channel 0 equivalent of this signal)
9	'0'	LABEL: rx_less_data_inten1 (read/write) (RESERVED for future use) If this bit is enabled and bit 4 in Receiver Control/Status Register is set, an interrupt will be generated when less data was received. '0' - disable '1' – enable (Reserved for use with second link interface channel – currently duplicate of the channel 0 equivalent of this signal)
10	'0'	LABEL: rx_more_data_inten1 (read/write) (RESERVED for future use) If this bit is enabled and bit 5 in Receiver Control/Status Register is set, an interrupt will be generated when more data was received. '0' - disable '1' – enable (Reserved for use with second link interface channel – currently duplicate of the channel 0 equivalent of this signal)

Bit	State after reset	Description
11	'0'	LABEL: txff_afull_inten1 (read/write) (RESERVED for future use) If this bit is enabled, an interrupt will be generated when the OS TX FIFO is nearly full (less than 64 bytes of space available). '0' - disable '1' – enable (Reserved for use with second link interface channel – currently duplicate of the channel 0 equivalent of this signal)
12	'0'	LABEL: tx_mssg_end_inten1 (read/write) (RESERVED for future use) If this bit is enabled, an interrupt will be generated when a message has been completely transmitted to the OS TX FIFO. '0' - disable '1' – enable (Reserved for use with second link interface channel – currently duplicate of the channel 0 equivalent of this signal)
[29:13]	'0'	All bits are hardwired to '0'
30	'0'	LABEL: err_int_enable1 (read/write) Reserved. Currently the bit is unused.
31	'0'	LABEL: int_enable1 (read/write) When this bit is cleared, all previous interrupt enable states are overwritten. No interrupt can be generated. '0' - disable interrupt generation '1' - enable interrupt generation

Interrupt Pending Register (Location: BASE + 0x1C)

Bit	State after reset	Description
0	'0'	LABEL: rx_new_hdr_intpd0 (read/write) This bit indicates if the interrupt request after receiving a new message header is pending. A '1' must be written to the bit to reset its status to '0'.

Bit	State after reset	Description
		'0' - no interrupt pending '1' - interrupt pending
1	'0'	LABEL: rx_mssg_end_intpd0 (read/write) This bit indicates if the interrupt request for the end of receiver message is pending. A '1' must be written to the bit to reset its status to '0'. '0' - no interrupt pending '1' - interrupt pending
2	'0'	LABEL: rx_less_data_intpd0 (read/write) This bit indicates if the interrupt request for less data received is pending. A '1' must be written to the bit to reset its status to '0'. '0' - no interrupt pending '1' - interrupt pending
3	'0'	LABEL: rx_more_data_intpd0 (read/write) This bit indicates if the interrupt request for more data received is pending. A '1' must be written to the bit to reset its status to '0'. '0' - no interrupt pending '1' - interrupt pending
4	'0'	LABEL: txff_afull_intpd0 (read/write) This bit indicates if the interrupt request for OS TX FIFO almost full is pending. A '1' must be written to the bit to reset its status to '0'. '0' - no interrupt pending '1' - interrupt pending
5	'0'	LABEL: tx_mssg_end_intpd0 (read/write) This bit indicates if the interrupt request for the end of transmitter message is pending. A '1' must be written to the bit to reset its status to '0'. '0' - no interrupt pending '1' - interrupt pending
6	'0'	Unused. Hardwired to '0'.
7	'0'	LABEL: rx_new_hdr_intpd1 (read/write) (RESERVED for future use) This bit indicates if the interrupt request after receiving a new message header is pending. A '1' must be written to the bit to reset its status to '0'. '0' - no interrupt pending

Bit	State after reset	Description
		'1' - interrupt pending (Reserved for use with second link interface channel – currently duplicate of the channel 0 equivalent of this signal)
8	'0'	LABEL: rx_mssg_end_intpd1 (read/write) (RESERVED for future use) This bit indicates if the interrupt request for the end of receiver message is pending. A '1' must be written to the bit to reset its status to '0'. '0' - no interrupt pending '1' - interrupt pending (Reserved for use with second link interface channel – currently duplicate of the channel 0 equivalent of this signal)
9	'0'	LABEL: rx_less_data_intpd1 (read/write) (RESERVED for future use) This bit indicates if the interrupt request for less data received is pending. A '1' must be written to the bit to reset its status to '0'. '0' - no interrupt pending '1' - interrupt pending (Reserved for use with second link interface channel – currently duplicate of the channel 0 equivalent of this signal)
10	'0'	LABEL: rx_more_data_intpd1 (read/write) (RESERVED for future use) This bit indicates if the interrupt request for more data received is pending. A '1' must be written to the bit to reset its status to '0'. '0' - no interrupt pending '1' - interrupt pending (Reserved for use with second link interface channel – currently duplicate of the channel 0 equivalent of this signal)
11	'0'	LABEL: txff_afull_intpd1 (read/write) (RESERVED for future use) This bit indicates if the interrupt request for OS TX FIFO almost full is pending. A '1' must be written to the bit to reset its status to '0'. '0' - no interrupt pending '1' - interrupt pending (Reserved for use with second link interface channel – currently duplicate of the channel 0 equivalent of this signal)
12	'0'	LABEL: tx_mssg_end_intpd1 (read/write) (RESERVED for future use) This bit indicates if the interrupt request for the end of transmitter message is

Bit	State after reset	Description
		pending. A '1' must be written to the bit to reset its status to '0'. '0' - no interrupt pending '1' - interrupt pending (Reserved for use with second link interface channel – currently duplicate of the channel 0 equivalent of this signal)
13	'0'	These bits are unused and hardwired to '0'.
14	'0'	LABEL: mstr_abrt_intpd (read-only) This bit indicates if the master has aborted a data transfer. It can only be reset by writing '1' to the appropriate bit in the PCI Configuration Register. '0' - no interrupt pending '1' - interrupt pending
15	'0'	LABEL: parity_err_intpd (read-only) This bit indicates if the parity error has occurred during data phases. It can only be reset by writing '1' to the appropriate bit in the PCI Configuration Register. '0' - no interrupt pending '1' - interrupt pending
16	'0'	LABEL: err_int_pending (read-only) Reserved. Currently the bit is unused.
[31:17]	'0'	All bits are hardwired to '0'

Receiver Address Register (Location: BASE + 0x24)

Bit	State after reset	Description
[1:0]	'0'	These bits have no meaning and are hardwired to '0'.
[31:2]	'0'	LABEL: rx_address (read/write) These bits contain the address of current memory location the device is pointing for memory-write operations. Its value increases by 1 after each successful data transfer. Writes to ALL message class base addresses are made via this register, with the class determined by bits 25:23 of the Command Register (BASE + 0x3C)

Receiver Assigned Message Header Register (Location: BASE + 0x28)

Bit	State after reset	Description
[7:0]	'0'	LABEL: Message Header 1 (read/write) This byte represents the 1st expected received message header.
[15:8]	'0'	LABEL: Message Header 2 (read/write) This byte represents the 2nd expected received message header. Unused in this implementation of the FT-PCI-OSLi and hardwired to '0'.
[23:16]	'0'	LABEL: Message Header 3 (read/write) This byte represents the 3rd expected received message header. Unused in this implementation of the FT-PCI-OSLi and hardwired to '0'.
[31:24]	'0'	LABEL: Message Header 4 (read/write) This byte represents the 4th expected received message header. Unused in this implementation of the FT-PCI-OSLi and hardwired to '0'.

Transmitter Message Header Register (Location: BASE + 0x2C)

Bit	State after reset	Description
[7:0]	'0'	LABEL: Message Header 1 (read/write) This byte represents the 1st message header.
[15:8]	'0'	LABEL: Message Header 2 (read/write) This byte represents the 2nd message header.
[23:16]	'0'	LABEL: Message Header 3 (read/write) This byte represents the 3rd message header.
[31:24]	'0'	LABEL: Message Header 4 (read/write) This byte represents the 4th message header.

Transmitter Address Register (Location: BASE + 0x30)

Bit	State after reset	Description
[1:0]	'0'	These bits are unused and hardwired to '0'.
[31:2]	'0'	LABEL: tx_address (read/write) These bits contain the address of current memory location the device is pointing. Its value increases by 1 after each successful data transfer.

Transmitter Message Length Register (Location: BASE + 0x34)

Bit	State after reset	Description
[19:0]	'0'	LABEL: tx_length (read/write) These bits contain the length for current DMA transfer. Its value decreases by 4 after each successful data transfer.
[31:20]	'0'	These bits are unused and hardwired to '0'.

Receiver Message Length Register (Location: BASE + 0x38)

Bit	State after reset	Description
[19:0]	'0'	LABEL: rx_length (read/write) These bits contain the length for current DMA transfer. Its value decreases by 4 after each successful data transfer.
[31:20]	'0'	These bits are unused, hardwired to '0' and masked by the Status Register.

Command Register (Location: BASE + 0x3C)

Bit	State after reset	Description
[18:0]	'0'	These bits are unused and hardwired to '0'.
19	'0'	LABEL: lnkdormant (read/write) Setting this bit configures the communications link as being dormant after periods of no link activity.
20	'0'	LABEL: rx_cmmd_20 (read/write)

Bit	State after reset	Description
		Class 3 Address Register Select Writing to the rx_addr_reg (Location: BASE + 0x24) when this bit is asserted writes to the Class 3 Address Register
21	'0'	LABEL: rx_cmmd_21 (read/write) Class 2 Address Register Select Writing to the rx_addr_reg (Location: BASE + 0x24) when this bit is asserted writes to the Class 2 Address Register
22	'0'	LABEL: rx_cmmd_22 (read/write) Class 1 Address Register Select Writing to the rx_addr_reg (Location: BASE + 0x24) when this bit is asserted writes to the Class 1 Address Register
23	'0'	LABEL: rx_cmmd_23 (read/write) Class 3 Buffer Clear for next message Setting this bit after a class 3 message transfer clears the class 3 message acknowledgement flag required to receive another class 3 message
24	'0'	LABEL: rx_cmmd_24 (read/write) Class 2 Buffer Clear for next message Setting this bit after a class 2 message transfer clears the class 2 message acknowledgement flag required to receive another class 2 message
25	'0'	LABEL: rx_cmmd_25 (read/write) Class 1 Buffer Clear for next message Setting this bit after a class 1 message transfer clears the class 1 message acknowledgement flag required to receive another class 1 message
26	'0'	LABEL: rx_cmmd_26 (read/write) Class 2 Length Preset Writing to the receiver header register (Location: BASE + 0x24) when this bit is asserted automatically classifies the message as belonging to class 2
27	'0'	LABEL: rx_cmmd_27 (read/write) Class 1 Length Preset Writing to the receiver header register (Location: BASE + 0x24) when this bit is asserted automatically classifies the message as belonging to class 1
28	'0'	LABEL: clr_cam (read/write) Delete all message IDs from the CAM Asserting this bit removes all message IDs from the CAM
29	'0'	LABEL: rx_osff_autoflush (read/write) Receiver Link Interface Buffer Autoflush Select Setting this bit flushes a message form the receiver link interface buffer

Bit	State after reset	Description
30	'0'	LABEL: rx_cmmd_30 (read/write) Message ID Probe Mode Select Writing a message ID to the Receiver Header Register (Location: BASE + 0x28) when this bit is asserted probes the contents of the CAM to determine if that message ID is stored in the CAM and its location. CAM contents are NOT deleted (including class 3 message IDs)
31	'0'	LABEL: rx_cmmd_31 (read/write) Message ID Delete Mode Select Writing a CAM location to the Receiver Length Register (Location: BASE + 0x38) when this bit is asserted deletes the contents of this CAM location

Class 1 Length Register (Location: BASE + 0x40)

Bit	State after reset	Description
[23:0]	'0'	LABEL: class1_lgth (read/write) Class 1 Length Preset Value This value is the maximum allowable message length for a Class 1 message
[31:24]	'0'	These bits are unused and hardwired to '0'.

Class 2 Length Register (Location: BASE + 0x44)

Bit	State after reset	Description
[23:0]	'0'	LABEL: class2_lgth (read/write) Class 2 Length Preset Value This value is the maximum allowable message length for a Class 2 message
[31:24]	'0'	These bits are unused and hardwired to '0'.

Status Register READ ONLY (Location: BASE + 0x38 – Upper section of Receiver Message Length Register)

Bit	State after reset	Description
[19:0]	'0'	These bits are unused, hardwaired to '0' and masked by the Receiver

Bit	State after reset	Description
		Message Length Register
[22:20]	'0'	LABEL: Message Transfer Complete Flags These flags were set for classes 3 to 1 respectively to acknowledge message transfer to memory. They were cleared following acknowledgement from the PC that the message had been handled and the memory allocation for that class could be overwritten.
[23]	'0'	Reserved for future use.
[26:24]	'0'	LABEL: Class Match Asserted following a successful write of an expected message ID to the CAM denoting which class it belonged to. Asserted until a successful write to a different class forced it to change. Bits 26 to 24 were set for classes 3 to 1 respectively.
[27]	'0'	LABEL: No Match Set if the received header did not match any stored in the CAM. Asserted until a match was found, whether for that header or for others.
[29:28]	'0'	LABEL: Incoming Message Length Mux Debug lines used to multiplex the message lengths following a received header match. 00_B – Class 1, 01_B – Class 2, 10_B – Class 3 (location 1), 11_B – Class 3 (location 2).
[30]	'0'	LABEL: CAM Write Failed Set following an unsuccessful expected header write to the CAM due to that particular class being full. Asserted until success occurred.
[31]	'0'	LABEL: CAM Write Successful Set following a successful expected header write to the CAM. Asserted until failure occurred.

Appendix D: FT-SARNet Control Token Definitions

This appendix details the control tokens that have been specified to date, for use with the FT-SARNet. Control tokens are denoted by an ID bit of zero. The eight data bits determine the nature of the control token. All unspecified combinations are available for future use. A note for each token is included specifying whether the token reaches the end nodes (FT-PCI-OSLi and FT-SARNIC) and if so, how far into the design it progresses.

Token Name	Coding [Type, LSB...MSB]	Token Function	Token Stripped at:
CONREQ	000000000	Connection Request	Link Interface
XON	001100000	Permit Transmission	Link Interface
XOFF	001000000	Inhibit Transmission	Link Interface
EOM	011000000	End of Message	Depacketiser
EOP	010000000	End of Packet	Depacketiser
BEOP	010100000	Bad End of Packet	Depacketiser
DLPRB	000100000	Deadlock Probe	Router
DLCLR	000110000	Deadlock Path Clear	Router
DLMOV	000111000	Deadlock Data Movement	Router

Appendix E: FT-PCI-OSLi Configuration Registers Contents

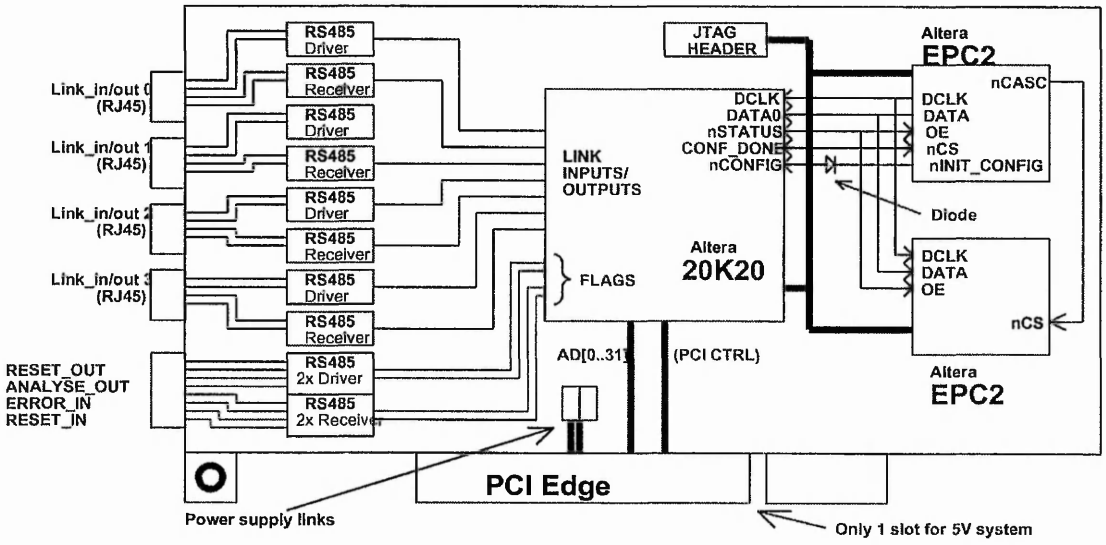
This appendix details the contents of the configuration registers of the FT-PCI-OSLi as defined by the PCI specification version 2.1.

Base Address Offset	Reg No	Contents
0x00	0	Device ID[31:16], Vendor ID[15:0]
0x04	1	Status[31:16], Command[15:0]
0x08	2	ClassCode[31:8], Revision ID[7:0]
0x0C	3	BIST[31:24], Header Type[23:16], Latency Timer[15:8], Cache Line Size[7:0]
0x10	4	Base Address Register 0[31:0] (this design decodes 1 MBytes memory locations)
0x14	5	Base Address Register 1 (not implemented)
0x18	6	Base Address Register 2 (not implemented)
0x1C	7	Base Address Register 3 (not implemented)
0x20	8	Base Address Register 4 (not implemented)
0x24	9	Base Address Register 5 (not implemented)
0x28	10	CardBus CIS Pointer[31:0] (not implemented)
0x2C	11	Sub-system ID[31:16], Sub-system Vendor ID[15:0]
0x30	12	Expansion ROM Base Address Register (not implemented)
0x34	13	[Reserved]
0x38	14	[Reserved]
0x3C	15	Maximum Latency[31:24], Minimum Grant[23:16], Interrupt Pin[15:8], Interrupt Line[7:0]

Appendix F: Specification for PCI/RS485 Interface board

Design Overview

The interface board is to have standard PCI edge connection at the bottom, with 4 off serial connections to the left of the board, mounting on the metal back-plate. Interface to the PCI bus is to be handled by a suitable PLD, for which settings need to be stored in compatible configuration device(s). PLD settings should be transferred using the JTAG ISP(In-System-Program) connection. Interface to the serial connections is made by RS485 balanced line transceiver circuits. The diagram below shows the basic building blocks. These blocks are described in the following sections.



System diagram for the PCI-RS485 interface board

Interface to the PCI bus

Board format is to be 32-bit PCI bus connection on a standard short card implementing the 5V system (single connector key furthest from the back plate).

Pin-out for PCI edge connector on 5V / 32bit system

Pin	Signal Name	Description
B1	-12V	(not used)
A1	TRST#	(PCI JTAG – not used)

Pin	Signal Name	Description
B2	TCK	(PCI JTAG – not used)
A2	+12V	+12V Supply to D-connect only
B3,B15,B17,A12,B12, A13,B13,A18,B22,A24, B28,A30,B34,A35,A37, B38,A42,B46,A48,A56, B57	GND	Ground-plane connection Decoupling capacitance > 0.01uF per Vcc pin, equally distributed
B49	M66EN	Connect to GND / (Only relevant for 66MHz)
A3	TMS	(PCI JTAG – not used)
B4	TDO	(PCI JTAG – not used – link to A4/TDI)
A4	TDI	(PCI JTAG – not used – link to B4/TDO)
B5,A5,B6,A8,B61,A61, B62,A62	+5V	+5V for RS485 drivers
A6	INTA#	PCI interrupt for PLD
B7	INTB#	(not used)
A7	INTC#	(not used)
B8	INTD#	(not used)
B9	PRSNT1#	Link – GND or NC for power requirement
A9,B10,A11,B14,A14, A19	(RESERVED)	(not used) – These pins should NOT be commoned
A10,A16,B19,B59,A59	5V_I/O	(not used) / universal board PLD power only
B11	PRSNT2#	Link – GND or NC for power requirement
A15	RST#	
B16	CLK	PCI clock output to PLD
A17	GNT#	
B18	REQ#	
A58,B58,A57,B56,A55, B55,A54,B53,B52,A49, B48,A47,B47,A46,B45, A44,A32,B32,A31,B30, A29,B29,A28,B27,A25, B24,A23,B23,A22,B21, A20,B20	[AD0..31]	32 bit PCI bus connections
A21,B25,A27,B31,A33, B36,A39,B41,B43,A45, A53,B54	3V3	3V3 power-plane connection

Pin	Signal Name	Description
B26	C/BE3#	
A26	IDSEL	
B33	C/BE2#	
A34	FRAME#	
B35	IRDY#	
A36	TRDY#	
B37	DEVSEL#	
A38	STOP#	
B39	LOCK#	
B40	PERR#	
A40	SDONE#	
A41	SB0#	
B42	SERR#	
A43	PAR#	
B44	C/BE1#	
A52	C/BE0#	
B60	ACK64#	64bit format only (not used)
A60	REQ64#	64bit format only (not used)

Maximum power requirement for the board is to be set by links between PRSNT1#/PRSNT2# pins and GND. Once known these short cut jumpers could be replaced with solder links.

Link settings for power requirement

PRSNT1#	PRSNT2#	Expansion Configuration
OPEN	OPEN	No board present
GND	OPEN	Expansion board present, 25W maximum
OPEN	GND	Expansion board present, 15W maximum
GND	GND	Expansion board present, 7.5W maximum

PCI Control through PLD

Interface to the PCI bus is to be handled by the Altera programmable logic device (PLD) – 'Apex 20K200', powered from 3.3Vdc, in PQFP package. The PCI signalling is at 3V3 level, whereas internal voltage is 1V8dc, therefore step-down on board is needed to supply these pins (VCCINT)
The PLD is not 5V tolerant, so any 5V logic will need converting before connection to PLD pins (Serial transceivers run at 5V, additionally reference White Paper A-WP-APEX5V-01.02 from Altera regarding the PCI connections).

FT-PCI-OSLi pin connections assigned to an EP20K200EQC240-1

Pin	Signal Name	Description
1,5,14,27,32,39,52,60, 90,122,127,140,144,145, 159,168,176,179,210	VCCINT	Connects to 1V8dc derived from +3V3 pins on edge connect
12,45,67,97,120,148,177, 199,229	VCCIO	Connects to +3V3 pins on edge connect
142	VCC_CLKOUT	(not used)
6,15,19,26,28,38,42,51,56, 78,89,108,128,132,137, 139,146,155,162,165,167, 175,188,211,218,240	GND	Dedicated ground pin, must be connected to GND
9,10,11,13,16,17,18,20,21,22, 23,24,25,35,36,37,40,41,43, 44,47,48,49,50,53,57,58,54, 55,59,72,73,106,107,129, 130,131,133,134,135,136, 138,143,156,157,160,161, 163,164,166,169,178,180, 181,185,187,189,190,191, 192,193,194,195,196,197, 198,200,201,202,203,204, 205,206,207,221,222,223, 224,225,226,227,228,230, 231,232,233,234,236,237, 238,239	GND*	Unused I/O pins. These pins can either be left unconnected or connected to GND. Connecting these pins to GND will improve the device's immunity to noise.
34,154,209,212	GND+	Unused inputs. These pins should be connected to GND
147,158	GNDINT	Dedicated ground pins, which must be connected to GND

Pin	Signal Name	Description
141	GND_CLKOUT	(not used)
125,124,123,121,119,118, 117,116,114,113,112,111, 110,109,105,104,84,83,82, 81,80,79,77,76,71,70,69,68, 66,65,64,63	AD[0..31]	32 bit PCI bus connections
29,30	MSEL0,1	Connect to GND
31	CLK	PCI clock input from edge connect
33	CONFIG#	
46	RST#	
61	GNT#	
62	REQ#	
74	C/BE3#	
75	IDSEL	
85	IRDY_O#	
86	TMS	JTAG mode select
87	TCK	JTAG clock signal
88	IRDY_I#	
91	TRDY_I#	
92	STATUS#	
93	CONF_DONE	
94	TRDY_O#	
95	C/BE2#	
96	FRAME#	
98	DEVSEL#	
99	STOP#	
100	PERR#	
101	SERR#	
102	PAR	
103	C/BE1#	
115	C/BE0#	
126	INTA#	PCI interrupt
149	TDI	JTAG data to device
150	CE#	
151	OS_CLK	Connects to 30MHz clock for links
152	DCLK	Configuration address clock
153	DATA0	Configuration data input

Pin	Signal Name	Description
208	TDO	JTAG data from device
213	CEO#	
214	TRST	
2	RESET_OUT	
3	ANALYSE_OUT	
4	ERROR_IN	
7	RESET_IN	
170	OS_LINK_IN1	
171	OS_LINK_OUT1	
172	LINK_SPEED_SEL1	
173	OS_LINK_IN2	
174	OS_LINK_OUT2	
182	LINK_SPEED_SEL2	
215	OS_LINK_IN3	
216	OS_LINK_OUT3	
235	LINK_SPEED_SEL3	
217	OS_LINK_IN4	
219	OS_LINK_OUT4	
220	LINK_SPEED_SEL4	

Configuration of the PLD

As the chosen PLD is SRAM based, configuration data must be re-loaded each time the system initialises, or when new configuration data is needed. As the configuration data for the 20K200 is too large for one storage device, it is necessary to cascade two Altera configuration devices – 'EPC2', both powered from 3.3Vdc (in 20pin PLCC format). These are cascaded using the nCASC pin of device one connecting to nCS of device two, providing the necessary handshaking.

The JTAG 'initiate Configuration' feature is supported by the insertion of a diode (with threshold voltage Vt less than or equal to 0.7V) between the 20KE #Config pin and the EPC2 #INIT_CONF pin, as shown in the system diagram on page1. The diode effectively makes the #INIT_CONF pin open-drain, and it will only be able to drive low or tri-state. [reference 20KE errata sheet M-ES-APEX-01.1].

When powered up and nCS pin driven low, device one controls configuration of the PLD. After device one has finished sending data, its nCASC pin is driven low, which being as it is connected to nCS on device two, causes device two to send the remaining configuration data. Device one clocks device two until configuration is complete. Once configuration is complete and the nCS pin on device one is driven high by PLD's CONF_DONE pin, device one continues to clock an additional 16 clock

cycles to initialise the PLD. Device one then goes into zero-power (idle) state. If nCS on the master EPC2 is driven high before all configuration data is transferred, the master EPC2 device drives the PLD nSTATUS pin low, indicating a configuration error.

Pin connections for the Altera EPC2 configuration devices (20 pin PLCC package)

Pin	Signal Name	Description
1	TDO	JTAG data output
2	DATA	Serial data output
3	TCK	JTAG clock
4	DCLK	Clock output from master EPC2 Clock output to slave EPC2 Drives low on configuration end
5	VCCSEL	Mode select for VCC (connect to VCC if powering from 3V3, or connect to ground if powering from 5V)
6,7,15,16,17	(N.C.)	(no connection)
8	OE/RST#	Low resets address counter High enables the counter
9	CS#	Chip select
10	GND	Ground pin (decouple with 0.2uF)
11	TDI	JTAG data input
12	CASC#	Cascade select output
13	INIT_CONF#	Allows JTAG INIT_CONF instruction to initiate configuration
14	VPPSEL	Mode select for VPP. Programming voltage set to 3V3 by connecting this pin to VCC or set to 5V by connecting to GND
18	VPP	Programming power pin, normally tied to VCC
19	TMS	JTAG mode select
20	VCC	Power supply, +3V3

'Technical' Index

No	Ref	Name	No	Ref	Name
1	190	A Route	51	167	Eliminate
2	239	A6	52	165	Energy
3	145	Aberration	53	25	Entity
4	50	Above The	54	58	Escape
5	I	Abstract	55	57	Evasion
6	16	Aerospace	56	27	Exclusive
7	21	After	57	91	Exit
8	28	Aid	58	167	Extra Over
9	29	Al	59	94	Extraction
10	66	Analysis	60	120	Fresh
11	38	And Others	61	245	Grounded
12	99	Andover	62	199	Hardcore
13	48	Another Route	63	200	High Performance
14	50	Another Time	64	31	Impede
15	86	Apex	65	46	In Excess
16	163	Approach	66	15	In Retrospect
17	72	Approaching	67	25	Inception
18	11	Artificial	68	145	Inclusion
19	50	Assembled	69	167	Incursion
20	59	Auto	70	181	Injection
21	74	Autonomy	71	65	Insertion
22	V	Avoidance	72	66	Integrity
23	17	Bottleneck	73	1	Introduction
24	42	Bypass	74	II	Isolation
25	88	Cascade	75	240	Jumpers
26	6	Catalyst	76	245	L'Index
27	165	Clit	77	53	Midway
28	176	Command	78	19	Oblique
29	25	Connect 4	79	5	Original Route
30	88	Creation	80	6	Parallel
31	1	Critical	81	1	Relative Ease
32	134	Crystal	82	37	Responsibility
33	58	Daisy Chain	83	240	Short Cut
34	58	Deflection	84	5	Simple Solution
35	91	Desire	85	86	Substitute
36	99	Destination	86	III	Thanks
37	43	Detour	87	188	The Boom
38	172	Deviation	88	245	The Chain
39	192	Differential	89	1	The Critic
40	28	Direct Route	90	26	The First One
41	23	Division of Labour	91	64	The Medium
42	77	Domination	92	69	The Message
43	101	Double	93	167	The Ramp
44	III	Drag	94	6	The Spur
45	128	Drought	95	54	The Start
46	X	Dynamic	96	158	The Steps
47	54	E Star	97	163	Tight Fit
48	54	Edge	98	55	Trigger
49	48	Ejection	99	145	Toward
50	165	Electron	100	83	Via Media

JTAG ISP connection

The PLD settings should be transferred using the JTAG ISP (In-System-Program) connection as outlined in IEEE Std 1149.1 specification. This connects to a suitable interface cable such as Altera MasterBlaster for download of configuration data and also in-circuit debugging from a remote computer. Since the PLD and configuration devices are powered at 3V3, programming is also done at this level by connection of VCC and VIO JTAG pins to the +3V3 power plane. Function described in previous section.

Pin connections for JTAG 10way header

Pin	Signal Name	Description
1	TCK	Clock Signal
2	GND	Signal grounded
3	TDO	Data from device
4	VCC	Power Supply
5	TMS	JTAG mode select
6	VIO	Reference voltage for programmer output driver
7	–	No connection
8	–	No connection
9	TDI	Data to device
10	GND	Signal ground

Interface to the serial links

Interface to the four serial links is via four independently controlled RS485 balanced line transceiver circuits. These links need to be capable of data transfer up to 20mbps. Circuits to be based around the Analog Devices part no.: ADM1485 powered from 5Vdc (split power-plane required). As these transceivers do not have separate driver and receiver sections, then a separate device will be necessary to perform these functions (i.e.: 8x ADM1485 necessary to facilitate the four bi-directional serial links).

Status flag inputs/outputs do not require the high speed switching allowed by the ADM1485 and as such a more compact device is available to interface these signals, namely National Semiconductors' DS34C86T/DS34C87T receiver/driver combination which allow 4 signals per device, saving board space. These are also powered from 5Vdc. Status flags to be incorporated are: [RESET_OUT / ANALYSE_OUT / ERROR_IN / RESET_IN]. Both the ADM1485 and DS34C8xT employ 5V signalling and must therefore be converted to 3V3 signals before connection to the PLD link inputs/outputs, using potential dividers.

Pin connections for ADM1485 Differential line transceiver

Pin	Signal Name	Description
1	RO	Receiver output
2	RE#	Receiver enable (Active low)
3	DE	Driver enable (Active high)
4	DI	Driver input
5	GND	Ground
6	A	Non-inverting receiver input A / Driver output A
7	B	Inverting receiver input B / Driver output B
8	VCC	Power supply, +5V

Pin connections for DS34C86TN Differential line receiver

Pin	Signal Name	Description
4,12	EN	Enable A+B, C+D
2,6,10,14	+IN	A,B,C,D non-inverting inputs
1,7,9,15	-IN	A,B,C,D inverting inputs
3,5,11,13	OUT	A,B,C,D outputs
16	VCC	+5V power supply
8	GND	Ground connection

Pin connections for DS34C87TN Differential line driver

Pin	Signal Name	Description
4,12	EN	Enable A+B, C+D

2,6,10,14	+OUT	A,B,C,D non-inverting outputs
3,5,11,13	-OUT	A,B,C,D inverting outputs
1,7,9,15	IN	A,B,C,D inputs
16	VCC	+5V power supply
8	GND	Ground connection

External connections to the serial inputs/outputs are to be made on a right-angle D-type plug connector mounted on the metal back plate. The +12V auxillary output will be dual powered from either the PCI edge-connect (with diode protection) or on-board disc-drive type power connect input (for higher power usage).

Pin connections for D37P connector

Pin	Signal Name	Description
1	LINK1_TX+	Link output1 (non-inverted)
20	LINK1_TX-	Link output1 (inverted)
2	LINK2_TX+	Link output2 (non-inverted)
21	LINK2_TX-	Link output2 (inverted)
3	LINK3_TX+	Link output3 (non-inverted)
22	LINK3_TX-	Link output3 (inverted)
4	LINK4_TX+	Link output4 (non-inverted)
23	LINK4_TX-	Link output4 (inverted)
5	RESET_OUT+	Reset output (non-inverted)
24	RESET_OUT-	Reset output (inverted)
6	ANALYSE_OUT+	Analyse output (non-inverted)
25	ANALYSE_OUT-	Analyse output (inverted)
7	LINK1_RX+	Link1 input (non-inverted)
26	LINK1_RX-	Link1 input (inverted)
8	LINK2_RX+	Link2 input (non-inverted)
27	LINK2_RX-	Link2 input (inverted)
9	LINK3_RX+	Link3 input (non-inverted)
28	LINK3_RX-	Link3 input (inverted)
10	LINK4_RX+	Link4 input (non-inverted)
29	LINK4_RX-	Link4 input (inverted)
11	ERROR_IN+	Error input (non-inverted)
30	ERROR_IN-	Error input (inverted)
12	RESET_IN+	Reset input (non-inverted)
31	RESET_IN-	Reset input (inverted)

18,19	GND	Ground connection
36,37	+12V	+12V output allowing 500mA

Appendix G: FT-PCI-OSLi Power Consumption Calculation

This appendix details the power consumption calculations, the result of which was noted in section 6.6.

The Icc_{INT} standby current for an Apex 20K200E 240 pin plastic quad flat pack (PQFP) device with a 1.8V internal voltage, for reduced power consumption, was specified at 10mA. P_{INT} (standby) was therefore 18mW.

The report file generated after synthesis specified that the FT-PCI-OSLi possessed 1435 flip-flops clocked at a frequency of 33MHz and 185 flip-flops clocked at 30MHz. These parameters gave internal Icc_{INT} values of 30.74mA and 20.07mA respectively. The internal power consumption was 55.34mW and 36.13mW respectively, based on a 1.8V internal voltage.

A total of 3109 Logic Elements (LEs) were implemented in the FT-PCI-OSLi. The total implemented in each clock domain was not specified so they were all calculated for a 33MHz clock frequency, to give a faster switching speed and thus greater power consumption. The report specified the average fan-out to be 3.90, and assumptions were made to determine the total number of LEs with a Carry Chain. The report file indicated that the design possessed:

10 carry chains between 0 and 2 LEs long,
21 carry chains between 3 and 5 LEs long,
10 carry chains between 6 and 8 LEs long,
1 carry chain between 9 and 11 LEs long,
3 carry chains between 15 and 17 LEs long,
2 carry chains between 18 and 20 LEs long and
9 carry chains between 30 and 32 LEs long.

The average length of the chains was determined, giving;

10 carry chains 1 LE long,
21 carry chains 4 LEs long,
10 carry chains 7 LEs long,

1 carry chains 10 LEs long,
3 carry chains 16 LEs long,
2 carry chains 19 LEs long and
9 carry chains 31 LEs long.

This gave a total of 539 LEs with Carry Chains. The literature specified an assumed average LE toggle of 12.5%, implying that only this percentage of outputs would change state at any one time, on average. The power calculator determined the internal power consumption to be 44.37mW and Icc_{INT} of 24.65mA.

The FT-PCI-OSLi contained 83 Embedded System Blocks (ESBs), all driven by the 33MHz PCI clock. All ESBs outputs were set for the Turbo mode of operation, increasing speed at a cost of increased power consumption. The ESB outputs had an assumed average specified toggle of 12.5%. The power calculator determined the Icc_{INT} and P_{INT} values to be 24.04mA and 43.27mW respectively.

The FT-PCI-OSLi possessed 46 output and bi-directional pins synchronised at 33MHz and one output pin driven at 30MHz (the communications link output). All outputs operated at the 3.3V PCI standard. All outputs were specified with an assumed average toggle of 12.5%. These outputs consumed Icc_{IO} current values of 3.13mA and 0.06mA respectively, and had a P_{IO} power consumption of 10.33mW and 0.2mW respectively.