A MESSAGE CONTROLLER FOR DISTRIBUTED PROCESSING SYSTEMS

KAR LEONG WONG

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

June 2000

ABSTRACT

This thesis presents a novel message-passing hardware network interface controller which has been designed and developed for integration into a router interconnection network for distributed parallel processing systems. It describes how the network interface controller can improve the efficiency of message transfers in packet switching based communications and reduce the overheads incurred to microprocessor tasks.

Initially, a review focusing on the architectures and techniques of interfacing inter-processor interconnection networks to parallel processor computing nodes was carried out. Various parallel processing system packet routing devices and network interface controllers have been investigated. Following the review, a novel network interface controller was designed, to link a microprocessor node to a parallel processor system, interconnection network.

The network interface controller design was captured in a Hardware Description Language (VHDL) following a top-down design methodology. A series of comprehensive tests were written to verify the functionality of the design model. The design was synthesised into a target programmable logic device, tested via a working prototype processor node incorporating the StrongARM SA-110 microprocessor. This was followed by the construction of a distributed parallel processing system using an ICR C416 packet routing interconnection network.

The successful implementation has demonstrated how an efficient inter-processor communication can be achieved using the network controllers to link to an ICR C416 packet routing network to StrongARM microprocessor nodes. This offers the processing power of high performance microprocessors in an embedded distributed parallel processing system. Key features of the system incorporating the network controller are discussed and the system is compared and contrasted with the state-of-the-art in parallel processing communication networks.

ACKNOWLEDGEMENTS

I would like to dedicate this dissertation to my parents. Without the courage and support from my father, I would not be able to learn that much. For my mother who is so far away, I hope she is happy and proud for what I have achieved today.

I am grateful to my supervisor, Brian O'Neill, who always found new ideas to push me forward. Also for those moments where he sat down with me, side by side, troubleshooting the tough problems faced. To my second supervisor, Steve Clark, thank him very much for his kind counciling and guidance, especially in preparing this thesis.

Special thanks to Robin Hotchkiss for technical assistance with StrongARM processing nodes, and for correcting my English. Thanks to Gary Coulson for guiding me at the initial stage, and also to Jien Hau Ng for the advice and 'coffee time' spent together. Not forgetting Kenny Liew who developed software and tests for the StrongARM-Router Network system.

Finally, I would like to thank Alec Crawley of Quantel Ltd. for his helpful discussion at the initial design stage of the project.

To all that I haven't mentioned, thanks.

TABLE OF CONTENTS

LIST OF ACRONYMS

ASIC	Application Specific Integrated Circuit
CAM	Content Addressable Memory
CPLD	Complex PLD
CPU	Central Processing Unit
DMA	Direct Memory Access
DRAM	Dynamic RAM
EAB	Embedded Array Block
FIFO	First In First Out
MIMD	Multiple Instructions Multiple Data
I/O	Input/Output
PC	Personal Computer
PCB	Printed Circuit Board
PCI	Peripheral Component Interconnect
PLD	Programmable Logic Device
RAM	Random Access Memory
RISC	Reduced Instruction Set Computing
ROM	Read Only Memory
RTL	Register Transistor Logic
SARNIC	StrongARM-Router Network Interface Controller
SDRAM	Synchronous DRAM
SRAM	Static RAM
UART	Universal Asynchronous Receiver Transmitter
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit
VLSI	Very Large Scale Integration

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

1. INTRODUCTION

Many hands make light work.

John Heywood, *Proverbs*

Computers were designed to solve problems: from basic calculation to complex simulation. Over the decades, new computer architectures and device technologies have always been exploited to improve system performance. However, for every technology breakthrough, there is still a limit to the performance that a single processor can provide; there are always problems that are beyond the capabilities of one individual processor [1]. In addition, there is a belief that the speed of the processor will soon reach the physical limits imposed by the speed of light and quantum physics effects [2], though computers today have reached clock speeds approaching GHz rates. To obtain truly significant performance improvements, and to handle ever-bigger tasks, a viable approach is parallel processing [3]. In general, parallel processing can be defined as a concept of several processing entities, either homogeneous (identical) or heterogeneous, co-operating together on the solution of a problem.

The idea of parallel processing for the solution of complex problems has only become possible in recent decades. A number of different parallel system architectures have been introduced. Early generations of parallel machines were supercomputers that were used essentially in highly numerical intensive research and scientific areas. These supercomputers relied on custom built, high speed circuits and mostly utilised the principles of pipeline processing or array processing as their underlying architecture [4]. However, with the advance of Very Large Scale Integration (VLSI) technology, microprocessors have become more powerful, cheaper and smaller in size. This has given a large boost to the development of parallel systems that are constructed by interconnecting low-cost, off-the-shelf microprocessors. By connecting relatively fewer but more powerful microprocessors, the cost of parallel systems was reduced, while offering performance approaching that of supercomputers. M. J. Flynn classified this architecture, of connecting multiple microprocessor devices to process multiple data concurrently, as the Multiple Instructions Multiple Data (MIMD) model [4, 5]. This MIMD model is the most common model for general parallel computation, examples include: nCUBE 2, Paragon XP/S, and Connection Machine CM-5 [6, 7].

A significant aspect of parallel computers is the mechanism that the processors use to exchange information. There are two general alternatives to the mechanism used, known as shared memory and message passing [8, 9]. The first alternative uses a global shared memory that can be accessed by all processors. A processor can communicate to another by writing into the global memory, and then having the second processor read that same location from the memory. This type of parallel system is generally known as multiprocessor system, and the processors are normally closely coupled. In the second alternative, each processor has its own local memory. Processors communicate by passing messages through an interconnection network consisting of communication links. This type of parallel system is generally known as multicomputer system, and the processors are normally distributed (sometimes physically separated by a large distance). Shared memory multiprocessor systems are attractive, as they are relatively simple to program, while distributed memory multicomputer systems have better system scalability. Over the years, as both classes of architecture improved, the boundaries between multiprocessors and multicomputers have become blurred. Most of the scaleable parallel systems nowadays have a similar architecture consisting of a group of processing nodes connected by an interconnection network [10, 11], as shown in Figure 1. Physically distributed memories can be globally shared through virtual address space, with the help of some memory-mapping techniques in the network interface, or with a software translation layer [12].

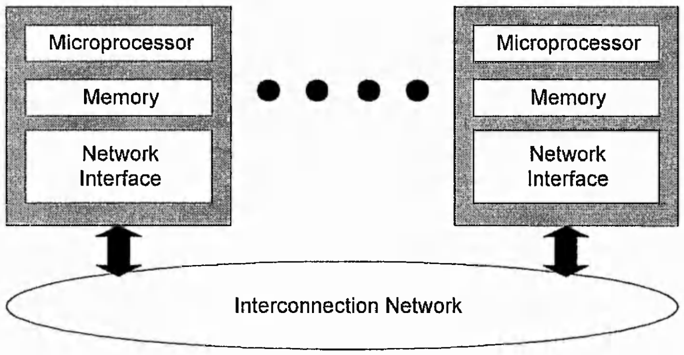


Figure 1: The standard architecture of scaleable parallel systems.

As illustrated in Figure 1, there are generally four major components in a scaleable parallel system: the microprocessor that features a processing unit; the memory that is used for program and data storage; the interconnection network that provides the communications path between processing entities; and the network interface that couples the processing entities to the communication network. Microprocessors and their associated memory chips have become inexpensive and widely available due to the dominance of uni-processor systems. Much research in parallel processing systems has thus focused on the interconnection network and the network interface. This includes design and development of hardware routing devices that provide efficient inter-processor communication, and hardware communication controllers in the processor node that offload most of the communication tasks of the microprocessor.

1.1 The Interconnection Network

An interconnection network is a medium that enables the establishment of communication channels amongst connected processing nodes. Ideally, an interconnection network should provide direct point-to-point connection to all processing nodes in the system. However, this type of network is only feasible for connecting a small number of processing nodes; when these systems scale up, packaging constraints and hardware costs tend to limit the number of connections that can be implemented [13]. Therefore, parallel processing systems normally have their nodes connected with a limited number of communication links, in a specific network topology.

Most of the early network topologies are configured in a static way where the processing elements are directly linked in a fixed network connection style. This reduces the cost and complexity of the interconnection network, but it is only suitable for parallel systems that have a specific communication patterns. For generality in parallel tasks, dynamic connections in the network topology are preferred, where the communication pattern can be defined based on applications [14]. This requires the use of switches or arbiters along the connecting paths. Two major categories of dynamic network topology that are generally used are bus systems and switch systems.

Bus topology networks are very common in multiprocessor systems. All processor nodes share the use of a common communication path called a bus [15]. Only one communication transaction can happen at a time. An arbiter is thus provided to resolve the bus contention. The processor nodes connected to a bus system are likely to be closely coupled. Many systems require the inter-processor node distances to be limited so that the signals propagation delay is bound for correct arbitration. Generally bus systems are low-cost and low latency with closely coupled architecture. However, they suffer the problem of scaling. As the number of processors trying to access the bus increases, the contention between the processors rises accordingly. Access to bus eventually becomes a bottleneck, limiting the speed of the computation system. Hence, most bus systems posses very high bandwidth, in order to cover the average bandwidth required by a predefined number of connected processors.

Switch topology networks utilise complicated devices called switches or routers that offer separate connection paths for each processor node. Thus simultaneous data transfer can be done across the network providing the simultaneous data are not heading for the same destination. Some switches are also equipped with a queuing function that queues the data transfers during contention at the destination output. The switch topology is better than the bus network in term of system scalability, as the technology provides dedicated bandwidth for each connected node. The processor nodes connected in these systems can be closely coupled or loosely coupled. The allocated communication path for each connected processor allows independent transfer operations. In switching networks, complexity increases exponentially with the number of connection ports implemented. However, switching networks offer high overall bandwidth and improved routing capability. In fact, most recent interconnection network research has focused on the switching topology; mainly encouraged by the improvements in silicon technology that has reduced the complexity and the cost of the switching device implementation. The Cray T3D [16] interconnection network, the Myrinet [17], and to a lesser extend, the ICR C416 [18] are some commercially successful examples of such switching devices.

1.2 The Processor Node and the Network Interface

As shown in Figure 1, the processor node of a scaleable parallel processing system generally consists of a microprocessor, memory, and a network interface. The co-operation between these components determines the efficiency of the information passing, to and from the communication network. Each interface offers different advantages and disadvantages, from loosely-coupled network interfaces that are connected through a system Input/Output (I/O) expansion bus, to tightly-coupled network interfaces that are integrated onto the same microprocessor silicon chip.

Tightly-coupled network interfaces are well known for high performance, especially in communications. They offer low message latency with a special tailored interface and specific instructions for communications. For instance some integrated network interfaces allow message transfers by reading from, and writing to, special registers or executing some specialised instructions [19, 20]. This type of architecture simplifies the processing node implementation, minimising the number of additional components used. The Inmos Transputer on-chip interface was a successful example [21]. Despite the excellent performance, the built-in tightly coupled network interface requires a great engineering effort to integrate it with the microprocessor core on the same silicon chip. Study of the Transputer family history has shown that the upgrade of either the microprocessor core or the interconnection network/network interface was a time consuming and costly job, mainly due to the total integration on a single chip.

Most of the parallel systems today utilise a loosely-coupled network interface that is connected through a system I/O expansion bus. This type of network interface can be easily adapted across a family of different microprocessors, workstations, and Personal Computers (PCs). Utilisation of modern workstations/PCs as parallel machine building blocks is an economical solution and allows technology migration closely following the rapid workstations/PCs evolution. These system units are high performance and reasonably cheap nowadays. Moreover, the operating system kernel for these parallel systems can be modified from the existing system software. Examples of these systems are the Myrinet/PCI host interface card [22] and the first version of the SHRIMP design [23]. However, due to the overhead of accessing the I/O bus, it is preferable to carry out large Direct Memory Access (DMA) transfers across the bus. Not only does this increase the message buffering latency in the network interface, it also dominates the memory bus usage over the whole period of transfer. Most I/O bus based network interfaces are designed with large amount of buffering to support this operation mode.

As a compromise between the advantages and disadvantages of tightly-coupled and loosely-coupled interfaces, some architectures have been proposed that place the network interface on the memory bus or even integrate it with the cache controller [24, 25]. Moving the interface to the microprocessor memory bus should provide better message latency than coupling through I/O expansion bus, while offering a certain level of flexibility of choice of state-of-the-art microprocessors as processing elements. By optimising a dedicated communication path between the network interface and the memory bus, unnecessary arbitration latency of crossing from I/O bus to memory bus and bandwidth competition with other I/O bus devices are avoided. It also allows the maximum communication bandwidth for the network interface, most likely limited by the memory bus bandwidth available and utilisation percentage of the Central Processing Unit (CPU), but not by the inherent I/O bus speed. However, this type of interface requires the aid of a custom hardware device to handle memory bus access requests and the network interfacing.

1.3 Research History and Objectives

The Nottingham Trent University parallel processing research group has been carrying out research and design in inter-processor communications since 1989 [26, 27, 28]. Initial research interest was in Transputer systems. Transputers were designed as building blocks for parallel processing systems. Each chip was equipped with some internal memory, external memory control logic, I/O device interface, and four bi-directional serial communication links for message passing. By connecting the Transputers through the serial communication links, a parallel system network could be easily constructed. However, as the size of Transputer arrays grew, the interconnections became more complex, and considerable amounts of the processing power of the Transputer had to be allocated for communication software control [26, 29].

As a solution to the communication difficulties described above, dynamic hardware packet routing switches were designed and developed in The Nottingham Trent University, which then led to the realisation of the commercial ICR C416 [18, 30] device. The ICR C416 is a 16-link packet routing switch that was used to provide non-blocking messages routing for Transputer parallel systems; off-loading the communication tasks from the Transputer microprocessors. This device could be cascaded to support scaleable systems. The use of the ICR C416 device in Transputer systems demonstrated the efficiency of routing devices as a solution to medium scale, low-cost, high performance inter-processor communications [31].

Following the successful implementation of the ICR C416, a diverse range of inter-processor communication analyses was carried out. This included implementation of a multicast function on the hardware routing device [32] and study of different types of communication link flow control effects [33]. The successful use of the ICR C416 device, for Transputer networks, also raised interest in using the router switch to interface other families of high performance microprocessors. In this way, the parallel system formed could benefit from the rapid advances of microprocessor design, while the efficient switching communication network core could still be maintained.

Since the demise of the Transputers, attention focused on incorporating state-of-the-art Reduced Instruction Set Computing (RISC) microprocessors into parallel processing systems. However, it was desirable to retain many of the advantages offered by the Transputer family. For instance, Transputers achieved low message latency and minimal processor intervention by using a mode of dedicated DMA transfer, and allowed the message transfer to interleave with normal memory access operations. With such techniques, computation could be carried out as usual with communication only adding a minimal overhead. In comparison, most of the modern RISC microprocessors contain an adequate amount of internal cache. Provided the cache is used efficiently, most of the computation tasks will operate from the processor cache, while the message transfers can take place on the memory bus, utilising cycle-stealing DMA transfers.

1.3.1 The New Distributed Processing System

The key aim of this project was to build a low-cost, medium-scale, high performance embedded distributed parallel processing system. The target was not to construct a system that is comparable to those of high-end supercomputers, but to build a powerful platform for embedded applications. Due to the physically distributed architecture of the system, the word 'distributed' was added. In fact, the term 'distributed processing system' will be generally used in the remaining of the thesis.

The new distributed processing system was designed based on three major underlying objectives:

- Applying switching or routing devices as the backbone of the scaleable interconnection network: providing non-blocking point-to-point communications, and providing a scalable interconnection network with individual channel communication bandwidth for processor nodes.

- Utilising standard high performance RISC microprocessors and high-speed memory devices in processor node implementation. By using commodity products, the component cost was reduced while allowing close tracking of current technology.
- Reducing latency and overhead of message passing by designing a single-chip custom network interface controller that tightly couples to the microprocessor memory bus. The highly integrated memory controller and network interface architecture must optimise the communication streaming path.

This proposed system has similarities to the Transputer system, especially in its efficient communication links, but offers a much higher processing power.

At the initial stage, the ICR C416 packet routing switch was chosen as the communication network core of the system. The use of the ICR C416 router in existing systems has shown its reliability and simplicity in connection and flow control. Serial communications devices such as the ICR C416 router have the advantages of reducing the wiring cost and complexity in distributed systems (especially loosely distributed systems), though the performance of a serial communication link is lower than that of a parallel communication link. Research into better performance and fault tolerance for the router is in progress [33]. However, any development of the communication link would retain bi-directional serial connections. Therefore, an upgrade of the ICR C416 routing network would not be a difficulty, as the physical connections remain.

An initial review had identified the StrongARM SA-110 as a suitable processing core for the building block. The StrongARM SA-110 is a low power, high performance RISC processor [34]. With the core operating at up to 233 MHz and the data bus at up to 66 MHz, this general-purpose 32-bit microprocessor is targeted for many embedded applications. The SA-110 has 32 kB internal cache, 128 byte write buffer, and hardware support for fast interrupt handling. The on-chip cache together with the write buffer substantially increased the average execution speed and reduced the average memory bandwidth required, allowing DMA transfers with minimal performance loss.

The combination of the ICR C416 packet routing switches and the StrongARM SA-110 microprocessors coupled with some memory would form the low-cost embedded distributed system, initially known as StrongARM Router Network (SARNet). The main original contribution to this research, however, was the design of the one chip solution, network interface message controller. This device utilises techniques specifically introduced for message passing efficiency and minimising processor overheads, and acts as the interface core to glue the major components together.

1.3.2 The Network Interface Controller Hardware

The underlying hardware of the inter-processor communication is the interconnection network and network interface. Ideally, a network system protocol should be designed considering the router and the interface together. However, this does not take place in many systems, where either the interface or the router is built first, thus one constraining the design of the other. For example, the ICR C416 router was built after the first generation of Transputers, and now the network interface for the StrongARM was built after the ICR C416.

In order to analyse the effect of coupling the network interface to the microprocessor memory bus, a custom network interface controller had to be designed, which closely integrated with the memory interface logic. Designing and implementing such hardware traditionally required a large engineering effort. However, with the availability of current large density, high performance CPLDs, the trend of logic design has changed. Synthesising and fitting a design into the target device can be carried out using re-programmable CPLDs. This enables almost unlimited modifications and re-designs without the excessive fabrication cost. Therefore, use of CPLDs for implementing the interface logic would ease the design changes required for incorporating a new RISC microprocessor. Nevertheless, there was still a challenge to push the performance of current CPLD design to somewhere comparable with the ASIC technology.

The Altera FLEX 10KA CPLD [35] was chosen to develop the novel network interface controller design. This network interface controller implied a message-passing model, which was why the design was also known as the message controller. In addition to providing the function of message passing, the design acted as the interface core that glued the microprocessor and memory module, as well as supporting many other features. The mandatory functions embedded include:

- Servicing the request from the microprocessor.
- Controlling the arbitration between DMA accesses and microprocessor accesses.
- Managing the access to memory module with correct timing.
- Supporting external I/O interfaces for slow devices.
- Providing timer function for software and process scheduling purposes.

The network interface controller design had been realised using Very-high-speed-integrated-circuit Hardware Description Language (VHDL) entry, in a top-down modular flow. A series of comprehensive tests were written to verify the functionality of the model through simulation. The tested model was then synthesised into hardware logic, followed by hardware prototyping, and real-time hardware testing.

Two versions of network interface controller design were implemented. The first version, previously known as OS link Processor Interface (OSPI), was implemented with only the basic set of requirements. It was used to study the feasibility of the design realisation in the Altera FLEX CPLD. The successful implementation of the network interface controller design and some early experiences with the first prototype design had led to the development of the second version design, named as StrongARM Router Network Interface Controller (SARNIC).

The finalised SARNIC hardware design was equipped with many message-processing functions that reduced communication tasks of the microprocessor. It contained a number of original features as listed below:

- Optimised data transfer path between the memory and network interface for continuous message streaming with minimal buffering, whilst keeping low interference to processor-memory accesses.
- An additional serial router communication link to double the effective network interface bandwidth, to provide better fault tolerance, and to offer a wider variety of network connection topologies.
- Multiple hardware message contexts/channels support for all communication links, hence reduced the complexity and inefficiency of implementing the same function in software.
- Ability to monitor router communication links status and the ability to communicate with the router device for fault detection and recovery.
- A booting from communication link option for ease of initialisation and real-time reconfiguration of a processor node.

The successful CPLD implementation of the SARNIC provided an ideal single chip solution for the formation of an embedded distributed processing system building block. By replicating the building block and connecting them together through the ICR C416 packet routing switch, a low-cost and efficient embedded distributed processing system, SARNet, could thus be constructed. This was followed by intensive software development before the system was ready as a total solution.

1.3.3 Loosely Coupled Network Configuration

In distributed processing systems, there could be a need for a loosely coupled network configuration, where many of the processing nodes are physically separated at different locations. If processing is installed at the locations where the computing power is required, then communication costs are reduced [36]. Such a system requires a method to provide greater distances of data transmission.

A side topic of this research was to carry out an investigation into extended distance communication for the new embedded distributed processing system. A solution was required to provide reliable and protected communication, preferable cost-effective, over a large range of distances. This would widen the area of applications for the embedded distributed processing system, for instance in a home-networking environment.

Differential signal transceiver circuitry was analysed in the investigation. Standard high speed transceiver devices and CAT 5 Unshielded Twisted Pair (UTP) cables were used. In addition to using standard differential transceiver circuitry and twisted-pair cables, experiments were carried out to improve the signal quality through the use of some added passive components. This was carried out in the aspect of line balancing.

1.4 Structure of the Thesis

This thesis presents research into distributed parallel processing systems leading to the design and development of a novel integrated network interface message controller device.

The arrangement of the thesis is summarised as the following:

- Chapter 2 gives an overview of inter-processor communications in parallel processing system, providing an insight into its fundamental principles, including latency, bandwidth, processor overhead, reliability, and scalability. The discussion of the architectures of inter-processor communication systems is concentrated on two main areas: the interconnection network and the network-interface. The advantages and disadvantages of common bus interconnection networks and switch interconnection networks are described, followed by technology case studies. The network interface technologies are categorised into three classes based on the level of coupling to the processing elements and memory: from loosely coupled I/O bus based, through memory bus based, to tightly coupled processor integration based. The chapter is concluded with examples of current network interface technologies.

- Chapter 3 highlights a feasibility study of an novel embedded distributed processing system, using commodity microprocessors and RAM chips, with the aid of a proven inter-processor communication network utilising ICR C416 routing devices. A custom design for a network interface hardware controller to link the router network to the microprocessor and memory is proposed. A review of possible choices of RISC microprocessor and memory technology is given, resulting in the choice of the StrongARM microprocessor and SDRAM. A basic network interface and memory controller was implemented to study the potential performance of the system. Finally, an analysis of differential transceiver circuitry was carried out to extend serial communication transmission distances.
- Chapter 4 describes the structure and implementation of the StrongARM-router network interface controller device, the SARNIC. The design is described in six main blocks, namely the Bus Controller, the Communication Controller, the Control Link, the Interrupt Controller, the Timer, and the Universal Asynchronous Receiver Transmitter (UART). Functions of each block, and implementation issues related to the specific CPLD architecture are detailed.
- Chapter 5 describes the functional simulation of the SARNIC device in an emulated distributed system platform. Performance issues relating to the synthesis of the design into CPLDs are explained. System integration, verification of the design, and some basic performance tests are reported. A SARNet system was constructed, comprising of StrongARM processing units linked with an ICR C416 router network using SARNIC network interface devices. Raw performance tests were carried out for various combinations of possible communications.
- Chapter 6 is devoted to the discussion of the work presented in this thesis compared to previous work and other systems in the research

literature. Key issues raised by the design and utilisation of the SARNIC, and the subsequent SARNet system, are given. This is followed by the conclusion of the thesis. Finally, potential areas of further work arising from the network interface controller are highlighted.

2. INTER-PROCESSOR COMMUNICATION REVIEW

This chapter presents the fundamentals associated with inter-processor communications of a parallel processing system, in particular efficiency, reliability, and scalability: efficiency affects the performance of the system; reliability states the stability of the system and fault management; scalability relates to system processing power and the ease of system expansion. A review of different inter-processor communication architectures is then described as two major components: the interconnection network and the network interface, followed by case studies of the current technology. Bus based interconnection networks and switch based interconnection network architectures are described. Issues relating to the incorporation of the network interface are also discussed.

2.1 Concepts and Goals

Communications take place using a predefined algorithm or paradigm. Since the early 1980's, several different inter-processor communication algorithms have been introduced. These algorithms were developed following two major tracks: the shared-memory model, implemented in shared-memory multiprocessor systems and the message-passing model, implemented in distributed-memory multicomputer systems.

Due to the advantage of scalability in distributed-memory architectures, most parallel machines conform to the message-passing computing model [37]. The development of shared-memory architectures has been moving towards using physically distributed memory for scalability while maintaining the globally shared memory address for programmability: these are generally called Distributed Shared Memory (DSM) systems [12]. There are also recent parallel systems that support both message-passing and shared memory communication paradigms [38]. The review given in this chapter is mostly focused on message-passing model as the system described in this thesis is based on this.

A message-passing action involves the transmission, by a sending process, of a set of data values through a specified communication mechanism and the acceptance, by a receiving process, of the set of data [39]. The practical implementation of message passing, between processes located in different processors, requires the use of a communication network for the transmission of data and synchronisation signals. This processor interconnection network technology involves several design considerations. The common considerations will be the network topology, deadlock possibilities, data transmission rate, probabilities of usage for each device, and queuing protocols at the sender and receiver [40]. The essential characteristics of the inter-processor communication fall into three categories: efficiency, reliability, and scalability.

2.1.1 Efficiency of an Inter-Processor Communication System

The aim of efficient inter-processor communications is to overlap computation and communication. As the overlap of the inter-processor communication system reduces, a processor node would need to wait longer for the data to arrive, or more involvement from the processor would be required. In this situation, programmers often create coarser grain computation that requires less frequent communications. Consequently, tasks that might otherwise execute in parallel on several different nodes must now be combined to execute serially on a single node, thus reducing the amount of parallelism within a task.

In general, the efficiency of an inter-processor communication system is represented by three associated parameters: latency, bandwidth, and processor overhead.

2.1.1.1 Message Latency

Latency is defined as the time required transferring an empty message between the relevant processors. It is calculated from the point a message enters the communication network until the point the message reaches the destination. Latency is an important factor for short communications such as control signals, synchronisation signals, and error messages, as this overhead will be huge compared to the data contents.

The contributing factors to the latency of a network mostly depend on the speed of the communication devices and the number of stages/hops that a message has to travel in the network topology. In bus based interconnection networks, the latency is low as there is only one stage that the messages have to traverse. However, the factors of collision and contention have to be taken into consideration. Some early switching based interconnection networks utilising the store-and-forward communication mechanism incurred a large latency. This is because they store a complete packet before forwarding the packet to the next intermediate hop. As the number of hops a message has to travel increases, the latency increases linearly. Luckily the introduction of virtual cut-through [41] and wormhole routing mechanisms [42] have significantly reduced this hop dependent latency. These mechanisms forward the message or packet to the next location once the routing information has been extracted and decoded.

2.1.1.2 Communication Bandwidth

Bandwidth is defined as the speed at which data can be transferred between processors once a transmission has begun. It is primarily determined by the capacity and the capability of the interconnection network. The higher the bandwidth figure, the more information can be transferred within a limited time frame. This factor is important for large message size communications.

With higher communication bandwidth, the data can be transferred through the interconnection network faster. However, the data transfer rate of the network interface to the processor also has to be taken into consideration. Ideally, the interconnection network bandwidth and the network interface bandwidth should be equal, so that a bottleneck does not occur. Moreover, care should be taken that a high data rate does not incur too many overheads on the processor memory bus.

To increase the physical bandwidth of the interconnection network, most closely coupled parallel systems use multiple data paths (parallel connections) instead of a single data path (serial connections). This method is very effective for closely coupled parallel systems, as the interconnection is either on the same board level, or only at very short distances between different boards. However, with loosely coupled parallel systems, multiple data path connections are a disadvantage. The cost of the cable increases and there is a problem with inter-signal skew for large distance communications.

2.1.1.3 Communications Related Processor Overhead

Processor overhead is defined as the slack time incurred during which the processor is busy initiating or receiving a message. It is an overhead that contributes directly to the latency of the current communication request, and indirectly to the latencies of subsequent requests.

Processor overhead is directly relevant to the network interface hardware. A good network interface design with communication function support will reduce the processor overhead significantly. For example, with the help of a communication co-processor, the protocol processing can be off-loaded from the processor. In this way, the processor is freed for independent computation work while the co-processor is being occupied with communication tasks. Another example is the occupancy of the memory bus during the transfer of a message through the network. A fair division of memory bus bandwidth between the network interface and the processor will reduce the bus access competition correspondingly.

Another approach to reduce processor overheads through using software methods. Traditional communication models usually provide message send and receive functions as services of the operating system. This is time-consuming because the service requires several crossings between the user level and the kernel level for each message in order to provide protection, buffer management and message-passing protocols. Recently, methods have been developed to reduce the software overhead by supporting the communication directly at the user level [22, 43].

2.1.2 Reliability of a Parallel Processing System

Reliability is defined as the probability that a system will operate for a long enough time to be useful [44]. The interconnection network is one of the key factors of a reliable parallel processing system. Another factor related to reliability is the fault tolerance of the system. Depending on the requirements and applications of the system, different levels of reliability and fault tolerance can be implemented accordingly.

An interconnection failure will manifest itself in one of two ways. Messages either will be transmitted with errors or will not be transmitted at all. In the former case, error detection can be implemented to catch the fault, for example using parity or checksum algorithms. In the latter case, where faults can be caused by an open link or a failed processor, time-out or message acknowledgements are methods to check the reception of messages at the destination. As these fault detection and possible fault correction methods are added as an overhead to the communication bandwidth, the balance between performance and reliability will depend on the system requirements.

Failures can be classified as hard (permanent) or soft (transient) [44]. Most failures, when originally detected, will be initially dealt with as if they are transient. Recovery will be attempted and will consist of a retry of the failed process. If the retry is successful, the system can be said to be recovered. If the retry is not successful, the failure will no longer be regarded as transient, and will be dealt with as a hard failure. In the case of hard failure, the isolation of the fault is required, followed by the replacement of the failed hardware and a system restart. In some critical systems, operation in a fallback mode that provides reduced capability for the system will be a better solution. A straightforward mechanism to provide some levels of fault tolerance against failure is by replicating the critical component.

2.1.3 Scalability of a Parallel Processing System

Scalability refers to the ability to retain efficiency as the number of processing elements increase [45]. It applies to the expansion flexibility for both the number and the location of processor nodes in the system, as well as the application program involved.

Parallel processing systems should be able to operate effectively and efficiently at many different scales. A system may start off with only two processors, but later be expanded to hundreds. An architecture is scalable if it continues to yield performance increases proportional to the number of processors used for a given application. When a particular installation expands, the expansion should occur without disrupting the system. In this sense, the scalability should reflect the ease of translation for the system software and application software. Ideally, no changes to the software are required, but this is rarely ever the case.

In the aspect of interconnection network scalability, a system expansion preferably should not affect the performance of the network. As the number of processor scale up, demand for a resource (e.g. a communication channel) may grow. Depending on the application and the topology, a performance bottleneck can be formed, since high amounts of accesses are requested to that particular resource. Thus, the demand for scalability in parallel processing systems has led to a design philosophy in which no single resource is assumed to be in restricted supply [39]. Rather, the system should allow possible extension to replicate this resource to avoid both performance and system bandwidth degradation. For example, due to the sequential access in a shared memory bus system, the shared medium becomes a bottleneck as more processors are connected. A switch network, on the other hand, can provide arbitrary scaling with dedicated bandwidth for each connected processor.

2.2 The Processor Interconnection Network

The interconnection network plays a central role in determining the overall performance of a parallel system. If the network cannot provide adequate performance, processor nodes will frequently be forced to wait for the arrival of data, causing performance drop in parallel systems.

The network technology always needs to be improved in order not to become a serious bottleneck for a parallel system. Unfortunately, microprocessors and their associated memory devices are currently, rapidly improving in performance. Throughout the evolution of parallel systems, many structures and topologies have been introduced to allow the interconnection of multiple processors. Generally, the interconnection technology falls into two major classifications: common busses and switches. Most of the recent parallel systems utilise switches as their interconnection network, as these switches provide point-to-point communication channels and ease of system scaling. However, there are still systems that use shared medium common busses because of the cost effectiveness.

The following sections describe the two major classifications of interconnection network: common bus systems and switched systems. The investigation focuses on distributed interconnection network architectures, and is biased towards switched systems.

2.2.1 Common Bus Systems

A bus system is simply a collection of wires for data transactions among connected processing nodes. Since it is a shared medium, only one transaction can be done at a time between a source and a destination. In order to solve the contention when multiple requests are occurring, bus arbitration logic is used to service the requests one by one.

Usually all the processors in a bus structured system are homogeneous. This interconnection style is very common in closely coupled multiprocessor systems. A moderate number of processor nodes connected on the bus can communicate efficiently with high throughput. However, when the number of processor nodes in the system increases, the inherent sequentiality of these communication methods produces bottlenecks and bandwidth degradation. Timing constraints are a difficulty too. In order to guarantee performance, a common bus system is typically built with a fixed number of slots on a backplane, and thus only allows a limited number of connected processors.

Although closely coupled bus system usually use multiple data connections (parallel), serial connection is preferred in loosely coupled bus systems. Obviously, a major factor is the amount of cable used, which is proportional to the distance of connections. However, the data transfer rate of a serially connected bus system is significantly less than a parallel connected bus system due to the reduced number of connections. Therefore, research in serial bus systems concentrates on pushing the speed of the data transactions across the network. The Ethernet system is an example of such a serial bus system.

2.2.1.1 Ethernet

Ethernet is a very common network connection technology used in modern Local Area Network (LAN) systems [46]. This technology is cheap and widely available due to its dominance of current PC networking. Traditionally, Ethernet was a shared-medium network, and therefore all stations attached to the network are competing for the use of a wire. Therefore one would not expect that Ethernet is suitable for high performance distributed parallel systems which need predicted low latency in message transfers. However, recent research has demonstrated the use of this commodity network as the interconnection of a parallel system [43].

As a distributed network topology, Ethernet systems utilise serial connections. Data packets are transmitted serially over the shared-medium to every attached station. Figure 2 shows the data packet format for the standard Ethernet systems, as specified in IEEE 802.3. The first 8 preamble bytes and last 4 frame check sequence bytes are normally generated by the Ethernet chipset, while the rest are the responsibility of the software. The preamble consists of 62 bits of alternating 1's and 0's, used by the receiver to acquire bit synchronisation, and 2 consecutive 1's to acquire byte synchronisation. The destination address contains the address of the intended receiver (broadcast address is all 1's). The source address contains the unique Ethernet address of the sending station. The length of data field contains the number of bytes in the data field. The data field can accommodate 46 to 1500 bytes of data, where shorter packets must be padded to 46 bytes. The frame check sequence consists of 32 bits of Cyclic Redundancy Check (CRC).

Bytes	8	6	6	2	46-1500	4
	Preamble	Destination address	Source address	Length of data field	Protocol header, data and pad	Frame check sequence

Figure 2: IEEE 802.3 Ethernet packet format

In order to solve the contention of stations competing for the network access and ensure access to the network channel is fair, Ethernet conforms to a medium access control mechanism called Carrier Sense Multiple Access with Collision Detection (CSMA/CD):

Carrier Sense - When an interface is transmitting, there will be a signal on the channel, which is called carrier. Before each interface begins transmitting, it will sense if any carrier exists on the channel, and must wait until the carrier ceases if there is one.

Multiple Access - All the interfaces in an Ethernet system are equal priority in their ability to send frames onto the network.

Collision Detection - Since signals take a finite time to travel from one end of an Ethernet system to the other, the first bits of a transmitted frame do not reach all parts of the network simultaneously. Therefore, it is possible for two interfaces to sense that the network is idle and to start transmitting their frames simultaneously, and this is referred to as a 'collision'. The interface has a way to detect the 'collision' condition, stop the transmission, and re-send the frames.

Most designs ensure that the majority of the collisions on an Ethernet will be resolved in microseconds. In the event of a collision the Ethernet interface waits for a few microseconds, then retransmits the data packet. However, as more computers are added to a given Ethernet, and as the traffic level increases, more collisions will occur.

It may also happen that there are multiple collisions for a given frame transmission attempt. After 16 consecutive collisions for a given transmission attempt, the interface will discard the Ethernet packet. This can happen only if the communication channel is overloaded for a fairly long period of time, or is broken in some way.

The original Ethernet system operated at 10 Mbps. There are four different media segments defined in the standard: 10 BASE 5 which uses thick coaxial cable with a maximum length of 500 m; 10 BASE 2 which uses thin coaxial cable with a maximum length of 185 m; 10 BASE T which uses twisted-pair cable; and 10 BASE F which uses fibre optic cable. 10 BASE T was the most widely used for PC network connections.

Recent Ethernet systems are operating at 100 Mbps. There are two LAN standards developed. Only the upgrade version of the original Ethernet system will be given here. This approach is called 100 BASE T Fast Ethernet. It produces a tenfold increase in the transmission speed, but keeping the existing frame format and media access control mechanism. Like the 10 Mbps Ethernet system, there are three different media segments defined: 100 BASE T4 which uses 4 pairs of telephone-grade twisted-pair wire; 100 BASE TX which uses 2 pairs of data-grade twisted-pair wire; and 100 BASE FX which uses fibre optic cable.

In the forth-coming Gigabit Ethernet, enhancements have been made to the speed of transmission, and in providing additional support for new applications and data types [47]. This includes support for video and multimedia traffic with reserved bandwidth and virtual LAN protocols. At gigabit speeds, fibre optic cable will be a more suitable communication medium. However, research in using copper cable still continues [48]. 1000 BASE CX uses Shielded Twisted Pair (STP) at up to 25 m, while 1000 BASE T uses four pairs of CAT 5 UTP (unshielded) cable at up to 100 m with special coding and technology. These systems are still in the stage of testing and development.

It is difficult to guarantee predicted message delivery in Ethernet based parallel processing environments due to the collision detection and resolution mechanisms. Ethernet systems also suffer from the inherent disadvantages of shared-medium networks like bandwidth sharing degradation and the lack of scalability. Nevertheless, the introduction of star-wired hubs and switches has continued to improve the situation. These hubs and switches offer each node a 'private' link to the network that can provide a full-duplex link. New protocols are introduced in Gigabit Ethernet to provide reserved channel bandwidth.

2.2.2 Switched Systems

Switching or routing devices can provide simultaneous message transfers across the network, providing the messages are not going to the same destination. These switches allow scalability of the system without bandwidth degradation. They also raise the possibility of connecting heterogeneous processors for a distributed system, through compatible data representations and message-passing protocols. However, there is a general problem with message delay, because of the routing function and the fact that a message might traverse multiple links before it reaches the destination.

Initial switching communication technology used a store-and-forward mechanism for its message passing. Each message or packet has to be stored in an intermediate node before it is retransmitted to the destination node or the next intermediate node. A disadvantage is that even if there is no blockage on the communication path, the process of store and forward is still repeated at each intermediate node, until the message reaches the destination. The virtual cut through mechanism was introduced later, where the message or packet is forwarded through the intermediate nodes as long as there is no blockage ahead [41]: when a blockage occurs, the message or packet will be stored at that location and waits until the blockage disappears.

Wormhole routing is another mechanism that can forward the message or packet through the communication path until a blockage is found. The message or packet is injected into the network as a train of smaller data groups, called flits [42]. In this way, only the small size flits need to be buffered, and thus only a small amount of buffers needed to be implemented in the switching device.

The following sections will focus on three common packet routing switches: the Nottingham Trent University ICR C416, the SGS-Thompson ST C104, and the

Myricom Myrinet. These designs embrace most of the features of current switching technology.

2.2.2.1 ICR C416 Based Systems

The Nottingham Trent University ICR C416 [18] is a dynamic hardware routing switch designed initially for the first generation of Transputers [21], utilising the protocol called Over Sampling (OS) Links. It uses a wormhole routing mechanism for its message passing, in which the routing decision is taken as soon as the routing information has been received. There are a total of 16 ports available on the device. Messages to an input port of the device can be routed to any output port. All messages can be routed independently of each other provided there is no contention for an output link.

Each link of the ICR C416 consists of a pair of full duplex wires. Serial data is sent down the link, starting with a start bit, followed by an identification bit of '1', data bits, and finally a stop bit. The speed of the transmission can be set as either 10 Mbps or 20 Mbps. As three extra bits are sent per character, the maximum data throughput can only be 14.55 Mbps, or 1.82 MB/s. No clock is sent over the channel, and an over sampling technique is used to synchronise the serial data at the receiving end.

The OS Link protocol implements a stop and wait mechanism. Each data byte sent out must be acknowledged before the following bytes are sent out. An acknowledge token consists of a start bit immediately followed by a stop bit. To allow data to be sent back-to-back (thus achieving maximum throughput), an acknowledge token is sent back once a data token is detected, provided there is room in the buffer for the current data byte. Thus, at least one byte of buffer is needed at the receiving end to ensure continuous flow of data. This acknowledge token, however, reduces the bandwidth on bi-directional transfers since data and acknowledge have to be multiplexed on the channel. As a result, a single byte will need 13 bits of time with data and acknowledge back-to-back. This gives a maximum bi-directional throughput of 3.08 MB/s.

The ICR C416 packet is a sequence of header bytes, a length byte, and a payload of specified length. Figure 3 shows the format of the packet. The header byte is the destination byte of the routing device output port. It will be stripped as the message is passed through the routing device. For a message that is going through more than one router, variable lengths of header bytes can be used: the Most Significant Bit (MSB) of the header byte (known as the cascade bit) is set to '1', except for the last header byte. In this way, headers are stripped at each passage of a router. The byte following the last header will be the length byte. This length value is used to count the number of subsequent bytes and disconnect the link at the end. With the 8-bit length byte, a variable length of payload can be sent, from 1 to 256 bytes of data[1].

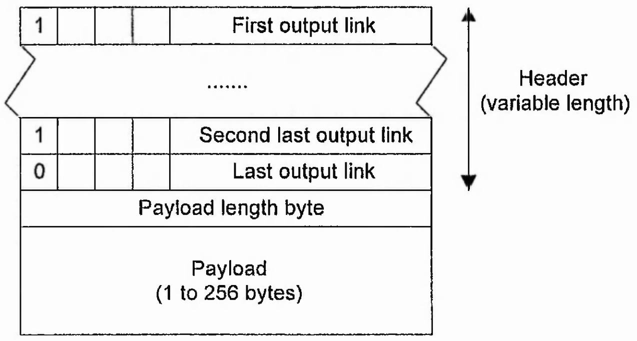


Figure 3: ICR C416 packet format.

The ICR C416 offers a simple and cheap solution to inter-processor communications. Though the link speed is not as fast as other networks, it does cover many applications without performance losses. In addition, the simplicity of the connection and wiring make it a favourable routing device in embedded or small-scale networks. One of the limitations of the ICR C416 is the lack of fault tolerance in the message passing. However, some forms of control can be achieved by monitoring the traffic via the software control port of the device, and implementing a fault detection function in the host. The operation and the special features of the device will be covered in more detail in Chapter 3.

[1] The valid packet length byte of 0 (equivalent to packet size of 256) is not documented in the data sheet.

2.2.2.2 ST C104 Based Systems

The SGS-Thompson ST C104 [49] is a routing switch built specially for the second generation of Transputers - the T9000 [50, 51]. Like the ICR C416, the router uses a form of wormhole routing. The device has 32 ports, providing high bandwidth, serial communication links to each other via a 32 by 32 way, non-blocking, crossbar switch.

The link channel of the ST C104 uses two pairs of full-duplex wires, one pair in each direction, which are referred to as Data Strobe (DS) Links. One wire is for data and the other carries a strobe signal. Each DS pair carries tokens and an encoded clock, and operates at data rates of up to 100 Mbps. Two types of tokens are available: data and control tokens. Data tokens are 10 bits long and consist of a parity bit, an identification bit of ‘0’, and 8 bits of data. Control tokens are 4 bits long and consist of a parity bit, an identification bit of ‘1’, and 2 bits for types of control. Thus, the maximum unidirectional throughput will be 80 Mbps, or 10 MB/s.

Token-level flow control of the DS Links employs a stop and wait mechanism to prevent a sender from overrunning the input buffer of a receiving link. A sender will transmit eight tokens and wait for a Flow Control Token (FCT) before transmitting any further tokens. Thus, each receiving link input must contain a buffer with at least eight tokens of space. Whenever the receiving end has space for more than eight tokens, the ‘FCT’ token is sent to the transmitting end to allow transmission of the next eight tokens. The provision of eight extra token buffer on each input link ensures that the ‘FCT’ token can be sent before the current eight tokens of data is fully transmitted. Hence, a total of sixteen token buffer spaces will allow continuous flow of data without restricting the maximum bandwidth of the link. However, this ‘FCT’ token does slightly reduce the data bandwidth on bi-directional transmission. For full utilisation of the link bandwidth, eight tokens of data will be followed by a ‘FCT’ token. This gives a bi-directional throughput of 19.05 MB/s.

Communication through ST C104 ports takes place in virtual channels, which always occur in pairs between the nodes in a network. The packet contains a header that identifies the virtual input channel of the node, and is also used to route the packet through the network. This is followed by the data section of the packet until a packet termination token is received. A packet termination token is either an End Of Packet (EOP) or an End Of Message (EOM). To ensure that no packet is lost, each packet of data sent along a virtual channel must be acknowledged before the next is sent. This is achieved by sending acknowledge packets at the receiving end, which consists of only a header and 'EOP' token. Figure 4 illustrates the structure of these packets.

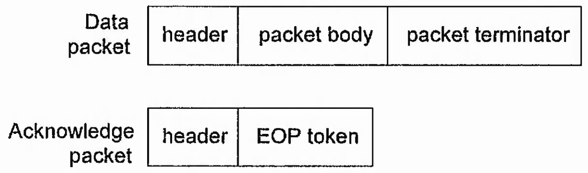


Figure 4: ST C104 structure of data and acknowledge packets.

Two types of error detection are implemented in the ST C104: parity and disconnection errors. Single bit odd parity error is checked at every token to ensure the link layer is reliable. A disconnection error is implemented by doing a time-out check for non-activity on the link. Null (NUL) tokens are sent whenever the link is idle, to ensure no disconnection error occurs.

The ST C104 is one of the more sophisticated routing devices available, which offers up to 32 ports of connection, and features support for large network connection. It has different levels of reset and error recovery modes, providing good fault tolerance.

2.2.2.3 Myrinet Based Systems

The Myricom Myrinet [17] is a new type of message passing network used for packet communication and switching within Massively-Parallel Processors (MPPs). Like the ST C104 and the ICR C416, the switching circuit employs a wormhole routing mechanism. The core of the switch is a pipelined crossbar, which introduces no internal conflicts between packet flows. Currently there are 4-port, 8-port, and 16-port switches available.

A Myrinet link channel is composed of nine full-duplex pairs of wires, making a total of eighteen. The message consists of a series of 9-bit parallel characters. The character can be either an 8-bit data byte or one of the five control bytes. Character transmission is at 80 MHz, resulting in a bandwidth of 720 Mbps per channel. However, the maximum unidirectional data throughput can only be 640 Mbps per channel as only 8 bits are data. Similar to the ICR C416, a sampling technique is used to synchronise the character at the receiving end.

The flow control in Myrinet is accomplished by injecting 'stop' and 'go' characters on the channel. When the receiving buffer reaches the stop limit, the receiving end sends a 'stop' character to the transmitting end so that the flow will stop before the buffer overflows. The receiving buffer is then read until the 'go' limit is reached, where a 'go' character is generated to resume the data flow before the buffer is totally empty. The 'stop' and 'go' limits are determined by measuring the number of characters which can transit on the round trip of Myrinet cable, and the time for 'stop' and 'go' characters' generation and receipt. Buffer size has to be carefully selected to ensure these control characters will not consume excessive bandwidth on the channel. Thus, there is a possibility of not sending the flow control characters at all if the buffer size is big enough, giving a full 160 MB/s bi-directional throughput (twice of the unidirectional throughput). This is certainly an advantage compared to the acknowledge flow control mechanisms of the ST C104 and the ICR C416.

A Myrinet packet consists of a sequence of bytes starting with a routing header, followed by an arbitrary-length payload, and terminated by a trailer that may include a checksum. The format of the packet is illustrated in Figure 5. The number of headers used is variable, depending on the number of switches used. The MSB of each header byte will be set to '1' to indicate the destination router outgoing port, except for the last header byte (dedicated for the host to identify packet type) which is set to '0'. A routing header byte will be stripped off when it enters the switch. Following the headers will be the arbitrary length of payload. At the end of payload, an 8-bit CRC character is computed on the entire packet, including the header. The marker of end of packet is the 'gap' character.

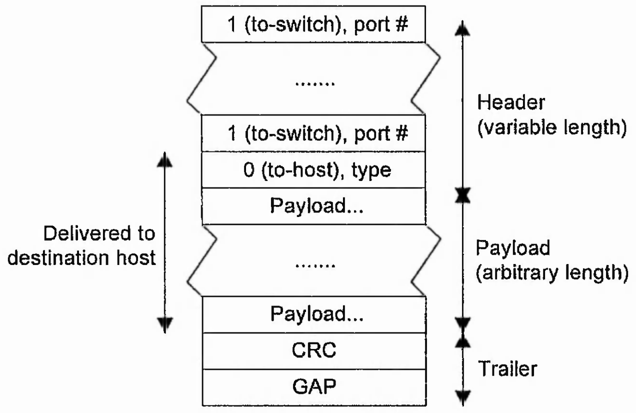


Figure 5: Format of a Myrinet packet.

To increase reliability, Myrinet uses a long period time-out mechanism to detect packets blocked for more than 50 ms, either caused by a software error or by a bit error in a header causing deadlock. Thus, the transmitter is required to send a non-idle character periodically to avoid time-out error. When a long-period time-out occurs, the transmitter will drop the blocked packet, and send a Forward Reset (FRES) character to the receiver to reset the blocked channel.

The Myrinet network flow control shows a good example of increasing the bandwidth on bi-directional communications. It also offers high bandwidth with the expense of multiple wire connections. Recent upgrades of the Myrinet switch has increased its bandwidth, achieving 1.2 Gbps per channel per direction.

2.3 The Processor Interconnection Network Interface

Although interconnection networks such as router switches are the key components in distributed multiprocessor systems, the interface between the router and the processors is of equal importance. A processor-network interface supports the communication protocols of a network, and data transfer to and from a processor node's local memory. These make the interface very dependent on the interconnection architecture, and the processor chosen. One processor-network interface can be optimised to support a specific communication paradigm to achieve high performance.

The following sections describe the architecture and implementation of the network interface technology. A number of architectures of integrating a network interface to the processor and the memory subsystem are described. Examples of systems are given in each architecture class. Finally, the last section discusses the amount of communication support that can be offered in the network interface, and types of solution in implementing the support.

2.3.1 Tightly-coupled Versus Loosely-coupled Interfaces

A communication network interface controller can be located at a different level from the microprocessor. The closer the network interface is to the microprocessor, the greater the performance, but less general the communication parts. Typically, common network hardware has a processor interface implemented attached to the I/O bus of the host system. One of the common examples is using Peripheral Component Interconnect (PCI) bus interfaces [52]. However, to allow very low network access latencies and simple protection schemes, more customised architectures have been proposed, that integrate the processor-network interface more tightly with the host system. For example, by placing the interface on the memory bus or integrating it with the cache controller. There are some even more tightly-coupled processor-network interfaces where the interface design is integrated onto the same chip as the processor, thus offering specific instructions for sending and receiving messages.

2.3.2 I/O Bus Based Interface

The I/O bus based network interface is connected to the microprocessor and the memory through the system I/O bus. It is an economical strategy where standard workstations can be applied as the processing elements, and the communication network is provided through the network interface card slotted in the workstation's

I/O bus. The other benefit is that a network interface (and thus the interconnection network) can be easily adapted across different families of processors or workstations. In this way, upgrades of processing elements would not a relatively easy task, allowing close tracking of the rapidly advancing PC/workstation technology.

The standard I/O bus architecture normally offers a few expansion slots for the attachment of different I/O cards. Therefore the I/O bus based network interface has to compete with all the other I/O devices attached to the bus for bandwidth. Depending on the I/O bus arbiter, the network interface might not be able to get a fixed interval bandwidth. This is not a desirable situation, as the data flow to and from the network will be affected without the help of a large buffer. Another problem is with the I/O bridge which handles data transfers between the I/O bus and the microprocessor memory bus: the overhead and bandwidth of the I/O bridge, which determine the message latency, are variable. For instance, the performance of PCI bus systems is very dependent on the PC chipset used. A variety of performance figures of PCI bus transfer rates for a series of different computer systems have been reported on the Myricom (manufacturer of Myrinet) website.

Although the latency measurement is difficult to define, most recent parallel systems use this I/O bus based method for implementing the network interface [22, 23, 53, 54]. The main supporting reason is the flexibility of using current low-cost PC/workstations as processing elements. Two examples of I/O bus based network interfaces are discussed in the following sections.

2.3.2.1 Myrinet/PCI Host Interface

The Myricom Myrinet/PCI Host Interface card is the network interface card designed for connecting standard PCs or workstations to the high performance Myrinet switch discussed in section 2.2.2.3 [22]. The adapters are programmable, providing a flexible interface to the host through the system PCI I/O bus. Each network interface consists of a processor and some memory, which is used to store the firmware program and data for the message handling.

Figure 6 illustrates the architecture of the network interface and the host system. There are three DMA engines located on the board: one for the transfer between the host memory and the network interface memory, one for sending messages from the network interface memory to the network, and one for receiving messages from the network to the network interface memory. The network interface memory is constructed from fast but relatively expensive Static Random Access Memory (SRAM), and therefore the size is limited [22]. All inbound and outbound network transfers are staged through this local memory. Although there are two DMA engines for inbound and outbound transfers between network interface and the network, there is only one DMA engine for the transfers between the network interface memory and the host memory. Thus, this might limit the maximum performance attainable where bi-directional message operations have to share the same DMA resource.

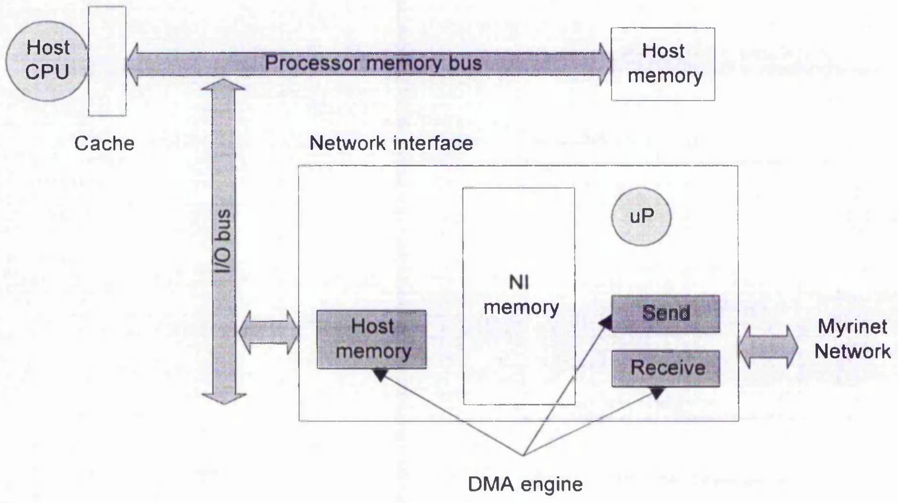


Figure 6: Myrinet host and network interface architecture.

An important feature of the Myrinet interface card is the availability of a programmable processor [55, 56]. The behaviour of the network adapter is determined by a firmware program, called Myrinet Control Program (MCP) that is written into the local SRAM from the host at start-up. This feature offers flexibility in designing the network protocol. However, the network interface processor is much slower than the host CPU, therefore the program must be carefully written to achieve efficient message handling in the network interface. In addition, due to the centralised control in the processor, the processor might become the bottleneck of the communication handling.

As a whole, the Myrinet/PCI host interface delivers high performance at reasonable cost, with latencies determined primarily by the messaging software and the time to transfer data on the host I/O bus. Many recent research parallel machines are focusing on using the Myrinet/PCI host interface and network because of its flexibility in protocol implementation and high network throughput [22, 57, 58].

2.3.2.2 SHRIMP Interface

The Scalable, High-performance, Really Inexpensive Multi-Processor (SHRIMP) system consists of commodity PC nodes connected by a high-speed interconnection network. The initial prototype was using Intel Pentium Xpress PC system as the processor node, and the Intel Paragon routing backplane as the interconnect [59]. The key components in the system are the custom SHRIMP network interfaces, which provide the connection from the PC node to the network [23, 60, 61]. Figure 7 illustrates the first version of SHRIMP network interface.

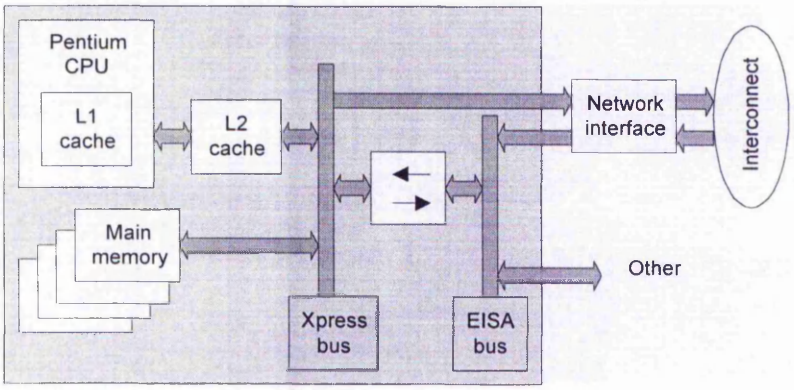


Figure 7: A SHRIMP node with network interface.

The first generation of SHRIMP network interface is designed to connect to both the Xpress memory extension slot and an Enhanced Industry Standard

Architecture (EISA) expansion slot. For outbound transfers, data is 'snooped' directly off the Xpress memory bus. However for inbound transfers, data has to be transferred to the main memory through the EISA bus, as the Xpress bus does not offer the capability of bus mastering. Both incoming and outgoing data transfers are assisted by two individual DMA engines, releasing the involvement of the CPU. One benefit of the Xpress bus is that the snooping cache architecture will insure cache consistency during message transfers. Therefore the SHRIMP system can use the cacheable memory region as the send and receive buffers for message passing without the help of extra hardware.

Other than supporting message-passing mechanism, the SHRIMP network interface also supports a Virtual Memory-Mapped Communication (VMMC) model. The virtual memory mapping communication uses a physical memory mapping mechanism. Each page of a local physical memory can be mapped to a physical page of some other node in the system. This will require a map system call to set up the appropriate physical mapping information in the network interface, and to perform protection checking. Once a map is established, either the network interface snoops all writes to the mapped memory and forward to the mapped received node (automatic-update), or a send primitive (user-level) is used to transfer data with minimal overhead (deliberate-update). Since the destination node already knows the mapping information, the network interface uses the destination address to transfer the data directly to the mapped-in physical memory without CPU assistance. This removes the need of a receiver interrupt, and thus reduces the message latency significantly.

Recently, the SHRIMP VMMC model has been ported on a Myrinet network of PCI-based PCs [58]. This was to analyse the feasibility of implementing the VMMC model on a commercially available hardware platform. The results have shown that the Myrinet implementation incurs a relatively higher overhead because of the demands of the network interface resources. These resources include the communication processor (LANai processor) and the network buffer (on board SRAM). However, the Myrinet based implementation requires less operating system support because of the help of the communication processor.

2.3.3 Integrated Processor Based Interfaces

An integrated processor based interface, commonly known as a tightly coupled processor-network interface, is integrated onto the same silicon chip as the processor. This type of processor architecture normally has its own instruction set that includes specific instructions for sending and receiving messages. Efficiency is the key to the success of processor integrated network interfaces. Due to its highly integrated architecture, the message latency is much lower than the I/O bus based network interface. Other than having the network interface integrated on the same chip, these processor designs are also normally embedded with some additional functions, such as memory control logic. This 'all in one' feature simplifies the implementation of processor nodes and the construction of a parallel processing system.

The problem with the processor integrated interface architecture is a lack of flexibility: the selection of microprocessor and the network interface architecture is fixed at the design stage. Also due to its highly integrated architecture, a large amount of engineering effort is needed to design and integrate the network interface.

There are quite a number of processor integrated interface designs available. Examples are the iWarp [62], MDP [20], and Transputer systems.

2.3.3.1 Transputer T800 Interface

The Transputer architecture is constructed using the OCCAM parallel processing language concepts of process concurrency and communication [37]. However, the hardware can also support traditional programming languages. OCCAM is based on Hoare's Communicating Sequential Processes (CSP) [63] and is designed to support explicit hardware concurrency. Every OCCAM program that is executed in Transputers is structured in a set of concurrent sequential processes that communicate through Input/Output instructions on point-to-point channels. This process is synchronous and unbuffered, where one process executes an I/O instruction on a given channel, it then stops until the opposite process begins to execute the complement I/O instruction.

Since the Transputer was designed with OCCAM in mind, message passing is supported in hardware, yielding efficient implementation. Through a careful choice of processor characteristics, each Transputer provides on chip: a few external links for inter-Transputer communications; internal timers; a large area of RAM; and bus logic for an external memory system. With all the necessity functions embedded into a single chip processor, the building block for a parallel system is simplified, minimising the components used.

The Transputer chip architecture is illustrated in Figure 8. Communication on channels between processes is achieved by using the input message (in) and output message (out) instructions. A channel can be either an internal memory location or one of the external links, depending on whether the message passing takes place between processes in the same processor or on a different processor. Because every Transputer link is memory-mapped, both internal and external communication can be implemented by means of the same instructions. Three operands are needed for the message passing instructions: message length, message address, and message identifier. Based on these operands, the external communication system will be able to access the Transputer memory through DMA without processor intervention.

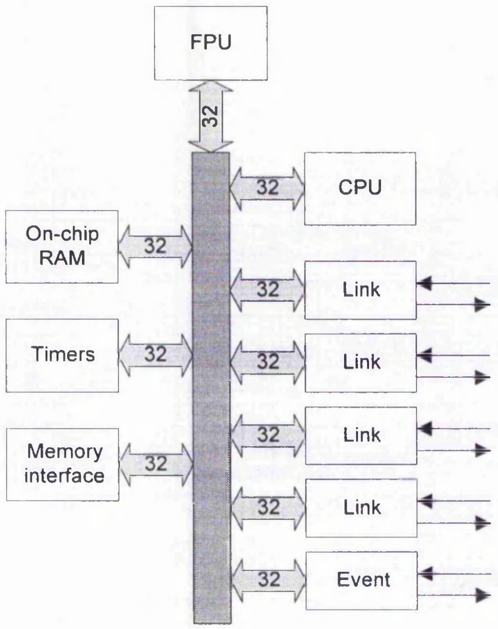


Figure 8: The Transputer chip architecture.

In addition to its internal RAM, the Transputer can also access a wide external memory space, directly mapped up to 4 Gbyte. The external memory interface module supports Dynamic Random Access Memory (DRAM), SRAM, Read Only Memory (ROM) and EPROM, and its timing can be software-configured to cater for the correct memory types and speed. Transputers have two timers based on free-running clocks, each for a different process priority level. The higher priority clock is running at a 1 μs clock tick, while the lower priority one running at a 64 μs clock tick.

The Transputer was the leading microprocessor for embedded parallel system in the late 1980's and the early 1990's. However, the integrated network interface architecture has made the Transputer upgrade complicated. In contrast, stand-alone microprocessors have advanced in a much faster curve throughout the years, both in terms of architecture and processing power.

2.3.4 Memory Bus Based Interfaces

Memory bus based network interfaces have an architecture that lies between I/O bus based network interfaces and processor integrated network interfaces. The performance is improved compared to I/O bus based network interfaces, due to several factors. The first factor is that message transfers do not need to cross over from the I/O bus to the memory bus, thus reducing the overhead. Secondly, the memory bus is conventionally faster than the I/O bus, thus elevating the performance and network interface bandwidth available. Thirdly, the number of devices connected on the memory bus are normally less than the number of devices attached on the I/O bus, thus bus competition is reduced. In contrast to processor integrated network interfaces, the memory bus based network interface offers better flexibility in selecting the appropriate processor and in upgrading the processor.

Integrating processor-network interfaces in memory control logic seems to be an effective and convenient solution. But care must be taken to make sure the network message transfer does not access the memory bus too often. A DMA method is normally used for message transfers, though it suffers from a slight start-up overhead. With cycle-stealing DMA, message accesses utilise unused processor cycles, thus minimising the bus congestion. More advantages will be seen if the memory used supports burst mode operation, so that each individual DMA access can be optimised to the burst size. An important note when using DMA for message transfer is that without the help of cache-coherent hardware, the message area must never be swapped to disk by the operating system (to avoid corruption of messages), nor made cacheable (to ensure data consistency).

2.3.4.1 MAGIC Interface Controller

The Stanford FLexible Architecture for SHared memory (FLASH) multiprocessor systems are aimed to provide efficient integration of both cache-coherent shared memory and high performance message passing mechanisms [64]. To accomplish these goals, a custom node controller, Memory And General Interconnect Controller (MAGIC) was designed. This controller is a highly integrated chip that handles all communications both within the node and between nodes. Figure 9 illustrates the FLASH node organisation.

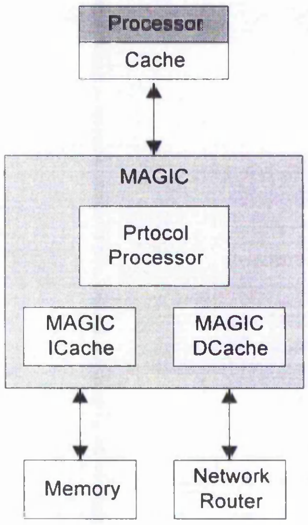


Figure 9: FLASH node organisation.

Each node in FLASH contains: an off-the-shelf microprocessor with caches, a portion of the machine's global memory made of DRAM, a connection port to the communication network, an I/O interface, and the MAGIC chip. The MAGIC chip forms the heart of the node, integrating the memory controller, I/O controller, network interface, and a programmable protocol processor. In this way, low hardware overheads are achieved.

The MAGIC chip supports both message-passing and shared-memory communication models [38]. In order to achieve efficient protocol handling, the MAGIC architecture splits the task into separate control and data processing paths. Message headers flow through the control macro-pipeline while message data flow through the data transfer logic. The control macro-pipeline consists of an embedded programmable processor that provides flexibility in supporting a variety of DSM and message-passing protocols, and additional hardware support for ensuring efficient protocol processing operations. For efficiency in data movement, the data transfer logic is hardwired. Data buffers are provided in the data transfer logic to stage data, thus achieving low latency and high bandwidth through data pipelining and elimination of multiple data copies.

The MAGIC design combines the benefit of integrating the network controller closely to the memory controller and flexible support for both memory-passing and the shared-memory model. This flexible support only slows the ideal performance by 2% to 12% [65].

2.3.5 Co-processor Versus Conventional Logic Interface Implementation

One of the main objectives to be achieved in implementing the network interface controller is to overlap computation over communication. Once the communication task has been assigned to the network interface controller, the microprocessor can continue with computational tasks. In this way, the amount of parallel computation performed is significantly increased. However, in order to achieve this, the network interface controller hardware has to be equipped with the intelligence to handle the communication tasks without much intervention from the microprocessor.

A network interface mechanism moves data from memory to the communication medium, and vice versa [13]. The physical implementation must be supported by logical mechanisms for message assembly, formatting, routing, and optional error checking. These mechanisms can be distributed between the processor and the network interface. The more functions supported in the interface, the fewer loads on the processor, however, the interface design will be more expensive and complicated. At a minimum, the interface will provide the translation between the information format of the communication medium and the data format of the processor. The network interface effects byte by byte transfer under the control of the processor, while the processor performs all other necessary functions to formulate the message and to transmit it to the destination process. At a more sophisticated level, the network interface will assemble complete message packets to pre-assigned buffer locations, and exert flow control on the processor when the buffer is full.

The implementation of the communication network interface controller can be varied, depending on how specialised the controller is in the tasks it performs. It can be a hardware state machine built from conventional logic, a customised special co-processor that runs protocol micro-code, or an inexpensive off-the-shelf general-purpose microprocessor. Conventional logic implementation normally requires an excessive amount of logic design and a long development cycle. It does not offer much flexibility to the communication models that can be applied. However, the operation speed of a hardware state machine is high, as the hardware has been designed for a specific configuration. In contrast, a co-processor solution, whether it is custom built or off-the-self general-purpose microprocessors, offers flexibility in supporting various communication models. The compromise is a reduction in speed and an increase in cost. For instance, the first SHRIMP design that was built using specialised hardware resulted in a relatively higher performance compared to the later SHRIMP design implemented on the Myrinet/PCI host that uses a communication co-processor [58].

3. SYSTEM DESIGN STUDY

This chapter discusses the support and requirements to implement a network interface hardware device to be used for the construction of a new distributed parallel processing system. The system is to be based on an ICR C416 router network. An investigation of microprocessors and memory devices suitable for constructing the processor node technology is carried out, resulting in the selection of the StrongARM SA-110 microprocessor and SDRAM. This is followed by the specification of a network interface controller to link the router network with the processing node. An initial feasibility study for the network interface controller is carried out along with some experiments on long distance communications.

3.1 System Overview

The aim is to build a low-cost embedded distributed parallel processing system which incorporates high performance RISC microprocessors as the processing power of the system; fast RAM as the program workspace; and efficient switching devices as the communication system. To interface, and to glue these components together, a custom network interface controller with integrated memory control logic was designed. This is focused on enhancing the inter-processor communication of the distributed system and maximising the features offered by the switching technology.

Taken from the name of the two main components, namely the StrongARM microprocessor and ICR C416 Router, the system is called StrongARM Router Network (SARNet). In order to ease the distributed system implementation and expansion, the system components are arranged modularly. The processor nodes were to be constructed as 'ready to use' building blocks, namely the SARNode. This SARNode building block consists of the StrongARM SA-110 microprocessor, the SDRAM module, and the custom interface chip, the SARNIC, that provided a communication link to the ICR C416 and other functional support. The block diagram of the SARNet system is illustrated in Figure 10 and the SARNode architecture is illustrated in Figure 11.

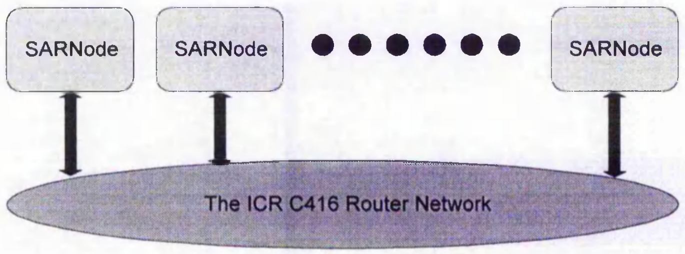


Figure 10: The SARNet scalable embedded distributed processing system.

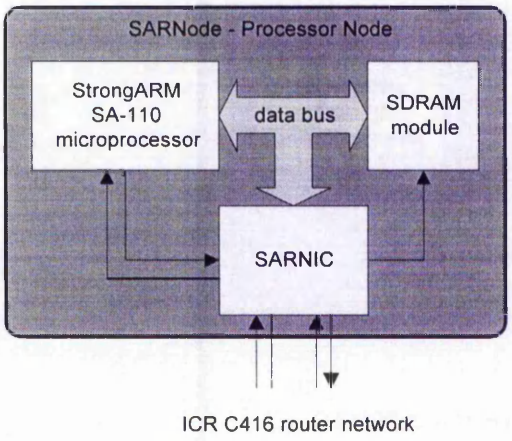


Figure 11: SARNode – the processor node of SARNet.

3.2 Microprocessor Selection

The selection of the processor for a distributed system can be complex. Two key aspects will be performance and cost. One can always use the microprocessor with the highest performance, but the system must be cost effective.

During the 1980's, most microprocessors applied Complex Instruction Set Computing (CISC) architectures. Generally, they provided a large set of complex instructions to simplify the software code development and to increase execution efficiency. However, through years of study, statistic analysis showed that the simplest instructions were used most often. Therefore, by replacing the rarely used complex instructions with the multiple instructions, the core can be optimised, which provided more powerful processor cores. Because of the reduction in the instruction-set complexity, and thus the reduction in hardware complexity, the processor core can operate at a higher clock rate and execute the instruction in lesser cycles. These microprocessors, called RISC microprocessors, typically contains less than 100 instructions and use simple addressing modes [66, 67].

Currently there are many fast RISC processors on the market including: MIPS Technologies Inc. R3000 family, ARM Ltd. StrongARM family[2], Motorola PowerPC family, and Sun Microsystems microSPARC. Although the microprocessor designs vary greatly, there are commonalties between all of them: address generation; arithmetic logic units; register files; and most importantly on-chip cache and memory management hardware features. The StrongARM SA-110 was selected [34]. This was due to its high performance and low price, in contrast to other choices of 32 bit RISC microprocessor at that moment of time. However, utilisation of other microprocessors could be easily achieved, as the interface support logic for the microprocessor was implemented in an easily adaptable manner using a modular VHDL design entry on a CPLD.

The StrongARM SA-110 microprocessor is an implementation of Advanced RISC Machine Ltd. (ARM) Version 4 instruction set [34]. It is a general purpose 32 bit RISC microprocessor targeted at embedded markets. With the core operating at up to 233 MHz, it achieves 268 Dhrystone (Version 2.1) MIPS. As well as the high performance, the power consumption is remarkably low, at a maximum of 420 mW. It has separate instruction and data caches, each 16 kB in size with 32-byte blocks and 32-way associativity. It also has an 8-entry write buffer with each entry able to contain 1 to 16 bytes. The onchip caches and the write buffer effectively reduce the average amount of memory accesses required by the processor. This allows the memory system to support DMA message channels with minimal performance loss.

2 The StrongARM family was transferred to Intel Corporation in 1998.

3.3 Processor Node Memory Selection

Over the years, DRAM has always been more favourable for microprocessor main memory organisation because it offers more memory bits per device, thus making it more economical in contrast to Static Random Access Memory (SRAM), although SRAM offers a faster access rate. Speed improvements on DRAM have continued on, historically from process and photolithography advances, and more recently due to architectural changes. This can be observed in the revolution from Fast Page Mode (FPM) DRAM, Extended Data Out (EDO) DRAM, to SDRAM [68]. FPM was the standard in DRAM functionality for more than ten years. It allowed faster continuous access of data bits (called the page mode cycles), provided those bits were in the same page. In early 1995, EDO DRAM was introduced, maintaining backward compatibility with FPM in terms of data width and packaging. Technically, it offered even shorter page mode cycles, and therefore increased the available data rates and the overall system performance.

Lately, SDRAM has become the new memory standard for computer systems. It is a technology that has provided the system designer with further advantages in terms of speed and performance and at the same time has reduced the development work required for the total system. Three major differences between the SDRAM and conventional DRAM are the synchronised operation, the burst mode operation, and the mode register. SDRAM uses a clock to synchronise the inputs, while DRAM is asynchronous. Burst mode is a mode where the column address is generated internally once the first access column address has been given, and the following data transactions are permitted to take place on every clock cycle. Mode register allows justification of the SDRAM operation and function into desired conditions.

The SDRAM devices were selected as the SARNet main memory. By having the synchronous access mode and capability to burst data without entering new addresses, the SDRAM is well suited for the StrongARM SA-110 cache line fills operation. In addition, the burst length and latency of the SDRAM can be programmed with a special instruction cycle and can thus be altered to suit different processors. For the flexibility in supporting different size of memory, standard SDRAM modules could be utilised.

3.4 Router Network

At present, the ICR C416 device is used in the construction of the embedded distributed system described here. An upgrade to new router switches is possible in the future without much modification since the physical serial connection remains.

The basic functionality and packet protocol of the ICR C416 have been given in Section 2.2.2.1. An ICR C416 device provides 16 OS Link connections, supporting a maximum of 16 processor nodes. For a larger system, multiple ICR C416 devices can be cascaded to form a network of routers. This offers arbitrary scaling of the network without sacrificing the bandwidth available to each communication channel. To support multiple router hops, the number of message routing header bytes can be varied. Each header byte is used for address decoding of individual router hop.

An advanced feature of the ICR C416 design is group adaptive routing [18]. It is designed for easing network communication bottlenecks by grouping links that create identical paths between two points. Hence, messages that require the same destination can actually be routed through any of the grouped links. The grouping feature also takes into account the effects that bi-directional communication has on message bandwidth, by attempting to select links such that uni-directional communication is prioritised. If a processor node contains multiple link connections to an ICR C416 device, this feature can be utilised in the same manner.

An example of a 5 router distributed network is shown in Figure 12. Links connected between the routers (4 links) are logically grouped using the ICR C416 group adaptive routing algorithm. In this way, it reduces the chances of bottleneck forming at the central router by allowing the routers to automatically select the best path within the grouped links.

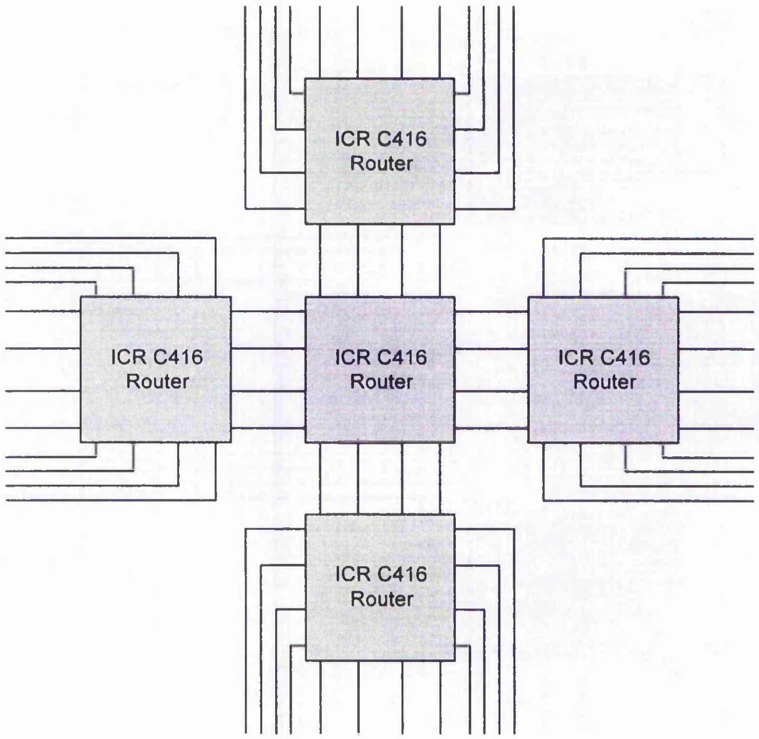


Figure 12: An example of 5 router network.

3.5 Interface and Control

To interface the microprocessor, the memory, and the router network, an interface chip was specifically designed. This interface chip, SARNIC, is to provide mandatory hardware support for the SA-110 processor, SDRAM control, and efficient communication links to the ICR C416. The following sections discuss the requirements, specification, and the feasibility of realising those features.

3.5.1 SARNIC Processor Interface

The processor interface is required to handle bus transaction requests from the StrongARM SA-110 microprocessor. It is also required to decode three major regions of the processor node memory space, which are the SDRAM main memory, the I/O device, and the SARNIC internal registers. For each incoming request, the address of the access must be latched and decoded. Depending on the region of the access, a proper response should be given. For instance, if the access falls within the main memory region, the request must be propagated to the memory interface for SDRAM access; or if the access is for the SARNIC internal registers, the relevant register must be connected to the data bus.

There is also a need for the processor interface to generate interrupts for the SA-110 microprocessor on important events. Example events are the arrival of a new message, end of a message transfer, timer interrupt, and external I/O interrupts. Each of the interrupt sources may be enabled or disabled separately, which is controlled by an interrupt enable register. Conventionally, modification of a particular bit in the enable register required a read of the register followed by a write with the bit updated. A better solution uses two write-only registers, one for setting and the other for clearing bits. A '1' in a particular bit will only modify the intended register bit. There are two interrupt levels for the SA-110 CPU, by which the SARNIC design can interrupt the microprocessor from its normal program execution at two different priorities.

3.5.2 SARNIC Memory Interface

SDRAM module was chosen as the main memory of the processor node. In order to map the SDRAM into the StrongARM SA-110 memory region, a memory interface was required to provide address decoding and control signalling. This allowed the SA-110 CPU to access to the SDRAM region for program execution and data storage accordingly.

Other than the SA-110 CPU, the SDRAM region has to be accessed by DMA channels for message transfers to and from the communication network. To provide efficient message passing, a continuous flow of data is required once a message is initiated. This requires constant data transfers between the SDRAM and the network interface controller across the memory bus, and non-blocking data transfers between the source and the destination network interface controllers across the router network. Also, the constant message transfers on the memory bus should only interfere at minimum level with CPU normal accesses. Therefore, a means of fair access and fair arbitration between the CPU and DMA channels was needed in the memory interface. Fair access is feasible if the DMA transaction size is not larger than the CPU access length requirements. Fair arbitration is achievable if the CPU and DMA channels take turns accessing the memory bus. With these settings, the DMA channels can freely access the SDRAM when the CPU bus is idling, but will never utilised the memory bus bandwidth for more than 50% when the CPU is busy accessing.

Ideally, it would be useful to have the option of supporting different sizes of memory and different SDRAM families. This can be done through some form of software programmable registers holding the configuration of the memory. However the boot from link feature of the design may affect this, as the boot program needs to be downloaded into the memory before the software can actually modify those registers to configure the memory (please see Section 3.5.4.1 for further description). Another possibility is to use hardware static configuration pins. Nevertheless, this flexibility might slow the memory interface and will increase the logic complexity, which should be avoided.

3.5.3 SARNIC Router Network Interface

The router network interface is responsible for handling message passing to and from the router network. It comprises of one or more bi-directional communication link to the router network, depending on the memory-network interface bandwidth available and the size of the target device. Sufficient buffers had to be implemented in order to de-couple the communication link from the memory bus. Additional support was required to include handling multiple message contexts and virtual channels.

3.5.3.1 Communication Link Architecture

The INMOS-style OS Link interfaces were used to connect to the ICR C416. This bi-directional communication link would be DMA assisted in order to achieve efficient transfer of messages. DMA assistance removes the need of processor involvement during message transfers, but incurs a slight overhead on setting up the DMA channel.

As the ICR C416 packet router is utilised for the communication network, the higher level of message passing protocol must conform to the ICR C416 packet protocol. An ICR C416 packet consists of a number of routing header bytes, followed by the packet length byte, and 1 to 256 bytes of payload. A higher level of message protocol, however, normally requires a message header for message identification. A straight forward solution to encapsulate the message header into the ICR C416 packet is to send it as routing header, as the number of routing header bytes allowed are variable. In this way, the true routing header bytes will be stripped off when the packet is routed through routers, while the first byte arriving at the receiving node is the message header, followed by packet length byte, and the payload. The SARNet packet format is shown in Figure 13.

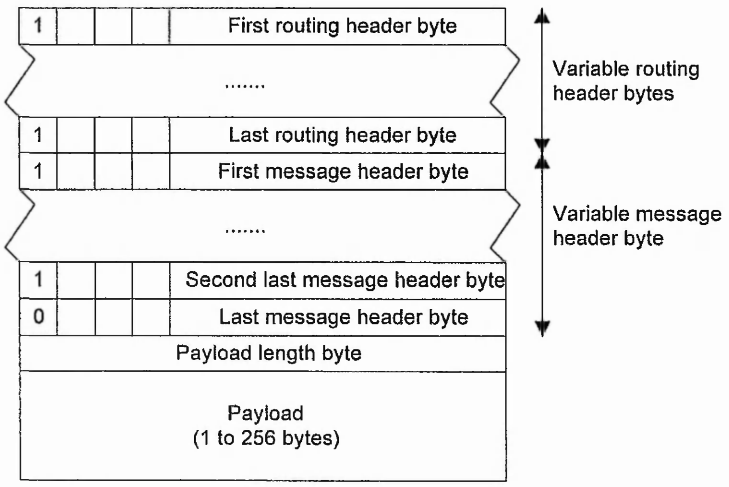

1
First routing header byte
....
Variable routing header bytes
1
Last routing header byte
1
First message header byte
....
Variable message header byte
1
Second last message header byte
0
Last message header byte
Payload length byte
Payload
(1 to 256 bytes)

Figure 13: The SARNet packet format.

As a packet can only be a maximum of 256 bytes in size, this relatively short packet length may be irksome in some applications. Therefore, support for long messages would be preferred, in a form of multi-packets. The operation of packetising and de-packetising large messages was implemented in hardware to minimise software overheads.

In transmit mode, the message formatting is relatively straightforward. A message length register is used to hold the total length of the message in bytes. If the message length register value is less than or equal to 256, only a single packet is sent. If the message length register value is greater than 256, a multi-packet message is transmitted. A packet of 256 bytes is sent with the packet length value of zero and the message length register is decremented accordingly. This process is repeated until the message length register value is less than 256, where the last packet is sent with remaining data bytes. Note that each packet must include the message header for identification purposes.

Receive mode is slightly more complex. On the first packet, an interrupt request to the processor is raised. When the software has analysed the header, it responds by copying the header to the header register and loading the total length of the expected message into the message length register. If the length value is less than or equal to 256, a single packet is received. If the value is more than 256, a multi-packet message is expected. Thus on the arrival of subsequent packets, the header of the packet will be compared with the content of the header register. If it matches, the packet is transferred into the memory and the message length register is decremented. If the header of an arriving packet does not match with the content of the header register, an interrupt is generated immediately, to request for handling of the new message. The process of multi-packet handling is repeated until the message length register value reaches zero. In order to remove the first interrupt request when a message arrives, the header register can actually be pre-loaded with the expected message header.

Ideally, the receiving node's length register and the transmitting node's length register will be set with the same figure for an intended message transfer. However, there might be cases where a software compilation error or some very rare exceptions that cause a difference between the expected incoming message length and the actual incoming message length. It is desirable to have some form of hardware fault tolerance for this situation. Therefore, there can be three different situations for termination of a receiving channel:

- Normal termination: the actual incoming message length is equal to the expected incoming message length. The DMA channel is finished exactly at the end of the message.
- Early termination: the actual incoming message length is less than the expected incoming message length. The DMA channel is not finished and is terminated earlier.
- Late termination: the actual incoming message length is more than the expected incoming message length. The DMA channel is finished with the rest of the message being flushed.

In order for the length fault tolerance to work accordingly, the receiver needs end of message information. With the multi-packet mode configuration mentioned earlier, the last packet is found when a packet is less than 256 bytes. However, for a message size that is multiple of 256 bytes, there is no way to identify the last packet without carrying some special code. In this case, a detection of late termination or normal termination can be difficult. For example, consider a case with the receiver channel expecting 256 bytes and the actual incoming message is 512 bytes, and a case with the receiver channel expecting 256 bytes and the actual incoming message is 256 bytes. Hence, special treatment has to be applied for messages that are a multiple of 256 bytes.

3.5.3.2 Link DMA Control & Buffers

DMA channels are normally used to support the process of sending and receiving messages through a communication link. Although DMA operations suffer from initial start-up overhead, once a channel is activated, no processor intervention is needed until the end of the operation. A processor sends messages by initiating DMA data transfers from memory to the network interface. The network interface then packetises the data and injects it into the network. To receive messages, the processor initiates DMA data transfers from network interface to the memory. Hence, two

DMA channels would be required per communication link to support concurrent bi-directional data transfers.

DMA operations can be classified into two mechanisms: block transfers where the entire data block is transferred across the memory bus in one unit; and cycle-stealing transfers where the data block is fragmented into a series of small transfers that utilise processor idle cycles. Cycle stealing mechanism is a better choice in this case, because smaller transactions offer fair multiplexing of CPU and DMA access on the memory bus. Moreover, the OS Link data transfer rate is much lower than the memory bandwidth. These infrequent transfers can be evenly distributed on the memory bus utilisation. With a block transfer mechanism, no CPU accesses are allowed until the DMA operation is finished.

In order to ensure fair arbitration of access between the link DMA and the CPU, the size of each DMA transfer should not be any bigger than the CPU access size. In particular for the StrongARM SA-110 CPU, the maximum size should only be 8 words. Configuring the access size to this maximum will reduce the address set-up overhead to a minimum. However, larger DMA buffers will be required to hold the data.

Link buffers are required to de-couple the communication link data transmission and the DMA transactions to and from the host memory. There are two factors that affect the buffer size: link transmission speed, and DMA transaction rate. Normally transfer rate over the host memory is faster than the transfer rate over communication link, thus, once a message is initiated, a continuous data stream to and from the communication link can be achieved. However, depending on the network interface architecture and amount of buffering given, the continuous message stream might not be attainable, and thus, this factor will worsen the network bandwidth utilisation. For instance, access to the host memory is shared by other DMA channels, and most frequently accessed by the CPU; DMA transactions have to compete for access on the memory bus. In this case, a guaranteed access rate of the DMA channel on the memory bus, and enough buffering will ensure a continuous data stream to and from the communication link (provided no blockage on the communication link occurs) while waiting for the DMA transaction to complete.

With an I/O bus based network interface, there is no guarantee of consistent DMA access rate to the host memory. It depends on the arbitration of the I/O bridge/memory controller. Thus it is better to carry out large DMA block transfers, even up to a packet size, before transmitting to a link or after receiving from a link. This normally requires SRAMs, or a large First In First Out (FIFO). In the case of the SARNIC design, control over the host memory bus is governed and thus continuous DMA access within a certain time frame is guaranteed. Depending on the link speed, a small amount of buffer is adequate, which can be implemented in Altera FLEX onchip RAM, or even with conventional logic elements.

If the transfer rate over the communication link is faster than the transfer rate over the host memory, the interface would become a bottleneck. Thus, continuous message data flow over the communication network would be impossible, as gaps in transfers would form due to data starvation. To utilise the network bandwidth better and ease the traffic, formatting the message into 'bursts' of packets or smaller flow groups are preferable, before injecting into the network. An output buffer of at least the burst size is required to achieve this.

3.5.3.3 Additional Communication Links

In the absence of a hardware routing device as the switch-based interconnection network, one processor node would need to be equipped with 2 or more communication links to allow it to connect to other processor nodes. Generally, increasing the number of links connected to each node will reduce the mean number of hops required to reach a particular destination. Hardware routing devices like the ICR C416 provides up to 16 point-to-point communication links with just one communication link to each processor node. Based on this fact, the StrongARM processor node can be implemented with only one communication link. Nevertheless, there are still reasons to implement more communication links on each processor node for hardware routing:

- To offer communication fault tolerance through the mean of link replication.
- To increase the bandwidth of a processor node, so that more messages can be received or transmitted at a time.
- To combine with the adaptive grouping feature of the ICR C416, so that the utilisation of the increased bandwidth is further enhanced.
- To provide extra flexibility to the interconnection pattern of the parallel system.
- To create separate communication routes for different types of messages to improve performance and reliability.

There are a number of factors that need to be considered when implementing extra links. The primary concern is the utilisation of the host memory bus. With each communication link implemented, the demand for bus access increases contention on the memory bus, which degrades the processing node performance. Transputers suffered from the same limitation. The designers stated an assumption that the communication overload of those links should not be more than 10% of the overall memory access [37].

Initially, a single communication link was implemented in the first version of the SARNIC. Only when the size of a single communication link and the memory interface controller bandwidth available had been analysed, the feasible number of communication links to be implemented could be decided. This is discussed in Section 3.6.3.

3.5.3.4 Virtual Channels Support

Virtual channels are defined as concurrent logical connections to different destinations or to different processes at a destination concurrently down the same physical connection [69]. In practice, the virtual channels are multiplexed onto the physical communication link, through software or hardware. Therefore, in principle, it is sufficient to connect just one link per processor node to the routing network with the virtual channel capability.

There are a number of different levels, in which the multiplexing scheme on the communication link hardware may be implemented. The highest level of sharing is at the message boundary, where one message is transmitted after another. Multiplexing at the packet boundary provides fairer arbitration access to the transmitter and receiver link hardware. Multiplexing at the flow group boundary is the lowest level of virtual channel implementation. It was introduced to ease the traffic of the routing network and avoid deadlock [70, 71]. As the network routing devices are generally connecting on a packet basis, it is better to use packet level virtual channel support. However, the frequency of arbitration often depends on the size of the packet. Overhead of the arbitration should be minimal to maintain efficiency. Figure 14 illustrates the method of virtual channel multiplexing on a physical link.

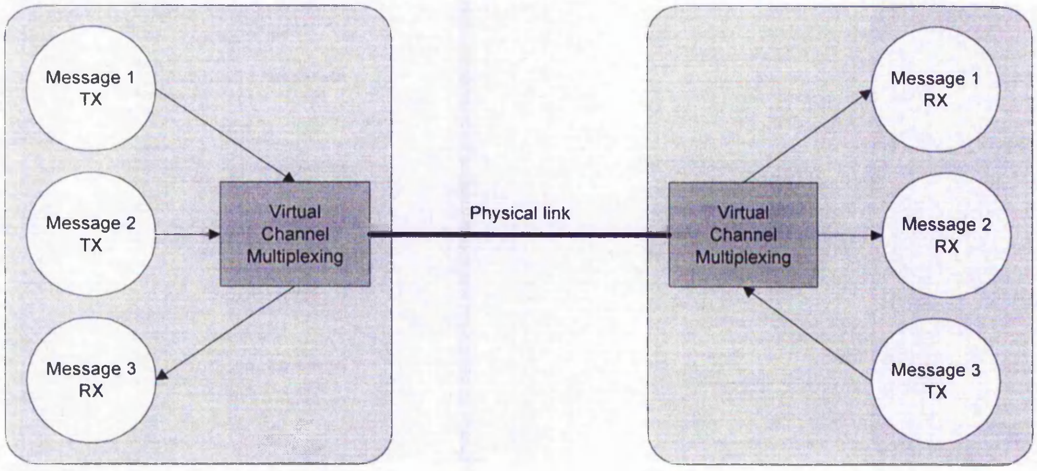


Figure 14: Virtual channel multiplexing of a physical link.

Without support of special multiplexing arbitration hardware, a purely software means of multiplexing can incur significant overhead. The information of an active communicating message channel needs to be stored at the arbitration point, so that the message channel can be continued later from the point it stopped. The task of sequentially matching and finding the corresponding message channel from the table of active messages for an incoming packet is also time-consuming. Depending on the level of multiplexing, this procedure can cause significant inefficiency and latency in servicing messages.

Appropriate hardware support for the virtual channel implementation improves the multiplexing arbitration efficiency. With the procedure of handling virtual message channels being carried out in hardware, intervention from the processor (software), and thus the overhead is significantly reduced. This requires logic to switch and allocate hardware message channels to the corresponding packets, and additional memory to store virtual channel control information. It is an added advantage if the block of virtual channel control information can be implemented with special hardware logic so that concurrent searches for a particular virtual channel can be done, hence reducing the time required.

As hardware logic is a scarce resource, only a limited amount of functionality is feasible. An optimised usage of the virtual channel hardware implementation, with additional support in software for larger numbers of virtual channels would be an advantage. The concept of caching fits well into this situation. Hardware virtual channels are utilised for frequently active messages. Replacement is only required if no match is found on the list of hardware virtual channels. This will speed-up the process of virtual channel arbitration on average.

3.5.4 Other Supports

Other than providing an efficient communication link to the interconnection network, the memory control logic, and the processor interface, the SARNIC design requires some general features to support the StrongARM microprocessor in an embedded parallel processing system. These include a booting option for the microprocessor, timer function, I/O device control, and some hardware debug facilities.

3.5.4.1 Booting options

Normal processor architecture boots up from ROM. If an additional boot option can be supplied through the link (like the 'boot from link' for Transputers), it would be a great advantage for slave subsystems in parallel processing systems. Not only can the cost of ROM be saved, rebooting or reconfiguration of remote processing nodes can also be done through the interconnection network in real time.

The SA-110 processor starts from reset by executing code from memory location zero. A possible solution is that after a reset of the system, the interface chip holds the SA-110 in a reset state, or stalls the first bus access to memory location zero. Then the incoming boot code from the link is DMA transferred to memory from location zero upwards. Of course, the memory at location zero must be RAM in this case. Some intelligence must be built into the communication link interface to recognise this boot message and initiate DMA transfer.

If the memory is software configurable, it will be necessary to download the memory configuration over the link in order to set up the memory before the code is loaded into it. This could be handled using special header bytes that direct the configuration data to control registers instead of memory. Alternatively, memory could be configured by a default power-on value that will work with most memory types.

As the 'boot from link' option requires RAM at memory location zero, while boot from ROM option requires ROM at memory location zero, a method to map the RAM or ROM to memory location zero according to the boot option is needed. Another solution is to keep the memory location zero as RAM, and utilise a bootstrap ROM interface to copy the boot code from ROM to memory location zero by a method similar to the link receiver mechanism.

3.5.4.2 Timer

The easiest way to achieve a timer operation is to supply the number of clock ticks (which is equivalent to the length of time) to a count down counter, and to generate an interrupt when the counter reaches zero. Further flexibility can be added to offer longer intervals by dividing the clock ticks by a factor before using it to decrement the counter. Thus, a wide range of timer periods is supported by using a dividing factor and the count down timer.

Alternatively, a single timer could be used to provide functions comparable to the Transputer on board timer. It would be a 32 visible bit counter that increments in microsecond ticks. This allows any number of concurrent processes to access the read-only clock. Also, a timer match register is needed, to provide a timer value at which an interrupt is to be generated. A value that is written to this register will be compared in relation to the current timer value. If the shortest route from the current timer value to the given value is clockwise, the interrupt event is considered as the time in the future, otherwise it is considered as the time in the past (please refer to Figure 15). An interrupt will be generated at once in the latter case.

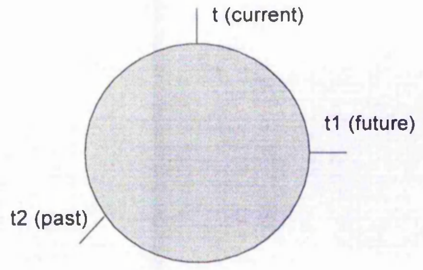


Figure 15: Examples of Transputer timer with past and future time references.

3.5.4.3 Peripheral I/O

This unit provides a low-cost solution for interfacing external passive peripheral I/O devices. These devices normally work at slower speeds and smaller data bus widths. If the number of peripherals is significant, it is probable that external latches and buffers will be required to separate the peripherals from the SA-110 CPU. Hence, direction and enable controls of such buffers are also driven by this interface. The timing of the interface signals may be fixed to a slow access rate, made compatible with the largest possible number of peripherals. Alternatively some level of configurability to the signal timing can be offered to drive the interface at different rate.

3.5.4.4 Debug Facilities

Since the CPLD is re-programmable, any internal signal can be routed out to the spare pins through design re-compilation and re-programming. Through reserving a number of pins of the CPLD device and connecting them to an external connector, device debug facilities can be provided. With the help of a logic analyser, this facility will allow monitoring of the internal and external operation of the device, which will aid the testing phase.

In addition to the hardware debugging pins, a port for console control and monitoring would be useful. A standard input like a keyboard and a standard output like a monitor could be used, but this requires additional interfaces. An alternative will be to implement a standard UART port. By connecting this UART port to a PC COM port and running a terminal program on the PC, a means of console input/output control is achieved.

3.6 Initial Design Feasibility Study

This section describes the initial stage of design and hardware implementation in CPLDs. A general overview of the CPLD technology is given, leading to the selection of the Altera FLEX 10KA CPLD family. Initial stages of development involved an investigation into the restrictions imposed in utilising this device. This included the analysis of OS Link engine implementation with required transmission speed, and the basic SARNIC design realisation with optimised memory bus operating frequency. An additional study was carried out for extended distance communications using differential transceiver circuitry.

3.6.1 CPLD Technology

Historically, FPGAs and CPLDs have been largely used as glue logic, reducing chip count and simplifying board design. Most designs were typically expressed using Boolean equations. Recent CPLDs, however, are high density and high performance. Many of them are equipped with on-chip memory. Nowadays they are employed for system-level integration and are often a preferred alternative to ASIC designs. Some of the advantages are:

- Design development and verification is faster as the CPLD only need to be programmed for operation, without having to wait for prototypes to be manufactured like the ASIC design.
- Design changes can be made without incurring any penalties, as the devices are software configurable and re-programmable by the user.
- Errors can be corrected and different algorithmic approaches can be explored, without further hardware expenses.
- Debugging of the device operation is easier, by configuring some of the CPLD pins as outputs to monitor any particular signals inside the device.
- The overall development cycle is shorter; hence, time to market is shorter.

There are two leading companies currently dominating the CPLD market, these being Altera and Xilinx. Each company offers different architectures, and an understanding of the architectures is required to produce optimum coding. After considering the factors of cost, performance, and availability, the Altera FLEX 10KA family was chosen [35]. The FLEX 10KA CPLDs use an SRAM structure, which means the design configuration has to be downloaded every time the power is re-applied to the devices. The available device size range was from 10 K gates to 100 K gates.

Figure 16 illustrates the architecture of the Altera FLEX 10KA CPLD. It is a series of building blocks connected with different level of interconnects. The smallest unit of the building blocks is the Logic Element (LE), which is sometimes called the Logic Cell (LC). Eight of these LE form a block known as a Logic Array Block (LAB), and the LEs within the same LAB communicate through a local interconnect. Multiples of the LAB are populated as a table of rows and columns. LABs within the same row communicate through the row interconnect, and LABs on different rows communicate through the column interconnect. Therefore, there are three different delays for signal propagation from one LE to another. The shortest will be for LEs that are located in the same LAB, followed by LEs that are on the same row but different LABs, and the longest for LEs that are located at different rows. The same theory applies for connection to external pins.

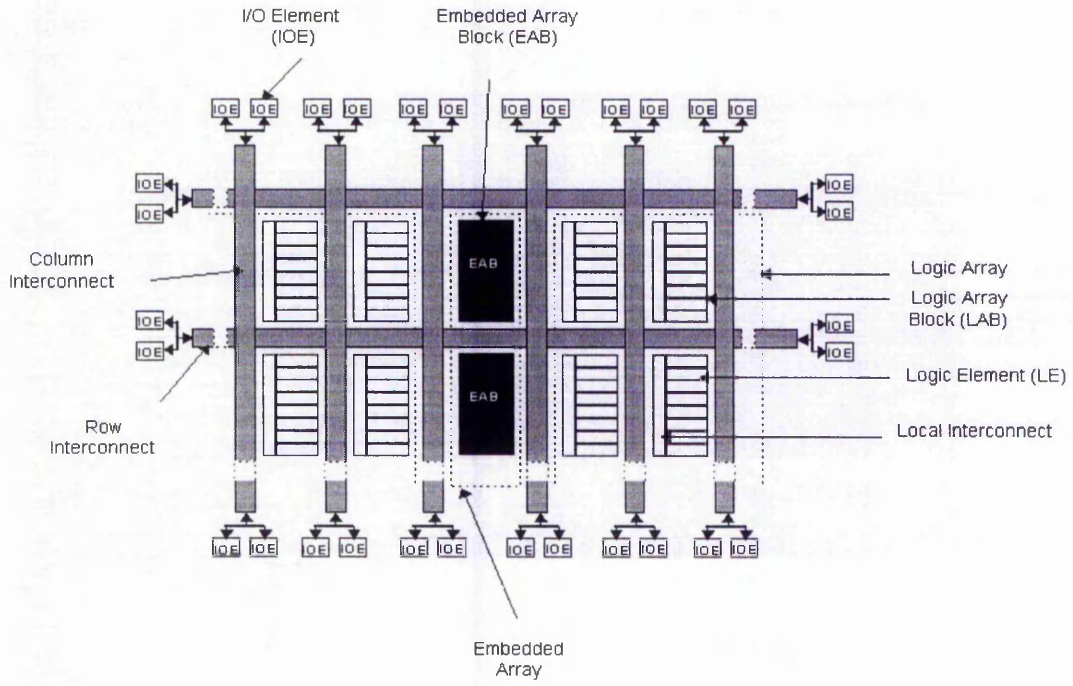


Figure 16: Altera FLEX 10KA architecture.

The construction of a LE is detailed in Figure 17. Basically, it is a four-input Look Up Table (LUT) and a programmable register with clock, preset, reset and enable controls. The dual-output structure allows the LE to be used as combinational output, sequential output, or combination of both. Two special input and output resources of the LE are the carry chain and the cascade chain. The function of the carry chain output is to provide a very fast carry forward path into the LUT input of the succeeding LE. This is especially useful for implementation of adders, accumulators and comparators. The function of the cascade chain is to provide a fast input to a logical 'AND' or 'OR' gate at the output of the succeeding LUT. This chain is ideal for implementing wide input 'AND' gates, wide input 'OR' gates, and multiplexers [72].

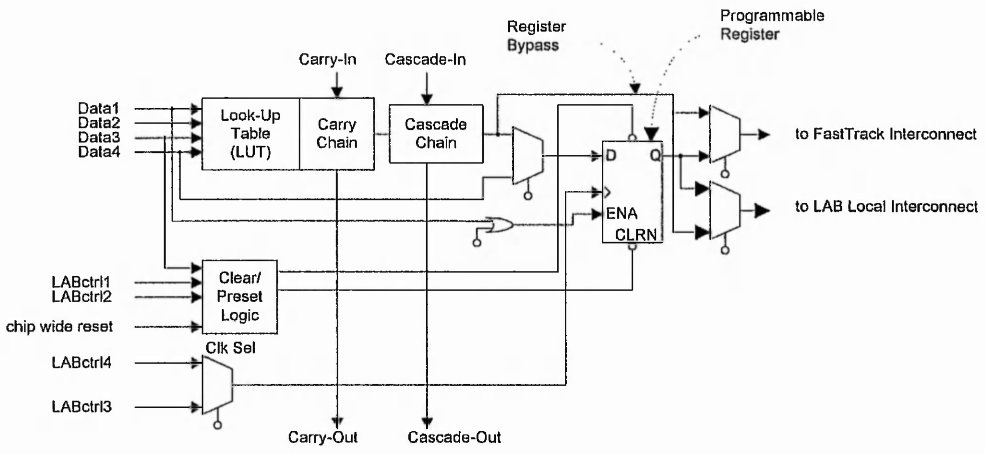


Figure 17: Altera FLEX 10KA logic element.

3.6.2 CPLD Prototype OS Link Implementation

The standard OS Link operates at 10 Mbps or 20 Mbps. To achieve this operating speed, the ICR C416 packet routing device uses a 30 MHz clock input, utilising 3 times over-sampling [29]. The design was fabricated in a 1.5 micron CMOS ASIC implementation. However, implementation of the OS Link design in CPLDs required a feasibility study to ascertain whether those devices were capable of providing suitable performance. The selected CPLD used for this study was from the Altera FLEX 8000 family [73]. This was due to the higher cost of the Altera FLEX 10KA family devices that had just been introduced. The Altera CPLD compiler and synthesiser software, Maxplus II [74], was used for the development of the OS Link interface. Due to incompatibilities of the component library used in the ICR C416 OS Link interface design, the schematic was effectively re-drawn for the CPLD compilation and synthesised in Maxplus II. The initial target choice was the Altera EPF8282ALC84-4, which is the smallest, slowest, but cheapest device in the family. The design fitted well in the device, but the timing analysis reported an operating frequency of 22 MHz, which did not meet the required 30 MHz operation for OS Link.

In order to improve the operating frequency of the design, a second implementation was carried out. VHDL design tools were utilised for this implementation, as it provides a clear representation of the device operation, and the flexibility of moving from one target technology to another with the availability of different synthesisers. The reduction in performance compared to the ICR C416 implementation was mainly due to the logic interconnect delay in the CPLD device. The worst case interconnect delay is almost double the logic delay. Therefore, the OS Link interface architecture was revised and modifications were made to optimise it for the CPLD technology. Using the same EPF8282ALC84-4 device, the performance was improved to an operating frequency of 33 MHz, which meets the required OS Link transmission speed.

To verify the functionality of the modified OS Link interface design, the original simulation vectors used for the ICR C416 OS Link interface were re-produced in a VHDL test bench. This was followed by the implementation of a hardware prototype to test the OS Link interface CPLD in real time.

As the OS Link communication between two nodes is asynchronous, there is a chance of metastability failure. The OS Link interface design implemented a 2-stage synchroniser to reduce the mean time between metastability failures. In addition, for off-board OS Link communications, the use of external link cable tends to distort the signal depending on the length and quality of the cable. The 3 times over-sampling operation of the OS Link should be able to cope with up to 33% of OS Link signal skew.

To analyse the reliability of the OS Link interface CPLD implementation in real time, some tests were carried out with two OS Link CPLD Printed Circuit Boards (PCBs) linked with an external cable. The tests were carried out with one OS Link CPLD PCB configured as transmitter, while the other OS Link CPLD PCB was configured as receiver. These two PCBs were linked with a 30-cm long twisted-pair cable. The CPLD design in the transmitter was modified to repetitively send out a continuous data byte stream, generated pseudo-randomly, following a 17-bit maximum length sequence. The receiver on the other end was modified to check for the correct incoming byte stream, and increment the counter for every correct data byte received. In the case of an error detected, the counter was stopped and error was flagged. The reliability tests were carried out for both 10 Mbps and 20 Mbps OS Link. Tests were run measuring the amount of data transferred until the first occurrence of failure, or in the case of no failure for the duration of approximately two days. The results show at least 72.03 GB for 10 Mbps and 128.04 GB for 20 Mbps. This gives a communication probability bit error rate of 1.74×10^{-12} for 10 Mbps link and 9.76×10^{-13} for 20 Mbps link.

3.6.3 Initial Network Interface CPLD Design

After the successful implementation of the OS Link engine, the SARNIC CPLD realisation was undertaken. The design and development of the SARNIC CPLD proceeded in two stages. The first stage, which resulted in the first SARNIC prototype, was a feasibility study using an Altera EPF10K50V-3 device, with just a minimal set of requirements implemented. These included: an interface to the processor, an interface to the memory, and a single communication link interface to the router network.

The realisation of the first SARNIC prototype went through the same design, simulation, and development stages as the second prototype described in Chapters 4 and 5. Following this, prototype processor node PCBs were made to verify the design. The main objective of the feasibility study, utilising the first prototype, was to investigate the speed of the StrongARM memory bus using the controller implemented in the CPLD. The speed of the memory bus directly affects the data transfer rate associated with the CPU and the DMA channels, and hence, indirectly affects the size of buffer required to overlap DMA transactions, link communications, and the number of communication links that can be supported without overloading memory bus bandwidth. Both the selected CPU and memory can support 66 MHz operation. The only concern was the performance of the Altera FLEX 10KA CPLD.

After refinements and synthesis of the first prototype SARNIC design, the timing analyser reported a device operating frequency of approximately 40 MHz. However, the input and output pin delay of the EPF10K50V device resulted in a system memory bus performance of 33 MHz. The critical timing paths were found to be the data masking bits of the SDRAM, where signals from the CPU must travel through the CPLD before reaching the SDRAM within the same clock cycle. Investigations demonstrated that recompilation with a faster speed grade of FLEX10KA CPLD would only boost the internal operating frequency, with minimal improvement to the input output pin delays.

To overcome the critical path problems, the first prototype SARNIC design was adapted to change to the propagation operation of the critical paths. The address pipeline enable mode of the StrongARM SA-110 was utilised. This mode allowed the memory address signals to be available during the negative phase of the bus clock, effectively half a clock cycle earlier. Hence, by using latches that provide signal feed-through for half of a clock cycle and signal latches for the other half of the clock cycle, an SDRAM mask signal latch pipeline was formed (a more detailed description is given in Section 5.3.4). This pipeline modification pushed the external bus operation up to 38 MHz.

The core clock of the CPLD device was derived from the StrongARM SA-110 microprocessor bus clock. As the SA-110 CPU generates this clock by dividing its core clock, the nearest frequency is 36.9 MHz with the CPU core running at 221.3 MHz. With the 32-bit memory bus operating at 36.9 MHz, the peak bandwidth achievable is 147.6 MB/s. However, this rate is not achievable, due to the overheads of SDRAM accesses and the SA-110 CPU maximum access size of 8 words. At 36.9 MHz, a maximum of 4 clock cycles overhead is incurred for a SDRAM access, plus one extra memory request cycle. Hence, the maximum access time of the SA-110 CPU (T_C(max)) is given by:

$$\begin{aligned} T_C(\max) &= \frac{1+4+8}{36.9\times10^6}\,(s) \\ &= \frac{13}{36.9\times10^6}\,(s) \end{aligned}$$

Equation 1: Maximum access time of the SA-110 CPU.

With the OS Link maximum bi-directional bandwidth of 3.08 MB/s and the size of the smallest DMA transfer (*DMASize*), the interval time between DMA transfers (*I*), the time required for a DMA transfer (T_D), and the memory bus utilisation percentage (*U*) are given by:

$$I = \frac{4\times DMASize}{3.08\times10^6}\,(s)$$

Equation 2: Interval time between DMA transfers.

$$\begin{aligned} T_D &= \frac{1+4+DMASize}{36.9\times10^6}\,(s) \\ &= \frac{5+DMASize}{36.9\times10^6}\,(s) \end{aligned}$$

Equation 3: Time required for a DMA transfer.

$$\begin{aligned} U &= \frac{T_D}{I}\,(\%) \\ &= \frac{5+DMASize}{36.9\times10^6}\times\frac{3.08\times10^6}{4\times DMASize}\,(\%) \\ &= \frac{5+DMASize}{47.92\times DMASize}\,(\%) \end{aligned}$$

Equation 4: Memory bus utilisation percentage.

Therefore, with the smallest DMA transfer size of 1 word, the percentage of memory bus usage is 12.52%. However, by increasing the DMA transfer size to 8 words, the percentage of memory bus usage can drop to 3.39%. Assuming the memory bus arbiter design was made so that the CPU and DMA take turns to access the memory bus, in order to support full bandwidth on all the number of communication links, the number of supported communication links (*NOLink*) is given by the following inequality:

$$
\begin{aligned}
I &> NOLink \times (T_C(\max) + T_D) \\
NOLink &< \frac{I}{T_C(\max) + T_D} \\
&< \frac{4 \times DMASize}{3.08 \times 10^6} \times \frac{36.9 \times 10^6}{13 + 5 + DMASize} \\
&< \frac{47.92 \times DMASize}{18 + DMASize}
\end{aligned}
$$

Equation 5: Number of fully supported communication links.

Consequently, if the DMA size is set to 1 word, the number of feasible communication links can only be 2. However, if DMA size is set to 8 words, the number of feasible communication links can be increased to 14.

Other than determining the feasible bus operating frequency, a problem related to the reset circuitry was found in the first implementation. The clock output from the SA-110 CPU is held high during the reset cycle. This effectively stalled the reset sequence generator that operates from the same clock source. Changes were made to the reset sequence generator, so that this portion of logic operates from the free running 30 MHz clock.

3.6.4 Differential Transceiver for Extended Distance Communications

For tightly coupled processing systems, the processing nodes and the routing devices are in very close proximity, most likely on the same PCB. However, there may be cases where the processing nodes are physically distributed at different locations. This kind of loosely coupled configuration, an example of which would be a home network application, require off-board cable connections.

A direct OS Link is excellent for communication on the same PCB level, and is also good for very short cable connections. As the distance of off-board communications increases, the signal quality deteriorates. The signal amplitude is attenuated and signal skew increases when a signal travels down the transmission line. The cable also tends to pick up noise along the transmission line, including possible hazardous voltage spikes that might destroy the communicating devices. The use of external transceivers and additional protection circuits are helpful in these situations. Depending on the transceiver circuitry, the cable quality, and the data transmission rate, there is a limit to the cable length achievable. The study here was dedicated to find an optimum transceiver circuitry solution for the OS Link for large distance transmissions, and thus the maximum cable length figures under various data transmission rates.

A transceiver device is a combination of a transmitter and a receiver. The transmitter converts a data value to its specified electrical level representation that is suitable for long distance transmission. The receiver detects signals that are above its acceptable threshold value for decoding to the corresponding data values. For high performance and reliable long-distance communications, differential transmission interfaces like RS-422 or RS-485 are commonly used. The standard RS-422/RS-485 devices can accommodate data transmission rates of 10 Mbps at distances of up to 10 m, or data transmission rates of 100 kbps at distance up to 4 km. Due to technological advances, new RS-485 transceivers offer the potential to transmit and receive at faster speeds [75, 76, 77, 78]. Nevertheless, there is always a constraint of maximum data transmission rate versus maximum distance regardless of which interface circuit used. At high-speed data transmission at distances up to 100 m, the effects of signal skew are more significant than the effects of amplitude attenuation.

· The OS Link interface test design in section 3.6.2 was reproduced in this study, using the EPF10K10ATC100-2 CPLD from the Altera FLEX 10KA. The CPLDs act as the terminal for generating and comparing random bytes of a data packet. With these advanced CPLDs, the OS Link implementation is capable of operating at 44 Mbps transmission rate. The transceiver used was the Analog Devices ADM1485 device [75], which provided differential transmission across a long distance twisted pair cable. The twisted pair cable chosen was the CAT 5 UTP cable from Belden (SM1720A) [79], which is designed for 100 Mbps LAN communication. The connection was full duplex, achieved by separate twisted pair lines for transmit and receive.

When measuring the quality of the signal received, eye pattern is the most common and effective method [80]. The eye diagram was obtained by using the infinite persistence display mode of the oscilloscope, triggered from the transmitting clock. From the opening of the eye, the tolerance of the transmission system that could be sustained was determined. The opening of the eye was measured in two aspects: the height and the width. The height information states whether the amplitude of the incoming signal is acceptable to be received and interpreted correctly, whilst the width information states whether the receiver system is capable of sampling the data correctly. With the ADM 1485 transceivers used, the height value had to be greater than the threshold value of 0.2 V. With the 3 times over-sampling of the OS Link interface implementation, the width value must be greater than 67% of the bit width. These two figures were used as the acceptable reference margin for the tests.

3.6.4.1 Basic Transceiver Circuitry Configuration

The test carried out in this section was devised to imitate realistic OS Link transmission by transmitting random bytes over the cable connection. It was generated using 17 bit Maximum Length Sequence (MLS) pseudo-random bit stream in the CPLD. Each group of eight bits of the MLS bit stream was formatted to an OS Link token, fed to the differential transceiver, and then sent along the twisted pair cable. The transceiver circuit was primitive, with only 100 Ω termination resistors across the two differential signals, at both the transmitting and receiving ends. The eye diagram was obtained at the receiver differential inputs to measure the signal quality. The test was then repeated for different cable lengths and different baud rates.

The results are shown in Figure 18 and Figure 19. Detailed values are given in Appendix D. These graphs show that the amplitude attenuates approximately linearly with distance. However, for signal skew, the eye opening reduces approximately logarithmically with distance. When the baud rate is increased, both amplitude attenuation and eye opening reduces significantly.

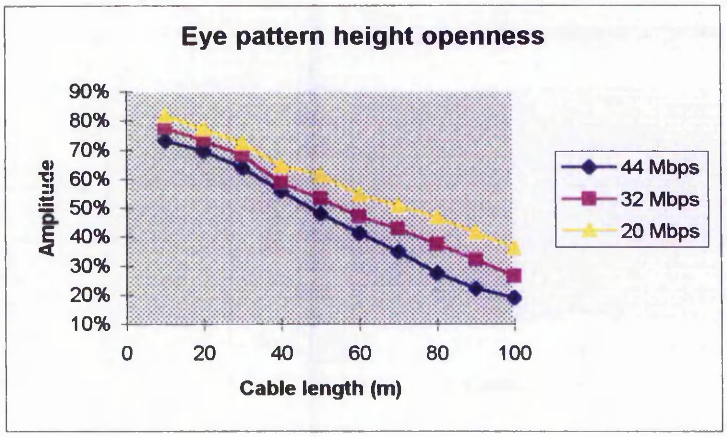


Figure 18: The eye pattern height openness versus cable length for basic configuration.

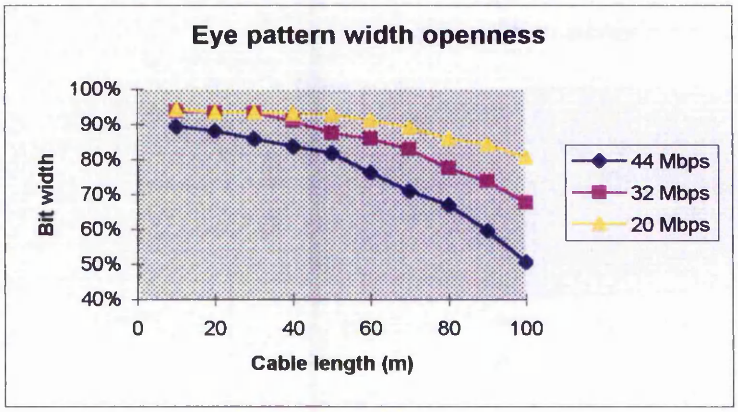


Figure 19: The eye pattern width openness versus cable length for basic configuration.

3.6.4.2 Transmission Line Balance Improvements

Following the initial test, the transmitter circuit was modified to improve signal transmission. From the results in previous section, 44 Mbps transmission reaches its limit at 80 m. Thus the experiment was run at this speed and this cable length, where changing the transmitter configuration would be expected to have the greatest effect on signal quality.

It was noted that the skew problem appeared to be dominated by the line balance caused by the imbalance of numbers of 1's and 0's sent. In order to highlight this, four different sets of data patterns were used to carry out the tests, and the signal quality was measured. These data patterns were:

- random - generated with 17 bit pseudo random MLS
- unbalanced positive - generated with data value of 11111111_b
- unbalanced negative - generated with data value of 00000000_b
- balanced - generated with data value of 00110011_b

The results are shown in Table 1. With the basic configuration, it can be seen that the balanced data pattern had the best signal quality, while the unbalanced data patterns were worst. To improve the line balance, some passive components were added to the transmitter output circuitry, which limited the unbalanced signal charges and terminated the line impedance. With this additional circuitry, the tests were repeated. The results showed that the eye opening was wider when compared to the previous results. Also, the variations in results between the data patterns were small.

Data pattern	Eye height	Eye Width
Without line balance improvement		
Random	1.52 V	15.9 ns
Unbalanced +	1.76 V	14 ns
Unbalanced -	1.66 V	16.7 ns
Balanced	1.6 V	18.9 ns
With line balance improvement		
Random	1.92 V	19.5 ns

Unbalanced +	2 V	20.4 ns
Unbalanced -	1.94 V	19.9 ns
Balanced	1.88 V	19.8 ns

Table 1: Result for line balance improvement tests.

3.6.4.3 Transceiver Circuitry Diode Protection

The desired protection had to suppress transient noise voltages induced on the twisted pair cable, which could damage the transceiver circuitry. Galvanic isolation provides good protection, but incurs a higher cost. Transient Voltage Suppressor (TVS) diodes are cheaper and more commonly used, although they provide lower levels of protection. The tests here only utilised diode protection.

Initially the Transil TVS diodes from SGS-Thomson [81] were selected based on characteristics and price. They can sustain peak pulse powers of up to a few thousand watts. However, they posses a fairly large capacitance, which will effectively distort the signal quality, especially at high speeds and long transmission distances. A second solution was selected, using lower power zener diodes, with much lower capacitance, in conjunction with a current limit resistor (to limit the transient current). Unfortunately, the current limit resistor further attenuated the signal amplitude. Both solutions described here have their pros and cons in system protection.

It is necessary to consider the absolute maximum voltage input of the transceiver when selecting the voltage clamping level of the protection diode. In this case, the absolute maximum voltage input of the ADM1485 transceiver was +/- 14 V. Thus, the TVS diode selected for the first configuration was the P6KE13CA from Fairchild (this is a direct equivalent of the Transil device). For the second configuration, the zener doide selected was the BZX7912 from Philips [82]. From calculations, the minimum value for the current limit resistor is estimated at 40 Ω for protection against a 100 V pulse. Tests were repeated for different values of resistors to examine the signal quality. For all configurations, the cable length was 100 m, and the bit rate was 44 Mbps. The transmitter was equipped with the line balance output filter, as described in the previous section.

The results are summarised in Table 2. From the result, it can be seen that TVS diodes are not suitable for high-speed transmission. The eye pattern is totally closed. This only left the option of using zener diode with a series current limit resistor. The zener diode protection circuitry only incurs about 1 ns extra skew on the signal, which was considered acceptable.

Protection	Eye height	Eye width
None	1.02 V	17 ns
TVS	0 V	0 ns
0.4 W zener diode, 0 Ω resistor	1.02 V	17 ns
0.4 W zener diode, 56 Ω resistor	0.92 V	16.6 ns
0.4 W zener diode, 82 Ω resistor	0.84 V	16.1 ns
0.4 W zener diode, 100 Ω resistor	0.8 V	15.5 ns
1.3 W zener diode, 0 Ω resistor	0.98 V	17 ns
1.3 W zener diode, 0 Ω resistor	0.88 V	16.2 ns
1.3 W zener diode, 0 Ω resistor	0.74 V	15.7 ns
1.3 W zener diode, 0 Ω resistor	0.74 V	15.2 ns

Table 2: Result for diode protection tests.

3.6.4.4 Overall Transceiver Circuitry Performance

With all the improvements and added protection, as shown in Figure 20, the transmission line random data tests were repeated to assess the overall performance. Figure 21 and Figure 22 show the results of the test (Appendix D gives detailed figures). The results were a significant improvement compared with the initial tests. It can be seen that the bit width remained fairly constant over the range of distances, and they are considerably higher than the limit of 67%.

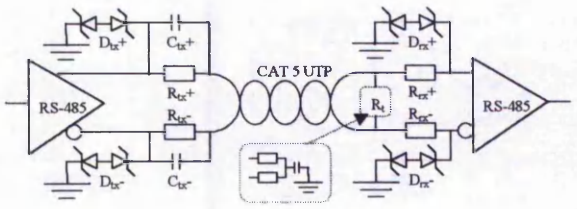


Figure 20: Enhanced RS-485 circuit diagram (for one direction).

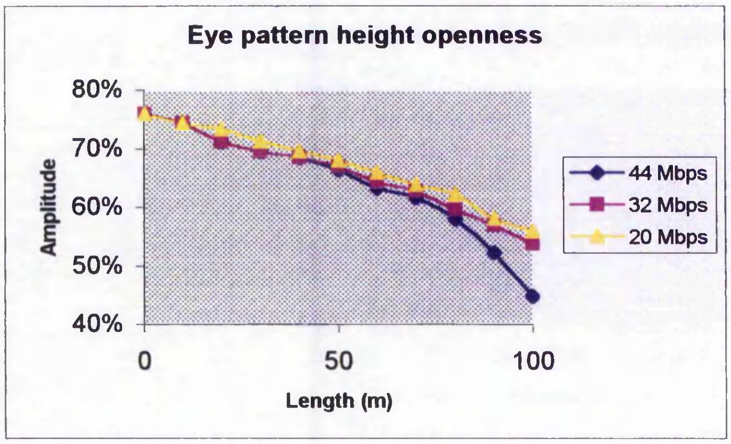


Figure 21: The eye pattern height openness versus cable length for improved configuration.

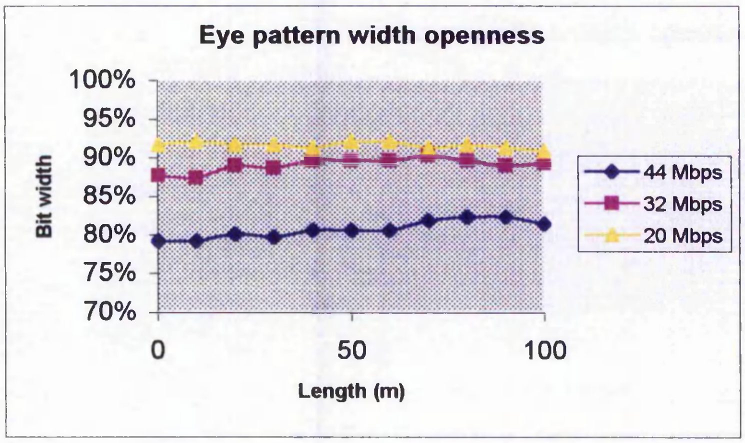


Figure 22: The eye pattern width openness versus cable length for improved configuration.

At the maximum cable length (100 m), the test was then run for extended periods, to discern the bit error rate. The results were: 1.45 x 10^{-12} at 20 Mbps, 1.68 x 10^{-13} at 32 Mbps, and 3.6 x 10^{-12} at 44 Mbps.

4. SARNIC IMPLEMENTATION AND STRUCTURE

This chapter discusses the approach and techniques used in the design of the novel network interface controller, the SARNIC. The design utilises VHDL in a modular, top down, hierarchical method. VHDL was selected as it provides a clear representation of the device operation, and the flexibility of moving from one target technology to another with the availability of different synthesisers. The software tools employed throughout the design cycle were the Mentor Graphics QHDL compiler and simulator, based on Sun Workstations.

It was decided to target the design in the Altera EPF10K50V CPLD [35]. The re-programmable feature of this SRAM based CPLD makes prototyping of the design more economical and eases design upgrade. In addition, it offers the flexibility of in-circuit debug, as simulation tests might not cover every detailed operation. The Altera CPLD software development tools, Maxplus II [74], provides a platform to construct designs from different entry formats (including VHDL) down to the complete end-product level. However, its simulator does not support proper test bench, and lacks debugging facilities throughout the design stage. Thus the Mentor Graphics QHDL compiler and simulator is used instead, until a full working VHDL model is ready, which is then transported onto the Maxplus II platform for synthesis.

4.1 Top-Down Design

The design of the SARNIC device was utilising top down design methodology. It was broken down into a modular and hierarchical structure. Each module was then constructed and simulated before inserting into a higher level module. This simplified the design and development process of the device. The structure of the design also eases future modifications or reuse of the modules in other designs.

The SARNIC design fragments into six main functional modules: the Bus Controller, the Communication Controller, the Control Link, the Interrupt Controller, the Timer, and the UART. Also, some reset circuitry, which occupies a small portion of the chip, controls the booting sequence of the whole device. Figure 23 illustrates the block diagram of the SARNIC. The functions of each module are described in the following sections in greater detail.

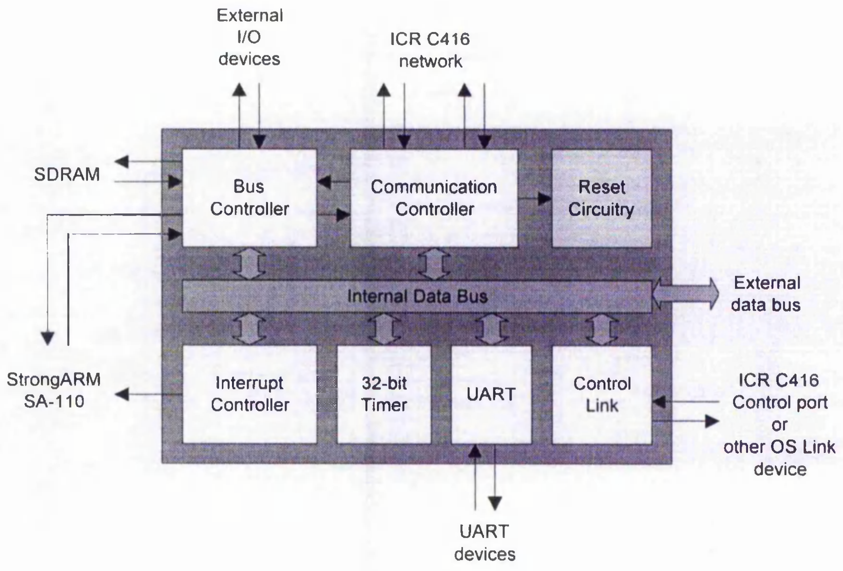


Figure 23: Block diagram of the SARNIC design.

4.2 Bus Controller

The bus controller interfaces the CPU and communication DMA channels to the addressable memory-mapped devices. These memory-mapped devices include the fast access SDRAM main memory, external peripheral I/O devices, and the SARNIC internal registers. Each of the memory-mapped devices will be control by separate blocks: the SDRAM Interface, the External I/O Interface, and the Internal Register Interface. The Arbiter Core is responsible to arbitrate the requests from the CPU and Communication Controller DMA, and to select which block of memory devices interface to activate. Priority is given to the CPU access, so that the overhead of DMA transactions on CPU normal operation is minimal. This is illustrated in Figure 24.

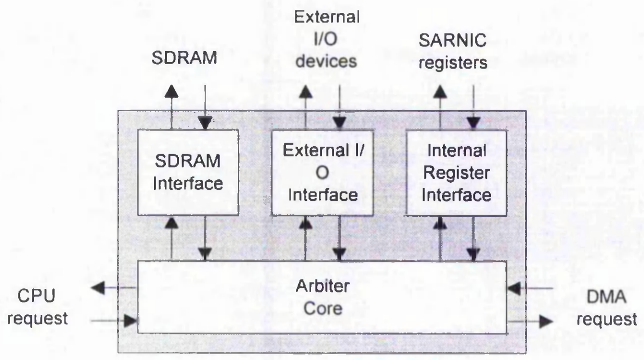


Figure 24: Block diagram of the Bus Controller.

The address space of the processor node is divided logically into three main regions: the SDRAM space, external I/O space, and SARNIC register space. For ease of logic decoding, the physical memory devices are only populated at the bottom of the logical region, and mirrored to the rest of the space. Table 3 summarises the address mapping of the SARNode.

Memory device	Start address	End address	Size
SDRAM	0000 0000h	01FF FFFFh	32 MB
SDRAM (mirror 1)	0200 0000h	03FF FFFFh	32 MB
.....	.....	.....	.....
SDRAM (mirror 63)	7E00 0000h	7FFF FFFFh	32 MB
External I/O (CS0)	8000 0000h	8FFF FFFFh	256 MB
External I/O (CS1)	9000 0000h	9FFF FFFFh	256 MB
External I/O (CS2)	A000 0000h	AFFF FFFFh	256 MB
External I/O (CS3)	B000 0000h	BFFF FFFFh	256 MB
SARNIC registers	C000 0000h	C000 00FFh	256 B
SARNIC registers (mirror 1)	C000 0100h	C000 01FFh	256 B
.....	.....	.....	.....
SARNIC registers (mirror 4194303)	FFFF FF00h	FFFF FFFFh	256 B

Table 3: Address mapping of the SARNet processor node.

4.2.1 Arbiter Core

The task assigned to the Arbiter Core is to handle requests from the StrongARM SA-110 microprocessor and Communication Controller DMA channels. This includes latching the memory access address and requests from the StrongARM SA-110 and the DMA engine individually.

The Arbiter Core is also required to arbitrate the memory bus access between the microprocessor and DMA engines. The requests from the CPU and the DMA engine are pipeline and queued. When the memory bus is idling, priority is always given to the CPU. However, when requests are pending, access to the memory bus is alternated between CPU and DMA requests. This provides a fair memory access for both sources. No one particular source can dominate the memory bus utilisation.

When a request is being served, the address will be decoded in this block to decide which block of memory device interface to activate. Corresponding control signals are then sent to the appropriate memory device interface, to proceed on the bus transaction cycles. Necessary wait states, depending on the memory device accessed, are inserted to stall the request source until the data bus is ready for data transaction.

4.2.2 SDRAM Interface

The function of the interface is to generate the physical row and column addresses, and provide control signals, for the SDRAM. It also generates memory refresh cycles at the required interval. The start-up sequence of the SDRAM module has also been implemented into the interface.

The current version of interface is designed for 16 MB or 32 MB SDRAM modules, with 4 chip select lines, 11 row address lines, 9 column address lines, and 1 bank address line. For connection to a 16 MB module, 2 of the chip select lines should be left unconnected. As the SARNIC operating core frequency is fed from the StrongARM SA-110, which is configurable, the memory bus can be set-up to different speeds. Therefore, it is advantageous to allow the timing of the SDRAM control signals, such as bank active and row precharge cycles, to be configurable. The

SDRAM Timing Register is used for this purpose. Moreover, different SDRAM families or manufacturers may have differences in the exact timing specifications. The SDRAM Timing Register offers the option to optimise the SDRAM interface timing to a specific SDRAM module. As most SDRAM modules [83, 84] conform to the standard rate of 4096 refresh cycles every 64 ms, the refresh interval for the SDRAM has been fixed to this number. Hence, the refresh interval was generated from the 30 MHz clock, as the core clock can be variable.

The SDRAM access consists of 4 consecutive states in a state machine: *idle*, *row-address*, *column-address*, and *precharge*. Each state can be a single bus cycle or multiple bus cycles, depending on the timing specification of individual SDRAM module. In the *idle* state, the SDRAM Interface waits for an access request. When a request is detected, the *row-address* state specifies the row address of the SDRAM location. This is followed by the *column-address* state that specifies the column address of the SDRAM location, together with the read or write command. For a read operation, the data output will only be available after several cycles delay, defined by the CAS latency of the SDRAM module. For a write operation, data can be written to the SDRAM module in the same cycle. The final state is the *precharge* state that commands the SDRAM to rewrite the bit line and reset the internal row address. The state machine then returns to the *idle* state, and is ready for new requests.

The SDRAM operation supports burst modes of 2, 4, or 8 words, with sequential or interleaved burst sequences. In this way, the column address is only specified once, and the burst continues from that location. Most accesses from the StrongARM SA-110 CPU are expected to be cache line operations of 8 words, which are suitable for the burst mode operation of the SDRAM. However, there are also shorter accesses from the SA-110 CPU, which range from a single word to a burst of 4 words. During these short accesses, the SDRAM interface does not know the number of cycles in the processor burst until the microprocessor's memory request signal de-asserts. This makes the use of SDRAM burst operation awkward. Therefore, the circuit implements processor bursts cycles by setting the SDRAM burst length to one cycle and issuing back-to-back read/write commands [85].

4.2.3 External I/O Interface

The External I/O Interface allows low-performance peripherals to be attached to the system. This interface will provide control signals such as chip selects, read, write, and buffer direction.

A total of four chip selects have been implemented, decoded from address lines 28 and 29. This gives a window of 256 MB for each I/O device. More devices can be attached by decoding the lower address externally. The read and write operation is controlled by a separate active low read signal and an active low write signal. A buffer direction control signal has also been provided to control the flow direction of the optional data buffer.

The timing of the External I/O Interface control signals is software configurable. This is done through the I/O Control Register. There are three fields in the register: cycle length, strobe mask, and frequency divisor. The cycle length field defines the number of cycles of each I/O access, up to a maximum of 8 cycles. The strobe mask represents the cycle state of the strobe, shifted out to the read or write, starting with bit 0. The frequency divisor pre-scales the frequency that is used to time the cycle length and shift the strobe mask, in the range from 1 to 4. By setting these fields, the user can impose the address set-up, the strobe duration, and the address hold for the I/O access, thus offering an interface for a wide range of different speeds of I/O devices.

4.2.4 Internal Registers Interface

This block provides the address decoding and read write controls to all the internal registers. The six lower order bits of the address bus (*a2* - *a7*) are used to access 64 different internal register locations. Note that the register access must be in a single word. Byte masking is ignored in this design, in order to reduce the logic complexity.

Because the Altera FLEX 10KA devices do not support internal bus implementation, the register output has to be implemented using multiplexers. For a register read, the 64 internal register outputs have to propagate through six stages of multiplexer busses (32 bit multiplexer). The first stage comprises of 32 2-to-1 multiplexer busses selected by address line *a2*, the second stage comprises of 16 2-to-1 multiplexer busses selected by address line *a3*, and so on until the final stage of a single 2-to-1 multiplexer bus selected by address line *a7*. This configuration would result in a total of 63 2-to-1 multiplexers busses used for 64 registers (effectively 2016 multiplexers if fully implemented) and large fan out to the address logic. Even though not all of the register locations contain physical register bits, the total logic and routing resources used for the multiplexers is a significant portion of the total design. The six stage multiplexer busses also resulted in 2 wait states (decided after synthesis and timing analysis) for each register read.

For a register write, data inputs of the selected register will be enabled following the address decoding. Only one wait state is needed to enable the CPU data bus.

4.3 Communication Controller

The Communication Controller is the largest part of the SARNIC design. It is the system network interface that controls every message passing operation to and from the interconnection network. There are two Communication Links implemented, each comprises of two independent incoming and outgoing links. A total of four DMA message channels, two incoming and two outgoing, are designed to support the Communication Links. One special feature of this Communication Controller is that the message channel connections to the communication links are not fixed: they are handled by the Message Allocater Switch. This allow flexible use of the message channel resource, so that both incoming channels or both outgoing channels can be allocated to one specific Communication Link for hardware virtual channel support. The block diagram of the Communication Controller is shown in Figure 25. The following sections describe the sub-blocks operation in more details.

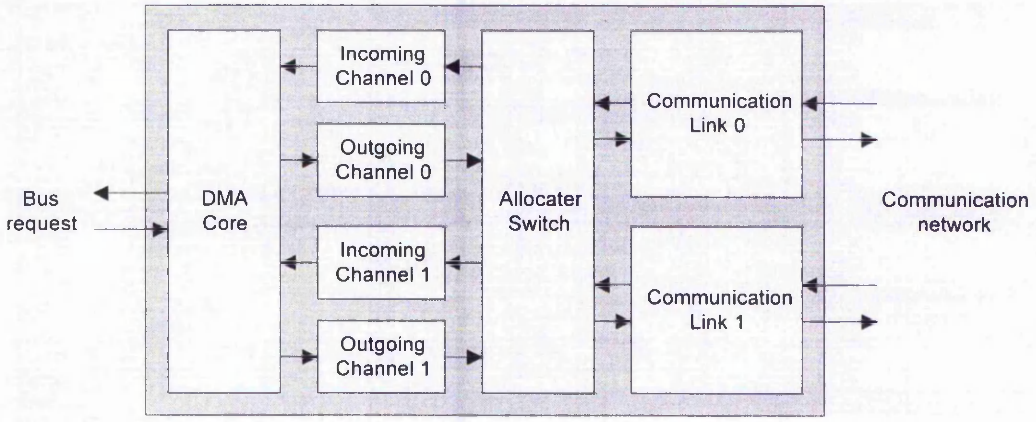


Figure 25: Block diagram of the Communication Controller.

4.3.1 DMA Message Channels

Four DMA assisted message channels have been implemented in order to support the Communication Links message-passing operations. Two of them are for the outgoing direction while the other two are for the incoming direction. The DMA mechanism utilised is a cycle-stealing mode, where the DMA transfer of a message is partitioned into a series of small transactions, instead of doing a single large block transfer across the memory bus. This is based on the fact that smaller block transfers allow better sharing of the memory bus among other memory bus devices, most importantly the CPU. The memory bus controller gives priority to the CPU access, so that the DMA transfer does not occupy the memory bus too often.

With integrated DMA engines and the pipeline design of the Bus Arbiter Core, memory bus switching inefficiencies are reduced. This is an advantage compared to a DMA message channel implementation that would require frequent short accesses to the memory bus. The DMA transfer size selected is a single word (discussed in Section 4.3.2.3).

The concept of virtual channels was implemented in the hardware design. Hence, the connection of message channels is not fixed to any particular communication link. Their use is assigned to a corresponding communication link through the allocation of the Message Allocater Switch. In this way, more than one message channel can be allocated to the same communication link, and the use of this communication link will be shared by the assigned message channels through a multiplexing scheme at the packet boundary. The hardware version of virtual channel support offers efficient switching and sharing of the communication link, thus achieving higher utilisation of communication bandwidth. For example, once a message channel has completed its transmission, the communication link can switch immediately to the other message channel to continue with the second data transmission.

The architecture of the transmit and receive DMA message channels are generally the same, each using three registers to control their operation. These registers are:

Header Register - used to hold the message header for incoming channels, and to hold routing headers and message headers for outgoing channels. The header register of the incoming channel is only 2 bytes wide. The header register of the outgoing channel is split into two physical 32-bit addresses (high and low) as it accommodates up to 6 bytes of header.

Address Register - the pointer to the message address. This is a 24-bit register, offering a memory access range of 16 MB, which can only cover the 16 MB of SDRAM.

Length Register – for counting the message length and controlling the state of operation. The 16-bit length field offers a maximum message size of 64 kB. The rest of the bits are for control and status purposes corresponding to the incoming or outgoing channels. These controlling status bits include: the *active* bit that indicates whether a message channel is active or not; the *done* bit that signify the completion of message transfer; and the *reset* bit that is used to reset the message channel in the case of errors.

The operation of each message channel is identical. Three states are involved in the state machine: *idle*, *active*, and *done* states. In the *idle* state, message channels are waiting for the assignment of a new message passing. To activate a message channel, information about that message needs to be specified in the Header, Address, and Length Registers. The final write must be the Length Register, as this will change the message channel to the *active* state, during which no modification to the registers is allowed. In the *active* state, outgoing DMA transactions will be initiated and the message continues to spill out on the assigned transmitter link, provided the link is not blocked; while incoming DMA transactions will be initiated whenever there are data coming in from the receiver link. At the end of the message transmission or reception, the message channel terminates and moves to the *done* state. At this state, the message channel will wait for the acknowledgement from the CPU that corresponds to the completion notification of the message transfer. Once the acknowledgement is received, the state machine resets to the *idle* state and the message channel is ready for use again.

In normal operation, the receiving DMA message channel terminates exactly at the end of message. However, for debugging and software development purposes, two extra modes of termination are added. They are the late-termination, where more data bytes are coming than the expected; and the early-termination, where less data bytes are transferred than expected. In the case of late-termination, redundant bytes have to be flushed. To cope with the difficulty of handling message sizes that are multiples of 256 in late termination or normal modes (described in Section 3.5.3.2), such message channels are treated as normal termination, whether the actual incoming message is exactly the same size or greater than expected. This results in subsequent packets of the same incoming message, that are greater than expected, being treated as a new message.

To support the boot from link feature, DMA message channel 0 is embedded with extra functionality. When a boot request is received, the Address Register will be reset to zero. The incoming data bytes will then be transferred to memory, starting from location zero where the SA-110 CPU boots from.

4.3.2 Communication Links

The main task of the communication links is to handle the operation of message passing through the ICR C416 router network. This includes packetising and depacketising of messages, insertion and extraction of the headers at the appropriate place, and the low-level link flow control. Additional features include decode of the boot header from the links, and selection of boot from link or from internal ROM.

The communication link utilises an OS Link. There are two external communication links implemented on the device. The two links effectively double the bandwidth that one communication link can provide. Coupled with the group adaptive routing function of the ICR C416, one single destination header can be used for both communication links if they are connected to the same ICR C416 router in a continuous sequence. At the same time, two communication links offer greater network system design flexibility. They also increase the level of fault tolerance through resource replication, where if one link fails, the other can still support the system, although with reduced capability. The flow diagram of the communication link control is illustrated in Figure 26.

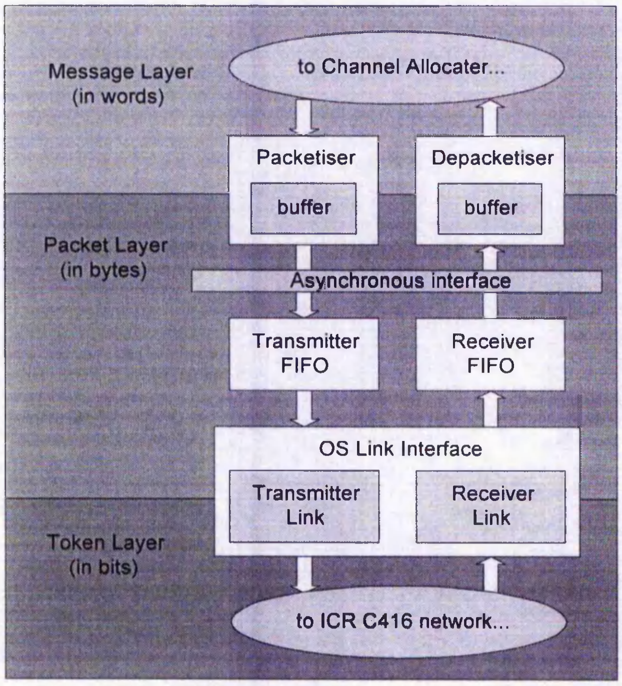


Figure 26: Flow diagram of the Communication Link control.

A message-passing mechanism involves the transmission of one message from the source communication link, through the communication network, and the receipt of the message at the destination communication link. The transmission of data through a communication link can be viewed at three different representation layers: message layer, packet layer, and token layer. The process of a message transmission includes: DMA transfer of an outgoing message from the memory (message layer); packetising the message and pushing the packets into the Transmitter FIFO, byte by byte (packet layer); and finally the injection of bytes onto the serial link by the OS Link Interface (token layer). As a complementary function, the process of a message receipt includes: formation of a byte from the incoming bit stream by the OS Link Interface (token layer); popping the packet from the Receiver FIFO, byte by byte and de-packetising the message (packet layer); and finally a DMA transfer of the message to the memory (message layer).

From the hardware point of view, the communication link is working on two different clock domains. In order to sample the OS Link bit stream correctly, the link FIFOs and the OS Link Interface are working on a 30 MHz clock, supplied from an external crystal module. The rest of the circuitry is working on the memory bus clock, hence, utilising the faster bus clock speed of the SA-110 microprocessor. The following sections give a more detailed explanation of each sub-module.

4.3.2.1 Packetiser

The function of the Packetiser is to request DMA transactions from the memory and to fill the DMA buffers with the outgoing message. The message will be divided into packets, and patched with corresponding routing information and message headers before the packets are pushed to the outgoing link FIFO.

When a start request is detected from the DMA controller, the control logic will shift the contents of the Message Header Register to the transmit FIFO. Although the register is 6 bytes wide, there can be less than 6 bytes of header sent. The last header byte is recognised by setting the MSB to ‘0’. At the end of the shifting operation, the packet length byte is generated and sent, based on the Message Length Register value. This is followed by the required number of data bytes DMA transferred from the memory.

For short messages that are less than or equal to 256 bytes, only one packet is sent. For long messages, the information is sent in multiple packets. Whenever the message size left is bigger than 256, the message header, a packet length byte value of zero, and the 256 bytes of data are sent as a packet. This process is repeated until the value in the Message Length Register is less than or equal to 256, then the last packet is sent with the remain value as the packet length byte.

4.3.2.2 De-packetiser

The function of the De-packetiser is to request DMA transactions to the memory for each incoming message. When a packet arrives and the packet header matches one of the active receiver channel's Message Header Register, the De-packetiser will start extracting the packet and filling the DMA buffers with incoming message bytes from the incoming link FIFO. A DMA request will be made when the DMA buffer is full or when the packet ends.

During idle mode, the control logic waits for the arrival of packets. Once the packet header is ready, it has to be checked to decide the correct operation before proceeding with the packet body. If the header is decoded as a boot header, a boot request is made to the Reset Circuitry for resetting appropriate sections. If not, the header is checked against all active message channels for a match. If a match with the Message Header Register is found, the extraction of the packet and data transfers proceed. If no match was found, notification will be generated to the processor through the Interrupt Controller to signify no available receiving channel for the current packet.

When processing the packet body, after the header extraction, the first byte received is the length of the packet. This value will be used to count the number of data bytes in the current packet. When the packet length counter reaches zero, the state machine returns to header detection. Note that a length value of zero indicates there are 256 data bytes in the packet (undocumented feature of ICR C416). In the case of late termination, as described in Section 4.3.1, a flush command will be given by the corresponding message channel. This command will force the removal of the remaining data bytes from the Receiver FIFO.

4.3.2.3 DMA Buffer

The DMA buffer is the first stage of the link buffer pipeline. Transfer of messages to and from memory is done through a series of small DMA block transfers. This DMA buffer is used to hold the DMA block of data while the communication link continues working on network data transmission to or from the link FIFO.

The SDRAM interface was configured to support a maximum burst size of eight words. Hence, increasing the DMA transfer size up to the burst length can reduce the overhead effect, as shown in Equation 4 in Section 3.6.3. However, a DMA block size of more than 8 words is not advisable as no extra overhead reduction can be seen, but the access time increases. The FLEX 10KA devices provides memory, called Embedded Array Blocks (EAB), but this is not suitable for the DMA buffer implementation. Each EAB is a distinct memory block with a single access port, and they are scarce resources: the FLEX 10KA50V only comes with 10 EABs. In addition, an EAB can only be a maximum of 8 bits wide. To implement a 32-bit wide DMA buffer, 4 EABs would be required. Hence, the DMA buffer was implemented using normal logic resources. The DMA buffer size was chosen to be 1 word, in order to save logic expenses and reduce control complexity.

Each DMA transaction steals memory bus cycles. It is important that the DMA transaction rate does not occupy the memory bus too often, causing overloading. With 36.9 MHz memory bus frequency, the maximum percentage of utilisation of each communication link (2 DMA channels) is 12.52% as calculated using Equation 4. With two communication links implemented, a total of 25.04% utilisation results. However, the maximum utilisation is 50% because of the arbiter design discussed in Section 4.2.1.

4.3.2.4 Link FIFO

The link FIFO is the second stage of the link buffer pipeline. It is designed to provide continuous data feeding to and from the communication network while DMA transaction is being requested or served.

The size of the link FIFO depends on the link speed and the DMA transaction rate. The higher the link speed, the faster the data is coming in, and thus the larger the amount of FIFO size needed. The higher the DMA transaction rate, the smaller the amount of FIFO needed. The deterministic factors for the DMA transaction rate, however, depend on the memory bus speed, the number of competing devices on the memory bus, and the access time allocated for each memory bus device. The longer the time the communication link has to wait for an access, the slower the DMA transaction rate.

In the SARNIC design, only the CPU and the DMA engines of two communication links have access rights to the host memory. Each communication link has two DMA engines. After each DMA access, the memory bus must be relinquished to the CPU if its request is pending. Therefore, the maximum time a communication link has to wait for a DMA transaction, T_{OSL} (in the units of OS Link byte time), with a memory bus frequency of 36.9 MHz, is:

$$
\begin{aligned}
T_{OSL} &= \frac{2\times 2\times T_D + 2\times 2\times T_C(\max)}{\dfrac{13}{20\times 10^6}} \; (\textit{link byte time}) \\
&= \left(4\times\left(\frac{4+DMASize}{36.9\times 10^6}\right) + 4\times\left(\frac{13}{36.9\times 10^6}\right)\right)\times\frac{20\times 10^6}{13} \; (\textit{link byte time}) \\
&= (4+1+13)\times\frac{80}{36.9\times 13} \; (\textit{link byte time}) \\
&= 3 \; (\textit{link byte time})
\end{aligned}
$$

Equation 6: Maximum duration for a DMA operation, units of OS Link byte time.

The time to transfer the data between the link FIFO and the DMA buffer is negligible as it only takes a few clock cycles. Furthermore, since the shift registers in the link transmitter and link receiver can be considered as a single byte storage, this gives extra tolerance to the DMA interval. Therefore, to cover the DMA interval, the link FIFO size should be at least 3 bytes.

For OS Link flow control buffering, as an acknowledge token is returned for each received data byte, an extra one byte link buffer is needed for the acknowledge token. This increases the required receiver link FIFO to 4 bytes. Again, instead of using EABs, the implementation of the small link FIFO utilises the normal logic resources of the FLEX 10KA device.

In order to support the boot from ROM and boot from link feature, the receiver FIFO for Communication Link 0 has two input sources: from the receiver link engine and the internal ROM. This implementation is used to save the logic of interfacing the internal ROM to the SDRAM interface (discussed further in Section 4.3.2.6).

4.3.2.5 OS Link Interface

The OS Link Interface is the data conversion interface between the packet layer and the token layer. It is to handle the low level flow control of the OS Link, transforming bytes from packet layer to a bit stream in token layer, and shifting the incoming bit stream in token layer to bytes in packet layer.

Each OS Link is an asynchronous communication channel, consisting of two unidirectional wires. Data bytes are transmitted by the link as a sequence of 11-bit tokens, each containing a start bit, a type bit of '1', eight data bits, and a stop bit. After transmission, the transmitter has to wait for an acknowledge token to arrive. This acknowledge token is used as flow control to prevent buffer overflow. An acknowledge token will not be sent out if no buffer is available at the receiver. To allow continuous flow of data on the link, the acknowledge token is sent out immediately after decoding the type bit at the receiving end. The OS Link data and acknowledge token format is shown in Figure 27.

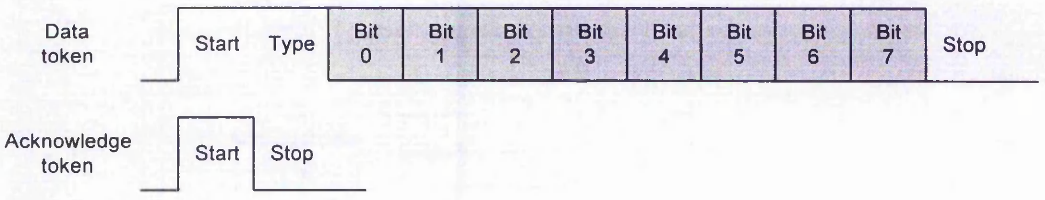


Figure 27: OS Link data and acknowledge token format.

The link engine utilises an over-sampling technique to read data. To overcome the signal skew and the possible variation of bit width on the OS Link, over-sampling is required. Taking an odd number of samples is always better than taking an even number of samples as the odd number sampling offers greater clearance from the ‘middle’ sample, and thus provides greater skew immunity. For example, 3 time over-sampling gives 33% of clearance on both sides of the ‘middle’ sample. 4 time over-sampling however gives 25% on one side and 50% on the other side, depending on whether sample 2 or sample 3 is taken as the ‘middle’ sample. To minimise the sampling frequency required, 3 times over-sampling was applied. For 5 times over-sampling, the clearance is 40%, which is only a 21% clearance improvement for a 67% increase in clock speed. Through the use of both edges of the clock for sampling, the sampling frequency was reduced to 30 MHz.

As the OS Link is an asynchronous channel, metastability can occur during the sampling of incoming bit stream. Metastability is a phenomenon that occurs when data input is changing while a flip-flop is sampling this undetermined data input. Hence, the output at this state can be undefined or oscillating, and can be propagated into the consequent logic, causing system failure. There is no absolute way of removing the problem of metastability, the probability of failure can only be reduced [86, 87]. A two-stage synchroniser has been implemented in the link input sampling circuitry to reduce the probability of metastability failure. At a synchronising frequency of 30 MHz with both phases of the clock, the Altera FLEX 10KA devices can reduce the probability of failure to less than once in 100 years [88].

4.3.2.6 Internal ROM

Other than the boot from link feature, there is a small internal ROM to store the fixed version of boot program. Altera FLEX 10KA devices have embedded RAM cells that can be used to implement RAM, ROM or FIFO. These EAB have been utilised to implement a 2 kB internal ROM, utilising 8 EABs.

When boot from internal ROM option is selected, data bytes in the internal ROM will be accessed using an address counter. In order to save logic, these data bytes are treated as normal OS Links data bytes and transferred into main memory through the Incoming DMA message channel 0. Thus, the format of the boot program stored in internal ROM must exactly match the ICR C416 packet protocol.

4.3.3 Channel Switching and Allocation

The purpose of channel switching and allocation is to allow flexible use of the DMA message channels among two communication links. It is used to allocate DMA message channels to the dedicated communication links. The hardware implementation of the channel switching and allocation reduces the overhead in supporting virtual message channels, and lessens the need of processor intervention in handling virtual message channels.

To match an outgoing message channel to a transmitter link, the 'PHYC' bit in the Length Register has to be written with the intended communication link number, i.e. '0' for communication link 0 and '1' for communication link 1 (see Appendix A for more details). This allows any outgoing message channel to be assigned to any communication link.

When both outgoing message channels are assigned to one particular communication link, hardware virtual channel transmission is in effect. Both message channels are multiplexing the message on the particular communication link in packet level, i.e. each message channel takes turns to send out a packet. The benefit of doing this is that in the case of a short message queued at the back of a long message, the short message has the chance to be sent out in between packets of the long message, reducing unnecessary latency.

To match an incoming message channel to a receiver link, the Header Register has to be written with the identical message header as the one received by the receiver link. Once the Allocater Switch has found a match between the incoming packet header and the Header Register, connection is made until the packet is transferred. Even though writing both incoming message channels with the same message header is unrealistic, in this case, protection has been provided to connect only one of the message channels to the receiver link at a time.

As the ICR C416 is a packet switch, there is a chance of the reception of one message being interfered with by packets from other messages. In this situation, the second incoming DMA message channel can be allocated to the same receiver link for handling the interfering message. This mode is known as hardware virtual channel receiving. Switching between the active DMA message channels is done automatically by the Allocater Switch depending on the incoming message header. To support more than two incoming virtual channels, however, software involvement would be required. One of the DMA message channels has to be freed to handle the packet of the third interfering message, through software means. Current message information will have to be swapped out to a temporary storage and restored at a later point. Nevertheless, the chance of handling more than two virtual channels is far less than the chance of handling up to two virtual channels.

4.3.4 DMA Core

The DMA Core is responsible for arbitration between the DMA message channel transactions and requesting the memory bus access for DMA operations. The progress of arbitration includes selecting the active channel for DMA transaction, multiplexing the selected channel address, and then generating the bus access request to the Bus Controller. The corresponding DMA transaction is then continued when the memory bus is ready. For transactions to the memory, multiplexing of the two incoming message channels data is required, while for transactions from the memory, de-multiplexing of the data to both outgoing message channels is required.

4.4 Control Link

The Control Link is designed specially for the control port of the ICR C416 packet switch. The link protocol used is the OS Link. The module is implemented as a register-mapped interface, where reading or writing to the Data Register will perform a byte pull or a byte push to the Control Link FIFO. This allows a direct interface to the ICR C416 control port through software. The ICR C416 software port commands can be sent as writes to the Data Register, and results can be collected as reads from the Data Register.

The Control Link module was also embedded with an optional mode of automatically detecting blocked links in the ICR C416, and reporting back to the CPU through interrupts. This special feature can be enabled in the Control Register: all accesses to the Data Register are ignored in this mode.

The process of detecting blocked links is achieved through the ICR C416 software port command *09h*, which will return the status of all 16 links. Bit 5 of each status byte indicates the data flow since last status check. Therefore, if one of the links returns a 'no data flow state' for two consecutive status checks, a link block is assumed. The polling rate of the status, however, can be difficult to decide, as the time a link is considered blocked depends on the communication network characteristics. The size, the topology, the routing algorithm, and the load of the communication network can vary the time a message needs to travel from the source to the destination. A starting point of deciding the polling time will be to presume the network consists of a single ICR C416 device. Since each output link of the ICR C416 device can queue up to 3 packets, the worst case queued message delay incurred by the ICR C416 device, $T_{delay}(\max)$, is given by Equation 7.

$$\begin{aligned} T_{delay}(\max) &= 3 \times PacketSize \times TokenTime\ (s) \\ &= 3 \times (PacketHeader + LengthByte + Payload) \times \frac{11}{10 \times 10^{6}}\ (s) \\ &= \frac{3 \times (6 + 1 + 256) \times 11}{10 \times 10^{6}}\ (s) \\ &= 867.9 \times 10^{-6}\ (s) \end{aligned}$$

Equation 7: Worst case queued message delay of the ICR C416.

Based on this calculation, a range of polling rates are offered. The user has the flexibility to select a suitable rate to define the time-out figure of a message. A pre-loadable 4 bit counter, clocked by 4 sets of selectable pre-scale clock, has been implemented for this purpose. This gives a time-out range from 64 μs to 4 s.

Other than connecting this link to the control port of the ICR C416 device, it can also be used for connections to other OS Link compatible device, such as a Transputer. However, in this mode, the user must make sure the special link blockage detection for the ICR C416 is disabled.

4.5 Interrupt Controller

The function of this block is to generate interrupt requests for the StrongARM SA-110 microprocessor, based on the status of the interrupt sources and the enabling state of the interrupt. There are a variety of interrupt sources implemented in the SARNIC design. These include DMA message channels interrupts, a Timer interrupt, Control Link interrupts, UART interrupts, and four additional external interrupts.

The interrupt controller supports two levels of StrongARM SA-110 interrupts: FIQ and IRQ. FIQ is the higher priority interrupt, and the StrongARM SA-110 contains 8 banked registers that allow fast interrupt handler switching. Two different set of registers are utilised for the FIQ and IRQ interrupts, but sharing the same interrupt sources. Therefore, only one type of interrupt should be active for each interrupt source. The interrupt registers have been designed in such a way that enabling one interrupt type will disable the other interrupt type automatically.

For each set of interrupt registers, there are: an internal Enable Register, a write-only Enable Set Register, a write-only Enable Clear Register, and a read-only Status Register. A Raw Status Register is shared by both sets of interrupt register. The Enable Register decides whether a particular interrupt bit is enabled or not. The Enable Set Register and the Enable Clear Register are used to modify the state of the Enable Register. Writing a logic '1' at the appropriate interrupt bit location in the Enable Set Register or the Enable Clear Register will enable or disable the interrupt correspondingly. These two separate write-only registers implementations ease the process of modifying individual bits in the Enable Register, without having to read the Enable Register, mask the corresponding interrupt bit, then write back to the Enable Register. The Raw Status Register displays the status of the original interrupt source. The Status Register is the masked version of the Raw Status Register: it is the bit-wise 'AND' function of the Raw Status Register and the Enable Register.

4.6 Timer

The 32-bit Timer provides a basic wrap-over counter that increases with clock ticks of 1 μs, giving a 1 MHz real-time timer. The use of this Timer is similar to the OCCAM timer function [37]. A special relation operator function decides whether a timer value supplied is before or after the current timer value. The result of the function is available in the Control Register, and an interrupt can be generated if enabled.

The Match Register is used to hold the timer value to be compared with the real-time timer value. The comparison is done with 'wrap around' arithmetic and magnitude comparison, so that if a value loaded into the register is one which has actually just passed, the interrupt occurs at once. To achieve this, the difference between the timer match register and the real time timer must be calculated. The most significant bit of the result is used to decide whether the requested timer event is in the past or future.

With the 32-bit Timer clocked at 1 MHz, the maximum magnitude of a timing event is 2^{32} μs, equivalent to 4,294,967,296 μs. Since the timing event has to be divided into 'past' and 'future', only half of the maximum magnitude is meaningful. This gives a maximum length of timing event of approximately 35 minutes and 47 seconds. Longer timing period can be handled in software without significant overhead.

4.7 UART Communication Port

The UART module design is to allow information display on a monitor and keyboard inputs, through a PC terminal program, by connecting it through the COM port. The UART module implemented here is a reduced function set of a standard UART device. The flow control circuitry has been totally removed. Only the serial communication link operational mode remains. Functions like parity generation and checking, framing error, and buffer overrun error are fully supported. The baud rate selection is reduced to only four options: 9600, 19200, 38400, and 57600 bps. No FIFO has been implemented in this version because of resource saving; this is acceptable, as the communication load is not heavy for development usage.

4.8 Reset Circuitry

The Reset Circuitry is used to generate a set of reset pulses to different portions of the interface chip, and to generate a reset pulse to the CPU. There are two sources that can cause a reset to the system. One is the global system reset switch, and the other is the boot request from the Communication Links.

To support the boot from link feature, the Reset Circuitry has to pass a reset pulse to the whole system when a boot request comes in. However, the reset must be avoided at the communication receiver link that generates the boot request, which will be in the state of receiving the boot message. Therefore, three different reset pulses are generated: one for the chip-wide reset, and the other two are for resetting each communication receiver link individually depending on which link the boot message comes in on.

Other than the internal reset pulses, the chip-wide reset has to be propagated to the StrongARM SA-110 CPU too. During the reset state, the SA-110 CPU will disable the clock output for 150 μs. This will effectively stop any SARNIC logic operations that are driven by the core clock, for that period of time. Hence, the Reset Circuitry logic is driven using the constant 30 MHz crystal clock.

5. SIMULATION, SYNTHESIS AND VERIFICATION

This chapter describes the simulation, synthesis, and verification of the SARNIC device. The design simulations utilised the Mentor Graphics QHDL simulator to interpret the VHDL code behaviour. Following simulation, the synthesis of the SARNIC design into the Altera EPF10K50VRC240-3 CPLD, using the Maxplus II [74] project development platform, was discussed. Finally the SARNIC design was verified through the hardware realisation and testing of a few processor nodes (SARNode). Tests, that were optimised to highlight the raw hardware performance, are presented.

5.1 SARNIC Design Flow

Figure 28 illustrates the design flow adopted for realising the design concepts into an operational CPLD. A top down design approach was used for the SARNIC development, utilising VHDL. Simulations were then carried out for verification of the design functionality. Each individual lower module was manually simulated before combining into higher level modules. At the top-level, the design was again simulated as a complete system, through the use of other interacting component models.

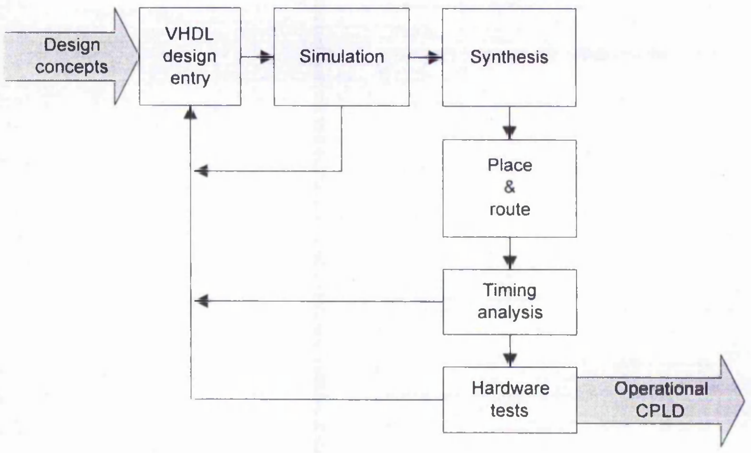


Figure 28: High-level design flow.

The processes of synthesis, place and route, and timing analysis were carried out using the Altera Maxplus II software. The process of synthesis is to generate a list of operational low-level components that are specific to the target CPLD family. These low-level components, known as Logic Elements (LE) in the FLEX 10KA CPLD were then placed and routed automatically on the selected size of device (EPF10K50VRC240-3 in this case). After the place and route process, the design is subjected to a static timing analysis that ensures the required operational speed of the design was met. The device was optimised for performance with smallest area used, by changing the design entry and careful placing of the logic elements. A programming file was also produced for configuring the SRAM based EPF10K50VRC240-3 CPLD.

To save design time, post-synthesis simulation was replaced by hardware tests of the SARNode prototype. Normally, re-simulation of the design after synthesis is required to confirm the correct synthesis process. The Maxplus II software has an option of producing a VHDL VITAL simulation model [74] that contain the timing information. By replacing the original SARNIC VHDL design with the VITAL simulation model, the post synthesis design could be simulated for correct device operation under the same test conditions. However, due to the detailed internal simulation of the VITAL model, long simulation times would be expected. Correct operation results could be obtained faster through hardware tests.

Once the design cycle was complete, the programming file was used to configure the CPLD for hardware operation. Hardware verification tests were carried out for real-time operation validation. In the case of errors found during verification, changes were made easily, following the complete design flow, since the CPLD used is re-programmable.

5.2 SARNIC Simulation

The purpose of simulating the design was to guarantee design functionality before implementation. Also, simulation enabled estimates of device performance to be made. Design simulation for a re-programmable technology like a CPLD is not critical, as the targeted design can be corrected or upgraded without extra cost. Therefore, the simulations generated did not cover 100% of the device functionality. However, finding design problems during simulation is preferable as the simulator has the capability of monitoring all of the internal nodes of the design.

The simulation involved two different approaches: manually generated and model generated. The functional testing of each individual design module was carried out by utilising stimulus from manually generated waveforms, before inserting into a higher module in the design hierarchy. The tests were manually generated, as it was easier to generate the direct test waveforms than to create a model that would be significantly changed during the initial design phase. The simulation cycle followed a bottom-up methodology until it reaches the top-level design module.

The complete SARNIC design was then simulated with other interacting component models like the SA-110 model, the SDRAM model, and the OS Link interface model. Due to difficulties in getting the industrial standard models in VHDL code, all the models have been produced for this work to emulate each device. From the emulation models, a processor node model (SARNode model), was constructed. This is shown in Figure 29. Multiples of this SARNode model were connected to an ICR C416 router model, to perform different simulations of a complete distributed system model.

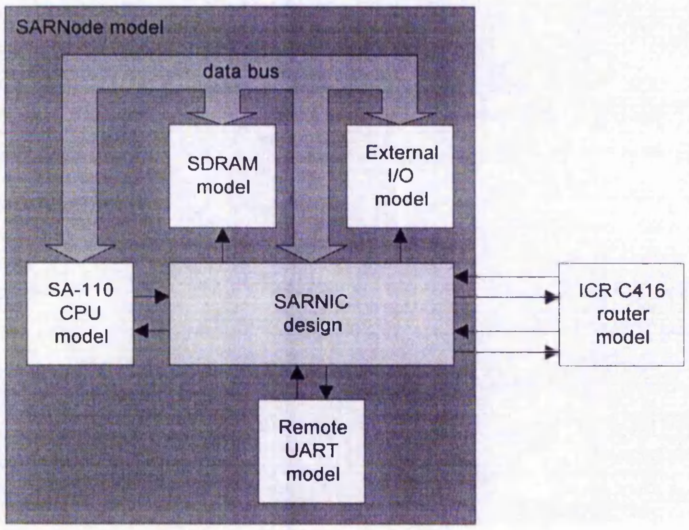


Figure 29: Simulation configuration of the SARNIC design model.

The individual emulation modules are described below:

SA-110 CPU model

This is the core of the model generated simulation. It emulates the CPU, executing instructions and issuing memory accesses to the SARNIC chip. To simplify the emulating function, only the bus transaction initiator was designed. Instructions are entered as memory requests with specific memory access type, address, and data for storing. Like the actual SA-110 CPU, the instruction execution is based on three priority levels from the highest to the lowest: FIQ mode, IRQ mode, and supervisor/user mode. This allows the emulation of interrupt service routines.

SDRAM model

The model emulates SDRAM functions following a reduced set of its standard state machine. Only the functions used by the SARNIC design were implemented, for instance refresh, mode register set, all bank precharge, and normal read/write cycles.

ICR C416 router model

The model was written as a synthesisable 4-link ICR C416 model. Only the basic direct routing function with queuing ability was implemented. The communications channels are non-blocking.

External I/O model

This model simulates the function of a slow memory peripheral. For a write access, it detects the active-low write pulse and writes the data into an internal register. For a read access, it detects the active-low read pulse and outputs data after a delay of 200 ns.

Remote UART model

This model contains a UART link interface. It waits for an incoming byte, and sends it back at a later time.

5.2.1 Functional Tests

The functional tests described here were carried out with a single SARNode model connected to an ICR C416 router model. These tests give an overall coverage of all the functions provide by the SARNIC design during 'realistic' operating conditions. The following is the list of functional tests executed through issuing memory accesses from the SA-110 CPU model:

- Testing the register access – writes followed by reads of some registers, to ensure the accessibility of the SARNIC registers. This is important since all the rest of the tests involved register access.
- Testing the SDRAM operation – writes to a block of SDRAM region, followed by verification of the content after some time. This was to test the correct SDRAM access and the SDRAM refresh cycles were working properly.
- Testing the external I/O access – a write followed by a read to the emulated external I/O model.
- Testing interrupt generation – generating an interrupt from emulated external I/O model to test the correct response from the SA-110 CPU model. The CPU model would go into an interrupt service mode to clear the interrupt source.
- Testing the timer – initialisation of a timer event by writing to the timer register and waiting for interrupt generation. This would be followed by clearing the timer interrupt.
- Testing the UART link – checking correct UART transmission under various baud rates and parity checking, with the help of the remote UART link model.

- Testing the Control Link – checking the automated link time-out error generation circuit, with the help of the ICR C404 model.
- Testing Message Channel communication – checking the message transmit and receive channels by initiating messages designated to the same SARNode, via the ICR C416 router model. Multiple simultaneous message transmissions were also simulated.

5.2.2 Estimated Design Performance

The simulation tests were configured with two SARNode models connected through an ICR C416 router model. The model simulations provided a means of measuring the performance figure for message transfer latencies and bandwidth. This allowed the device parameters to be recorded before design synthesis was carried out, thus allowing any final enhancements to be undertaken. The simulation also tested the correct operation of the design with varying message sizes.

The performance tests carried out were based on round-trip message time calculations. Model SARNode 1 begins by initiating a message transmission to model SARNode 2. At the end of the message reception on model SARNode 2, it initiates the same message transmission back to model SARNode 1. The time when model SARNode 1 has completely received the message was measured, and thus the round trip time was calculated.

Both 10 Mbps and 20 Mbps OS Link communications were tested. Different configurations of uni-directional, bi-directional, and multiple message DMA operations were also carried out. The message size used was varied from 1 byte to 512 bytes, with a fixed single byte of routing header and a single byte of message header. No transmission line delay was simulated in these tests. Complete plots of the message time and bandwidth versus packet size are shown in Figure 30, Figure 31, Figure 32, and Figure 33. Detailed results are given in Appendix E.

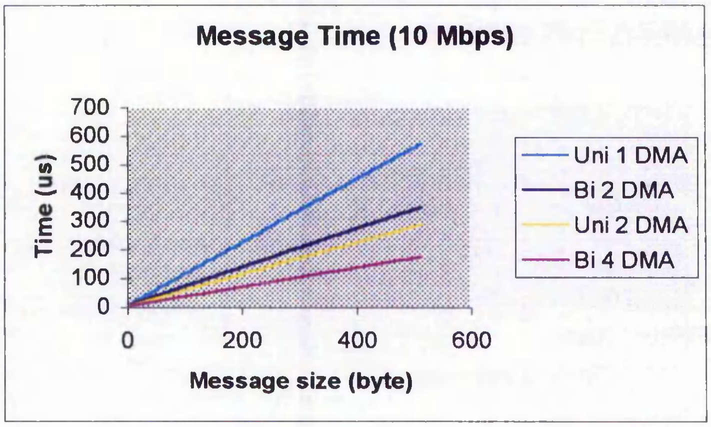


Figure 30: Message time simulation results for 10 Mbps link.

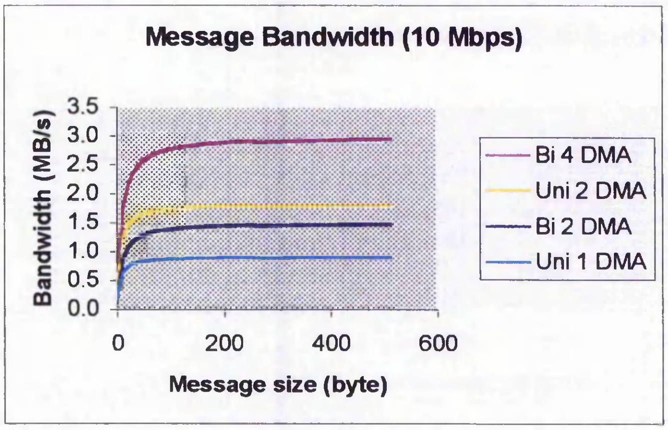


Figure 31: Message bandwidth simulation results for 10 Mbps link.

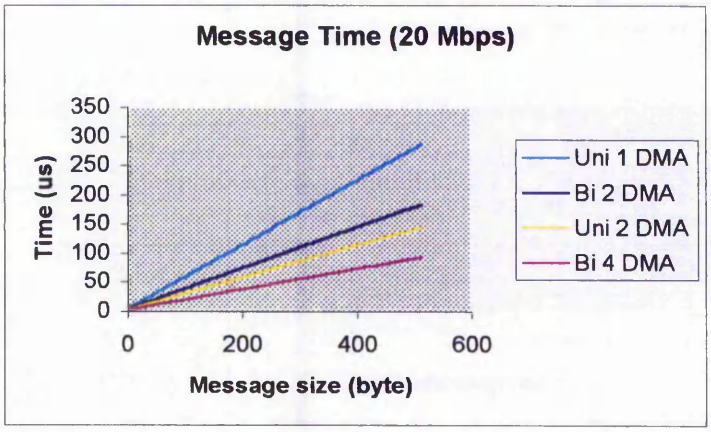


Figure 32: Message time simulation results for 20 Mbps link.

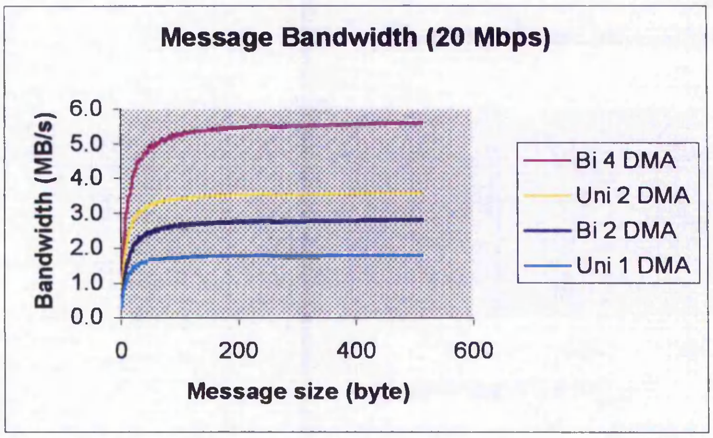


Figure 33: Message bandwidth simulation results for 20 Mbps link.

From the results, it can be seen that the message time incurred is proportional to the size of the message. This means the message overhead stays constant across varying message sizes. These overheads include the time of sending: the routing header byte, the message header byte, the length byte; and the time of message handling incurred by the router and the network interface. As 3 extra bytes are added for each packet, a small 'bump' can be seen at message size of 256 bytes on the message time graph. This happens at every packet boundary. By deducting the time spent on the 3 overhead bytes, the message handling latency is found to be a constant, equal to approximately 2.1 μs at 10 Mbps and 1.8 μs at 20 Mbps.

From the bandwidth graphs, the results show that as the message size increases the effective bandwidth increases, reducing the overhead effect of the routing information, which must accompany each packet. The step reduction in effective bandwidth due to the overheads of the second packet is visible at the 256-byte boundary. The graphs also show that the bandwidth is nearly saturated when the message size is over 64 bytes. At 10 Mbps, uni-directional data achieves a bandwidth of 0.9 MB/s, effectively 72 % of the link raw bandwidth. Bi-directional data only achieves 58.8 % (1.47 MB/s) of the link raw bandwidth; this is due to the penalty of sending an acknowledge token per data token. At 20 Mbps, an uni-directional bandwidth of 71.6 % (1.79 MB/s) and a bi-directional bandwidth of 56 % (2.8 MB/s) were recorded.

To further analyse the components that contribute to the message latency, the time needed for one particular message transmission was broken down into different sections as shown in Figure 34.

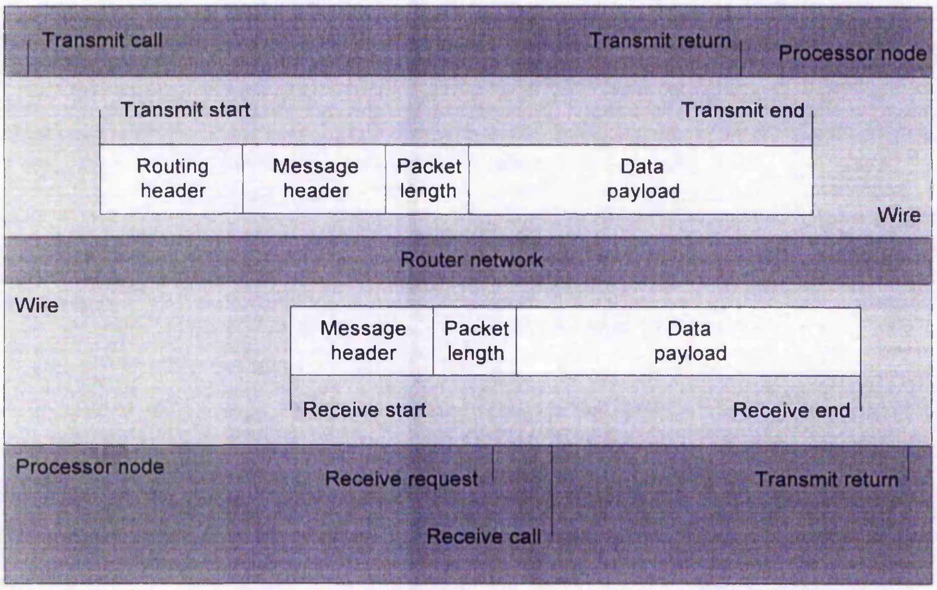


Figure 34: Communication transmission time break down.

The explanation of the break-down points are listed below:

- *Transmit call* – the write of the Transmitter Length Register initiates DMA channel transmission.
- *Transmit start* – the first bit of routing header appears on the wire.
- *Transmit return* – the last byte of the message has been pushed to the link FIFO and a *DONE* interrupt is generated to the SA-110 CPU.
- *Transmit end* – the last bit of the last data byte is sent.
- *Receive start* – the first bit of the message header appears on the wire.
- *Receive request* – the decoding of the message header generates an interrupt request to the SA-110 CPU if no input message channel is ready.

- *Receive call* – the write of the Receiver Length Register sets up the receiving DMA channel.
- *Receive end* – the last bit of the last data byte is received.
- *Receive return* – the last byte of the message is transferred to memory and a *DONE* interrupt is generated.

The transmission starts by setting up a DMA transmit channel. *Transmit call* is the point where the write of the Transmitter Length Register initiating the transfer. The first bit of the routing header then appears on the wire, as noted by *Transmit start*. The router network extracts the routing header and forwards the rest of the packet to the output. At the end of the DMA transmit channel transfers, the SARNIC device generates an interrupt request to the SA-110 CPU, noted by *Transmit return*. *Transmit end* is the point where the last bit of the last data byte appears on the wire.

At the receiver end, *Receive start* is the point where the first bit of message header appears on the wire. If no incoming message channel was activated, the SARNIC generates an interrupt request to the SA-110 CPU, as noted by *Receive request*. The processor then responds by setting up the receiver DMA channel. *Receive call* is the point where the write of the Receiver Length Register initiating the transfer. When the last bit of the last data byte has entered the processor node, as noted by *Receive end*, the SARNIC generates an interrupt request to the SA-110 CPU, as noted by *Receive return*.

A capture of the simulation waveforms at 10 Mbps and 20 Mbps for a message size of 4 bytes is given in Figure 35 and Figure 36 respectively. These waveforms show that the latency of sending a message from the point the channel is activated (*Transmit start* – *Transmit call*) is 199 ns at 10 Mbps and 206 ns at 20 Mbps. At the receiver node, the latency of generating the interrupt from the point the last bit of the message is received (Receive return – Receive end) is 153 ns at 10 Mbps and 234 ns at 20 Mbps. No receive request interrupt was found as the message channel is setup before the message arrives. The router latency is 1.1 μs at 10 Mbps and 616 ns at 20 Mbps. From the previous message time simulation, the message handling latency was 2.1 μs at 10 Mbps and 1.8 μs at 20 Mbps. By subtracting out the network interface and router hardware latency, this leaves an emulated software latency of

648 ns at 10 Mbps and 744 ns at 20 Mbps for setting up the DMA channels. This software latency will increase accordingly in real system, depending on the complexity of the software.

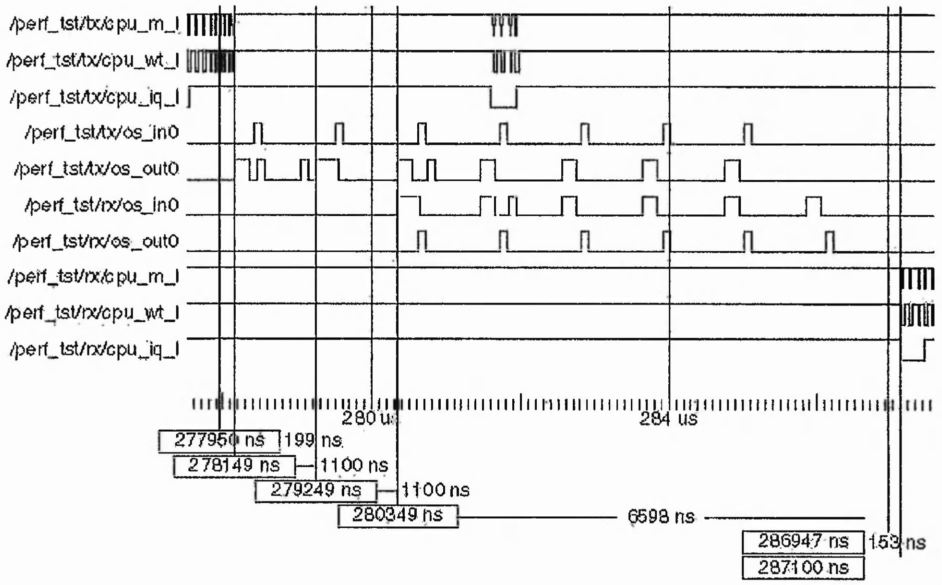


Figure 35: Simulation waveform for a 4-byte message at 10 Mbps.

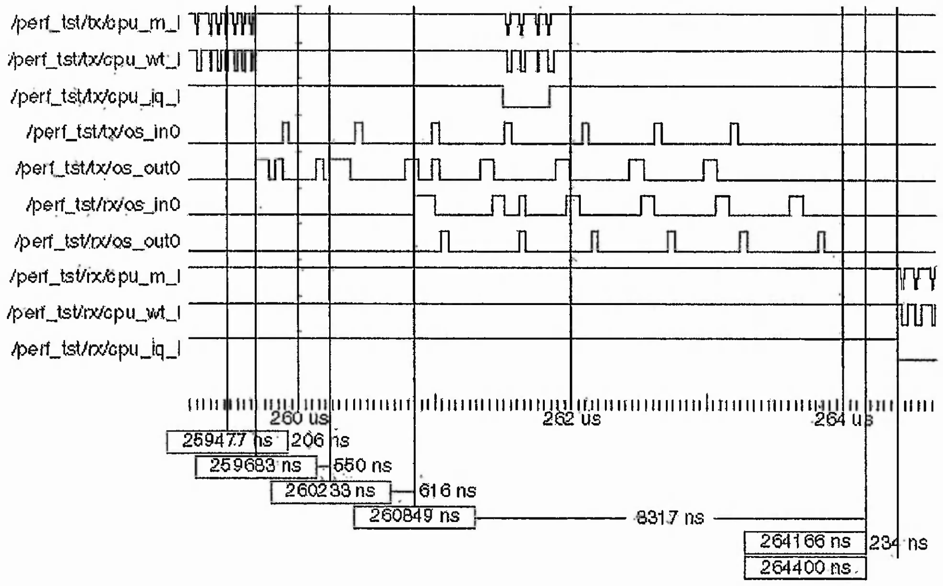


Figure 36: Simulation waveform for a 4-byte message at 20 Mbps.

5.3 SARNIC Synthesis

The next stage after the functional simulation was to synthesise the functional Register Transfer Level (RTL) VHDL code into a net-list of low-level operational components for the Altera CPLD place and route tool. VHDL code synthesis is different for schematic entry where schematic entry uses standard parts like flip-flop or multiplexer modules that synthesise directly into the corresponding logic. VHDL code on the other hand is a hardware description code, which describes the behaviour of the flip-flop or multiplexer function. Hence, the code has to be carefully written so that the synthesiser can interpret the functional behaviour into correct logic. Also, depending on how sophisticated the synthesiser is, VHDL code might be required to be written as a very low level, detailed description, instead of in an 'easy-to-understand' high level.

Two types of optimisation were used to ensure the synthesised VHDL code meets the design specification, these are: area optimisation and speed optimisation. Optimisation requires an understanding of the target technology architecture, how to use its dedicated special resources, and how to achieve efficient synthesis with a VHDL code format that suits the compiler/synthesiser. Area optimisation was carried out first. Through logic reduction, signal path lengths are reduced, and thus some speed optimisation would normally be achieved indirectly. However, this is not always the case, for instance, speed improvement might only be possible by replicating common logic for two different partitions.

5.3.1 Area Optimisation

Area optimisation is related to the efficiency of the Altera Maxplus II compiler/synthesiser. From experience of the feasibility study of the first version of SARNIC implementation, The Maxplus II software could not interpret high-level behavioural writing style of the VHDL code precisely and efficiently, usually resulting in more logic than expected. The Maxplus II software might even result in incorrect functional logic due to some unsupported VHDL features. Therefore in the feasibility study, there was much familiarisation with the VHDL code writing style for correct and efficient Maxplus II synthesis. The process involved porting the high-level code to a low-level code interpretation to achieve optimum results, including consideration of modularisation of the logic functions down to the individual CPLD logic elements.

Besides modifying VHDL code, area optimisation could also be achieved by utilising the special resources of the target CPLD. Obvious examples of this are the use of the carry chain and the cascade chain of the FLEX device family. The carry chain is especially useful in implementing counters or adders/subtracters, as it provides the carry forward function in the same logic element and a dedicated chain which links to the next logic element. This reduces logic and improves carry forward speed. The cascade chain on the other hand is useful for wide-input 'AND' gate or 'OR' gate implementations. It provides an additional 'AND' function fan in a chain from the previous logic element[3]. In this way, a portion of a wide-input 'AND' gate (in one logic element) can be fanned in directly to the next portion of the wide-input 'AND' gate (in the next logic element), without using an extra 'AND' gate. The cascade chain is also valuable for multiplexer implementation [72].

Another resource that can be used to reduce logic is the enable input of the register. Consider the following VHDL code implementation of the register:

```
if reset = '1' then
    q <= '0';
elsif clk'event and clk = '1' then
    if enable = '1' then
        if sel = '1' then
          q <= a;
        else
          q <= b;
        end if;
    else
        q <= q;
    end if;
end if;
```

If the code is written as above, the function requires 5 variable inputs, resulting in 2 logic elements used, since a logic element can only support a 4 variable input. By

[3] OR gate functions are achieved through De-Morgan conversion.

utilising the enable input pin, the variables *enable* and *q* are removed from the function. Hence, only 1 logic element would be required. The code becomes:

```
if reset = '1' then
    q <= '0';
elsif enable = '0' then
    q <= q;
elsif clk'event and clk = '1' then
    if sel = '1' then
        q <= a;
    else
        q <= b;
    end if;
end if;
```

5.3.2 Speed Optimisation

Speed optimisation is related to careful layout of the design on the CPLD. For the high-performance FLEX 10KA family CPLD, the general interconnect delay is greater than the delay of the logic cell. Therefore, for a signal path that feeds through multiple logic cells, the physical placement of these logic cells on the CPLD will affect the delay from the first logic cell output to the last logic cell input. To minimise the routing delay effect, the top-level modules of the SARNIC design were assigned to dedicated rows. This decision was made based on the fact that the longest delay between two logic cells are when they are located in different rows, where the signal travels through a column interconnect and two row interconnects before fanning into the destination logic cell. By fitting each module in a single row (if possible), the intra-signal delays are localised to only row interconnects.

In the case where the interconnect delay is critical, the related logic cells should be kept as close as possible to each other. The Maxplus II software offers this function with the 'clique' assignment. Logic cells that are assigned to a 'clique' will be grouped closely together during the Maxplus II fitting process. However, for designs that occupy more than 80% of the CPLD, 'clique' assignments are often ignored by the Maxplus II compiler due to the increase complexity in routing the logic cells. For more secure logic group placement, the critical path logic cells could be forced into certain LABs using absolute assignments.

Utilising target device special resources are another way of optimising the design operation performance. As mentioned previously, the carry chain improves the performance of a counter implementation. However, too long a carry chain is not recommended as the total carry signal delay is much greater than the delay of the conventional carry signal generation method. A long carry chain also reduces the flexibility of the other logic cells and routing resource usage as the logic cells used for the carry-chained counter are fixed in a predefined manner. Nevertheless, an implementation of a 32 bit carry-chained counter still operates faster than a conventional 32 bit counter implementation. As for the cascade chains, the dedicated connection allows faster wide-input 'AND' or 'OR' functions. Again, similar to the carry chain, too long a cascade chain will cause larger delay than the conventional routing interconnects. From experience, a cascade chain that is more than 4 logic cells is no longer beneficial.

5.3.3 Synthesis Results

With the combination of area optimisation and speed optimisation, a successful fit of the SARNIC design was produced. The CPLD fit utilised 85% of the logic cells, 80% of the memory bits, and it is capable of operating at the bus frequency of 39.37 MHz. A breakdown of the synthesis results is detailed in Table 4. Both the resource usage and operating frequency of each module are listed.

There are two clock domains in the SARNIC design. The major domain is driven by the memory bus clock, which is fed from the SA-110 CPU. The minor domain is driven by a constant 30 MHz crystal oscillator. The 30 MHz clock is required for precise sampling of the 10/20 Mbps OS Link operation. This clock is also utilised for generating constant Timer clock tick and UART baud rate.

Generally, most of the blocks in the SARNIC design operate within the region of 40 MHz to 50 MHz. No specific bottleneck was found. The largest block of the SARNIC design is the Communication Controller, which occupies about 56% of the 50 K gates CPLD.

Module	Area Logic Cells (LC)	Area Memory Bits	Speed (Bus) Frequency (MHz)	Speed (Link) Frequency (MHz)
SDRAM Interface	81		48.07	125
XIO Interface	44		74.62	
Register Interface	21		125	
Bus Core	153		72.46	
Bus Controller	304		43.1	125
OS Link Engine	58			38.75
ROM Interface	13	16384		45.45
Transmitter FIFO	38			83.33
Receiver FIFO	64			51.81
Link Transmitter	127		59.52	
Link Receiver	88		52.35	
Link Controller (x2)	381		49.01	37.87
DMA TX Channel (x2)	102		54.94	
DMA RX Channel (x2)	73		53.19	
DMA Core	131		42.55	
Switch & Allocator	210		65.35	
Communication Controller	1602		40	36.76
Control Link Engine	58			38.75
Control Link RX FIFO	14			59.52
Control Link Transmitter	16		106.38	
Control Link Receiver	16		106.38	
Control Link Timer	75		59.52	64.93
Control Link	208		59.52	35.97
UART Core	37		125	42.73
UART Transmitter	45		59.52	79.36
UART Receiver	49		59.52	87.71
UART	140		59.52	40.98
Interrupt Controller	97		64.93	
32-bit Timer	115		47.16	125
Reset Circuitry	16			80.64
SARNIC	2444	16384	39.37	35.71

Table 4: Resource allocation and performance of SARNIC.

5.3.4 Pin Assignments & Floor-Plan

The output of a logic cell in the FLEX 10KA CPLD can be connected to both column interconnects or row interconnects directly. Therefore, both the row pins and column pins can be accessed directly from the output of a logic cell. Both the row pins and column pins are suitable to be used for output pin assignments. However, as the column interconnects are normally shorter than the row interconnects, the delay will be shorter. Column pins are thus slightly better as a fast output.

The input of a logic cell can only be connected to row interconnects. Signal input from the column pins have to travel through a column interconnect and switch to a row interconnect before reaching the designated logic cell. Hence, column pins are generally not suitable for input assignments in the sense that they require longer input set-up time and use up additional column interconnect resource. However, there is an exception, where if the input fans out to many logic cells on different rows, column pins are advantageous as inputs. This is because signals from row input pins in this case will have to travel through two row interconnects and a column interconnect before fanning into the logic cells located at different rows.

Although the first version of SARNIC design can generally operate at the frequency of near 40 MHz, the SARNode performance drops to approximately 33 MHz due to external timing requirements. The input setup time and output delay time of the SDRAM data mask signals are the critical SARNode paths. The output delay of the address and control signals from the SA-110 is already 9 ns, leaving very little slack for the propagation delay time through the SARNIC design and the setup time for SDRAM inputs.

Enhancements were made in this, the second version of the SARNIC, to remove the critical paths. By enabling the address pipeline of the SA-110 CPU, the address signal outputs are available half a clock cycle earlier than the memory request signal. With latches that allow signal propagation during the first half clock cycle and the signal latching on the other half, pipelines are formed, which provide one and a half clock cycle of effective signal propagation slack time. This is illustrated in Figure 37, where the output is half a clock cycle delayed. The enhancement pushes the SARNIC device operation back to 38 MHz. At a CPU core clock speed of

221.3 MHz, the nearest step of bus clock is 36.9 MHz. The next higher step is at 44.3 MHz, which would require extra optimisation on the SARNIC design, both internally and externally.

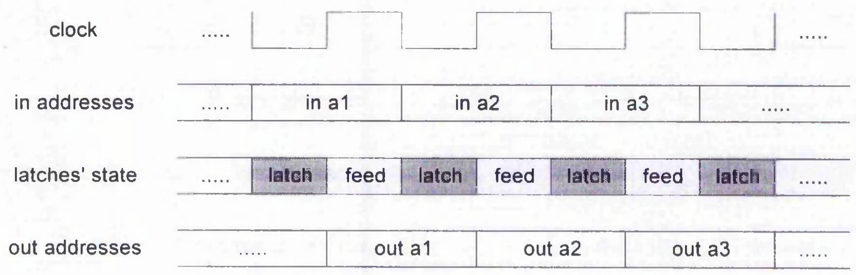


Figure 37: Address pipeline latches operation.

Improving the performance of the SARNIC design can be achieved by using a faster speed grade of CPLD. However, this is only effective for pushing the internal operating frequency, not for shorten the external pins input output delay. This is because the improvement on speed grade is achieved through faster logic cell operation, not by the shorter routing interconnect delay. The pin assignments for the first version of the SARNIC design were decided without much of the above floor plan considerations. They were placed for easy routing of the SARNode PCB. Therefore, this produced an extra constraint on the second version of the SARNIC design. The routing to the input output pins could not be changed much, limiting the optimisation to the input output delay that can be made. Without changing the pin assignments, significantly improving the performance of the SARNIC design is difficult.

5.4 SARNet System Integration and Verification

Following the successful design fitting, initial work on the hardware was centred on the design and production of a processor node. This consists of three major components: the StrongARM SA-110 microprocessor, the SDRAM module, and the SARNIC CPLD, which were integrated on a PCB for the processor node implementation. Multiples of this PCB were then connected to a ICR C416 routing switch to construct a basic StrongARM-ICR C416 distributed processing network, namely the SARNet. The ICR C416 packet routing switch is located on the B816 Transputer-ICR C416 ISA board [89], which is plugged into an AMD K6 233 MHz, 128 MB memory PC system. The same PC system was used as the host to develop the SA-110 machine codes, to download the task to the SARNet, to collect result from the SARNet, and also to act as the I/O console for the SARNet through the COM port of the PC.

5.4.1 The Processor Node - SARNode

A standard Eurocard size, four-layer PCB was designed and manufactured for the processor node implementation. This complete processor node hardware provided a method of verifying the SARNIC device functionality and a platform for software development. A block diagram of the main system components implemented on the prototype board is given in Figure 38. Figure 39 shows the photograph of the SARNode prototype PCB.

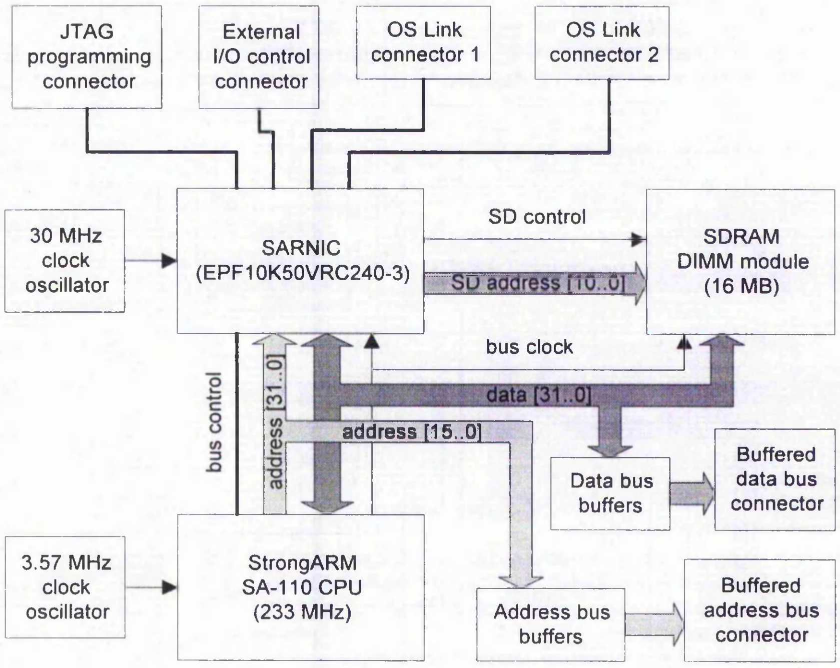

JTAG programming connector
External I/O control connector
OS Link connector 1
OS Link connector 2
30 MHz clock oscillator
SARNIC (EPF10K50VRC240-3)
SD control
SD address [10..0]
SDRAM DIMM module (16 MB)
bus clock
bus control
address [31..0]
data [31..0]
address [15..0]
Data bus buffers
Buffered data bus connector
3.57 MHz clock oscillator
StrongARM SA-110 CPU (233 MHz)
Address bus buffers
Buffered address bus connector

Figure 38: Block diagram of the SARNode PCB.

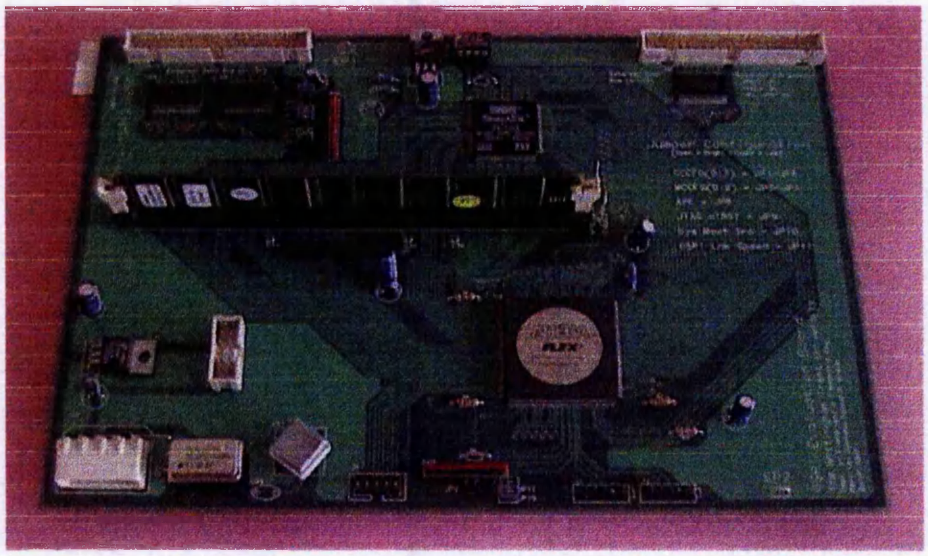


Figure 39: Photograph of the SARNode prototype PCB.

The SA-110 CPU generates the core clock from an external clock oscillator of 3.57 MHz. The bus clock, to which all other components of the system are synchronised to, was generated by the SA-110 CPU core clock. This method provides the system with a simple and low-cost bus clock solution, but the bus clock is then limited to certain frequencies in steps relative to the processor core clock. With this configuration, the SA-110 CPU core can operate at any of the 12 clock frequencies ranging from 87.5 MHz to 221.3 MHz, whilst the bus peripherals operate by dividing the core clock with a divider of 1 to 9. The core frequency of 221.3 MHz and the bus frequency of 36.9 MHz were selected. A separate 30 MHz clock oscillator is provided on the processor node PCB for OS Link operations and other functions that require a constant clock, such as the 32-bit timer and UART link.

The SARNIC design has provided the processor node two options of booting up the SA-110 CPU. The first method is to boot up via the OS Link through the ICR C416. On power up or hardware reset, the processor is released from reset but held in a stalled state until the incoming boot message has been completely copied to the SDRAM. In normal working conditions, receiving a boot message will reboot the processor in the same way. The second method utilised a small boot program that is stored in the internal ROM of the SARNIC CPLD. It is similar to the first method, except that the data is copied from the SARNIC internal ROM, and can only occur on power up or hardware reset. This method is targeted for use as a boot loader that copies the executive programs from an external I/O device, which could be a ROM.

5.4.2 The B816 Transputer-ICR C416 ISA Board

The ICR B816 is the first board developed by IC-Routing for use with Transputer systems [89]. The B816 is a motherboard providing direct connection of up to 8 TRAM boards to the ICR C416 router. This board plugs into the ISA slot of a PC and can be used with any Transputer development tool kit.

To keep the prototyping cost low, standard device packages were used for the SARNIC CPLD and the SDRAM modules. The prototype SARNode PCBs are standard Eurocard size, four-layer PCBs, and hence can not fit into the existing TRAM slots on the B816 board. Hence, connections to the B816 board were made through the eight external OS Links connections through the DB37 pin connector at the rear of the PCB. Twisted-pair cables were used to link this connector to the SARNode PCBs.

5.4.3 The High Performance RS485 Differential Transceiver Circuit

As the host PC is separated from the SARNodes by up to 2 metres, the OS Link communication through external cables was only reliable on 10 Mbps operations (problems were only observed for 20 Mbps operations). Hence, to retain 20 Mbps operations, external cable connections utilising the RS485 differential transceiver circuit, as studied in Section 3.6.4, were utilised. The transceiver circuits were built as daughter boards, sitting on top of the SARNode PCBs. However, the transceivers for the B816 board have to be placed outside of the host PC. Connections were made using RJ45 socket and CAT 5 UTP cables.

5.4.4 Functional Verification

A series of tests were generated for the SARNet system to validate hardware functionality. The test codes were written using the ARM instruction set and the ARM Debugger Software Development Kit was utilised for test code development. All the simulation tests were repeated in the hardware verification by porting the same test functions into the SA-110 instruction codes. This was followed by second phase of hardware testing with more thorough tests for verification of each device function. The overall hardware tests include:

- Testing the register access – writing to the SDRAM Timing Register to configure the correct operation mode and access cycle length.
- Testing the SDRAM access – writing to the complete SDRAM region with different patterns, followed by verification of the content.
- Testing the external I/O access – using a buffer and switch circuit to test the external I/O read write accesses.
- Testing interrupt generation – using the buffer and switch circuit to generate an interrupt.
- Testing the UART link – connecting the UART link to the COM port of a PC and testing it under various baud rates and parity checks.
- Testing the Control Link – enabling the automated link time-out function and intentional blocking of a communication link to emulate a link failure.
- Testing Message Channel communication – initiating message transfers across the ICR C416 router using different configurations and message sizes.

5.4.5 Initial Performance Results

After the functionality of the SARNIC CPLD was confirmed, a series of performance tests were carried out. These tests were written in a way that the software overhead was minimised, in order to highlight the raw hardware performance. The tests ran on prototype SARNodes, with a bus frequency of 36.9 MHz and the SA-110 core frequency of 221.3 MHz.

5.4.5.1 Communication Performance

The first communication performance test was the round-trip time calculation, as carried out in the initial simulation, described in Section 5.2.2. Two SARNodes were connected to the ICR C416 router on the B816 board. The OS Link communications were tested with 10 Mbps and 20 Mbps operation. The message size was varied from 1 byte to 512 bytes. Additional performance test results for simultaneous two OS Link operation were also given. Complete plots of the message time and bandwidth results are shown in Figure 40, Figure 41, Figure 42, and Figure 43.

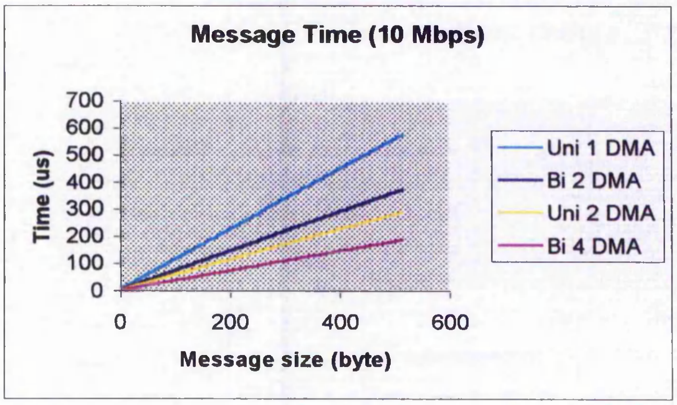


Figure 40: Message time results for 10 Mbps link.

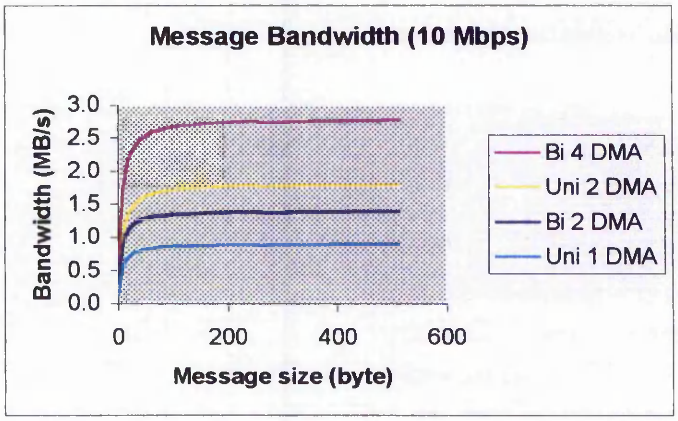


Figure 41: Message bandwidth results for 10 Mbps link.

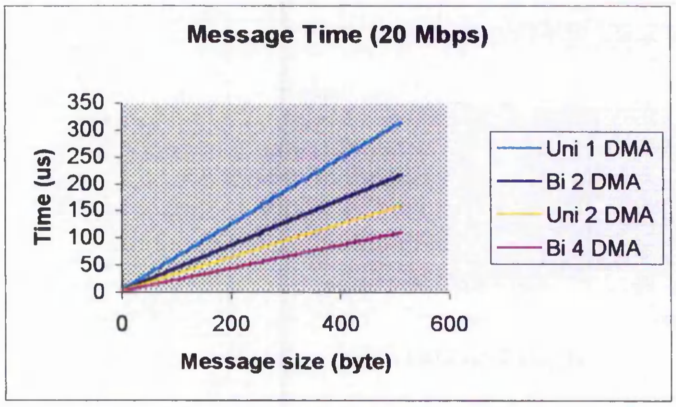


Figure 42: Message time results for 20 Mbps link.

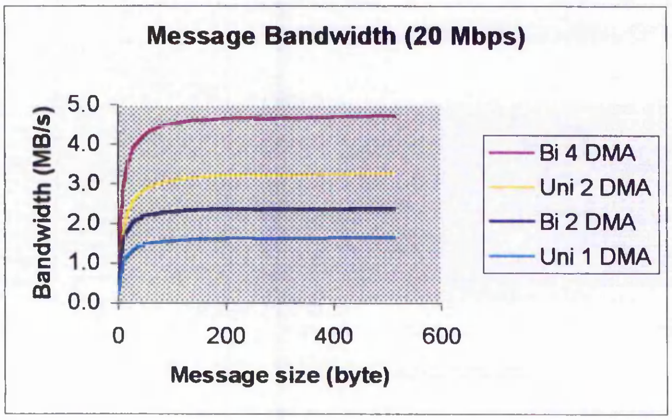


Figure 43: Message bandwidth results for 20 Mbps link.

The results obtained generally closely matched the simulation. However, there were some interesting minor differences relating to performance of the ICR C416 router. Figure 44 shows the result comparison between simulation and hardware tests for the 10 Mbps link: there was no noticeable difference between the simulation and hardware test for uni-directional transfers, other than a slight increase in the real case message handling latency (mainly due to the delay incurred by software processing of the messages). However, for bi-directional transfers, the hardware test graph was steeper than that of the simulated graph. The average time of a byte transfer on the link was higher than expected. In practice, the implementation of the OS Link protocol in hardware could not achieve 13 bits per byte bi-directional data transfers. This problem is not uncommon: devices such as the Transputer can demand up to 17 bits per byte for bi-directional data transmission [21]. In addition, the propagation delay of the transceiver chips (and to a lesser extent the wires) worsens the problem.

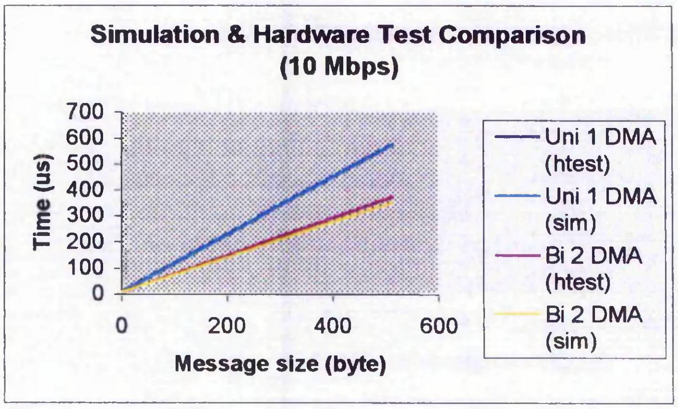


Figure 44: Simulation and hardware test result comparison for 10 Mbps link.

Figure 45 shows the result comparison between simulation and hardware tests for the 20 Mbps link. In this case, the hardware test graphs were both steeper than the simulated graphs, signifying the fact that the average byte transfer time was greater than expected. For uni-directional transfers, the ICR C416 hardware implementation requires 12 bits for sending a data token, as opposed to the 11 bits in the simulation. For bi-directional transfers, the situation is even worse for the same reason as in 10 Mbps link results.

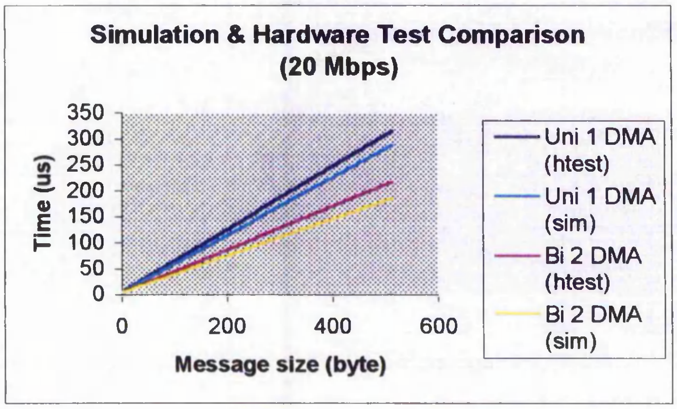


Figure 45: Simulation and hardware test result comparison for 20 Mbps link.

Table 5 summarises the comparison of performance figures between simulation and hardware tests.

Performance Figures	Simulation	Hardware Tests
Message handling latency (10 Mbps)	2.1 μs	3.08 μs
Message handling latency (20 Mbps)	1.8 μs	2.87 μs
Maximum bandwidth (10 Mbps uni-directional)	0.9 MB/s	0.89 MB/s
Maximum bandwidth (10 Mbps bi-directional)	1.47 MB/s	1.38 MB/s
Maximum bandwidth (20 Mbps uni-directional)	1.79 MB/s	1.63 MB/s
Maximum bandwidth (20 Mbps bi-directional)	2.8 MB/s	2.37 MB/s

Table 5: Performance comparison between simulation and hardware tests.

5.4.5.2 Overloading Effects of Communication on Computation Performance

A major aim of a communication system is to provide overlapping of communication and computation. Data transfers should be carried out with minimal overhead to the processing entities. To analyse this communication overhead, a standard benchmark program was used to gauge the performance of the processor during the transfer of data. The Dhrystone MIPS benchmark program was used, as quoted in the SA-110 CPU data sheet. This benchmark program is a short synthetic program written to represent system integer programming.

The source code for Dhrystone MIPS version 2.1 (dated May 1988) was used to build the benchmark process, and was compiled with the ARM C compiler 4.90 (optimisations turned off). Message transfers were carried out as a background process during the entire Dhrystone program execution. The message size selected was 256 bytes; this was an attempt to resemble 'real case' communications. Use of even larger message sizes could have been considered. This would minimise the number of interrupts over the test period, which would emphasise the efficiency of the arbitration between the DMA channels and processor rather than the software overhead. Table 6 shows the resultant Dhrystone MIPS performance ratings of the SARNode computation and the throughput of the communication subsystem.

Message activity	Dhrystone rating (MIPS)	Performance drop (%)	Interface throughput (MB/s)
No transfer active	149.7	0	N/A
1 DMA channel active 10 Mbps (unidirectional)	148.6	0.73	0.91
2 DMA channel active 10 Mbps (uni-directional)	148.1	1.07	1.84
4 DMA channels active 10 Mbps (bi-directional)	146.7	2	2.86
1 DMA channel active 20 Mbps (uni-directional)	148.3	0.94	1.68
2 DMA channels active 20 Mbps (uni-directional)	146.6	2.07	3.36
4 DMA channels active 20 Mbps (bi-directional)	144.7	3.34	4.9

Table 6: Measured Dhrystone figures and interface throughput.

The results show the integer performance decreases by only 3.34% with four active DMA channels continually operating at the maximum bit rate of 20 Mbps. Comparing the interface throughput figure with the maximum bandwidth results from previous communication performance tests (with no computation), the bandwidth results were slightly higher even though the maximum message size was not used. This shows that transfer rates are maintained with active processor computation.

6. DISCUSSIONS, CONCLUSIONS AND FURTHER WORK

This chapter begins with a discussion on the SARNet distributed processing system, the SARNIC interface core design, and areas of interest within them. The benefits of the memory bus based network interface controller and associated features are detailed and compared with existing systems. This is followed by the conclusion of the thesis, summarising the work and the original contributions made. Finally, some comments are given on possible further work.

6.1 Design Discussions

Recent work on parallel system architectures have emphasised incorporating low-cost, high performance, and rapidly advancing microprocessors, as discussed in Chapter 1 and further described in Chapter 3. This includes inter-processor communication and synchronisation hardware devices that are separate from the main processor. The Connection Machine CM-5 is an early machine that uses off-the-shelf microprocessors as its processing elements [7]. Recent examples include Cray T3D that uses DECchip 21064 microprocessors [90]. The SARNet system, presented in this thesis, incorporates the StrongARM SA-110 microprocessors.

For efficient inter-processor communications, switching devices were utilised in the SARNet system. The use of ICR C416 packet routing devices in the SARNet has demonstrated: efficiency in providing concurrent communication channels for the connected processing elements; ease of scalability of the number of processing elements; and flexibility in constructing a specific network topology. In fact, as discussed in Chapter 2, many parallel processing systems have focused on utilising switching devices for inter-processor communications over the last few years, and this trend is likely to continue into the future. Cray T3D communication network and recent Myrinet routing devices are such examples. In addition to the switching core, high performance parallel systems like Cray T3D and those utilising Myrinet also rely on massively parallel interconnects and high clock speeds. However, using multiple serial interconnects will be more feasible and flexible for connecting the distributed parallel processing system targeted for embedded applications, such as the SARNet system.

In the area of communication and network interfacing, the SARNIC design raised a number of discussion points. These are mainly associated with the areas of coupling the network interface to the processor node architecture, streaming and buffering of the message, connections to the communication network, and the hardware virtual channel implementation.

6.1.1 Coupling of The Network Interface

As reviewed in Chapter 2, the coupling of the network interface to the processor node architecture can be classified into three generic types: integrated processor based, memory bus based, and I/O bus based. In general, the closer the network interface is to the processor node architecture, the higher the efficiency; the further the network interface is from the processor node architecture, the better the generality.

Integrated processor based network interfaces offer high efficiency and low latency in message handling by integrating the dedicated control hardware closely with the processor architecture. Designing a tightly coupled network interface and the processor on the same IC requires a large engineering effort and is time consuming. These factors cause upgrade difficulties to both the processor architecture and the network interface architecture. In comparison, microprocessors design cycles for stand-alone systems are much shorter, which is one of the factors that have allowed uni-processor systems to take the lead in the microprocessor market. The use of a memory bus based network interface, like the SARNIC, provides nearly the efficiency of a tightly coupled system, but separates the network interface hardware from the processor. This allows better technology migration, following the advances of microprocessor development. A processor upgrade would only require modification of the processor interface in the SARNIC CPLD design. In a similar way, the upgrade of the interconnection network would only require the modification of the network interface logic.

A standard I/O bus based network interface can be easily adapted across a different processor node, for example workstations or PCs. An example of this is the PCI/host Myrinet interface that is currently used in many parallel research machines [22]. An advantage of the Myrinet interface is the flexibility of the onboard communication co-processor that can be customised with protocol micro-code according to requirements. Despite the high efficiency, there is still a potential problem with the latency of crossing from the I/O bus to the memory bus. This latency is composed of both the overheads of initiating and synchronising a transaction through the I/O bus bridge, and the bandwidth sharing with other possible competing I/O bus devices. The SARNIC device offers a better latency figure as both the memory interface and the network interface is integrated in the same CPLD. As the arbitration is on-chip, fair utilisation of the memory bus is also balanced between the message transfers and CPU accesses.

MAGIC [64] is a network interface design that is similar to the SARNIC. Both utilise the memory bus based network interface architecture. Both are single chip solutions with an integrated memory interface, network interface, and processor interface. MAGIC is more sophisticated with its greater flexibility in network protocol implementation. It uses a micro-coded protocol processor for handling communications. SARNIC uses a hardwired state machine for efficient message handling. Coupled with the CPLD realisation, these have made the development of the SARNIC device far less complex.

6.1.2 Streaming Versus Buffering for Network Interfaces

The issue of streaming and buffering is related to the injection and reception of a message. A buffering network interface stores the complete message into the network interface buffers before it is injected or received. A streaming network interface allows the message header to go through the system using a worm-hole routing method, without waiting for the end of the message. Buffered transmissions offer smooth message transmission on the communication network with the compromise of increased buffering latency. Streaming transmissions overlap message assembly at the network interface with injection or reception time, hence, reducing the latency of buffering. However, if there are inconsistent data transfers between the host and the network interface, a possibility of data starvation on injection or network back pressure on reception can occur. Data starvation during message injection can cause ‘bubbles’ to be inserted, which cause inefficient bandwidth utilisation. Network back pressure on the other hand might cause congestion to the communication network.

The Myrinet PCI/host network interface design uses buffered transmission. The network interface buffers the whole packet in its SRAM before DMA transfers into the network or to the host. Buffering packets/messages in these I/O bus based systems is an advantage: mainly because the interval time between data transactions over the I/O bus varies depending on the host system memory bus arbiter, and the number of competing devices. In addition, the relatively high network communication bandwidth compared to the I/O bus bandwidth eventually increases the chance of data starvation and network back pressure. For instance, although the SHRIMP network interface uses streaming transmission, the limitation of 33 MB/s bandwidth on the EISA bus has become a bottleneck that limits the communication bandwidth and increases network back pressure [23].

Streaming transmission was implemented in the SARNIC design. As the SARNIC design has the network interface and the memory interface integrated on the same chip, the arbitration time on the memory bus is minimised and predicted. The network interface and the CPU take turns to access the memory, and therefore the interval between data transactions to the network interface is bounded. The use of streaming transmission in the network interface has not only reduced the buffering latency, but has also reduced the need of large buffering logic in the design. The buffer size is kept to minimum: just enough to cover the interval time between two consecutive data transactions. However, if the OS Link performance was comparable with the Myrinet, the buffer size would need to be much larger, and might even force the design back to utilising buffered packet transmission.

6.1.3 Physical Connection Ports

Most current network interfaces only provide a single, high performance connection port to the communication network. These interfaces normally utilise a multiple wire (parallel) connection port to achieve high communication bandwidth. Implementation of more than one connection port would not be advisable, nor necessary, as the communication link bandwidth is high enough to fill the system memory bandwidth with just the network interface bandwidth required. For instance, the PCI/host Myrinet interface has a peak network interface bandwidth of 132 MB/s (32-bit)[4], while the maximum communication link bandwidth is 160 MB/s. Other constraints to the connection port expansion would be the large increase required in both the pin count of the device package and the wire connections. To keep the external wire connections to a minimum, some communication links work in half duplex mode [91], i.e. only one direction of communication is allowed at one time. In this way, the effective bi-directional bandwidth is halved, while still maintaining full uni-directional bandwidth. However, the overall performance should not drop significantly, as the amount of uni-directional communications would normally be much higher than bi-directional communications.

With serial connection links, like the OS Link used in the SARNet system, it is desirable to have more than one physical link. Extra links will proportionally increase the effective communication interface bandwidth, as long as the system memory to network interface bandwidth is large enough to cover the sum. Extra links also offer better fault tolerance, through critical resource replication. Another advantage of multiple links would be the increased flexibility of the network topology construction.

One interesting aspect with the double link SARNIC design is the usefulness of the 'group adaptive routing' feature of the ICR C416 packet routing switch [18]. This feature can be helpful in the case where both links of the SARNIC are connected to the same router. By grouping the links in the router, both links can share a single port address. The ICR C416 router will automatically route a message to the other

4 Latest Myrinet/PCI Host design is targeted at 64-bit PCI, which provides 264 MB/s peak network interface bandwidth.

free grouped link if the destined link is occupied. In this way, two messages destined for the same address can be routed to a SARNode concurrently via both links.

6.1.4 Virtual Channels

Virtual channels are used to provide concurrent communication channels by sharing messaging facilities according to need, while preventing any channel from causing the starvation of others by monopolising the communication link resources. This idea applies in both the interconnection network and the network interface hardware.

Early switching networks relied on circuit switching mechanisms, where a communication channel is created and held by the two communicating entities. No other entities can access the allocated channel resource until it is released. To allow better communication channel resource sharing, the information passing was later broken down into smaller sections: from the largest basis (message basis) through packet basis, to the smallest basis (flow-group basis). Each virtual channel takes turns to pass a small section of information over the network. Most of the recent switching devices operate on a packet basis, although virtual channels on a flow-group basis [70, 71] provide better utilisation of the communication resources.

The theory of virtual channels also applies to network interfaces. Multiple virtual channels on a processing node can share the use of a network interface link. For instance, a large message is split into smaller packets at the transmitting end, and recombined at the receiving end. However, there is a need for identification information to be carried within each smaller packet in order to identify the dedicated receiving virtual channel.

The function of virtual channels in the SARNIC device is achieved through multiplexing multiple message channel access to the link resources on a packet basis. The link resource is allocated to a virtual channel for the transmission or reception of a packet, then on completion is relinquished to other virtual channels. The control process for the allocation can be carried out in software by swapping the DMA information of a link engine in and out of a virtual channel table in memory.

However, this incurs switching latency and processor occupancy for each virtual channel swap. The SARNIC design was implemented with hardware support for virtual channel handling. Multiple message virtual channels can share the link engine resource without software intervention. Normal logic cells in the CPLD are utilised for this hardware virtual channel implementation. Due to the CPLD size limitations, support for only two virtual channels per direction (a total of 4 per device) is implemented. More virtual channels can be added by using a larger CPLD, or in software through the caching concept (refer to Section 3.5.3.4). Another approach is by changing the hardware virtual channel implementation using the advanced memory blocks available in the newer CPLDs (refer to Section 6.3).

In the transmitting interface, hardware virtual channels will take turns to transmit a packet on the physical link resource. This will guarantee fair access to the link between the hardware virtual channels, avoiding one virtual channel monopolising the link. In this way, a situation where a long message blocks a short message will never happen, as the short message has the chance to transmit before the long message has finished. Nevertheless, a link stalled within a packet will effectively block all the virtual channels, even though this situation is very rare.

For SARNIC message virtual channel identification purposes, a 2-byte message header is added to each packet. An incoming packet header will be compared with the hardware virtual channel header in the receiving interface. A match will result in the routing of the whole packet to the corresponding hardware virtual channel. A mismatch will result in an interrupt being generated to signify there is no active receiving channel for the current packet. An issue raised for the current receiving hardware virtual channel implementation is that when a 'swap-out' is required, the decision of which virtual channel to 'swap-out' totally relies on software. A replacement algorithm like a 'least recently used' hardware implementation would ease the decision-making.

6.2 Conclusions

The objectives of this research were to investigate: the implementation of a distributed processing system using a packet routing switch network; the efficiency of a memory bus based network interface; and the resulting properties that effect minimum microprocessor support for message passing. This research has resulted in two main original contributions:

- The design and development of a CPLD based interface controller, the SARNIC, that integrates a bus-based network interface controller, a memory interface controller, and a processor interface controller in a single chip. The chip also contains the core logic for the functions required for processor node and multi processor node operation.
- The formation of a processor node PCB that is used as the building block of a low-cost embedded distributed processing system. This was used for the construction of a basic 4-node system for embedded distributed parallel processing applications, namely the SARNet.

The thesis has detailed the SARNet implementation, which has been constructed, tested and analysed. This system possesses a flexible switched network architecture that uses a modern RISC microprocessor, namely the StrongARM SA-110, for low-power high performance distributed processing. The switch architecture was built using the ICR C416, a 16-port intelligent crossbar switch, for dynamic message routing. The SARNIC, in addition to providing links between the processing entity and the communication network, also supplies data bus management and memory interfacing.

The SARNet system has been shown to work well with the switch topology. The test results showed that the communication layer of the system architecture imposes only a 3.08 μs latency on message transfers. This low overhead was achieved through the streaming network interface and the wormhole routing switch network architecture. The efficiency of the communication hardware was reaffirmed by the results obtained from the Dhrystone benchmark tests, with only 3.34% computation overhead arising from the communication layer. The tests showed the internal cache of the SA-110 CPU minimised data bus requirements, which allowed good use of cycle-stealing DMA to deliver messages. The bus arbitration of the SARNIC provided fair sharing of the data bus between the processor and communication DMA channels without compromising the computing performance of the StrongARM CPU.

The SARNIC device was described in VHDL and implemented in a re-programmable CPLD architecture. This has aided in maintaining low prototype costs and allowed scope for re-use. The support features of the SARNIC device have been demonstrated in the SARNet system implementation, especially providing the message-passing based network interfacing to the router network without significant interference to the processor. These SARNIC features include:

- A memory bus arbitration system, configured to provide guaranteed bandwidth for each message channel, while minimising the interference overhead of DMA accesses to normal CPU execution.
- DMA assisted message channels, offloading the packetising and de-packetising task of a message in software.
- Two communication links, providing improvement of the effective network interface bandwidth and better fault tolerance. This also offers extra flexibility in the network topology that can be formed.
- Hardware virtual channels, providing concurrent message transmission/reception and offloading the link resource allocation task from software. The hardware virtual channels are free to be allocated to either of the two physical communication link resources.
- A routing communication control link for connection to the control port of the ICR C416, providing automatic detection of blocked channels and acquiring minimum assistance from the processor.

- A boot from communication link feature, which is an advantage for remote slave system operation and real-time system reconfiguration.

The use of CPLDs for the SARNIC design implementation proved to be a suitable target technology for the SARNIC prototype. It minimised the time required for the hardware realisation cycle. In comparison to an ASIC implementation, it saved a large amount of simulation that would have been needed to verify the design functionality before prototype production. The re-programmable feature of the CPLD has also eased the debugging cycle of the hardware operation, and offers flexibility in future modifications or upgrades. In spite of these advantages, the SRAM based CPLD requires the use of a configuration ROM and consumes a considerable amount of power in comparison to other fixed logic devices such as ASIC designs.

The research has also highlighted some issues in extended distance communications using differential transmission. Standard RS-485 industry interface devices are utilised to achieve high performance, long distance communications in many distributed systems. With some enhancements made to reduce the pattern dependent jitter, a result of 44 Mbps communications at up to 100 m was demonstrated. Some level of protection has also been provided with conventional zener diodes, without sacrificing performance.

The memory bus based network interface architecture and the re-programmable feature of the SARNIC CPLD implementation will ease any communication system upgrades. The network interface hardware and the switch fabric can be altered to investigate modifications to the protocol without any changes to the software layers. This would allow concurrent development of software and hardware to increase the effectiveness of the system. The low cost loosely distributed processing system implementation will further its contribution in embedded control systems, certainly if used in a home networking environment.

6.3 Further Work

The current prototype SARNode building blocks have been successfully implemented and tested. Coupled with the existing ICR C416 packet router switch, a basic SARNet system has been formed. At present, a standard software development library for the SARNet platform is being developed in the research group. This will contribute to a low-cost and efficient embedded distributed system for parallel processing applications.

In parallel with the software development, there are a few minor hardware reworks that could be carried out. The task will be to redesign the SARNode PCB with the new SARNIC pin layout and corrections to the prototype PCB, while focusing on shrinking the PCB size. This eventually will give a more compact outlook to the SARNode PCB and a very high chance of 44 MHz data bus operation. At the end, a system rig could be built to hold 8 SARNode PCBs, with the ICR C416 router held in the system rig or inside the host PC. The host PC will be utilised to download parallel processing tasks into the SARNet system. A complete 8-node SARNet system will thus be constructed.

In terms of performance improvement, the rapidly advancing CPLD technology could be utilised. During the design stage of the SARNIC device, the only high performance, widely available CPLD from Altera was the FLEX 10KA family, utilised for the design. One year later, the FLEX 10KE [92] family was introduced, and more recently the APEX 20K family [93]. The smaller micron technology effectively pushed the underlying performance of these new CPLDs. At the same time, the architecture improvements like the dual-port RAM and Content Addressable Memory (CAM) have also benefited potential design implementations.

6.3.1 Dual-Port RAM Link Buffer Implementation

In the current design, normal logic resources were utilised to implement the DMA buffer and link FIFO. The decision was made due to the limitation of the single port RAM architecture EAB of the FLEX 10KA family. Furthermore, the maximum data width of each EAB can only be 8 bits. With the dual-port RAM in the FLEX 10KE or APEX 20K devices, implementing the link FIFO in EAB would not need to sacrifice device performance as cycle sharing the single access port of the FLEX 10KA EAB does. In terms of data width, the new EAB can support up to 16 bits. This only requires 2 EABs for each DMA buffer, which allows better utilisation of the EAB resources. The enhanced EAB architecture of the FLEX 10KE is highly recommended for future DMA buffer enlargement. This will effectively reduce the DMA accesses on the data bus, and reduces the number of logic elements required for the DMA buffer implementation.

6.3.2 Virtual Message Channel Improvements

The present SARNIC design was equipped with two hardware virtual message channels per direction. Ideally, more hardware virtual message channels could be implemented by replicating the existing channels in a larger CPLD. On the transmitting direction, no complexity is added, as each hardware virtual channel takes turns to use the physical link resource. On the receiving side, however, difficulties will be encountered. The matching of the incoming message header to a correct hardware virtual channel will take a longer time, if the checking process is to cycle through all the active virtual channels. At the other extreme, the decoding logic will become complicated, if comparison of all the hardware virtual channels is done simultaneously. Introduction of the CAM in the APEX 20K CPLD would make the message header comparison much easier. By storing the active virtual channel headers in a CAM block, matching of an incoming message header can ideally be done in one clock cycle. Assuming the message header size is kept to 16 bit wide, a total of 32 hardware message channels can be supported with one CAM block.

6.3.3 Internal Boot ROM Enhancement

In the existing SARNIC CPLD design, 2 kB of the memory resource (8 EABs) are allocated for internal ROM implementation. This internal ROM is used to store a small boot program, thus offering an alternative way of initialising the StrongARM SA-110 CPU other than booting from communication link. In order to save logic, the current architecture copies the ROM data to the SDRAM region using the same boot from communication link DMA mechanism. A switch is used to select the byte input of the channel from the internal ROM or from the communication link engine.

The Altera FLEX10KA family CPLD actually implements ROMs by initialising EABs with the set of ROM data. Instead of wasting the capability of EABs as only a read-only ROM, an enhancement can be made by changing the boot ROM implementation into RAM, with initialised boot code on power up. This can be achieved by mapping the EAB blocks to the system memory region at location zero, with the help of additional memory decoding logic. This enhancement would provide the SA-110 CPU with a small amount of built-in fast SRAM, which is similar to the internal memory offered in Transputers. The SARNode processor node would be capable of operating with limited amount of memory, removing the necessity of SDRAM.

6.3.4 Link Transmission Speed & Protocol

Over time, advances in silicon technology have pushed the performance of data transmissions. If the 3 times over-sampling resolution and the double phase sampling architecture of the existing OS Link interface is to maintained, a higher link transmission speed would be predicted. As discussed in Section 3.6.4 the same link implementation is capable of achieving 44 Mbps transmission speed. Hence, if the ICR C416 router architecture is to be ported to the current CPLD technology, a higher communication network throughput can be achieved.

The SARNet latency tests have also highlighted the requirements to investigate other communication protocols, which may operate more efficiently. For instance, there is a significant reduction in the useable bit rate due to the low-level serial communication protocol, especially with concurrent bi-directional data transfer. The byte acknowledgement flow control of the OS Link reduces the bi-directional bandwidth to 85% of double the uni-directional bandwidth[5]. As the delay time of the transmission line increases proportionally with the distance, the turn around time of the acknowledge token will eventually not be enough to allow continuous flow of data tokens. This would utilise flow controls like the flow-group acknowledge or the stop-go protocol.

[5] This figure is quoted for the ideal 13 bits per byte bi-directional transmission. In most cases, the figure might be down to 16 bits per byte due to rapid synchronisation between acknowledge and data tokens transmission.

PUBLICATIONS

1 O'Neill, B. C., Wong, K. L., Coulson, G., Clark, S., Thomas, P. D., Cook, B. M., Walker, C. P. H., "*A Low-cost High-performance Multicast Routing Switch Chip for Communication Networks*", European Multimedia, Microprocessor Systems and Electronic Commerce, Conference, Florence, November 1997, pp. 836-843.

2 O'Neill, B. C., Wong, K. L., Coulson, G., Hotchkiss, R., Ng, J. H., Clark, S., Thomas, P. D., Cawley, A., "*A Distributed Parallel Processing System for the StrongARM Microprocessor*", Concurrent Systems Engineering, ISSN 1383-7575, 1998, Vol. 52, pp. 39-48.

3 O'Neill, B. C., Wong, K. L., Coulson, G., Hotchkiss, R., Ng, J. H., Clark, S., Thomas, P. D., "*An Interface Device to Support a Distributed Parallel System for the StrongARM Microprocessor*", Lecture Notes in Computer Science, ISSN 0302-9743, 1998, Vol.1401, pp. 1047-1050.

4 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.

5 Wong, K. L., O'Neill, B. C., Hotchkiss, R., Ng, J. H., Clark, S., Thomas, P. D., "*Interfacing StrongARM Microprocessors in a Parallel Network*", Postgraduate Research in Electronics, Photonics & Related Fields (PREP'99), January 1999, pp. 382-385.

6 Liew, K. W. K., O'Neill, B. C., Wong, K. L., Clark, S., Thomas, P. D., Cant, 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.

REFERENCES

1 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.

2 Sze, S. M., "*Semiconductor Devices: Physics and Technology*", Wiley, 1985, pp. 499-504.

3 Valiant, L. G., "*General Purpose Parallel Architectures*", Technical report TR-07-89, Aiken Computation Laboratory, Harvard University.

4 Hockney, R. W., Jeshope, C. R., "*Parallel Computers 2: Architecture, Programming and Algorithms*", 2nd Ed., UK, IOP Publishing Ltd., 1988, pp. 1-53.

5 Hwang, K., Briggs, F. A., "*Computer Architecture and Parallel Processing*", USA, McGraw-Hill, 1984, pp. 32-35.

6 Quinn, M. J., "*Parallel Computing – Theory and Practice*", 2nd Ed., USA, McGraw-Hill, 1994, pp. 67-80.

7 Hord, R. M., "*Parallel Supercomputing in MIMD Architectures*", USA, CRC Press Inc., 1993, pp. 13-60, 95-103.

8 Hwang, K., "*Advanced Computer Architecture - Parallelism, Scalability, Programmability*", USA, McGraw-Hill, 1993, pp. 1-50, 331-393.

9 Bertsekas, D. P., Tsitsiklis, J. N., "*Parallel & Distributed Computation – Numerical Methods*", USA, Prentice-Hall International Inc., 1989, pp. 1-8.

10 Culler, D., Karp, R., Patterson, D., Sahay, A., Schauser, K. E., Santos, E., Subramonian, R., Eicken, T. von, "*LogP: Towards a Realistic Model of Parallel Computation*", Communications of the ACM, November 1996, Vol. 39 (11), pp. 78-85.

11 Holt, C., Heinrich, M., Singh, J. P., Rothberg, E., Hennessy, J., "*The Effects of Latency, Occupancy, and Bandwidth in Distributed Shared Memory Multiprocessors*", Technical Report CSL-TR-95-660, Computer Systems Laboratory, Stanford University, January 1995

12 Protic, J., Tomasevic, M., Milutinovic, V., "*Distributed Shared Memory: Concepts and Systems*", IEEE Parallel & Distributed Technology: Systems & Applications, 1996, Vol. 4(2), pp. 63-71.

13 Reed., D. A., Fujimoto, R. M., "*Multicomputer Networks – Message-Based Parallel Processing*", USA, The MIT Press, 1987, pp. 80-82.

14 Hwang, K., "*Advanced Computer Architecture - Parallelism, Scalability, Programmability*", USA, McGraw-Hill, 1993, pp. 75-96.

15 Stone, H. S., "*High-Performance Computer Architecture*", 3rd Ed., USA, Addison-Wesley Publishing Company, 1993, pp. 358-363.

16 Cray Research Inc., "*Cray T3D System Architecture Overview*", Cray Research Inc., Part no. HR-04033.

17 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.

18 "*ICR C416 - 16-port Dynamic Routing Switch for Transputer Links - Data Sheet*", IC-Routing Ltd., 1996.

19 Joerg, C. F., Henry, D. S., "*A Tightly-Coupled Processor-Network Interface*", Computation structures group memo 342, Massachusetts Institute of Technology, March 1992.

20 Dallly, 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.

21 "*The Transputer Databook*", 2nd ed., Trowbridge, Inmos Limited, 1989.

22 Bhoedjang, R., Ruhl, T., Bal, H. E., "*User-Level Network Interface Protocols*", IEEE Computer, Novermber 1998, Vol. 31(11), pp. 53-60.

23 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.

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

25 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.

26 Ellis, J. W., O'Neill, B. C., Mills, E., "*Enhanced Communications for Transputer Arrays*", Transputer and OCCAM Engineering, ISSN 0925 4986, Vol 18, 1991, pp. 475-480.

27 Ellis, J. W., O'Neill, B. C., Clark, S., "*A Router Design for T800 Compatible Transputer Arrays*", Transputer Applications, ISSN 0969-9341, Vol. 1(2), 1993, pp. 12-18.

28 Curtis, K. M., Wilde, S, O'Neill, B. C., Ellis, J. W., Jelly, I., Lloyd, D., "*A Dynamic Routing Strategy for Transputer Networks*", Transputer and OCCAM Engineering, ISSN 0925-4986, Vol. 41, July 1994, pp. 235-246.

29 Ellis, J. W., "*A Hardware Routing Device for Transputer Arrays*", Ph.D. Thesis, The Nottingham Trent University, October 1995.

30 O'Neill, B. C., Coulson, G. C., Ellis, J. W., Clark, S., "*Design and Exploitation of a 26,000 Gate Message Routing Device*", Fifth EUROCHIP Workshop on VLSI Design Training, Dresden, Germany, October 1994, pp. 290-303.

31 Coulson, G., "*Optimisation of a Processing Farm Using Hardware Routing*", Transputer Applications and Systems, IOS Press, 1995, Vol. 46, pp. 70-77.

32 Coulson, G., "*An ASIC Implementation of A Multicast Message Routing Switch for Interprocessor Communications*", Ph. D. Thesis, The Nottingham Trent University, September 1998.

33 Hotchkiss, R., "*A Fault Tolerant Multicast Message Routing Switch for Interprocessor Communications*", Transfer Report, The Nottingham Trent University, September 1999.

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

35 "*Altera 1998 Data Book*", Altera Corporation, 1998, pp. 21-132.

36 Chambers, F. B., Duce, D. A., Jones, G. P., "*Distributed Computing*", London, Academic Press 1984, pp. 141-142.

37 Carlini, U. de, Villano, U., "*Transputers and Parallel Architectures - Message-Passing Distributed Systems*", UK, Ellis Horwood, 1991, pp. 1-69.

38 Heinlein, J., Gharachorloo, K., Dresser, S., Gupta, A., "*Integration of Message Passing and Shared Memory in the Stanford FLASH Multiprocessor*", Proceeedings of the 6th International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, October 1994, pp. 38-50.

39 Coulouris, G., Dollimire, J., Kindberg, T., "*Distributed Systems - Concepts and Design*", 2nd Ed., Wokingham, Addison-Wesley, 1994, pp. 1-124.

40 Thurber, K. J., Masson, G. M., "*Distributed-Processor Communication Architecture*", Lexington, Mass, 1979, pp. 239-247.

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

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

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

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

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

46 Held, G., "*Data Communications Networking Devices – Operation, Utilization and LAN and WAN Internetworking*", 4th. Ed., John Wiley & Sons Ltd., 1999, pp. 300-328.

47 "*Gigabit Ethernet – Accelerating the Standard for Speed*", Whitepaper, Copyright 1998 Gigabit Ethernet Alliance.

48 "*Gigabit Ethernet – 1000 BASE-T*", Whitepaper, Copyright 1997 Gigabit Ethernet Alliance.

49 "*ST C104 - Asynchronous Packet Swtich - Data Sheet*", SGS-Thompson, June 1994.

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

51 Hinton, J., Pinder, A., "*Transputer Hardware and System Design*", UK, Prentice Hall International Ltd., 1993, pp. 195-212.

52 Shanley, T., Anderson, D., "*PCI System Architecture*", 3rd. Ed., US, Addison Wesley, 1995, pp. 19-35.

53 McKenzie, N. R., Bolding, K., Ebeling, C., Snyder, L., "*Cranium: An Interface for Message Passing on Adaptive Packet Routing Networks*", Proceedings of the 1994 Parallel Computer Routing and Communication Workshop, Springer-Verlag, May 1994.

54 Boosten, M., Dobinson, R. W., Martin, B., Van De Stok, P. D. V., "*A PCI Based Network Interface Controller for IEEE 1355 DS Links*", Wotug-21, IOS Press, 1998, pp. 49-68.

55 "*LANai 4*", SAN-Chipset Draft Documentation, Myricom, February 1999.

56 "*LANai 7*", SAN-Chipset Draft Documentation, Myricom, June 1999.

57 Buzzard, G., Jacobson, D., Mackey, M., Marovich, S., Wilkes, J., "*An Implementation of the Hamlyn Sender-Managed Interface Architecture*", The Second Symposium on Operating Systems Design and Implementation (OSDI'96) Proceedings, Seattle, WA, October 1996, pp. 245-259.

58 Dubnicki, C., Bilas, A., Li, K., "*Design and Implementation of Virtual Memory-Mapped Communication on Myrinet*", Proceedings of the 11th International Parallel Processing Symposium, April 1997.

59 Felten, E. W., Alpert, R. D., Bilas, A., Blumrich, M. A., Clark, D. W., Damianakis, S. N., Dubnicki, C., Iftode, L., Li, K., "*Early Experience with Message-Passing on the SHRIMP Multicomputer*", Proceedings of the 23rd Annual International Symposium on Computer Architecture, May 1996, pp. 296-307.

60 Blumrich, M. A., Dubnicki, C., Felten, E. W., Li, K., Mesarina, M., R., "*two Virtual Memory Mapped Network Interface Designs*", The Hot Interconnects II Symposium Record, August 1994, pp. 134-142.

61 Blumrich, M. A., "*Network Interface for Protected, User-level Communication*", Ph. D. Dissertation, Princeton University, June 1996.

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

63 Hoare, C. A. R., "*Communicating Sequential Processes*", Hemel Hempstead, Prentice-Hall International, 1985.

64 Kuskin, J., Ofelt, D., Heinrich, M., Heinlein, J., Simoni, R., Gharachorloo, K., Chapin, 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.

65 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 the 6th International Conference on Architectural Support for Programming Languages and Operating Systems, San Jose, CA, October 1994, pp. 274-285.

66 Hwang, K., "*Advanced Computer Architecture - Parallelism, Scalability, Programmability*", USA, McGraw-Hill, 1993, pp. 157-177.

67 Esponda, M, Rojas, R., "*The RISC Concept – A Survey of Implementations*", Technical report B-91-12, Freie University Berlin, September 1991.

68 R. Weidlich, "*Speeding up DRAMs*", Siemens Memory Technical Document, Siemens, May 1997, pp. 22-24.

69 Walker, C. P. H., "*Hardware Implications of Virtual Channels, particularly for the Bus Interface*", Transputer Applications and Systems '94, IOS Press, 1994, pp. 223-234.

70 Dally, W. J., "*Virtual-Channel Flow Control*", IEEE Transactions on Parallel and Distributed System, Vol. 3, No. 2, March 1992, pp. 194-205.

71 Dally, W. J., Aoki, H., "*Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels*", IEEE Transactions on Parallel and Distributed System, Vol. 4, No. 4, April 1993, pp. 466-475.

72 "*Unlocking the Power of HDL-based design for programmable Logic*", Seminar with the Experts from Altera, Synplicity & Model Technology, Seminar Handouts, September 1998.

73 "*FLEX 8000 Programmable Logic Device Family Data Sheets*", Ver 6, Altera Corporation, March 1995, pp. 37-93.

74 "*Mentor Graphics & MAX+Plus II Software Interface Guide*", Ver. 8, Altera Corporation, San Jose, 95134 USA, June 1997.

75 "*ADM1485 - +5V Low Power EIA RS-485 Transceiver*", Data Sheet, Rev. 0, Analog Devices, July 1993.

76 "*LTC1686/1687 – 52 Mbps Precision Delay RS485 Fail-Safe Transceivers*", Data Sheet, Linear Technology, November 1997.

77 "*DS26C31T/DS26C31M CMOS Quad TRI-STATE Differential Line Driver*", Data Sheet, National Semiconductor Corporation, June 1998.

78 "*DS26C32AT/DS26C32AM Quad Differential Line Receiver*", Data Sheet, National Semiconductor Corporation, June 1998.

79 "*SM1720A Datatwist 100 – Technical Data Sheet*", Rev. 7, Belden Wire & Cable Company, 1999.

80 Haas, S, Liu, X, Martin, B., "*Long Distance Differential Transmission of DS Links over Copper Cable*", Rev. 2, CERN – European Organization for Nuclear Research, ECP Division, CH-1211, Geneve 23, July 1993.

81 Peter, J. M., "*Transil Clamping Protection Mode*", Application Note AN316, SGS-Thomson Microelectronics, October 1997.

82 "*BZX79 Series Voltage Regulator Diode*", Data Sheet, Philips Discrete Semiconductors, April 1996.

83 "*168 pin unbuffered COB-DIMM Modules HYS64/72V2200GCU-10, HYS64/72V4220GCU-10*", Siemens Semiconductor Group, February 1998.

84 "*2, 4 MEG x 64 SDRAM DIMMs MT8LSDT264A, MT16LSDT464A*", Micron Technology Inc., June 1998.

85 Novak, M., "*Use a CPLD to Implement an SDRAM Controller*", EDN Access, June 1997.

86 Chaney, T. J., Molnar, C. E., "*Anomalous Behaviour of Synchronizer and Arbiter Circuits*", IEEE Transactions on Computers, 1973, pp. 421-422.

87 Beaston, J., Tetrick, R. S., "*Designers confront Metastability in Boards and Buses*", Computer Design, March 1986, pp. 67-71.

88 "*Metastability in Altera Devices*", Ver. 2, Application Note 42, Altera Corporation, June 1996, pp. 435-444.

89 "*B816 Evaluating Board*", Data Sheet, IC Routing Ltd., 1996.

90 Scott, S, Thorson, G., "*Optimised Routing in the Cray T3D*", Cray Research Inc., May 1994.

91 Bolding, K., Cheung, S. C., Choi, S. E., Ebeling, C., Hassoun, S., Ngo, T. A., Wille, R., "*The Chaos Router Chip: Design and Implementation of an Adaptive Router*", Proceedings of VLSI'93, IFIP, 1993, pp. 311-320.

92 "*FLEX 10KE Embedded Programmable Logic Family Data Sheet*", Ver. 2.02, Altera Corporation, August 1999.

93 "*APEX 20K Embedded Programmable Logic Family Data Sheet*", Ver. 2.05, Altera Corporation, November 1999.

APPENDICES

A. TECHNICAL DOCUMENTATION OF SARNIC

The control and status registers of the chip are accessible at address offset of C000 0000h.

Register	Offset
RISR	00h
IER/IESR	04h
IECR	08h
ISR	0Ch
Reserved	10h
FER/FESR	14h
FECR	18h
FSR	1Ch
SDTR	20h
Reserved	24h
IOCR	28h
Reserved	2Ch
URBR/UTHR	30h
ULCSR	34h
Reserved	38h-3Ch
TER/TECR	40h
TVR/TMR	44h
Reserved	48h-5Ch
OCRDR/OCTDR	60h
OCCSR	64h
Reserved	68h-6Ch
O0RBR	70h
O1RBR	74h
Reserved	78h-80h
DO0RHR	84h
DO0RAR	88h
DO0RLR	8Ch
DO0THHR	90h
DO0THLR	94h
DO0TAR	98h
DO0TLR	9Ch
Reserved	A0h
DO0RHR	A4h
DO0RAR	A8h
DO0RLR	ACh
DO0THHR	B0h
DO0THLR	B4h
DO0TAR	B8h
DO0TLR	BCh

Reserved	C0h-FCh

RISR: *Raw Interrupt Status Register: (offset 00h)*

This read-only register gives information on status of unmasked interrupt sources.

Word bit	Name	R/W	Description
0	UART receiver interrupt (UAR)	R	A '1' indicates the UART receiver interrupt is active. Identical to DR bit of URBR.
1	UART transmitter interrupt (UAT)	R	A '1' indicates the UART transmitter interrupt is active. Identical to THRE bit of ULCSR
3:2	-		
4	Timer interrupt (TIM)	R	A '1' indicates a timer event has occurred.
7:5	-		
8	OSLC receiver interrupt (OCR)	R	A '1' indicates the OS Link Control Port receiver interrupt is active. Identical to DR bit of OCRDR.
9	OSLC transmitter interrupt (OCT)	R	A '1' indicates the OS Link Control Port transmitter interrupt is active. Identical to THRE bit of OCCSR
10	OSLC error interrupt (OCE)	R	A '1' indicates the OS Link Control Port error interrupt is active. Identical to ERR bit of OCCSR.
11	-		
12	OSL0 receiver interrupt (OR0)	R	A '1' indicates OS Link 0 has received a new message header.
13	OSL1 receiver interrupt (OR1)	R	A '1' indicates OS Link 1 has received a new message header.
15:14	-		
16	DMA OS receiver interrupt (DOR0)	R	A '1' indicates the DMA OS 0 receiver channel interrupt is active. Identical to DONE bit of DO0RLR.
17	DMA OS receiver interrupt (DOR1)	R	A '1' indicates the DMA OS 1 receiver channel interrupt is active. Identical to DONE bit of DO1RLR.
19:18	-		
20	DMA OS transmitter interrupt (DOT0)	R	A '1' indicates the DMA OS 0 transmitter channel interrupt is active. Identical to DONE bit of DO0TLR.
21	DMA OS transmitter interrupt (DOT1)	R	A '1' indicates the DMA OS 1 transmitter channel interrupt is active. Identical to DONE bit of DO1TLR.
23:22	-		
27:24	I/O interrupt (IO)	R	A '1' in the bit position indicates the associated external I/O interrupt source is active.
31:28	-		

IER: *IRQ enable register: (offset 04h)*

This read-only register is used to mask the IRQ interrupt input sources and defines which active sources generate an interrupt request to CPU on IRQ.

Word bit	Name	R/W	Description
31:0	IRQ enable	R	A '1' indicates the associated interrupt source is enabled and allows an interrupt request. A '0' indicates the interrupt is disabled.

IESR: *IRQ enable set register: (offset 04h)*

This write-only register is used to set bits in the IER.

Word bit	Name	R/W	Description
31:0	IRQ enable set	W	A '1' in this bit will set the corresponding bit in the enable register. A '0' will have no effects.

IECR: *IRQ Enable Clear Register: (offset 08h)*

This write-only register is used to clear bits in the IER.

Word bit	Name	R/W	Description
31:0	IRQ enable clear	W	A '1' in this bit will clear the corresponding bit in the enable register. A '0' will have no effects.

ISR: *IRQ Status Register: (offset 0Ch)*

This read-only register gives information on masked interrupt status for IRQ. Masked status bits are a bit-wise 'AND' of unmasked status bits and IRQ enable bits.

Word bit	Name	R/W	Description
31:0	IRQ masked status	R	A '1' in a bit position indicates the associated interrupt source is both active and enabled.

FER: *FIQ Enable Register: (offset 14h)*

This read-only register is used to mask the FIQ interrupt input sources and defines which active sources generate an interrupt request to CPU on FIQ.

Word bit	Name	R/W	Description
31:0	FIQ enable	R	A '1' indicates the associated interrupt source is enabled and allows an interrupt request. A '0' indicates the interrupt is disabled.

FESR: *FIQ Enable Set Register: (offset 14h)*

This write-only register is used to set bits in the FER.

Word bit	Name	R/W	Description
31:0	FIQ enable set	W	A '1' in this bit will set the corresponding bit in the enable register. A '0' will have no effects.

FECR: *FIQ Enable Clear Register: (offset 18h)*

This write-only register is used to clear bits in the FER.

Word bit	Name	R/W	Description
31:0	FIQ enable set	W	A '1' in this bit will clear the corresponding bit in the enable register. A '0' will have no effects.

FSR: *FIQ Status Register: (offset 1Ch)*

This read-only register gives information on masked interrupt status for FIQ. Masked status bits are a bit-wise 'AND' of unmasked status bits and FIQ enable bits.

Word bit	Name	R/W	Description
31:0	FIQ masked status	R	A '1' in a bit position indicates the associated interrupt source is both active and enabled.

SDTR: *SDRAM Timing Register: (offset 20h)*

This register controls the timing and format of the SDRAM access.

Word bit	Name	R/W	Description
0	RAS to CAS delay (TRCD)	R/W	The number of cycles from a row activate command to the first read or write command. 0 - 1 cycle 1 - 2 cycles
1	Last data in to precharge (TDPL)	R/W	On write cycles, this is the minimum number of cycles from the last data being written to the start of precharge. 0 - 1 cycle 1 - 2 cycles
3:2	Active to Precharge delay (TRAS)	R/W	The minimum number of cycles from row activate command to precharge command. 00 - 1 cycle 01 - 2 cycles

			10 - 3 cycles 11 - 4 cycles
6:5	Row cycle time (TRC)	R/W	The minimum number of cycles from an auto-refresh command to the next row activate command. 00 - 3 cycle 01 - 4 cycles 10 - 5 cycles 11 - 6 cycles
31:7	-		

IOCR: *I/O Control Register (offset 28h)*

This register is used to control the timing and strobe states of the external I/O device. All 4 I/O interfaces shared the same timing in this register.

Word bit	Name	R/W	Description
7:0	Strobe mask (SM)	R/W	Strobe mask shifted out to io_rd_l or io_wr_l (for read or write respectively) during access to external I/O device.
10:8	Cycle length (CL)	R/W	Contains the number of bus cycles for an access to the external I/O device.
11	-		
13:12	Frequency divider (FD)	R/W	Clock frequency is divided by the value in this field to determine the frequency of the clock that is used to time the cycle length and shift the strobe mask. 00 – 1 01 – 2 10 – 3 11 – 4
31:14	-		

URBR: *UART Receiver Buffer Register (offset 30h)*

This read-only register contains the data received from the UART, and the status of the byte. Flags include the buffer states of receiver, and errors associated with the data received.

Word bit	Name	R/W	Description
7:0	Data	R	Contains the valid data byte received.
8	Data ready (DR)	R	A '1' indicates a complete incoming character has been received. This bit will causes an interrupt to the CPU if enabled in FER or IER by setting UAR bit. Bit reset when CPU reads the contents of URBR.
9	Overrun error (OE)	R	A '1' indicates the data in URBR was not read by the CPU before the next character has completely received in the receiver shift register. The character in the shift register is overwritten without

			transferred to URBR. Bit reset when CPU reads the contents of ULSR.
10	Parity error (PE)	R	A '1' indicates the received character does not have the correct even or odd parity, as selected by the EPS bit in ULSR. Bit reset when CPU reads the contents of ULSR.
11	Framing error (FE)	R	A '1' indicates the received character does not have valid stop bits. A logic '0' sampled during the stop bits position will set the error. Bit reset when CPU reads the contents of ULSR.
31:12	-		

UTHR: *UART Transmitter Hold Register (offset 34h)*

This write-only register contains the data to be transmitted to the UART.

Word bit	Name	R/W	Description
7:0	Data	W	Contains the data byte to be transmitted.
31:8	-		

ULCSR: *UART Line Control & Status Register (offset 38h)*

This register controls the format of the asynchronous data communication exchange. It also provides status information to the CPU concerning the data transfer. Flags include the buffer states of transmitter, and redundant receiver status bits.

Word bit	Name	R/W	Description
1:0	Baud rate select (BRS)	R/W	This field is used to select 1 of the 4 preset baud rates: 00 - 9600 bps 01 - 19200 bps 10 - 38400 bps 11 - 57600 bps
2	-		
3	Parity enable (PEN)	R/W	This bit is used to enable or disable parity generation and checking. When PEN = 1, parity bit is generated in data transmitted, and is checked in data received, according to the EPS bit. When PEN = 0, no parity is generated or checked.
4	Even parity select (EPS)	R/W	This bit selects whether odd or even parity should be used by the transmitter and receiver logic. When EPS = 1, even parity is selected. When EPS = 0, odd parity is selected.
7:5	-		
8	Data ready (DR)	R	Identical to DR bit in URBR
9	Overrun error (OE)	R	Identical to OE bit in URBR
10	Parity error (PE)	R	Identical to PE bit in URBR
11	Framing error (FE)	R	Identical to FE bit in URBR
12	-		

13	Transmitter holding register empty (THRE)	R	A '1' indicates UTHR is empty and is ready to accept a new character for transmission. This bit will causes an interrupt to the CPU if enabled in FER or IER by setting UAT bit. Bit reset when CPU writes to UTHR.
31:14	-		

TER: *Timer Enable Register (offset 40h)*

This read-only register informs whether the timer match event is enabled or disabled.

Word bit	Name	R/W	Description
0	Enable (EN)	R	A '1' in this position indicates the timer comparison is active. A '0' indicates it's disabled.
31:1	-		

TECR: *Timer Enable Clear Register (offset 60h)*

This write-only register is used to disable the timer.

Word bit	Name	R/W	Description
0	Clear (CLR)	W	A '1' in this position will disable the timer comparison and clear the interrupt. A '0' will be ignored.
31:1	-		

TVR: *Timer Value Register (offset 44h)*

This read-only register holds the current value of the 32 bit 1 μs counter.

Word bit	Name	R/W	Description
31:0	Current counter value	R	Contains the current counter value, which ticks every microsecond.

TMR: *Timer Match Register (offset 44h)*

This register holds the value to be compared with the current counter.

Word bit	Name	R/W	Description
31:0	Compare value	R/W	The writing to this register set the EN bit in TER and enables the current value of timer to be compared with this value. Interrupt will be generated if TVR value is equal to the compared value or just passed within the first half of 32-bit modulo region, provided that the TIM bit in FER or IER is set.

OCRDR: *OSL Control port Receiver Data Register (offset 60h)*

This read-only register contains the data received from the OS Link Control port, and the status of the byte.

Word bit	Name	R/W	Description
7:0	Data	R	Contains the valid data byte received.
8	Data ready (DR)	R	A '1' indicates a complete incoming character has been received. This bit will causes an interrupt to the CPU if enabled in FER or IER. Bit reset when CPU reads the contents of OCRDR.
31:9	-		

OCTDR: *OSL Control port Transmitter Data Register (offset 60h)*

This write-only register contains the data to be transmitted to the OS Link Control port.

Word bit	Name	R/W	Description
7:0	Data	W	Contains the data byte to be transmitted.
31:8	-		

OCCSR: *OSL Control port Control & Status Register (offset 64h)*

This register is dedicated to the control port of the ICR C416 router. There is a built in link time-out check for the ICR C416 router through this port, which can be enable or disabled. Whenever the internal time-out counter counts to zero, a link check command will be sent to the control port of the ICR C416 router. Error will be reported if there is blocked link. If the OS Link Control port is only used to connect to conventional OS Link devices, the time-out check feature should be disabled.

Word bit	Name	R/W	Description
0	Time-out check enable (CEN)	R/W	This bit enables or disables the ICR C416 router links idling time-out check. The check is done by sending command byte (0x9) to the control port of the ICR C416 whenever the time-out counter reaches zero.
2:1	Time-out clock select (CCS)	R/W	These bits pre-scale the clock which count the timer, from the 30 MHz clock input. Selections are as follows: • 00 ~ 64 us (2K/30 MHz = 68.27 us) • 01 ~ 1 ms (32K/30 MHz = 1.09 ms) • 10 ~ 16 ms (512K/30 MHz = 17.48 ms) • 11 ~ 256 ms (8M/30 MHz = 279.62 ms)
6:3	Time-out counter load value (CLV)	R/W	The load value for the time-out check timer. When time-out check enabled and counter reaches zero, this value will be

			reloaded into the counter again.
7	Error (ERR)	R	Indicates blocked link has been found. An interrupt will be generated if enabled in IER or FER.
11:8	Error link (ELNK)	R	Link number that has been found blocked, associated with the ERR bit.
13:12	-		
14	Data ready (DR)	R	Identical to DR bit of OCRDR.
15	Transmitter holding register empty (THRE)	R	A '1' indicates OCTDR is empty and is ready to accept a new character for transmission. This bit will cause an interrupt to the CPU if enabled in FER or IER. Bit reset when CPU writes to OCTDR.
31:16	-		

O0RBR: *OSL0 Receive Buffer Register: (offset 70h)*

This read-only register contains 1 or 2 message tags from the OS Link 0 during OR0 interrupt is active. At other time, it is used as temporary data buffer for OS Link 0.

Word bit	Name	R/W	Description
7:0	Message tag 1	R	This byte carries message tag 1.
14:8	Message tag 2	R	An MSB of '1' in message tag 1 indicates this field carries a valid message tag 2. If not, this field will be read as zero. (Note that bit 15 is tied down to zero since it's always zero according to the protocol)
31:15	-		

O1RBR: *OSL1 Receive Buffer Register: (offset 74h)*

This read-only register contains 1 or 2 message tags from the OS Link 1 during OR1 interrupt is active. At other time, it is used as temporary data buffer for OS Link 1.

Word bit	Name	R/W	Description
7:0	Message tag 1	R	This byte carries message tag 1.
14:8	Message tag 2	R	An MSB of '1' in message tag 1 indicates this field carries a valid message tag 2. If not, this field will be read as zero. (Note that bit 15 is tied down to zero since it's always zero according to the protocol)
31:15	-		

DO0RHR: *DMA OS 0 Receiver Header Register: (offset 84h)*

This register contains message tags for the comparison of incoming OS Link message. Either 1 or 2 message tags are allowed. This register is read-only when the channel is activated.

Word bit	Name	R/W	Description
7:0	Message tag 1	R	This byte carries message tag 1.
14:8	Message tag 2	R	An MSB of '1' in message tag 1 indicates this field carries a valid message tag 2. If not, this byte is ignored for comparison. (Note that bit 15 is tied down to zero since it's always zero according to the protocol)
31:15	-		

DO0RAR: *DMA OS 0 Receiver Address Register: (offset 88h)*

This register contains the SDRAM destination address for the DMA transfer. The register is loaded with the address at the start of the transfer and is updated internally after each SDRAM transaction as the transfer proceeds. The address value needs to be word aligned. Bit 1 and bit 0 of the address will be ignored. This register is read-only when the channel is activated.

Word bit	Name	R/W	Description
1:0	-		
23:2	Destination SDRAM address	R/W	Contains the destination address within the SDRAM for the incoming DMA transfer. It's updated internally as the DMA operation progresses.
31:24	-		

DO0RLR: *DMA OS 0 Receiver Length Register: (offset 8Ch)*

This register contains the length of message in bytes that needs to be received. The value must be at word boundary. Bit 1 and bit 0 of the length value will be ignored. This register is read-only (except DACTV and RST) when channel is activated.

Word bit	Name	R/W	Description
1:0	-		
15:2	Word count	R/W	Indicates the number of words in the message to be transferred to SDRAM. It's updated internally after each write as the DMA operation progresses.
16	Channel active (ACTV)	R	When '0', channel not active. When '1', channel enabled to receive message. Bit resets when DMA transfer is done.
17	Channel packet active (PACTV)	R	When '0', no packet is active. When '1', the DMA channel is active receiving a packet.
18	Channel done (DONE)	R/W1C	When the DMA transfer is done, this bit is set to '1' internally and can interrupt the CPU if enabled in FER or IER. Write of '1' to this bit will reset the bit.
19	Channel error (ERR)	R	When the actual incoming message length is different from the expected one, this bit will be set to indicate error. Early or

			late termination can be determined by the byte count value. Write of '1' to this bit will reset the bit.
20	Flush channel (FLSH)	R/W	This bit is used to flush the incoming data without transferring it to the memory. Byte counts and address pointer unchanged during flush.
21	Deactivate channel (DACTV)	W	This bit is used to deactivate the receiver channel, which only effective when ACTV = '1' and PACTV = '0'. Please note that a read of PACTV = '0' followed by a write of DACTV = '1' doesn't mean receiver channel definitely be deactivated, since PACTV might just changed to '1' after the read.
22	Reset channel (RST)	W	When '1', the receiver channel will be reset to idle state. The bit will be cleared automatically after the reset.
23	Physical channel connected (PHYC)	R	This bit indicates which physical OS Link does the DMA receiver channel allocated to.
31:24	-		

DO0THHR: *DMA OS 0 Transmitter Header High Register: (offset 90h)*

DO0THHR and DO0THLR are considered as 6 continuous byte register, holding the routing headers and message tags for the OS Link protocol which will be sent at the beginning of each packet transfer. The organisation of header bytes is little endian. Variable lengths of header are allowed. Note that the MSB of each header byte must be a '1', except the last header byte which is a '0'. This register holds the higher 2 bytes of the header. The register is read-only when the channel is activated.

Word bit	Name	R/W	Description
7:0	Header 5	R/W	Carries value of header byte 5.
14:8	Header 6	R/W	Carries value of header byte 6. (Note that bit 15 is tied down to zero since it's always zero according to the protocol)
31:15	-		

DO0THLR: *DMA OS 0 Transmitter Header Low Register: (offset 94h)*

DO0THHR and DO0THLR are considered as 6 continuous byte register, holding the routing headers and message tags for the OS Link protocol which will be sent at the beginning of each packet transfer. The organisation of header bytes is little endian. Variable lengths of header are allowed. Note that the MSB of each header byte must be a '1', except the last header byte which is a '0'. This register holds the lower 4 bytes of the header. The register is read-only when the channel is activated.

Word bit	Name	R/W	Description
7:0	Header 1	R/W	Carries value of header byte 1.
15:8	Header 2	R/W	Carries value of header byte 2.
23:16	Header 3	R/W	Carries value of header byte 3.
31:24	Header 4	R/W	Carries value of header byte 4.

DO0TAR: *DMA OS 0 Transmitter Address Register: (offset 98h)*

This register contains the SDRAM address of the DMA transfer. It's the address of the source of data to transmit down the OS link. It is loaded with the start address of the transfer and is updated internally after each SDRAM transaction as the transfer proceeds. The address value needs to be word aligned. Bit 1 and bit 0 of the address will be ignored. This register is read-only when the channel is activated.

Word bit	Name	R/W	Description
1:0	-		
23:2	Source SDRAM address	R/W	Contains the source address within the SDRAM for the outgoing DMA transfer. The value is updated internally as the DMA operation progresses.
31:24	-		

DO0TLR: *DMA OS 0 Transmitter Length Register: (offset 9Ch)*

This register contains the length of message in bytes that need to be transmitted. The value can be byte boundary. This register is read-only (except RST) when the channel is activated.

Word bit	Name	R/W	Description
15:0	Byte count	R/W	Indicates the number of bytes to be transferred from SDRAM. It is updated internally after each read as the DMA operation progresses.
16	Channel active (ACTV)	R	When '0', channel not active. When '1', channel enabled to transmit message. Bit resets when DMA transfer is done.
17	Channel packet active (PACTV)	R	When '0', no packet is active. When '1', the DMA channel is active transmitting a packet.
18	Channel done (DONE)	R/W1C	When the DMA transfer is done, this bit is set to '1' internally and can interrupt the CPU if enabled in FER or IER. Write of '1' to this bit will reset the bit.
21:19	-		
22	Reset channel (RST)	W	When '1', the transmitter channel will be reset to idle state. The bit will be cleared automatically after the reset.
23	Physical channel connected (PHYC)	R/W	This bit indicates which physical OS Link does the DMA transmitter channel allocated to.
31:24	-		

DO1RHR: *DMA OS 1 Receiver Header Register: (offset A4h)*

This register contains message tags for the comparison of incoming OS Link message. Either 1 or 2 message tags are allowed. This register is read-only when the channel is activated.

Word bit	Name	R/W	Description
7:0	Message tag 1	R	This byte carries message tag 1.
14:8	Message tag 2	R	An MSB of '1' in message tag 1 indicates this field carries a valid message tag 2. If not, this byte is ignored for comparison. (Note that bit 15 is tied down to zero since it's always zero according to the protocol)
31:15	-		

DO1RAR: *DMA OS 1 Receiver Address Register: (offset A8h)*

This register contains the SDRAM destination address for the DMA transfer. The register is loaded with the address at the start of the transfer and is updated internally after each SDRAM transaction as the transfer proceeds. The address value needs to be word aligned. Bit 1 and bit 0 of the address will be ignored. This register is read-only when the channel is activated.

Word bit	Name	R/W	Description
1:0	-		
23:2	Destination SDRAM address	R/W	Contains the destination address within the SDRAM for the incoming DMA transfer. It's updated internally as the DMA operation progresses.
31:24	-		

DO1RLR: *DMA OS 1 Receiver Length Register: (offset ACh)*

This register contains the length of message in bytes that needs to be received. The value must be at word boundary. Bit 1 and bit 0 of the length value will be ignored. This register is read-only (except DACTV and RST) when channel is activated.

Word bit	Name	R/W	Description
1:0	-		
15:2	Word count	R/W	Indicates the number of words in the message to be transferred to SDRAM. It's updated internally after each write as the DMA operation progresses.
16	Channel active (ACTV)	R	When '0', channel not active. When '1', channel enabled to receive message. Bit resets when DMA transfer is done.
17	Channel packet active (PACTV)	R	When '0', no packet is active. When '1', the DMA channel is active receiving a packet.
18	Channel done (DONE)	R/W1C	When the DMA transfer is done, this bit is set to '1' internally and can interrupt the CPU if enabled in FER or IER. Write of '1' to this bit will reset the bit.
19	Channel error (ERR)	R	When the actual incoming message length is different from the expected one, this bit will be set to indicate error. Early or

			late termination can be determined by the byte count value. Write of '1' to this bit will reset the bit.
20	Flush channel (FLSH)	R/W	This bit is used to flush the incoming data without transferring it to the memory. Byte counts and address pointer unchanged during flush.
21	Deactivate channel (DACTV)	W	This bit is used to deactivate the receiver channel, which only effective when ACTV = '1' and PACTV = '0'. Please note that a read of PACTV = '0' followed by a write of DACTV = '1' doesn't mean receiver channel definitely be deactivated, since PACTV might just changed to 1 after the read.
22	Reset channel (RST)	W	When '1', the receiver channel will be reset to idle state. The bit will be cleared automatically after the reset.
23	Physical channel connected (PHYC)	R	This bit indicates which physical OS Link does the DMA receiver channel allocated to.
31:24	-		

DO1THHR: *DMA OS 1 Transmitter Header High Register: (offset B0h)*

DO1THHR and DO1THLR are considered as 6 continuous byte register, holding the routing headers and message tags for the OS Link protocol which will be sent at the beginning of each packet transfer. The organisation of header bytes is little endian. Variable lengths of header are allowed. Note that the MSB of each header byte must be a '1', except the last header byte which is a '0'. This register holds the higher 2 bytes of the header. The register is read-only when the channel is activated.

Word bit	Name	R/W	Description
7:0	Header 5	R/W	Carries value of header byte 5.
14:8	Header 6	R/W	Carries value of header byte 6. (Note that bit 15 is tied down to zero since it's always zero according to the protocol)
31:15	-		

DO1THLR: *DMA OS 1 Transmitter Header Low Register: (offset B4h)*

DO1THHR and DO1THLR are considered as 6 continuous byte register, holding the routing headers and message tags for the OS Link protocol which will be sent at the beginning of each packet transfer. The organisation of header bytes is little endian. Variable lengths of header are allowed. Note that the MSB of each header byte must be a '1', except the last header byte which is a '0'. This register holds the lower 4 bytes of the header. The register is read-only when the channel is activated.

Word bit	Name	R/W	Description
7:0	Header 1	R/W	Carries value of header byte 1.
15:8	Header 2	R/W	Carries value of header byte 2.
23:16	Header 3	R/W	Carries value of header byte 3.
31:24	Header 4	R/W	Carries value of header byte 4.

DO1TAR: *DMA OS 1 Transmitter Address Register: (offset B8h)*

This register contains the SDRAM address of the DMA transfer. It's the address of the source of data to transmit down the OS link. It's loaded with the start address of the transfer and is updated internally after each SDRAM transaction as the transfer proceeds. The address value needs to be word aligned. Bit 1 and bit 0 of the address will be ignored. This register is read-only when the channel is activated.

Word bit	Name	R/W	Description
1:0	-		
23:2	Source SDRAM address	R/W	Contains the source address within the SDRAM for the outgoing DMA transfer. The value is updated internally as the DMA operation progresses.
31:24	-		

DO1TLR: *DMA OS 1 Transmitter Length Register: (offset BCh)*

This register contains the length of message in bytes that need to be transmitted. The value can be byte boundary. This register is read-only (except RST) when the channel is activated.

Word bit	Name	R/W	Description
15:0	Byte count	R/W	Indicates the number of bytes to be transferred from SDRAM. It's updated internally after each read as the DMA operation progresses.
16	Channel active (ACTV)	R	When '0', channel not active. When '1', channel enabled to transmit message. Bit resets when DMA transfer is done.
17	Channel packet active (PACTV)	R	When '0', no packet is active. When '1', the DMA channel is active transmitting a packet.
18	Channel done (DONE)	R/W1C	When the DMA transfer is done, this bit is set to '1' internally and can interrupt the CPU if enabled in FER or IER. Write of '1' to this bit will reset the bit.
21:19	-		
22	Reset channel (RST)	W	When '1', the transmitter channel will be reset to idle state. The bit will be cleared automatically after the reset.
23	Physical channel connected (PHYC)	R/W	This bit indicates which physical OS Link does the DMA transmitter channel allocated to.
31:24	-		

B. TIMING SPECIFICATION OF SARNIC

Symbol	Parameter	Time (ns)	Freq (MHz)	Formula
General				
Tskew	Signal skew & wire delay	1.00		
SA-110 CPU				
Tmck	MClk and nMClk skew	1.00		
Tmsd	nMREQ output delay	10.00		
Tws	nWAIT input setup	1.00		
Taddr1	Addr output delay	13.00		
	(APE high)			
Taddr1seq	Addr sequence output delay	9.00		
	(APE high)			
Tdout(C)	Data output delay	10.00		
Tdbe(C)	DBE to data output delay	9.00		
SDRAM				
Tsdd	SDRAM data input setup	3.00		
SARNIC CPLD				
Tctrlout	SD Ctrl output delay	19.20	41.32	Tclk = Tskew+Tmck+Tsdd+Tctrlout
Tdqmout1	SD DQM[3:0] output delay	14.50	50.85	1.5 X Tclk = 2Tskew+Tmck+Taddr1seq+Tsdd+Tdqmout1
	(timed from CPU Addr in)			
Tdqmout2	SD DQM[3:0] output delay	18.40	42.74	Tclk = Tskew+Tmck+Tsdd+Tdqmout2
	(timed from rising Clk in)			
Tmris	CPU nMREQ input setup	9.40	49.02	Tclk = Tskew+Tmsd+Tmris
Twout1	CPU nWAIT output delay	18.00	48.39	1.5Tclk = 2Tskew+Tmsd+Tws+Twout1
	(timed from CPU mREQ in)			
Twout2	CPU nWAIT output delay	31.40	44.91	1.5Tclk = Tskew+Tws+Twout2
	(timed from rising Clk in)			
Tais	CPU Addr input setup	6.40	49.02	Tclk = Tskew+Taddr1+Tais
Tdbe	CPU DBE output delay	13.00	76.92	Tclk = Tskew+Tdbe(C)-Tdout(C)+Tdbe
Tdis	Data input setup	14.10	39.84	Tclk = Tskew+Tdout(C)+Tdis
Tdout	Data output delay	15.10	49.75	Tclk = Tskew+Tmck+Tsdd+Tdout

C. SCHEMATIC DIAGRAMS OF SARNET

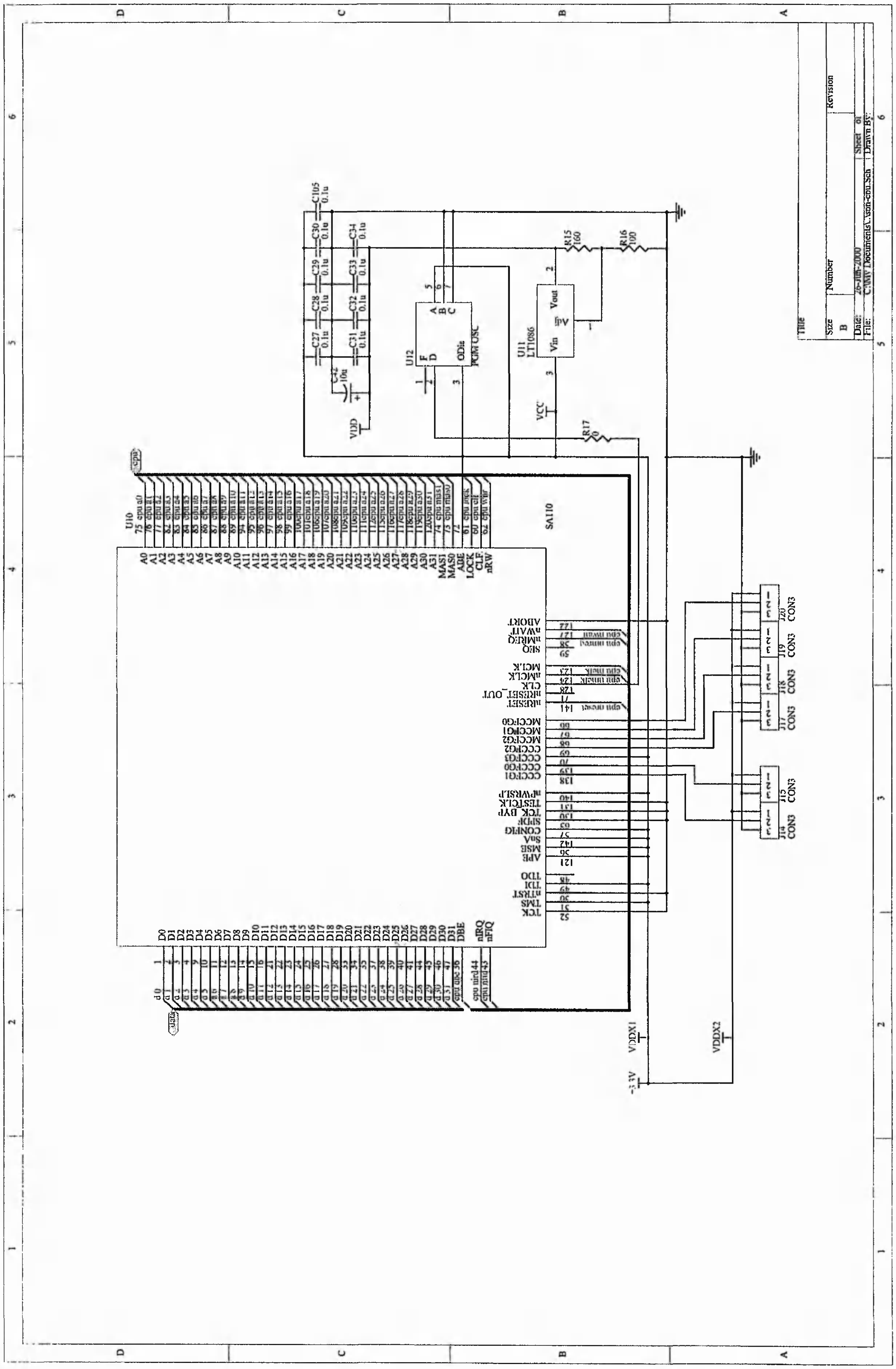
Sheet 1 – The StrongARM SA-110 microprocessor

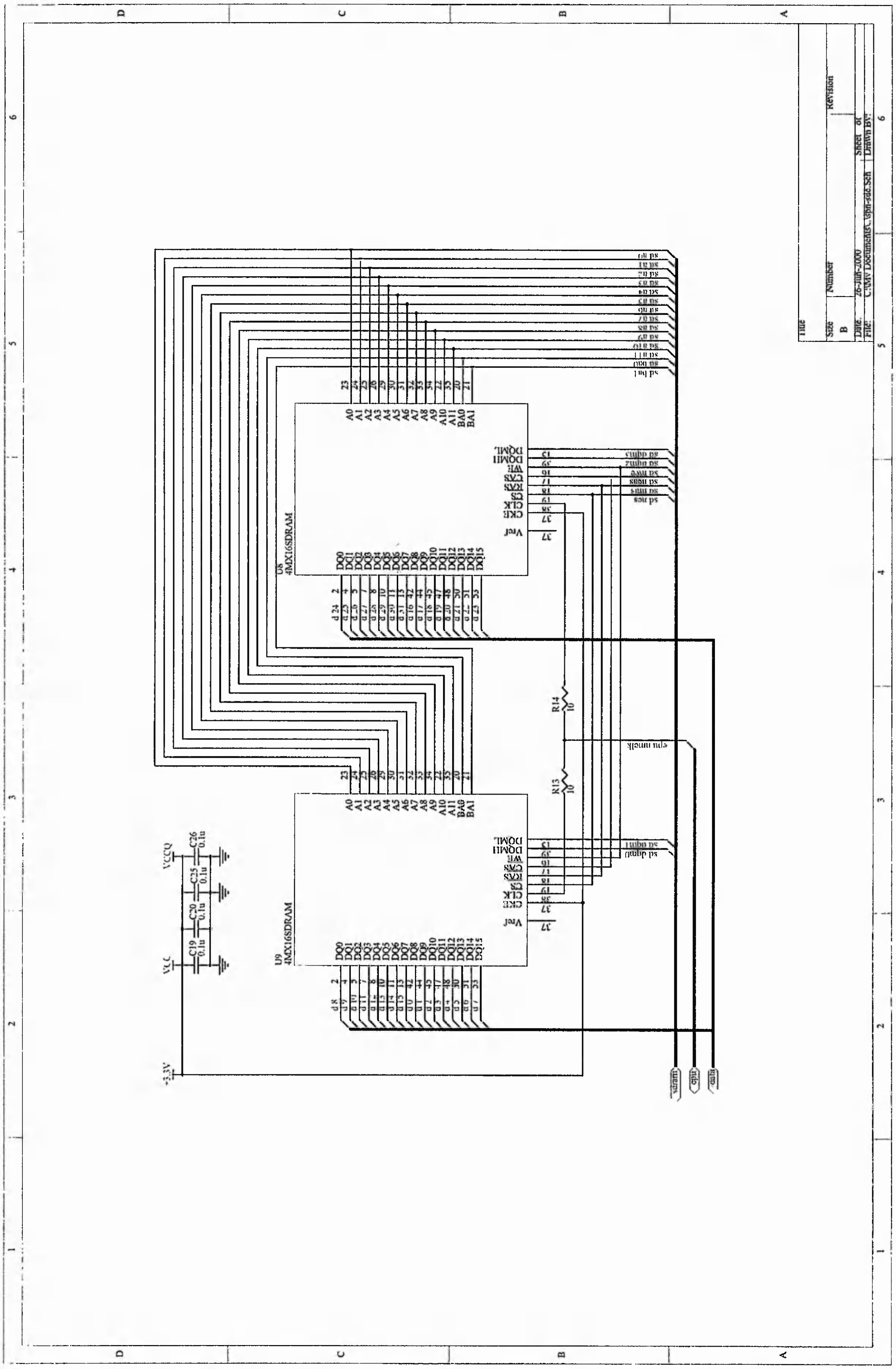
Sheet 2 – 4M X 16 SDRAM chips

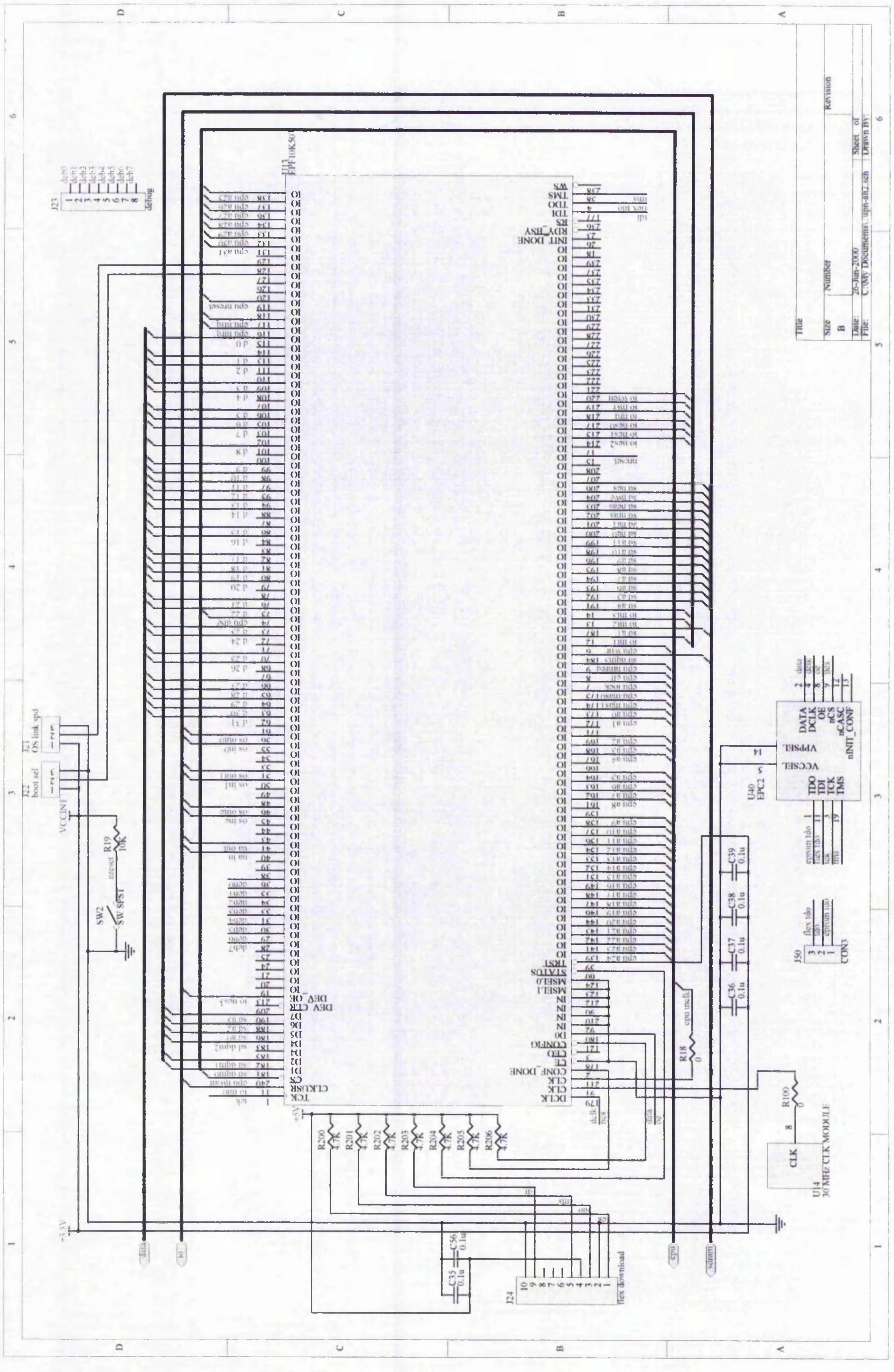
Sheet 3 – The SARNIC (Altera EPF10K50VRC240-3) CPLD

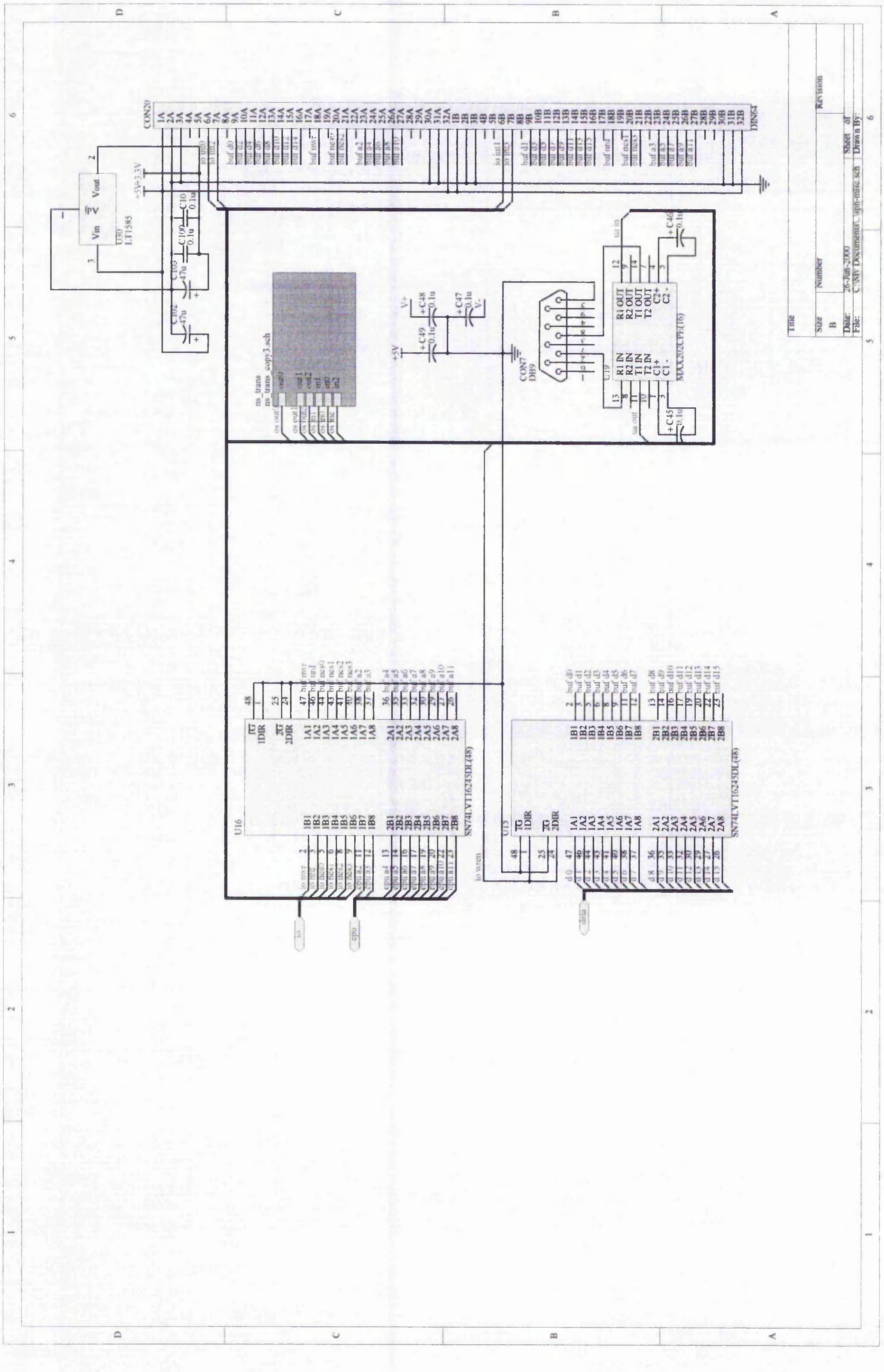
Sheet 4 – Miscallaneous (Data buffers, I/O buffers, and RS-232 transceiver)

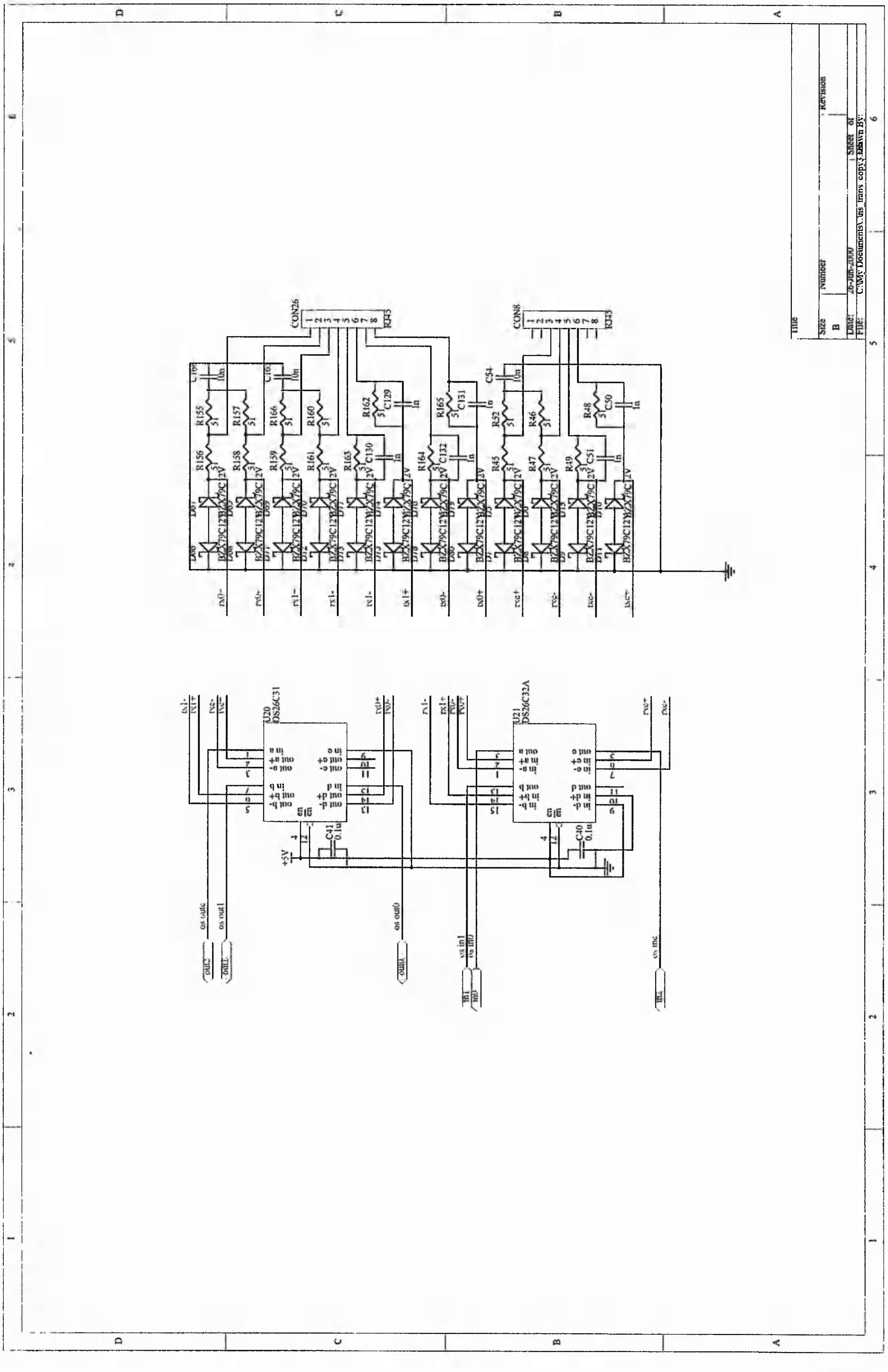
Sheet 5 – OS Link differential transceiver circuitry


U10
SA110
U12
PGM OSC
U11
LT1086
Vin
Vout
Adj
C27 0.1u
C28 0.1u
C29 0.1u
C30 0.1u
C105 0.1u
C31 0.1u
C32 0.1u
C33 0.1u
C34 0.1u
C42 10u
R15 160
R16 100
R17 0
VDD
VCC
VDDX1
VDDX2
+3.3V
J14 CON3
J15 CON3
J17 CON3
J18 CON3
J19 CON3
J20 CON3
cpu
data
Title
Size
B
Number
Revision
Date: 26-Jun-2000
File: C:\MW\Documents\...\sarn-cpu.Sch
Sheet of
Drawn By:


U8
4MX16SDRAM
U9
4MX16SDRAM
R14
10
R13
10
C19
0.1u
C20
0.1u
C25
0.1u
C26
0.1u
VCC
VCCQ
+3.3V
cpu mmclk
sdram
cpu
data
Title
Size
B
Number
Revision
Date: 26-Jun-2000
Sheet of
Drawn By:




CON20
DIN64
U30
LT1585
Vin
Vout
+5V
+3.3V
C100
C101
C102
C10
C47
C48
C49
C45
C46
ns_trans
ns_trans_copy3.sch
CON7
DB9
U19
MAX202CPE(16)
R1 IN
R2 IN
T1 IN
T2 IN
R1 OUT
R2 OUT
T1 OUT
T2 OUT
U16
U15
SN74LVT16245DL(48)
1DIR
2DIR
io
cpu
data
io wren
Title
Size
B
Number
Revision
Date:
26-Jun-2000
Sheet of
File:
Drawn By:


CON26
RJ45
CON8
RJ45
U20
DS26C31
U21
DS26C32A
+5V
C41
0.1u
C40
0.1u
BZX79C12
os out0
os out1
os outc
os in0
os in1
os inc
Title
Size
B
Number
Revision
Date:
26-Jun-2000
Sheet of
File:
C:\My Documents\..\us trans copy.Sch
Drawn By:

D. RESULTS FOR DIFFERENTIAL TRANSMISSION TESTS

Basic Configuration			
Length	Bit Height		
(m)	44 Mbps	32 Mbps	20 Mbps
10	73.16%	77.37%	81.58%
20	69.47%	73.16%	76.84%
30	63.68%	67.89%	72.11%
40	55.79%	58.42%	64.21%
50	47.89%	53.16%	61.05%
60	41.05%	46.84%	54.21%
70 .	34.74%	42.63%	50.53%
80	27.37%	37.37%	46.32%
90	22.11%	32.11%	41.05%
100	18.95%	26.32%	35.79%

Basic Configuration			
Length	Bit width		
(m)	44 Mbps	32 Mbps	20 Mbps
10	89.32%	93.44%	94.00%
20	88.00%	93.12%	93.20%
30	85.80%	93.12%	93.20%
40	83.60%	90.88%	92.80%
50	81.84%	87.36%	92.40%
60	76.12%	85.76%	90.80%
70	70.84%	82.88%	88.80%
80	66.88%	77.44%	85.60%
90	59.40%	73.60%	84.00%
100	50.60%	67.52%	80.40%

Enhanced Configuration			
Length	Bit height		
(m)	44 Mbps	32 Mbps	20 Mbps
0.1	75.79%	75.79%	75.79%
10	74.21%	74.21%	74.21%
20	71.05%	71.05%	73.16%
30	69.47%	69.47%	71.05%
40	68.42%	68.42%	69.47%
50	66.32%	66.84%	67.89%
60	63.16%	64.21%	65.79%
70	61.58%	62.63%	63.68%
80	57.89%	59.47%	62.11%
90	52.11%	56.84%	57.89%
100	44.74%	53.68%	55.79%

Enhanced Configuration			
Length	Bit width		
(m)	44 Mbps	32 Mbps	20 Mbps
0.1	79.20%	87.68%	91.60%
10	79.20%	87.36%	92.00%
20	80.08%	88.96%	91.60%
30	79.64%	88.64%	91.60%
40	80.52%	89.60%	91.20%
50	80.52%	89.60%	92.00%
60	80.52%	89.60%	92.00%
70	81.84%	90.24%	91.20%
80	82.28%	89.60%	91.60%
90	82.28%	88.96%	91.20%
100	81.40%	89.28%	90.80%

E. RESULTS FOR SARNIC SIMULATION AND HARDWARE TESTS

Simulation (10 Mbps)								
Message Size	Uni 1 DMA		Uni 2 DMA		Bi 2 DMA		Bi 4 DMA	
(byte)	Time (us)	BW (MB/s)	Time (us)	BW (MB/s)	Time (us)	BW (MB/s)	Time (us)	BW (MB/s)
4	9.50	0.42	5.00	0.80	8.00	0.50	4.00	1.00
8	14.00	0.57	7.25	1.10	11.00	0.73	5.50	1.45
12	18.50	0.65	9.50	1.26	13.50	0.89	6.75	1.78
16	23.00	0.70	11.75	1.36	16.50	0.97	8.25	1.94
20	27.00	0.74	14.00	1.43	19.00	1.05	9.50	2.11
24	31.50	0.76	16.00	1.50	21.50	1.12	10.75	2.23
28	36.00	0.78	18.25	1.53	24.00	1.17	12.00	2.33
32	40.50	0.79	20.50	1.56	27.00	1.19	13.50	2.37
36	45.00	0.80	22.75	1.58	29.50	1.22	14.75	2.44
40	49.00	0.82	25.00	1.60	32.50	1.23	16.25	2.46
44	53.50	0.82	27.25	1.61	35.00	1.26	17.50	2.51
48	58.00	0.83	29.25	1.64	37.50	1.28	18.75	2.56
52	62.50	0.83	31.50	1.65	40.00	1.30	20.00	2.60
56	67.00	0.84	33.50	1.67	42.50	1.32	21.25	2.64
60	71.00	0.85	35.75	1.68	45.50	1.32	22.75	2.64
64	75.50	0.85	38.00	1.68	48.00	1.33	24.00	2.67
68	80.00	0.85	40.25	1.69	51.00	1.33	25.50	2.67
72	84.50	0.85	42.50	1.69	53.50	1.35	26.75	2.69
76	89.00	0.85	44.75	1.70	56.50	1.35	28.25	2.69
80	93.00	0.86	47.00	1.70	59.00	1.36	29.50	2.71
84	97.50	0.86	49.00	1.71	61.50	1.37	30.75	2.73
88	102.00	0.86	51.25	1.72	64.00	1.38	32.00	2.75
92	106.50	0.86	53.50	1.72	67.00	1.37	33.50	2.75
96	111.00	0.86	55.75	1.72	69.50	1.38	34.75	2.76
100	115.00	0.87	58.00	1.72	72.50	1.38	36.25	2.76
104	119.50	0.87	60.00	1.73	75.00	1.39	37.50	2.77
108	124.00	0.87	62.25	1.73	77.50	1.39	38.75	2.79
112	128.50	0.87	64.50	1.74	80.00	1.40	40.00	2.80
116	133.00	0.87	66.50	1.74	82.50	1.41	41.25	2.81
120	137.00	0.88	68.75	1.75	85.50	1.40	42.75	2.81
124	141.50	0.88	71.25	1.74	88.00	1.41	44.00	2.82
128	146.00	0.88	73.25	1.75	91.00	1.41	45.50	2.81
132	150.50	0.88	75.25	1.75	93.50	1.41	46.75	2.82
136	155.00	0.88	77.50	1.75	96.50	1.41	48.25	2.82
140	159.00	0.88	79.75	1.76	99.00	1.41	49.50	2.83
144	163.50	0.88	82.00	1.76	101.50	1.42	50.75	2.84
148	168.00	0.88	84.25	1.76	104.00	1.42	52.00	2.85
152	172.50	0.88	86.50	1.76	107.00	1.42	53.50	2.84

156	177.00	0.88	88.75	1.76	109.50	1.42	54.75	2.85
160	181.00	0.88	91.00	1.76	112.50	1.42	56.25	2.84
164	185.50	0.88	93.00	1.76	115.00	1.43	57.50	2.85
168	190.00	0.88	95.25	1.76	117.50	1.43	58.75	2.86
172	194.50	0.88	97.50	1.76	120.00	1.43	60.00	2.87
176	199.00	0.88	99.75	1.76	122.50	1.44	61.25	2.87
180	203.00	0.89	102.00	1.76	125.50	1.43	62.75	2.87
184	207.50	0.89	104.00	1.77	128.00	1.44	64.00	2.88
188	212.00	0.89	106.25	1.77	131.00	1.44	65.50	2.87
192	216.50	0.89	108.50	1.77	133.50	1.44	66.75	2.88
196	221.00	0.89	110.50	1.77	136.50	1.44	68.25	2.87
200	225.00	0.89	112.75	1.77	139.00	1.44	69.50	2.88
204	229.50	0.89	115.00	1.77	141.50	1.44	70.75	2.88
208	234.00	0.89	117.25	1.77	144.00	1.44	72.00	2.89
212	238.50	0.89	119.25	1.78	147.00	1.44	73.50	2.88
216	243.00	0.89	121.75	1.77	149.50	1.44	74.75	2.89
220	247.00	0.89	123.75	1.78	152.50	1.44	76.25	2.89
224	251.50	0.89	126.00	1.78	155.00	1.45	77.50	2.89
228	256.00	0.89	128.25	1.78	157.50	1.45	78.75	2.90
232	260.50	0.89	130.50	1.78	160.00	1.45	80.00	2.90
236	265.00	0.89	132.75	1.78	162.50	1.45	81.25	2.90
240	269.00	0.89	135.00	1.78	165.50	1.45	82.75	2.90
244	273.50	0.89	137.00	1.78	168.00	1.45	84.00	2.90
248	278.00	0.89	139.25	1.78	171.00	1.45	85.50	2.90
252	282.50	0.89	141.50	1.78	173.50	1.45	86.75	2.90
256	287.00	0.89	143.50	1.78	176.50	1.45	88.25	2.90
260	295.00	0.88	147.50	1.76	180.50	1.44	90.25	2.88
264	299.00	0.88	149.75	1.76	183.50	1.44	91.75	2.88
268	303.50	0.88	152.00	1.76	186.00	1.44	93.00	2.88
272	308.00	0.88	154.00	1.77	188.50	1.44	94.25	2.89
276	312.00	0.88	156.25	1.77	191.50	1.44	95.75	2.88
280	317.00	0.88	158.50	1.77	194.00	1.44	97.00	2.89
284	321.00	0.88	160.75	1.77	197.00	1.44	98.50	2.88
288	325.50	0.88	163.00	1.77	199.50	1.44	99.75	2.89
292	330.00	0.88	165.25	1.77	202.00	1.45	101.00	2.89
296	334.00	0.89	167.25	1.77	205.00	1.44	102.50	2.89
300	339.00	0.88	169.50	1.77	207.00	1.45	103.50	2.90
304	343.00	0.89	171.75	1.77	210.50	1.44	105.25	2.89
308	347.50	0.89	173.75	1.77	213.00	1.45	106.50	2.89
312	352.00	0.89	176.00	1.77	215.50	1.45	107.75	2.90
316	356.00	0.89	178.50	1.77	218.00	1.45	109.00	2.90
320	361.00	0.89	180.50	1.77	220.50	1.45	110.25	2.90
324	365.00	0.89	182.50	1.78	223.50	1.45	111.75	2.90
328	369.50	0.89	184.75	1.78	226.00	1.45	113.00	2.90
332	374.00	0.89	187.00	1.78	228.50	1.45	114.25	2.91
336	378.00	0.89	189.25	1.78	231.50	1.45	115.75	2.90
340	383.00	0.89	191.50	1.78	234.00	1.45	117.00	2.91

344	387.00	0.89	193.75	1.78	237.00	1.45	118.50	2.90
348	391.50	0.89	196.00	1.78	239.50	1.45	119.75	2.91
352	396.00	0.89	198.25	1.78	242.00	1.45	121.00	2.91
356	400.00	0.89	200.25	1.78	245.00	1.45	122.50	2.91
360	405.00	0.89	202.50	1.78	247.50	1.45	123.75	2.91
364	409.00	0.89	204.75	1.78	250.50	1.45	125.25	2.91
368	413.50	0.89	206.75	1.78	253.00	1.45	126.50	2.91
372	418.00	0.89	209.00	1.78	255.50	1.46	127.75	2.91
376	422.00	0.89	211.50	1.78	258.00	1.46	129.00	2.91
380	427.00	0.89	213.50	1.78	260.50	1.46	130.25	2.92
384	431.00	0.89	215.50	1.78	263.50	1.46	131.75	2.91
388	435.50	0.89	217.75	1.78	266.00	1.46	133.00	2.92
392	440.00	0.89	220.00	1.78	268.50	1.46	134.25	2.92
396	444.00	0.89	222.25	1.78	271.50	1.46	135.75	2.92
400	449.00	0.89	224.50	1.78	274.00	1.46	137.00	2.92
404	453.00	0.89	226.75	1.78	277.00	1.46	138.50	2.92
408	457.50	0.89	229.00	1.78	279.50	1.46	139.75	2.92
412	462.00	0.89	231.25	1.78	282.00	1.46	141.00	2.92
416	466.00	0.89	233.25	1.78	285.00	1.46	142.50	2.92
420	471.00	0.89	235.50	1.78	287.50	1.46	143.75	2.92
424	475.00	0.89	237.75	1.78	290.50	1.46	145.25	2.92
428	479.50	0.89	239.75	1.79	293.00	1.46	146.50	2.92
432	484.00	0.89	242.00	1.79	295.50	1.46	147.75	2.92
436	488.00	0.89	244.50	1.78	298.00	1.46	149.00	2.93
440	493.00	0.89	246.50	1.78	300.50	1.46	150.25	2.93
444	497.00	0.89	248.50	1.79	303.50	1.46	151.75	2.93
448	501.50	0.89	250.75	1.79	306.00	1.46	153.00	2.93
452	506.00	0.89	253.00	1.79	308.50	1.47	154.25	2.93
456	510.00	0.89	255.25	1.79	311.50	1.46	155.75	2.93
460	515.00	0.89	257.50	1.79	314.00	1.46	157.00	2.93
464	519.00	0.89	259.75	1.79	317.00	1.46	158.50	2.93
468	523.50	0.89	262.00	1.79	319.50	1.46	159.75	2.93
472	528.00	0.89	264.25	1.79	322.00	1.47	161.00	2.93
476	532.00	0.89	266.25	1.79	325.00	1.46	162.50	2.93
480	537.00	0.89	268.50	1.79	327.50	1.47	163.75	2.93
484	541.00	0.89	270.75	1.79	330.50	1.46	165.25	2.93
488	545.50	0.89	273.00	1.79	333.00	1.47	166.50	2.93
492	550.00	0.89	275.00	1.79	335.50	1.47	167.75	2.93
496	554.00	0.90	277.50	1.79	338.00	1.47	169.00	2.93
500	559.00	0.89	279.50	1.79	340.50	1.47	170.25	2.94
504	563.00	0.90	281.50	1.79	343.50	1.47	171.75	2.93
508	567.50	0.90	283.75	1.79	346.00	1.47	173.00	2.94
512	572.00	0.90	286.00	1.79	348.50	1.47	174.25	2.94

Simulation (20 Mbps)								
Message Size	Uni 1 DMA		Uni 2 DMA		Bi 2 DMA		Bi 4 DMA	
(byte)	Time (us)	BW (MB/s)	Time (us)	BW (MB/s)	Time (us)	BW (MB/s)	Time (us)	BW (MB/s)
4	5.50	0.73	3.00	1.33	4.50	0.89	2.25	1.78
8	8.00	1.00	4.00	2.00	6.00	1.33	2.75	2.91
12	10.00	1.20	5.00	2.40	7.50	1.60	3.50	3.43
16	12.00	1.33	6.25	2.56	8.50	1.88	4.50	3.56
20	14.50	1.38	7.25	2.76	10.50	1.90	5.00	4.00
24	17.00	1.41	8.50	2.82	11.50	2.09	5.75	4.17
28	18.50	1.51	9.50	2.95	13.00	2.15	6.25	4.48
32	21.00	1.52	10.50	3.05	14.00	2.29	7.00	4.57
36	23.50	1.53	11.75	3.06	15.50	2.32	8.00	4.50
40	25.50	1.57	12.75	3.14	17.00	2.35	8.50	4.71
44	27.50	1.60	14.00	3.14	18.50	2.38	9.25	4.76
48	29.50	1.63	15.00	3.20	19.50	2.46	9.75	4.92
52	31.50	1.65	16.00	3.25	21.00	2.48	10.50	4.95
56	34.00	1.65	17.25	3.25	23.00	2.43	11.50	4.87
60	36.00	1.67	18.25	3.29	24.00	2.50	12.00	5.00
64	38.50	1.66	19.50	3.28	25.50	2.51	12.75	5.02
68	41.00	1.66	20.50	3.32	27.00	2.52	13.25	5.13
72	43.00	1.67	21.50	3.35	28.50	2.53	14.00	5.14
76	45.00	1.69	22.75	3.34	29.50	2.58	15.00	5.07
80	47.50	1.68	23.75	3.37	31.50	2.54	15.50	5.16
84	50.00	1.68	25.00	3.36	32.50	2.58	16.25	5.17
88	51.50	1.71	26.00	3.38	34.00	2.59	16.75	5.25
92	54.00	1.70	27.00	3.41	35.00	2.63	17.50	5.26
96	56.50	1.70	28.25	3.40	36.50	2.63	18.50	5.19
100	58.50	1.71	29.25	3.42	38.00	2.63	19.00	5.26
104	60.50	1.72	30.50	3.41	39.50	2.63	19.75	5.27
108	62.50	1.73	31.50	3.43	40.50	2.67	20.25	5.33
112	64.50	1.74	32.50	3.45	42.00	2.67	21.00	5.33
116	67.00	1.73	33.75	3.44	44.00	2.64	22.00	5.27
120	69.00	1.74	34.75	3.45	45.00	2.67	22.50	5.33
124	71.50	1.73	36.00	3.44	46.50	2.67	23.25	5.33
128	74.00	1.73	37.00	3.46	48.00	2.67	24.00	5.33
132	76.00	1.74	38.00	3.47	49.50	2.67	24.75	5.33
136	78.00	1.74	39.25	3.46	50.50	2.69	25.25	5.39
140	80.50	1.74	40.25	3.48	52.50	2.67	26.00	5.38
144	83.00	1.73	41.50	3.47	53.50	2.69	26.75	5.38
148	84.50	1.75	42.75	3.46	55.00	2.69	27.50	5.38
152	87.00	1.75	43.75	3.47	56.00	2.71	28.25	5.38
156	89.50	1.74	44.75	3.49	57.50	2.71	28.75	5.43
160	91.50	1.75	45.75	3.50	59.00	2.71	29.50	5.42
164	93.50	1.75	47.00	3.49	60.50	2.71	30.25	5.42
168	95.50	1.76	48.25	3.48	61.50	2.73	31.00	5.42

172	97.50	1.76	49.25	3.49	63.00	2.73	31.75	5.42
176	100.00	1.76	50.25	3.50	65.00	2.71	32.25	5.46
180	102.00	1.76	51.25	3.51	66.00	2.73	33.25	5.41
184	104.50	1.76	52.50	3.50	67.50	2.73	33.75	5.45
188	107.00	1.76	53.75	3.50	69.00	2.72	34.50	5.45
192	109.00	1.76	54.75	3.51	70.50	2.72	35.25	5.45
196	111.00	1.77	55.75	3.52	71.50	2.74	35.75	5.48
200	113.50	1.76	56.75	3.52	73.50	2.72	36.75	5.44
204	116.00	1.76	58.00	3.52	74.50	2.74	37.25	5.48
208	117.50	1.77	59.25	3.51	76.00	2.74	38.00	5.47
212	120.00	1.77	60.25	3.52	77.00	2.75	38.75	5.47
216	122.50	1.76	61.25	3.53	78.50	2.75	39.25	5.50
220	124.50	1.77	62.25	3.53	80.00	2.75	40.00	5.50
224	126.50	1.77	63.25	3.54	81.50	2.75	40.75	5.50
228	128.50	1.77	64.50	3.53	83.00	2.75	41.50	5.49
232	130.50	1.78	65.75	3.53	84.50	2.75	42.25	5.49
236	133.00	1.77	66.75	3.54	85.50	2.76	42.75	5.52
240	135.00	1.78	67.75	3.54	87.00	2.76	43.50	5.52
244	137.50	1.77	68.75	3.55	88.50	2.76	44.25	5.51
248	140.00	1.77	70.00	3.54	90.00	2.76	45.00	5.51
252	142.00	1.77	71.25	3.54	91.50	2.75	45.75	5.51
256	144.00	1.78	72.25	3.54	92.50	2.77	46.25	5.54
260	148.00	1.76	74.25	3.50	95.00	2.74	47.50	5.47
264	150.00	1.76	75.25	3.51	96.50	2.74	48.50	5.44
268	152.50	1.76	76.25	3.51	98.00	2.73	49.00	5.47
272	154.50	1.76	77.25	3.52	99.00	2.75	49.75	5.47
276	157.00	1.76	78.50	3.52	100.50	2.75	50.50	5.47
280	159.00	1.76	79.75	3.51	102.00	2.75	51.25	5.46
284	161.00	1.76	80.75	3.52	103.50	2.74	51.75	5.49
288	163.50	1.76	81.75	3.52	105.00	2.74	52.50	5.49
292	165.50	1.76	83.00	3.52	106.00	2.75	53.00	5.51
296	167.50	1.77	84.00	3.52	107.50	2.75	53.75	5.51
300	170.00	1.76	85.25	3.52	109.00	2.75	54.50	5.50
304	172.50	1.76	86.25	3.52	110.50	2.75	55.25	5.50
308	174.00	1.77	87.25	3.53	112.00	2.75	56.00	5.50
312	176.50	1.77	88.50	3.53	113.00	2.76	56.75	5.50
316	179.00	1.77	89.50	3.53	114.50	2.76	57.50	5.50
320	181.00	1.77	90.75	3.53	116.00	2.76	58.00	5.52
324	183.00	1.77	91.75	3.53	117.50	2.76	58.75	5.51
328	185.50	1.77	92.75	3.54	119.00	2.76	59.50	5.51
332	187.50	1.77	94.00	3.53	120.00	2.77	60.00	5.53
336	190.00	1.77	95.00	3.54	121.50	2.77	60.75	5.53
340	192.00	1.77	96.25	3.53	123.00	2.76	61.50	5.53
344	194.00	1.77	97.25	3.54	124.50	2.76	62.25	5.53
348	196.50	1.77	98.25	3.54	126.00	2.76	63.00	5.52
352	198.50	1.77	99.50	3.54	127.00	2.77	63.75	5.52
356	200.50	1.78	100.50	3.54	128.50	2.77	64.50	5.52

360	203.00	1.77	101.75	3.54	130.00	2.77	65.00	5.54
364	205.50	1.77	102.75	3.54	131.50	2.77	65.75	5.54
368	207.00	1.78	103.75	3.55	133.00	2.77	66.50	5.53
372	209.50	1.78	105.00	3.54	134.00	2.78	67.00	5.55
376	212.00	1.77	106.00	3.55	135.50	2.77	67.75	5.55
380	214.00	1.78	107.25	3.54	137.00	2.77	68.50	5.55
384	216.00	1.78	108.25	3.55	138.50	2.77	69.25	5.55
388	218.50	1.78	109.25	3.55	140.00	2.77	70.00	5.54
392	220.50	1.78	110.50	3.55	141.00	2.78	70.75	5.54
396	223.00	1.78	111.50	3.55	142.50	2.78	71.50	5.54
400	225.00	1.78	112.75	3.55	144.00	2.78	72.00	5.56
404	227.00	1.78	113.75	3.55	145.50	2.78	72.75	5.55
408	229.50	1.78	114.75	3.56	147.00	2.78	73.50	5.55
412	231.50	1.78	116.00	3.55	148.00	2.78	74.25	5.55
416	233.50	1.78	117.00	3.56	149.50	2.78	75.00	5.55
420	236.00	1.78	118.25	3.55	151.00	2.78	75.75	5.54
424	238.50	1.78	119.25	3.56	152.50	2.78	76.25	5.56
428	240.00	1.78	120.25	3.56	154.00	2.78	76.75	5.58
432	242.50	1.78	121.50	3.56	155.00	2.79	77.75	5.56
436	245.00	1.78	122.50	3.56	156.50	2.79	78.25	5.57
440	247.00	1.78	123.75	3.56	158.00	2.78	79.00	5.57
444	249.00	1.78	124.75	3.56	159.50	2.78	79.75	5.57
448	251.50	1.78	125.75	3.56	161.00	2.78	80.50	5.57
452	253.50	1.78	127.00	3.56	162.00	2.79	81.25	5.56
456	256.00	1.78	128.00	3.56	163.50	2.79	82.00	5.56
460	258.00	1.78	129.25	3.56	165.00	2.79	82.75	5.56
464	260.00	1.78	130.25	3.56	166.50	2.79	83.25	5.57
468	262.50	1.78	131.25	3.57	168.00	2.79	83.75	5.59
472	264.50	1.78	132.50	3.56	169.00	2.79	84.50	5.59
476	266.50	1.79	133.50	3.57	170.50	2.79	85.25	5.58
480	269.00	1.78	134.75	3.56	172.00	2.79	86.00	5.58
484	271.50	1.78	135.75	3.57	173.50	2.79	86.75	5.58
488	273.00	1.79	136.75	3.57	175.00	2.79	87.50	5.58
492	275.50	1.79	138.00	3.57	176.00	2.80	88.25	5.58
496	278.00	1.78	139.00	3.57	177.50	2.79	89.00	5.57
500	280.00	1.79	140.25	3.57	179.00	2.79	89.75	5.57
504	282.00	1.79	141.25	3.57	180.50	2.79	90.25	5.58
508	284.50	1.79	142.25	3.57	182.00	2.79	91.00	5.58
512	286.50	1.79	143.50	3.57	183.00	2.80	91.50	5.60

Hardware Tests (10 Mbps)								
Message Size	Uni 1 DMA		Uni 2 DMA		Bi 2 DMA		Bi 4 DMA	
(byte)	Time (us)	BW (MB/s)	Time (us)	BW (MB/s)	Time (us)	BW (MB/s)	Time (us)	BW (MB/s)
4	10.77	0.37	5.81	0.69	6.41	0.62	3.41	1.17
8	15.17	0.53	7.99	1.00	9.26	0.86	4.83	1.66
12	19.57	0.61	10.21	1.18	12.11	0.99	6.26	1.92
16	23.97	0.67	12.40	1.29	14.96	1.07	7.67	2.09
20	28.37	0.71	14.60	1.37	17.80	1.12	9.10	2.20
24	32.78	0.73	16.79	1.43	20.66	1.16	10.53	2.28
28	37.17	0.75	18.99	1.47	23.50	1.19	11.95	2.34
32	41.57	0.77	21.19	1.51	26.35	1.21	13.38	2.39
36	45.97	0.78	23.40	1.54	29.20	1.23	14.80	2.43
40	50.37	0.79	25.59	1.56	32.05	1.25	16.22	2.47
44	54.77	0.80	27.79	1.58	34.90	1.26	17.65	2.49
48	59.17	0.81	29.99	1.60	37.75	1.27	19.07	2.52
52	63.57	0.82	32.19	1.62	40.59	1.28	20.51	2.54
56	67.97	0.82	34.39	1.63	43.45	1.29	21.92	2.56
60	72.37	0.83	36.59	1.64	46.30	1.30	23.35	2.57
64	76.77	0.83	38.79	1.65	49.15	1.30	24.76	2.58
68	81.17	0.84	40.99	1.66	52.00	1.31	26.20	2.60
72	85.61	0.84	43.19	1.67	54.85	1.31	27.62	2.61
76	89.96	0.84	45.39	1.67	57.70	1.32	29.05	2.62
80	94.37	0.85	47.59	1.68	60.53	1.32	30.48	2.62
84	98.76	0.85	49.79	1.69	63.40	1.33	31.90	2.63
88	103.17	0.85	51.99	1.69	66.25	1.33	33.32	2.64
92	107.57	0.86	54.19	1.70	69.10	1.33	34.74	2.65
96	111.97	0.86	56.39	1.70	71.95	1.33	36.17	2.65
100	116.37	0.86	58.58	1.71	74.80	1.34	37.60	2.66
104	120.77	0.86	60.79	1.71	77.65	1.34	39.03	2.66
108	125.17	0.86	62.98	1.71	80.50	1.34	40.46	2.67
112	129.57	0.86	65.19	1.72	83.34	1.34	41.87	2.67
116	133.97	0.87	67.39	1.72	86.20	1.35	43.30	2.68
120	138.38	0.87	69.59	1.72	89.05	1.35	44.72	2.68
124	142.77	0.87	71.78	1.73	91.90	1.35	46.16	2.69
128	147.18	0.87	73.99	1.73	94.75	1.35	47.57	2.69
132	151.57	0.87	76.18	1.73	97.61	1.35	48.99	2.69
136	155.98	0.87	78.39	1.73	100.45	1.35	50.42	2.70
140	160.37	0.87	80.58	1.74	103.31	1.36	51.86	2.70
144	164.78	0.87	82.79	1.74	106.16	1.36	53.28	2.70
148	169.17	0.87	84.98	1.74	109.01	1.36	54.70	2.71
152	173.58	0.88	87.19	1.74	111.85	1.36	56.12	2.71
156	177.97	0.88	89.38	1.75	114.71	1.36	57.55	2.71
160	182.38	0.88	91.59	1.75	117.54	1.36	58.97	2.71
164	186.77	0.88	93.78	1.75	120.41	1.36	60.40	2.72
168	191.18	0.88	95.98	1.75	123.24	1.36	61.82	2.72

172	195.57	0.88	98.18	1.75	126.11	1.36	63.24	2.72
176	199.98	0.88	100.39	1.75	128.96	1.36	64.68	2.72
180	204.37	0.88	102.58	1.75	131.81	1.37	66.09	2.72
184	208.78	0.88	104.79	1.76	134.66	1.37	67.52	2.73
188	213.17	0.88	106.98	1.76	137.51	1.37	68.95	2.73
192	217.58	0.88	109.19	1.76	140.36	1.37	70.37	2.73
196	221.97	0.88	111.38	1.76	143.21	1.37	71.79	2.73
200	226.38	0.88	113.59	1.76	146.06	1.37	73.22	2.73
204	230.77	0.88	115.78	1.76	148.91	1.37	74.65	2.73
208	235.18	0.88	117.99	1.76	151.76	1.37	76.07	2.73
212	239.58	0.88	120.18	1.76	154.61	1.37	77.50	2.74
216	243.98	0.89	122.39	1.76	157.46	1.37	78.93	2.74
220	248.37	0.89	124.58	1.77	160.31	1.37	80.33	2.74
224	252.78	0.89	126.79	1.77	163.16	1.37	81.75	2.74
228	257.17	0.89	128.98	1.77	166.01	1.37	83.17	2.74
232	261.58	0.89	131.19	1.77	168.86	1.37	84.60	2.74
236	265.97	0.89	133.38	1.77	171.71	1.37	86.03	2.74
240	270.38	0.89	135.59	1.77	174.56	1.37	87.45	2.74
244	274.78	0.89	137.78	1.77	177.41	1.38	88.88	2.75
248	279.19	0.89	139.99	1.77	180.26	1.38	90.30	2.75
252	283.57	0.89	142.19	1.77	183.11	1.38	91.72	2.75
256	287.99	0.89	144.40	1.77	185.96	1.38	93.15	2.75
260	295.67	0.88	148.24	1.75	190.81	1.36	95.57	2.72
264	300.08	0.88	150.43	1.75	193.66	1.36	96.99	2.72
268	304.47	0.88	152.63	1.76	196.51	1.36	98.42	2.72
272	308.87	0.88	154.83	1.76	199.36	1.36	99.85	2.72
276	313.27	0.88	157.03	1.76	202.21	1.36	101.27	2.73
280	317.67	0.88	159.24	1.76	205.06	1.37	102.70	2.73
284	322.07	0.88	161.43	1.76	207.91	1.37	104.12	2.73
288	326.47	0.88	163.63	1.76	210.77	1.37	105.55	2.73
292	330.87	0.88	165.83	1.76	213.61	1.37	106.97	2.73
296	335.28	0.88	168.03	1.76	216.46	1.37	108.39	2.73
300	339.67	0.88	170.24	1.76	219.31	1.37	109.82	2.73
304	344.08	0.88	172.44	1.76	222.16	1.37	111.25	2.73
308	348.47	0.88	174.64	1.76	225.01	1.37	112.67	2.73
312	352.88	0.88	176.83	1.76	227.86	1.37	114.10	2.73
316	357.27	0.88	179.03	1.77	230.71	1.37	115.52	2.74
320	361.67	0.88	181.23	1.77	233.56	1.37	116.95	2.74
324	366.07	0.89	183.44	1.77	236.41	1.37	118.37	2.74
328	370.47	0.89	185.64	1.77	239.27	1.37	119.80	2.74
332	374.87	0.89	187.84	1.77	242.12	1.37	121.22	2.74
336	379.27	0.89	190.04	1.77	244.97	1.37	122.65	2.74
340	383.67	0.89	192.23	1.77	247.82	1.37	124.07	2.74
344	388.07	0.89	194.43	1.77	250.67	1.37	125.50	2.74
348	392.47	0.89	196.63	1.77	253.51	1.37	126.92	2.74
352	396.87	0.89	198.84	1.77	256.36	1.37	128.35	2.74
356	401.27	0.89	201.04	1.77	259.21	1.37	129.77	2.74

360	405.67	0.89	203.23	1.77	262.06	1.37	131.20	2.74
364	410.08	0.89	205.44	1.77	264.91	1.37	132.62	2.74
368	414.48	0.89	207.63	1.77	267.77	1.37	134.05	2.75
372	418.88	0.89	209.84	1.77	270.61	1.37	135.47	2.75
376	423.27	0.89	212.04	1.77	273.46	1.37	136.90	2.75
380	427.67	0.89	214.23	1.77	276.31	1.38	138.32	2.75
384	432.08	0.89	216.43	1.77	279.17	1.38	139.75	2.75
388	436.48	0.89	218.64	1.77	282.02	1.38	141.17	2.75
392	440.87	0.89	220.84	1.78	284.87	1.38	142.60	2.75
396	445.27	0.89	223.04	1.78	287.71	1.38	144.02	2.75
400	449.67	0.89	225.24	1.78	290.57	1.38	145.45	2.75
404	454.07	0.89	227.44	1.78	293.42	1.38	146.87	2.75
408	458.47	0.89	229.63	1.78	296.27	1.38	148.30	2.75
412	462.88	0.89	231.83	1.78	299.12	1.38	149.73	2.75
416	467.28	0.89	234.04	1.78	301.97	1.38	151.15	2.75
420	471.67	0.89	236.24	1.78	304.82	1.38	152.57	2.75
424	476.08	0.89	238.43	1.78	307.67	1.38	154.00	2.75
428	480.48	0.89	240.64	1.78	310.52	1.38	155.42	2.75
432	484.88	0.89	242.83	1.78	313.37	1.38	156.85	2.75
436	489.27	0.89	245.04	1.78	316.22	1.38	158.27	2.75
440	493.68	0.89	247.24	1.78	319.07	1.38	159.70	2.76
444	498.08	0.89	249.44	1.78	321.92	1.38	161.13	2.76
448	502.48	0.89	251.63	1.78	324.77	1.38	162.55	2.76
452	506.87	0.89	253.84	1.78	327.61	1.38	163.97	2.76
456	511.27	0.89	256.04	1.78	330.47	1.38	165.40	2.76
460	515.67	0.89	258.24	1.78	333.32	1.38	166.82	2.76
464	520.08	0.89	260.44	1.78	336.17	1.38	168.25	2.76
468	524.48	0.89	262.64	1.78	339.02	1.38	169.68	2.76
472	528.88	0.89	264.83	1.78	341.87	1.38	171.10	2.76
476	533.28	0.89	267.04	1.78	344.72	1.38	172.53	2.76
480	537.68	0.89	269.24	1.78	347.57	1.38	173.95	2.76
484	542.08	0.89	271.44	1.78	350.42	1.38	175.38	2.76
488	546.48	0.89	273.64	1.78	353.27	1.38	176.80	2.76
492	550.88	0.89	275.84	1.78	356.12	1.38	178.23	2.76
496	555.28	0.89	278.04	1.78	358.97	1.38	179.65	2.76
500	559.68	0.89	280.24	1.78	361.82	1.38	181.07	2.76
504	564.08	0.89	282.44	1.78	364.67	1.38	182.50	2.76
508	568.48	0.89	284.64	1.78	367.52	1.38	183.92	2.76
512	572.88	0.89	286.84	1.78	370.37	1.38	185.35	2.76

Hardware Tests (20 Mbps)								
Message Size	Uni 1 DMA		Uni 2 DMA		Bi 2 DMA		Bi 4 DMA	
(byte)	Time (us)	BW (MB/s)	Time (us)	BW (MB/s)	Time (us)	BW (MB/s)	Time (us)	BW (MB/s)
4	6.81	0.59	3.80	1.05	4.06	0.99	2.29	1.75
8	9.24	0.87	5.00	1.60	5.71	1.40	3.12	2.57
12	11.61	1.03	6.21	1.93	7.36	1.63	3.94	3.05
16	14.03	1.14	7.40	2.16	9.01	1.78	4.77	3.36
20	16.43	1.22	8.61	2.32	10.67	1.87	5.60	3.57
24	18.83	1.27	9.81	2.45	12.33	1.95	6.43	3.73
28	21.24	1.32	11.01	2.54	13.98	2.00	7.26	3.86
32	23.61	1.36	12.22	2.62	15.63	2.05	8.09	3.96
36	26.03	1.38	13.42	2.68	17.29	2.08	8.92	4.04
40	28.43	1.41	14.63	2.73	18.94	2.11	9.75	4.10
44	30.83	1.43	15.82	2.78	20.60	2.14	10.58	4.16
48	33.23	1.44	17.03	2.82	22.24	2.16	11.41	4.21
52	35.62	1.46	18.23	2.85	23.90	2.18	12.23	4.25
56	38.02	1.47	19.45	2.88	25.56	2.19	13.07	4.29
60	40.43	1.48	20.63	2.91	27.21	2.21	13.89	4.32
64	42.83	1.49	21.84	2.93	28.87	2.22	14.72	4.35
68	45.23	1.50	23.04	2.95	30.52	2.23	15.56	4.37
72	47.63	1.51	24.23	2.97	32.17	2.24	16.39	4.39
76	50.03	1.52	25.44	2.99	33.83	2.25	17.22	4.41
80	52.43	1.53	26.66	3.00	35.48	2.26	18.06	4.43
84	54.83	1.53	27.86	3.02	37.14	2.26	18.88	4.45
88	57.23	1.54	29.06	3.03	38.79	2.27	19.71	4.46
92	59.62	1.54	30.26	3.04	40.44	2.27	20.55	4.48
96	62.03	1.55	31.45	3.05	42.10	2.28	21.38	4.49
100	64.43	1.55	32.66	3.06	43.75	2.29	22.21	4.50
104	66.83	1.56	33.87	3.07	45.40	2.29	23.04	4.51
108	69.23	1.56	35.06	3.08	47.07	2.29	23.87	4.52
112	71.63	1.56	36.24	3.09	48.72	2.30	24.71	4.53
116	74.03	1.57	37.46	3.10	50.38	2.30	25.55	4.54
120	76.44	1.57	38.67	3.10	52.04	2.31	26.38	4.55
124	78.84	1.57	39.87	3.11	53.68	2.31	27.22	4.56
128	81.23	1.58	41.07	3.12	55.34	2.31	28.04	4.57
132	83.64	1.58	42.27	3.12	57.01	2.32	28.88	4.57
136	86.03	1.58	43.44	3.13	58.66	2.32	29.70	4.58
140	88.44	1.58	44.66	3.13	60.31	2.32	30.55	4.58
144	90.83	1.59	45.88	3.14	61.98	2.32	31.38	4.59
148	93.24	1.59	47.09	3.14	63.62	2.33	32.21	4.60
152	95.63	1.59	48.28	3.15	65.28	2.33	33.05	4.60
156	98.04	1.59	49.46	3.15	66.93	2.33	33.88	4.60
160	100.43	1.59	50.67	3.16	68.61	2.33	34.72	4.61
164	102.83	1.59	51.87	3.16	70.24	2.33	35.55	4.61
168	105.24	1.60	53.11	3.16	71.92	2.34	36.39	4.62

172	107.66	1.60	54.21	3.17	73.56	2.34	37.21	4.62
176	110.04	1.60	55.47	3.17	75.23	2.34	38.05	4.63
180	112.44	1.60	56.68	3.18	76.88	2.34	38.88	4.63
184	114.83	1.60	57.88	3.18	78.54	2.34	39.72	4.63
188	117.24	1.60	59.09	3.18	80.20	2.34	40.54	4.64
192	119.64	1.60	60.29	3.18	81.86	2.35	41.39	4.64
196	122.05	1.61	61.49	3.19	83.53	2.35	42.22	4.64
200	124.45	1.61	62.70	3.19	85.15	2.35	43.04	4.65
204	126.86	1.61	63.90	3.19	86.82	2.35	43.88	4.65
208	129.24	1.61	65.09	3.20	88.49	2.35	44.72	4.65
212	131.64	1.61	66.29	3.20	90.15	2.35	45.56	4.65
216	134.05	1.61	67.49	3.20	91.79	2.35	46.39	4.66
220	136.45	1.61	68.72	3.20	93.46	2.35	47.23	4.66
224	138.85	1.61	69.89	3.21	95.11	2.36	48.05	4.66
228	141.24	1.61	71.10	3.21	96.77	2.36	48.89	4.66
232	143.69	1.61	72.29	3.21	98.43	2.36	49.73	4.66
236	146.04	1.62	73.50	3.21	100.08	2.36	50.56	4.67
240	148.44	1.62	74.69	3.21	101.75	2.36	51.40	4.67
244	150.84	1.62	75.89	3.22	103.38	2.36	52.23	4.67
248	153.24	1.62	77.09	3.22	105.06	2.36	53.08	4.67
252	155.65	1.62	78.30	3.22	106.71	2.36	53.92	4.67
256	158.05	1.62	79.51	3.22	108.37	2.36	54.74	4.68
260	162.45	1.60	81.68	3.18	111.27	2.34	56.20	4.63
264	164.84	1.60	82.91	3.18	112.94	2.34	57.03	4.63
268	167.25	1.60	84.11	3.19	114.59	2.34	57.86	4.63
272	169.63	1.60	85.31	3.19	116.24	2.34	58.69	4.63
276	172.04	1.60	86.46	3.19	117.90	2.34	59.53	4.64
280	174.45	1.61	87.69	3.19	119.58	2.34	60.36	4.64
284	176.84	1.61	88.93	3.19	121.23	2.34	61.19	4.64
288	179.24	1.61	90.13	3.20	122.87	2.34	62.03	4.64
292	181.64	1.61	91.33	3.20	124.55	2.34	62.86	4.65
296	184.04	1.61	92.56	3.20	126.22	2.35	63.71	4.65
300	186.45	1.61	93.77	3.20	127.84	2.35	64.53	4.65
304	188.84	1.61	94.96	3.20	129.51	2.35	65.37	4.65
308	191.25	1.61	96.17	3.20	131.16	2.35	66.21	4.65
312	193.65	1.61	97.35	3.21	132.83	2.35	67.03	4.65
316	196.05	1.61	98.54	3.21	134.48	2.35	67.86	4.66
320	198.46	1.61	99.76	3.21	136.16	2.35	68.70	4.66
324	200.85	1.61	100.97	3.21	137.80	2.35	69.52	4.66
328	203.26	1.61	102.17	3.21	139.45	2.35	70.36	4.66
332	205.66	1.61	103.37	3.21	141.10	2.35	71.20	4.66
336	208.05	1.61	104.58	3.21	142.77	2.35	72.04	4.66
340	210.46	1.62	105.78	3.21	144.43	2.35	72.88	4.67
344	212.86	1.62	107.01	3.21	146.08	2.35	73.71	4.67
348	215.22	1.62	108.44	3.21	147.72	2.36	74.54	4.67
352	217.65	1.62	109.33	3.22	149.41	2.36	75.39	4.67
356	220.05	1.62	110.56	3.22	151.05	2.36	76.21	4.67

360	222.44	1.62	111.76	3.22	152.72	2.36	77.05	4.67
364	224.87	1.62	112.99	3.22	154.39	2.36	77.87	4.67
368	227.24	1.62	114.15	3.22	156.01	2.36	78.71	4.68
372	229.66	1.62	115.38	3.22	157.68	2.36	79.55	4.68
376	232.07	1.62	116.61	3.22	159.32	2.36	80.38	4.68
380	234.46	1.62	117.79	3.23	161.00	2.36	81.23	4.68
384	236.86	1.62	118.99	3.23	162.67	2.36	82.06	4.68
388	239.25	1.62	120.16	3.23	164.33	2.36	82.90	4.68
392	241.65	1.62	121.39	3.23	165.99	2.36	83.72	4.68
396	244.05	1.62	122.61	3.23	167.62	2.36	84.56	4.68
400	246.44	1.62	123.84	3.23	169.30	2.36	85.39	4.68
404	248.87	1.62	125.01	3.23	170.94	2.36	86.22	4.69
408	251.27	1.62	126.19	3.23	172.61	2.36	87.05	4.69
412	253.66	1.62	127.37	3.23	174.25	2.36	87.90	4.69
416	256.05	1.62	128.60	3.23	175.92	2.36	88.75	4.69
420	258.46	1.63	129.79	3.24	177.59	2.37	89.57	4.69
424	260.85	1.63	131.01	3.24	179.24	2.37	90.40	4.69
428	263.26	1.63	132.22	3.24	180.90	2.37	91.26	4.69
432	265.66	1.63	133.37	3.24	182.57	2.37	92.05	4.69
436	268.06	1.63	134.59	3.24	184.22	2.37	92.90	4.69
440	270.45	1.63	135.81	3.24	185.87	2.37	93.76	4.69
444	272.88	1.63	137.03	3.24	187.52	2.37	94.58	4.69
448	275.29	1.63	138.18	3.24	189.21	2.37	95.41	4.70
452	277.66	1.63	139.43	3.24	190.82	2.37	96.25	4.70
456	280.06	1.63	140.61	3.24	192.50	2.37	97.10	4.70
460	282.46	1.63	141.81	3.24	194.15	2.37	97.92	4.70
464	284.85	1.63	143.02	3.24	195.82	2.37	98.75	4.70
468	287.27	1.63	144.32	3.24	197.46	2.37	99.62	4.70
472	289.68	1.63	145.41	3.25	199.12	2.37	100.41	4.70
476	292.05	1.63	146.60	3.25	200.79	2.37	101.24	4.70
480	294.46	1.63	147.82	3.25	202.46	2.37	102.09	4.70
484	296.86	1.63	149.02	3.25	204.10	2.37	102.93	4.70
488	299.27	1.63	150.21	3.25	205.77	2.37	103.77	4.70
492	301.66	1.63	151.41	3.25	207.41	2.37	104.60	4.70
496	304.07	1.63	152.60	3.25	209.08	2.37	105.42	4.71
500	306.47	1.63	153.81	3.25	210.73	2.37	106.26	4.71
504	308.86	1.63	154.95	3.25	212.40	2.37	107.11	4.71
508	311.27	1.63	156.21	3.25	214.07	2.37	107.93	4.71
512	313.67	1.63	157.38	3.25	215.69	2.37	108.80	4.71