Communication Interfaces for A Distributed Embedded Multiprocessor System

Wei Kiong, Chin

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

School of Computing and Informatics
The Nottingham Trent University
Clifton Lane
Nottingham
NG11 8NS

Abstract

This thesis documents a research project on communication interfaces for a distributed embedded multiprocessor system. This resulted in the development of a novel embedded distributed multiprocessor system on a single chip.

The initial feasibility studies involved a review of the relevant embedded distributed multiprocessor systems and their inter-processor communication. The research aimed to expand the potential of a multiprocessor communication system on a single chip. System designs were adapted to achieve more efficient Direct Memory Access (DMA) and reduced processor intervention.

A development board with advanced FPGA technology is used to implement the designed modules. The single chip solution consists of two processing nodes and an 'off-the-shelf' hardware message router. Each processing node includes: a NIOS II processor, a memory module, and a network interface controller. The network interface controller, which interconnects the processor and the embedded routing network, was developed using VHDL.

All basic routing features and functions of this novel VHDL system model have been proven and verified through hardware testing and simulation. The system was synthesised and implemented into a single FPGA chip as a System-on-Programmable-Chip (SOPC). A test program was written to test the functionality of the interface. The research resulted in a fully operational prototype. The features of the system are discussed and compared and contrasted with the state-of-the-art research literature.

The router and NIOS II processors with their interface form the building blocks of a robust, embedded network on the single chip platform. The router interconnects all the processing nodes and allows them to operate in the same network simultaneously, thus increasing system flexibility and applications. The in-built differential output feature on the FPGA chip enables the system to be cascaded to more processing nodes off-chip.

Acknowledgements

This thesis is dedicated to my parents, who have given me full support and concern all the time. Without their support, mentally and financially, I would not have come here to further my studies.

I would like to take this opportunity to thank my director of studies, Dr. Steve Clark, for offering me the opportunity for this research post, also for his helpfulness and understanding of my difficult moments during my studies. I'd also like to thank the rest of my supervisor team: Dr. David Downes, Prof. Brian O'Neill, and Dr Richard Germon, for their technical support and guidance.

Special thanks to my best friend, Mr. Fook Chang, Ooi, and his girlfriend, Ms. Phoebe Jiang for their tolerance and support during my studies. They are like a family to me as they take care of all the cooking, washing up and housework in the busiest time so that I will be least distracted from my work. They also keep my spirits up all the time when going through difficult times.

Last but not least, to two of my very good friends I met in Nottingham, Mr. Ian Rollinson and Dr. Lee Wholton for being supportive as well, and not letting me take thing too seriously. Life has become more fun and interesting knowing these two friends.

To anyone whom I have not mentioned, thank you very much.

Table of Contents

List of Acronyms and Abbreviations

AHB	Advanced High Performance Bus of AMBA
ALM	Adaptive Logic Module
ALU	Arithmetic Logic Unit
ALUT	Adaptive Look-up Table
AMBA	Advanced Microcontroller Bus Architecture
APB	Advance Peripheral Bus of AMBA
ASB	Advanced System Bus of AMBA
ASIC	Application Specific Integrated Circuit
BEOP	Bad End of Packet
CAD	computer aided design
CAM	Content Addressable Memory
CAT 5	Category 5 unshielded twisted pair
CPU	Central Processing Unit
CSP	Communicating Sequential Processes
DDR SDRAM	Double Data Rate Synchronous Dynamic Random Access Memory
DMA	Direct Memory Access
DMIPS	Dhrystone Million Instruction per Second
EOM	End of Message token
EOP	End of Packet token
ESB	Embedded System Block
EX	Instruction Execution (as in Instruction Pipeline)
FIFO	First-In-First-Out
FPGA	Field Programmable Gate Array
FPU	Floating Point Unit
FSL	Fast Simple Link
FT-PCI-Li	Fault Tolerant PCI Link
I/O	Input/Output
ID	Identification
ID	Instruction Decode (as in Instruction Pipeline)
IF	Instruction Fetch (as in Instruction Pipeline)
IP	Intellectual Property
JTAG	Joint Test Action Group
JTAG-UART	Joint Test Action Group-Universal Asynchronies Receiver and Transmitter
LE	Logic Element
LMB	Local Memory Bus
MEM	Memory access (as in Instruction Pipeline)

MPP	Massively Parallel Processing
NIOSNIC	NIOS II based Network Interface Controller
NoC	Network-on-Chip
NTU	Nottingham Trent University
OPB	On-chip Peripheral Bus
OPC	Open Core Protocol
OS	Over Sampling
PC	Personal Computer
PCB	Printed Circuit Board
PCI	Peripheral Component Interface
PDA	Personal Data Assistants
PIO	Programmable Input/Output
PLA	Programmable Logic Arrays
PLD	Programmable Logic Device
PLL	Phase Lock Loop
PN	Processing Node
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
SDRAM	Synchronous Dynamic Random Access Memory
SMP	Symmetric Multiprocessing
SoC	System-on-a-Chip
SoPC	System-on-Programmable-Chip
SPU	Synergistic Processor Unit
SRAM	Static Random Access Memory
ST2SS	Stratix II Sub System
UART	Universal Asynchronies Receiver and Transmitter
VDHL	Very High Speed Integrated Circuit Hardware Description Language
VLSI	Very Large Scale Integration
WB	Write-Back (as in Instruction Pipeline)

List of Figures

1 Introduction

1.1 Introduction

The principle of 'Parallel Processing' is to achieve a solution to a problem, which is too complicated or time consuming for a single processor, by task division to multiple processors[1]. In a parallel processing system, a problem is first broken down into many sub-problems. All the sub-problems will then be solved through task distribution to the interconnected processors in the same network, which operate concurrently.

With increasing requirements of higher computation power, there will always be computationally intensive problems that are beyond the capability of a uniprocessor system[2 3]. Parallel processing becomes a way of overcoming the limitation of traditional computer architectures. Early generations of parallel machines, such as the CRAY-1 supercomputer[4], were used in highly numerical intensive research and scientific areas. They used custom built, high speed circuits and utilized array processing as their underlying architecture.

The rapid development of parallel processing has been motivated by the demand of high computational power in current applications and the advance of Very Large Scale Integration (VLSI) technology. Microprocessors have become more powerful, cheaper and smaller in size, therefore a basic parallel system can be constructed by interconnecting multiple processors[5] utilising the advantages such as: low cost, high performance and 'off the shelf' market availability.

A multiprocessor System-on-a-Chip (SoC) is the latest incarnation of VLSI technology[6] . A 'SoC' is an integrated circuit that implements the necessary functions of an electronic system, including microprocessor cores. The components that are implemented on a SoC vary with the application. The system may consist of memory

blocks, processors, interface busses, and other custom digital functions. The architecture of a system is more application specific than general purpose.

SoC devices are implemented in many products ranging from daily consumer applications to high-end industrial systems. For example: cellular phones for signal processing and user's telephony applications[7]; networking for data handling from modern communication equipment; video games for real-time game action rendering[8]. The applications often use parallel processing to handle real time applications. The use of general purpose computers in such systems would often be unsuitable[9]. General purpose machines would not perform as well for reliable real-time control and would not match the data rates for high-end video[10].

One of the requirements of many real-time applications is to 'embed' the processor within a larger system so that the 'embedded system' can interact and respond within that system. Embedded systems usually require low power consumption and a small physical size to be used in applications such as: power plants, automobile control, home networks and in monitoring and control operations[11 12]. Large scale parallel systems, such as supercomputers, are too large, consume too much power and are too expensive to be embedded for use in many real-time commercial applications.

The Parallel Processing Group in Nottingham Trent University (NTU) was initially involved in research based on the Transputers [13] but following the demise of Transputer has focused on other specialised custom embedded processors. One of the main achievements of the research group was the development of series of hardware message routing devices to improve network efficiency and fault tolerance for distributed embedded multiprocessor systems[14 15]. The intention of this research project was to build on previous research and utilise the new SoC technology to investigate an embedded distributed multiprocessor prototype platform, interconnected by routers, on a single programmable chip.

1.2 The Transputer

The Transputer was a 32 bit bus microprocessor designed by INMOS (now SGS-Thomson Microelectronic) to be used as a processing building block in parallel processing systems [16] [17]. Due to its Reduced Instruction Set Computer (RISC) like architecture, state-of-the-art performance (at that time), high speed serial links and low power consumption, it was used in the area of high performance embedded parallel computing systems. And because of that, it was able to support real-time programming[16]. The Transputer consisted of internal memory, an external memory control interface, Input/Output (I/O) devices, and four bi-directional serial communication links.

The Transputer had four high speed serial links to communicate with other Transputers (or other devices): called OS Links[5]. It was based partly on Hoare's Communicating Sequential Processes (CSP) model proposal that each processing node is connected via physical point-to-point connections [18]. The Over Sampling (OS) communication links between four neighbouring processing nodes were full duplex bi-directional serial communication links, illustrated by the example in Figure 1-1.

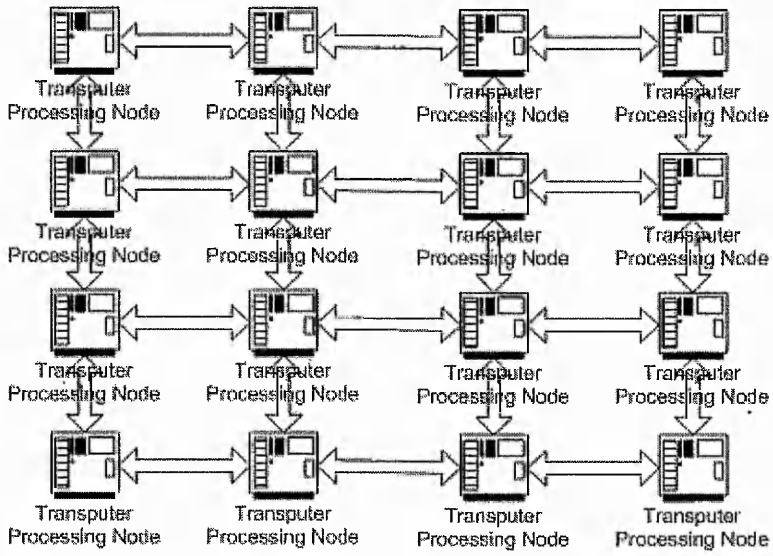


Figure 1-1: Transputer Network.

This mesh network topology was very efficient when only four or less processing nodes were utilised because communication was almost immediate between adjacent processing nodes. When utilising more than four Transputers, messages had to be forwarded via at least one intermediate processing node. This message forwarding not only reduced the efficiency of resource distribution, it also introduced latency and increased the message overhead to the message handling tasks (required to forward the message to other processors). As a result, processor performance was reduced, due to the increase of communication loading (as the processor needed to handle forwarding the messages from other processing nodes as well processing its own messages[19]). The 'Store and Forward' methodology of this point-to-point connection produced a message delay each time the message was forwarded. The message delay produced was proportional to the message size and the distance that the message had to travel[20]. This resulted in a variation of time in communication between processors, based on these factors. A large inter-processor latency reduced the overall performance of the Transputer system.

A reason for the success of Transputer was its built-in communication controller on the same chip. This provided efficient point-to-point message passing between adjacent Transputers[21]. However, this used up silicon space that could be used to implement useful functionality to the processor. Therefore extra engineering design effort was required to build dedicated interfaces for external functions. With the Application Specified Integrated Circuit (ASIC) technology available at that time, upgrading the processor or the network interface was time consuming and costly.

1.3 Research Background and Objectives

The Nottingham Trent University Parallel Processing Research Group has been investigating and designing inter-processor communication devices, for embedded parallel processing systems, for many years. The multiprocessor system research began with the implementation of Transputer systems[22 23 24 25 26].

The design and fabrication of a commercial 16-channel dynamic hardware routing switch, the ICR C416 device [27] [28] [29] [30] [31], overcame some of the limitations of the Transputer system. The ICR C416 offloaded much of the communication tasks from the Transputer System by providing direct point-to-point connections to up to 16 Transputers. In addition to that, this device could be cascaded to support scalable larger systems. The use of the ICR C416 in the Transputer systems, successfully demonstrated the efficiency of a routing device as a simple solution to medium scale, low cost, high performance embedded inter-processor communications[32]. This device could be implemented as the backbone of an embedded distributed multiprocessor system, as shown in the example in Figure 1-2, where each PN block is a processing node. It can also be connected to a PC via a custom Peripheral Component Interface (PCI) [33] [34].

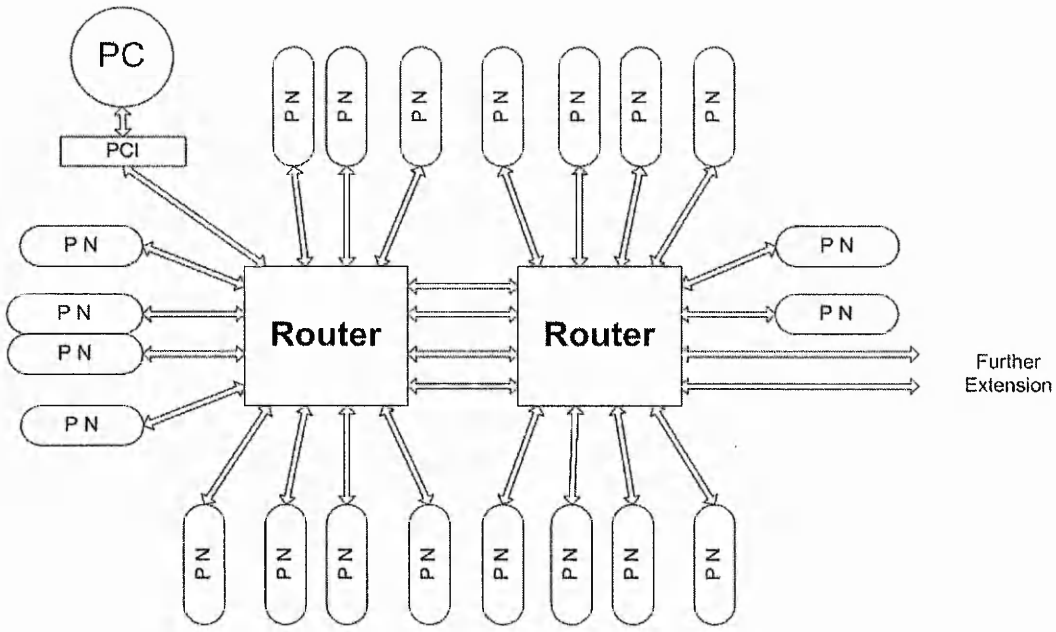


Figure 1-2: Transputer Network with ICR C416 Routers.

The rapid increase of computational power has prompted the need to upgrade the processors used in the ICR C416 network in order to keep pace with other embedded systems. Since the demise of the Transputer, attention switched to the design of routing

networks for other processors, while looking to maintain the successful features of the Transputer network (such as a router based serial communication network with low message latency and minimal processor intervention; i.e. minimise communication overheads).

The research group then extended the research attention to incorporating a state-of-the-art RISC processor into a parallel network. The StrongARM SA-110 microprocessor[35] was chosen (as a replacement for the Transputer) because it was a low cost, low power, easily available processor that offered state-of-the-art performance. The 32 bit SA-110 RISC processor could support a core bus and a data bus of up to 233 MHz and 66 MHz respectively. The SA-110 had 32 kBytes of internal cache memory and 128 bytes of write buffer, and it also supported fast interrupt handling. The on-chip cache and write buffer increased the average instruction execution speed and reduced the average memory bandwidth usage of the processor; this enabled the memory bus to be accessed for data transfer to and from a custom network interface device.

The data transfer between the memory module and the network controller was done during the time when the processor was not accessing memory. This technique is called Direct Memory Access (DMA)[36]. This 'cycle stealing' is a special hardware arrangement utilising the free memory bus cycle to read from or write to memory very quickly without incurring overhead in accessing the data bus.

The research group eventually developed the StrongARM Router Network Interface Controller (SARNIC)[37 38 39 40 41] to perform the communication interface role between the chosen processor and the router network. The SARNIC was developed and implemented on a Programmable Logic Device (PLD). The PLD consists of a bus-based SARNIC, a memory interface controller and a processor interface. The PLD was then mounted onto a Printed Circuit Board (PCB), interconnecting the memory module and the StrongARM processor, forming a processing node. The processing node could then be used as a building block in a scalable distributed parallel processing system,

interconnected by ICR C416 routers. In other words, the Transputer in the previous system has been directly replaced by the processing node (PN) as shown in Figure 1-3.

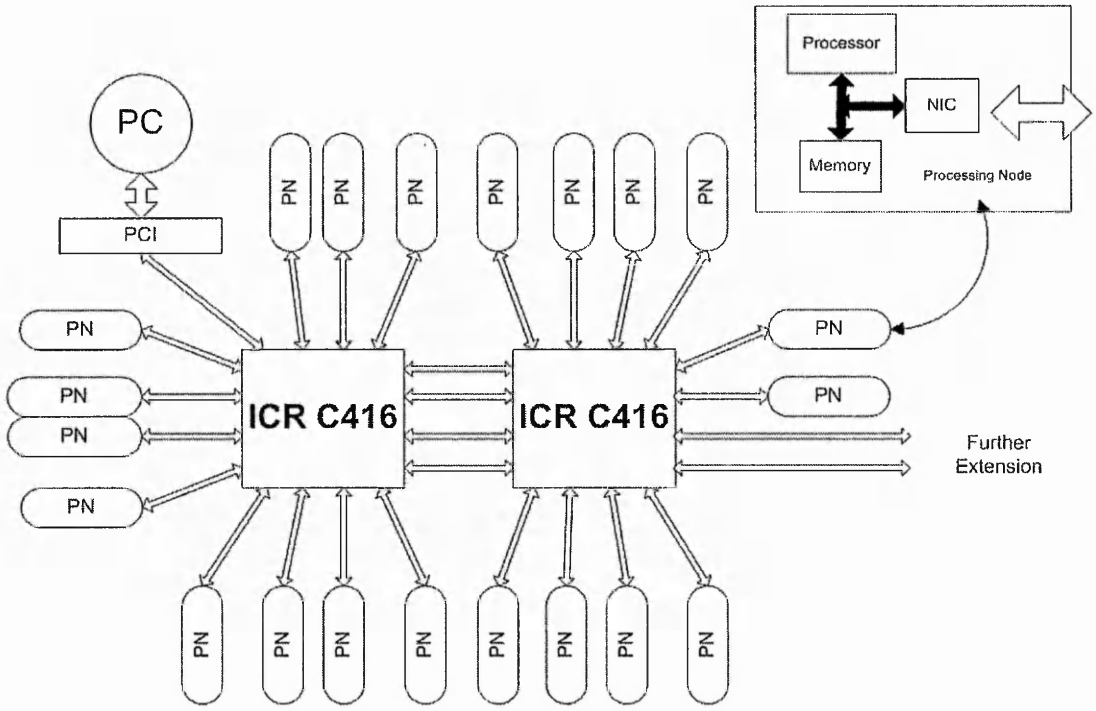


Figure 1-3: Block Diagram of a processing node and the distributed parallel processing Network.

The NTR-FTM08 router was the most recent routing device developed by the NTU research group. It was adapted from the ICR C416 Router design with enhanced fault tolerance, alongside support of a new adapted OS-Link protocol. It was integrated with basic fault detection and isolation methodology. Removing faults from the system will free resources held by faulty messages and would often allow communications (in a network with a fault) to be re-established automatically[42].

1.4 The New Distributed Multiprocessing System

The key aim of this project was to build a low cost, medium scale, high performance embedded distribute parallel processing network on a single chip. The target was not to construct a parallel processing system to compete with high-end and expensive

supercomputer, but to build a powerful single chip multiprocessor parallel platform for embedded systems.

The following are the objectives of this research project:

- ✓ To investigate the relevant router architectures used in the literature and especially in the NTU research group and then apply the routing device as the backbone of a scalable interconnect network, to provide non-blocking point-to-point communication for processing nodes.
- ✓ To build a multiprocessor network prototype platform on a System-On-Programmable-Chip. Implementing the multiprocessor embedded system onto a single chip to obtain optimised performance.
- ✓ To conduct performance analysis on the prototype platform. Design and run tests on the platform to ensure functionality and performance of the system.

At the initial stage, the NTR-FTM08 routing switch was chosen as the communication backbone for the network. The use of the NTR-FTM08 routing switch was to build on the advantage in the existing system. The serial communication routing device used would offer the advantage of reduced wiring and pin connection, and complexity in constructing a distributed system.

The design work started with the investigation and study of the design of recent Fault Tolerant PCI Link (FT-PCI-Li) designs with NTR-FTM08 router on an APEK20K[43] chip. The designs were later optimised; in particular by reducing the processor's intervention when passing a message and/or the message header capacity was increased.

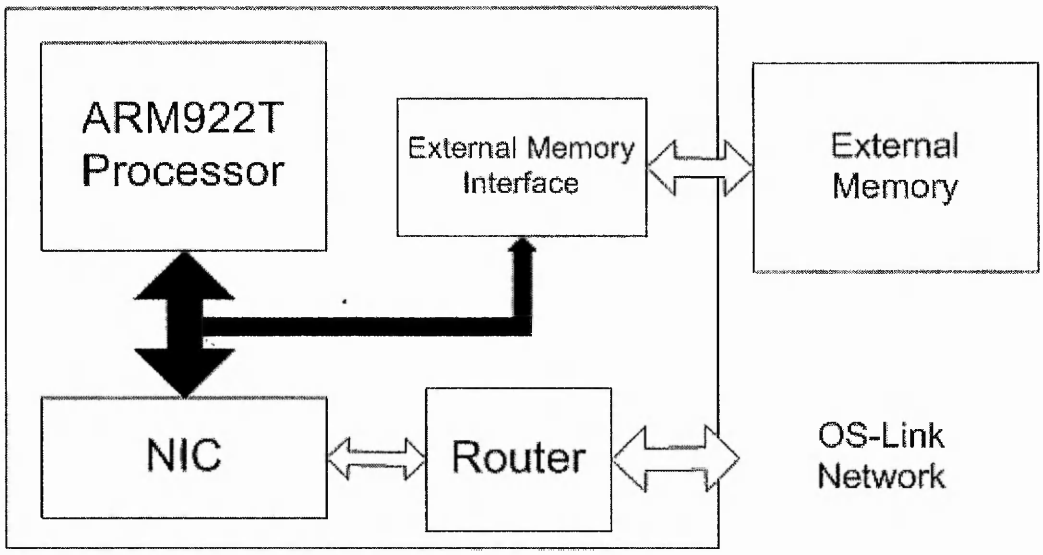


Figure 1-4: StrongARM Processing Node.

The design work then moved to the implementation of single chip solution for the StrongARM processing node. The Excalibur chip [44] was selected to develop the StrongARM processing node. The Excalibur is the combination of ARM 922T™ [45]32 bit RISC processor system and programmable logic on a single device. The network interface controller, external memory interface controller, and a routing device were developed in the programmable logic of the chip, as shown in Figure 1-4. A multiprocessor network can be achieved by interconnecting the processing nodes, via the routing device implemented on each Excalibur chip, on a PCB. Therefore this board, called the XA1, was developed.

The introduction of the NIOS II processor[46] and Stratix II [47]family FPGA made the group realise that there were potential benefits implementing a distributed embedded multiprocessor system on this new technology. NIOS II is a 32 bit softcore RISC processor (as compared to the ARM 922T on an Excalibur, which is a hardcore (built-in) processor). A key benefit was that more than one NIOS II processor can be implemented on the same Stratix II chip, where there is only one ARM 922T processor available in the Excalibur. NIOS II also offered advantages in terms of flexibility and resource saving,

while the Stratix II chip offered higher densities of logic elements and on-chip memory compared to the Excalibur. The features of the NIOS II and Stratix II have given an opportunity for System-on-Chip (SoC) prototyping and more space for further expansion and development.

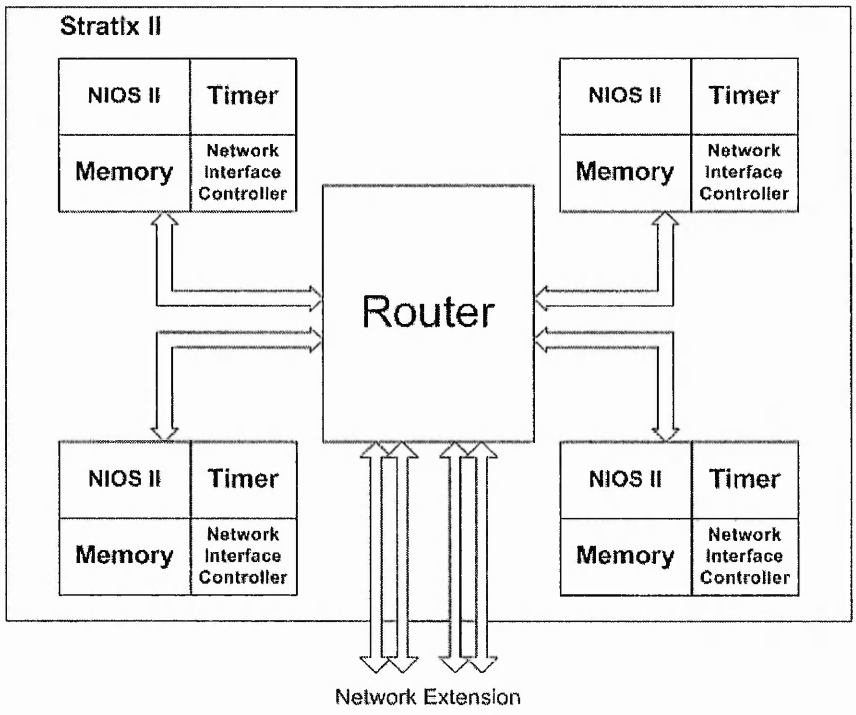


Figure 1-5: Distributed embedded multiprocessor system on Stratix II chip.

As shown in Figure 1-5, a basic distributed embedded multiprocessor SoC solution has been designed, tested and verified, with a novel Avalon Bus based OS-Link network interface controller. The OS-Link Network Interface Controller, which was AMBA bus interface, was modified to Avalon Bus interface so that it can be implemented into NIOS II processing node. Each processing node consists of a NIOS II processor, on-chip memory, timer, and network interface controller. All the processing nodes will be interconnected in a single chip by an OS-Link based router. The distributed embedded multiprocessors system was prototyped on the NIOS II development kit.

All the design was described using VLSI Hardware Description Language [48] (VHDL) and partitioned into a modular, top-down hierarchy. VHDL was the chosen language for design because it is an ANSI standard language used to describe hardware components and systems and the research group was very experienced with its use. VHDL is very powerful for digital system design, the written model is easy to modify and verify functionality, and it supports concurrent design. The language itself is publicly available, human and machine readable [49]. It is a recognised design entry standard, therefore the designed model is transferable between chip vendors and between different target technologies.

The PC based Quartus II design software[50] was used throughout the design cycle. It was provided by Altera [51] and also included the design synthesis tool and post-synthesis Stratix II chip programming tool[52]. Quartus II can give timing analysis reports to indicate whether the timing requirements are met before implementation of the design (onto the FPGA, or hardware in the future). The NIOS II Integrated Development Environment software[53] was used to compile programs, written to run in the processing nodes, to test the functionality of each processing node and the whole system. After compilation of both VHDL codes and testing codes, they were downloaded onto the Stratix II chip for real-time functionality verification and performance measurement.

The research described in this thesis has resulted in the design and realisation of SoC building blocks for an embedded multiprocessor network system. Multiple processing nodes were constructed by using the 'off-the-shelf' soft-core processors, available memory space and Adaptive Look-up Table (ALUT) in the target FPGA chip. The adapted OS-Link based routing switch, which was previously developed to interconnect and to provide a robust communication network for a distributed multiprocessor system, was implemented as a point-to-point communication medium for all the implemented processing nodes. The network utilised a serial adapted Over Sampling link (adapted OS-Link) based protocol [54] for message passing between processors.

1.5 Structure of the Thesis

Chapter Two gives an overview of the subject area. It is an introduction to the inter-processor communications methods, with a review of their characteristics. This chapter also reviews how the implementation of different inter-processor communication systems affects the performance of a network. The three routing network systems reviewed (interprocessor and not general purpose such as Ethernet) are some of the main ones used in packet switched multiprocessor systems: the OS-Link based systems, the Myrinet[55] based system and the Xpipes[56] system are compared and contrasted.

Chapter Three examines the characteristics of a Multiprocessor SoC. Modern SoC devices show a trend towards integrating processor core(s) on a single ship alongside memory, I/O support and customisable Programmable Logic Arrays (PLAs). This chapter reviews the challenge and necessary considerations when constructing a SoC device for use in parallel embedded systems. It also focuses on the influence of the multiprocessor architectures on the overall system performance.

Chapter Four documents implementing the OS-Link based network in a prototype system based on the XA1 board. It details the design, architecture used and how each processing node (an Excalibur[57] processing node), was interconnected using the OS-Link based network. It also describes why this approach was ultimately abandoned for an improved system.

Chapter Five describes the implementation of the final selected design methodology. This used a new multiprocessor system on a SoC, the Stratix II chip with NIOS processing cores. It discusses the design on a modular basis: broken down, in order of hierarchy, with description of the functionality and interface of each module.

Chapter Six discusses the hardware tests that were created and their results. A test program was written including functions to perform read and write to the mapped

memory and registers. The result of these tests demonstrated the basic network performance of the new designs.

Chapter Seven concludes the thesis, documenting the main achievements of the research. It also describes the potential expansion of utilising a multiprocessor system on a single chip alongside some other potential avenues of further work.

2 Some Characteristics of Multiprocessor communications

The increasing complexity of on-chip system integration means that more functional units require to be integrated. The effective use of multiprocessors in embedded systems does not just rely on the processing power of the processors, it is also affected by the availability and latency constrains of other system resources such as interfaces, routers and memory, and the design structure of that system. The accessibility of resources for computation or manipulation by the processing element (processor) is often crucial to determine an efficient multiprocessor system.

2.1 Symmetric Multiprocessing and Massively Parallel Processing

The data before and after it is processed must be stored somewhere in the system; either in registers or in memory, so that the processor can carry out the next dedicated task. The processing element must often be able to access these locations very quickly. There are basically two different ways of utilising the system memory: Shared Memory Systems and Distributed Memory Systems[2]. The choice of implementation depends on the requirements of the application and the requirements of the system.

Symmetric Multiprocessing (SMP) is a multiprocessor system, where two or more processors are housed together with shared memory resources. It is a type of Shared Memory System. The SMP system allows any implemented processor to execute a task regardless of the location of the data in the memory. By using a suitable operating system, an SMP system can move tasks amongst the available processors to balance the workload of the system efficiently[58]. The memory resources are centralised and can be accessed by any processor (master) or other type of bus masters. There will be central arbitration control to manage the access of memory resource from all the processors. This system is simple to design and offers fast data access with very low latency. However, this kind of system suffers from lack of scalability. As the number of processors increase, each

processor will be allocated a lower average access time, reducing the effective bandwidth of each processor. Therefore, memory management is a crucial issue in Shared Memory Systems, to prevent memory access from becoming the performance bottleneck of the system. The programmer must also ensure that fewer processors, if possible, are attempting to access the memory simultaneously and preventing one processor from monopolising the memory resource because this might contribute to a performance bottleneck of the system[59].

Another problem with Shared Memory Systems is a security issue. Since the memory is accessible by all processors, it is very difficult for one processor to determine whether the content of a memory location has been modified by another processor. Once again it is up to the careful design by the programmer, when mapping and dividing the memory resources, for each processor to avoid problematic memory usage overlap.

Early SMP systems were accessing memory that ran much slower than the processor accessing them. As a result, the processors spent a considerable amount of time waiting for data from memory resources. Modern memory resources are able to overcome this problem as they are running at comparatively higher clock speed and faster access time. However, they still face a memory access bandwidth problem as only one processor can access memory at a time; while one processor is accessing the memory, the rest of the processors must wait.

Massively Parallel Processing (MPP)[60] is another technique for implementing multiprocessor systems. The principle operation of parallel processing in MPP, similar to SMP, is to break a problem into smaller pieces which are then distributed to the processing nodes, to be solved concurrently. This is particularly useful in scientific and complex mathematics calculations that can more easily be split in this way. The number of processors in an MPP system is not such a crucial issue as the system architecture. In MPP systems, each processor has its own memory resource to form a basic processing node. By having a private memory resource, without memory competition from other processors, the processor will have full bandwidth to access its memory resources. Each

processing node has a copy of its allocated application tasks. All the processing nodes communicate by message passing to each other via a high speed interconnect.

To achieve a solution to a problem by task division to multiple processors, it is highly probable that the resultant data from one processor will be used as an input to another processor. Data exchange between processors/processing nodes is required and this is usually in the form of messages. Message passing in an MPP system has been presented as a solution to overcome the disadvantages suffered in Symmetric Multiprocessing systems, as proven by Hoare[18]. However this involves a trade-off as the system may have latency problems during message passing which depends on the efficiency of the interconnection network used and the distance between the processing nodes.

2.2 Inter-processor Communications

2.2.1 Parallel and Serial Communications

The technology of the microprocessor has been improved rapidly over the past few years. As the speed and data bus width of the processor increases, the communication and data exchange between processors become more and more critical to prevent data starvation. The decision of using parallel or serial communications (in order to optimise the overall efficiency of the system) relies on a few factors: network topology, architecture, speed requirement of the application and development costs.

Early embedded systems utilised bus based parallel communication because multiple bits can be transferred simultaneously. At lower data rates, parallel communication performs adequately. However, as the clock speed and distances increase, synchronisation of parallel data becomes a problem as setup times fall, propagation delays increase, noise has more effect (especially crosstalk), and additionally there is a requirement for multiple line drivers[61]. Pin count is also a problem with parallel communications because the number of pins available in an IC is usually limited. A high

number of parallel bits also involve a more complex PCB design, increasing track and wiring space as well as the cost.

Parallel communication could be clocked at higher data rate and travels longer distance if correctly terminated differential signal were implemented for each signal. However serial communications offers a better solution when I/O pin count of a chip become a constraint in parallel communications. This is because two pins will be utilised for each signal when differential signal was implemented, it will actually double the I/O pin requirement of a system. Serial communication is capable of being clocked at a higher rate because a correctly terminated differential serial link is less susceptible to noise[54]. Using serial communication can also reduce the track on PCB or wiring space when comparing to parallel communication.

Distributed systems, by utilising serial communication, could minimise the cost of development because of the requirement for a lower pin count therefore resulting in a simpler PCB design. Implementation of serial communications in distributed systems also provides a more reliable and robust point-to-point communication due to the reduced effect of clock skew. Having the key feature of scalability, it is more suitable to utilise serial based communications to reduce the complexity of the network design. Asynchronous data communication can be achieved by either encoding the clock signal to be sent with the data[62] or by using an over-sampling technique, at a higher data rate, to recover the data at the receiving side[63].

2.2.2 Bus-based Topologies

A majority of multiprocessor systems interconnection architectures fall within the SMP design type[8 64 65 66 67], or global shared bus system; one of the simplest interconnect structures. In this, the communication medium (the 'backbone' bus) is shared by all integrated devices and only one device (bus master) can drive the network or access the bus slave at a time. A basic block diagram for a SMP System is shown in Figure 2-1.

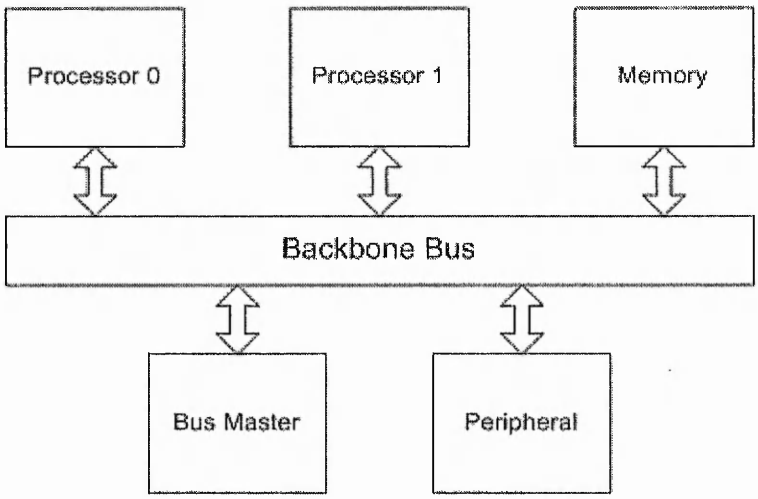


Figure 2-1Block Diagram for Shared Medium System

The bus used is usually a convenient and low overhead interconnection for a small number of active processors and bus masters, and a large number of passive modules (bus slaves) that only respond to the request from bus masters. The bus bandwidth must be shared by all the bus masters that can access that system. This results in effective bandwidth for each processor being inversely proportional to the number of bus masters[68]. Due to nature of central memory sharing, an error in memory might cause the whole system to crash.

Bus arbitration mechanisms are required when more than one processor or bus master attempts to access the bus simultaneously. A critical issue in the design of a shared medium bus is the 'arbitration strategy' that will assign the bus ownership of the system and resolve the access conflicts by multiple bus masters. Arbitration is performed in a centralised fashion by a bus arbiter module. The arbiter must be carefully designed to prevent any processing node from monopolising the memory access, and to resolve access contentions. Therefore any bus master wishing to access the memory module or peripheral must gain bus ownership from the arbiter.

2.2.3 Point-to-point and Switch-based Topologies

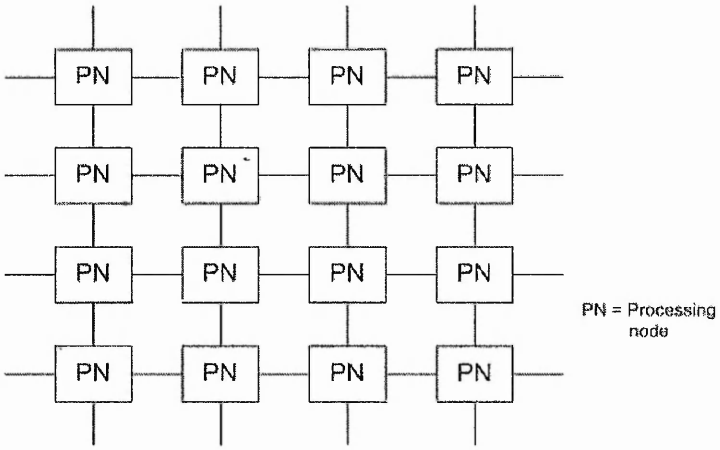


Figure 2-2: 2-DMesh Network Topology

Point-to-point networks were used to overcome the scalability problem of SMP systems. They were the network architecture used in MPP systems, as a communication medium between processing nodes. Each node was connected directly with the adjacent neighbouring processing nodes. Figure 2-2 shows an example of point-to-point network. The processing nodes here are arranged in a 2 dimensional Mesh topology where each processing node is connected to 4 adjacent nodes. This network topology offers guaranteed bandwidth and low latency between two adjacent processing nodes, due to the exclusive link connection between them. Therefore no arbitration is required and no access conflict occurs as there is only one communication channel per connection link. However, when sending a message to a non-adjacent processing node, for example from processing node 1, through processing node 2, to processing node 3, extra effort and time must be used by the intermediate processing node 2 to forward that message instead of running their own dedicated task. This kind of regular network topology benefits from reduction in cost and complexity, however were only optimised for systems with specific communication patterns.

Switch-based networks are an alternative to point-to-point networks. In switch-based networks, interconnections between processing nodes are via a set of message routing devices (routers). Each processing node will have a network adaptor, which connects to a port of a router. The implemented switch does not perform any information processing. Instead, they provide a programmable connection between their ports. Communication paths can be setup or changed over time, depending on the application requirements [69]. The routers pass messages throughout the network and allowed simultaneous transfer of messages provided there is no contention for the same destination node. Because there is no direct route between processing nodes (all communications are via routers) switch-based networks allow one processing node with a single communication channel to connect via an *n* channel router to communicate with *n*-1 of processing nodes. Routers that implement full crossbar architecture allow *n/2* bi-directional communication to take place simultaneously when there is no contention for the same destination or output port, as shown in Figure 2-3 which illustrates the NTU research groups ICR C416 (16 channel) commercial message router [27].

The switch-based network offers bandwidth guarantee irrespective of the network size or topology. Due to its flexibility, switch-based networks were very suitable to form irregular networks where network layout is independent of size and application. As the number of processing node in the system increases, latency will be introduced because a message might have to pass through more than one router. The latency introduced by a router will depends on the type of router used and the traffic conditions.

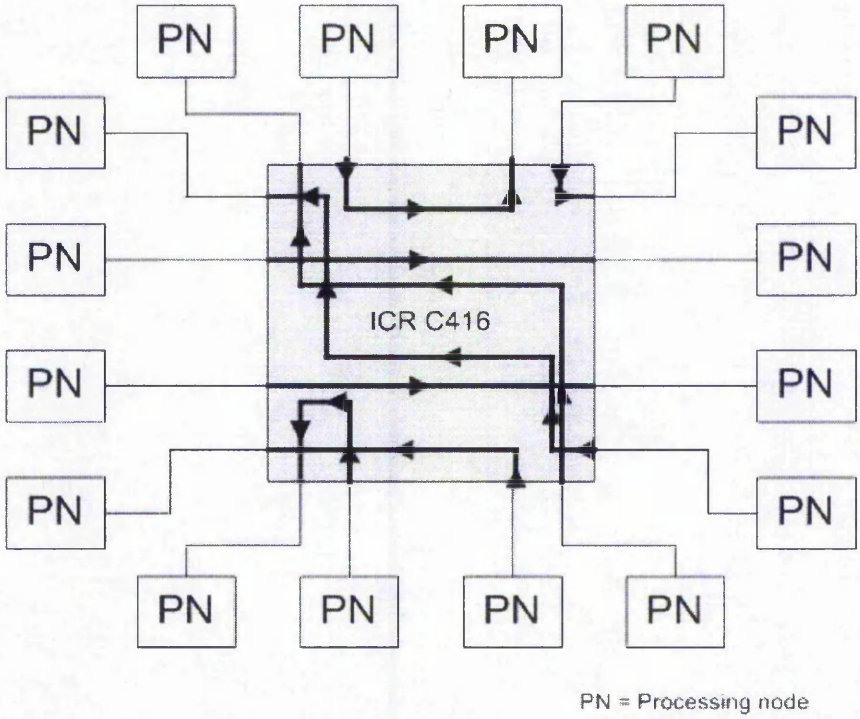


Figure 2-3: Simultaneous communication using ICR C416 hardware router

2.2.3.1 Switching Methodologies in Switch Based Network

The Store-and-forward methodology is one of the communication methodologies used in multiprocessor systems. As named, it stores first and forwards later: a packet or message will be forwarded from one switch to another when the latter has enough storage space available for the entire message. The entire packet has to be stored in the network switch before being transmitted to the next destination. This approach demands very high storage for buffering purposes in the switch and it can incur very high communication latency. The latency of any message is directly proportional to the message size and increases with the number of switching devices between the source and destination, as well as the traffic conditions of the network. This Store-and-forward approach is rarely used as a SoC communication method due to limited memory resources on the chip.

The Virtual Cut-through methodology[70] is an approach to overcome the penalty introduced by store-and-forward. A packet is forwarded as soon as the header is received and resources, such as buffer and channel approval, are acquired without having to store the entire packet. The message's tail information will release the output channel as it passes. In this approach, if a packet is blocked due to the utilisation of the same path by another message, the content of the message will then be buffered until the message path is free. More advanced versions of the Virtual Cut-Through approach, such as used in BLAM[71], utilise bypass buffers, which allow the new arrival to pass through the switch by storing the entire blocked message into the bypass buffer. This provides the advantage of low latency communication but requires enough buffering for blocked messages.

Comparing with Store-and-forward and Virtual Cut-through methods, the Wormhole Switching[29 72] method has more efficient use of buffer space. It connects the incoming message to the output channel as soon as the routing information has been received and the connection resources are available. As a result, switching latency is minimised. Wormhole switching can be implemented with minimal buffering resources at each switching device as the message is effectively distributed across the network. The primary advantages offered by this approach are minimum message latency and minimum buffer requirement. However, because the channel bandwidth is dedicated to one message, a message block can cause the channel to go idle or deadlock.

2.2.3.2 Switching Network Flow Control

All routing/switching methods using buffering need to communicate with the neighbouring processing nodes or switching devices when message passing takes place, to ensure the availability of buffers. Resource management will inform the upstream node when they should stop sending flits (a flit is the smallest possible unit of information in a message) due to the downstream buffer being full and vice versa.

In 'credit-based' flow control[73], each channel consists of a flit control counter to manage the number of free flits. Each input flit will consume an empty flit at the output buffer, the flit counter will decrease by one until it reaches zero, and the buffer is full. Therefore no further flits can be received until the buffer is free again. Once a flit is forwarded and an associated flit is freed, a credit will be sent to the upstream router. Meanwhile, the control counter will be incremented by one. For each flit received, a corresponding credit is eventually returned. Forwarding a flit to the downstream router will involve immediately returning a credit flit to the upstream router. Credit based flow control requires significant amounts of upstream signalling: it can introduce large overheads, especially for small flits.

In 'ACK/NACK' flow control[74], the upstream router will send a flit whenever bandwidth is available. The downstream router will accept all the flits as long as the buffer is available and an 'ACK' flit will be sent upstream. However, the flit will be dropped if no buffer is available and an 'NACK' will be sent upstream as notification. The upstream router must remain and hold the previous flit until an 'ACK' flit is received. An 'ACK' flit will only be sent by the downstream router when its buffer is freed. When the upstream router receives the 'NACK' flit, it will retransmit the dropped flit and in addition to the N succeeding flits that were transmitted during the round trip delay before continuing to transmit the remainder of the message. This flow mechanism is effective when it is applied to a routing device with a large buffer resource because the sent flit must be retained (in case a 'NACK' flit is received). A blocked downstream resource will incur poor link utilisation.

Finally, 'On-Off' flow control[74] is a widely used flow control mechanism that greatly reduces the amount of upstream signalling compared to 'ACK/NACK'. A signal will be sent to the upstream router only when it is necessary to change the state of permission. An 'On' flit indicates a flit transmit is permitted and 'Off' means a flit transmit is not permitted. In some case, an 'Off' can be sent to indicate that the number of free buffer spaces is equal or below a pre-defined threshold, so that the upstream router knows how many more flits can be transmitted before it has to stop and wait for the

permission flit. Similar to the credit-based flow control, one must take consideration of the availability of the buffer before it becomes empty, then sending an 'Off' flit (so that there is always enough buffer to receive those flits, that are sent before the 'Off' flit is received by the upstream router). However, with adequate buffering and management mechanisms, 'On/Off' flow control can operate efficiently with very little upstream signalling.

2.3 Message Router System Comparisons

2.3.1 Myrinet System

Myrinet is a cost effective, high performance, packet switching technology that is widely used in distributed computing systems[55]. It is used to interconnect clusters of workstations, PCs, servers or single-board computers. The Myrinet network system consists of two main components, the Myrinet's computer interface component and the Myrinet switch[55].

Myrinet's computer interface connects a processing node to the network. There are two memory blocks in the Myrinet computer interfaces and they are used for transmit and receive packet buffering. A DMA engine transfers the data packet between the processing node's memory and Myrinet's network interface. Meanwhile the Myrinet switches are multiple-port switches that employ Virtual Cut-through routing. If the selected outgoing channel is not already occupied by another packet, the head of the incoming packet is advanced into this outgoing channel, as soon as the head of the packet is received and decoded. The packet is then spooled through this established path until the path is broken by the tail of the packet. If the selected outgoing channel is occupied by another packet or is blocked, the incoming packet is blocked. Switches are powered separately from hosts, so that the network will continue to function even when some of the hosts are turned off.

2.3.2 Xpipes System

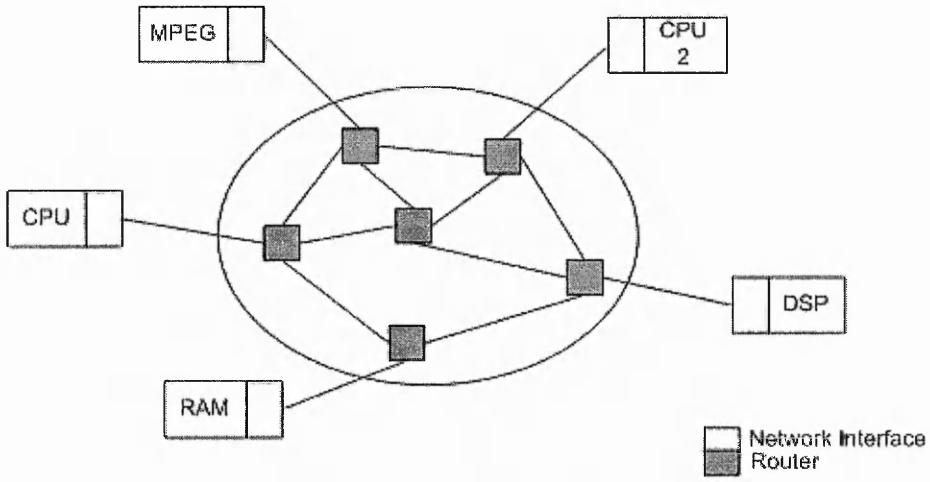


Figure 2-4: Example of Xpipes NoC.

Xpipes architecture utilises on-chip packet-switched micro-network of interconnects, known as a Network-on-Chip (NoC) architecture. Xpipes architecture consists of two main components: the network interface and the router. Each component interconnected to the on-chip micro-network via network interface. Routers were used to interconnect implemented components (master components and slave components) in SoC as shown in Figure 2-4. The network interfaces use Open Core Protocol (OCP) and convert the OCP to adapt to the network protocol. Designers can specify the arbitrary network topologies to meet the system requirement and to optimise the overall performance of the SoC.

Routers implemented in the Xpipes utilise Wormhole switching methodology to reduce router memory requirement and allowing low latency communication. The retransmission policy (GO-BACK-N) was implemented as 'ACK/NACK' flow control was used. Flits were transmitted continuously without waiting for the 'ACK'. When a 'NACK' is received, the transmitter will retransmit the negatively acknowledged flit in addition to the N succeeding flits that were transmitted during the round trip delay. The

downstream node will discard N-1 of the received flits following the corrupted flits regardless of they are error free or not.

2.3.3 OS-Link Based System

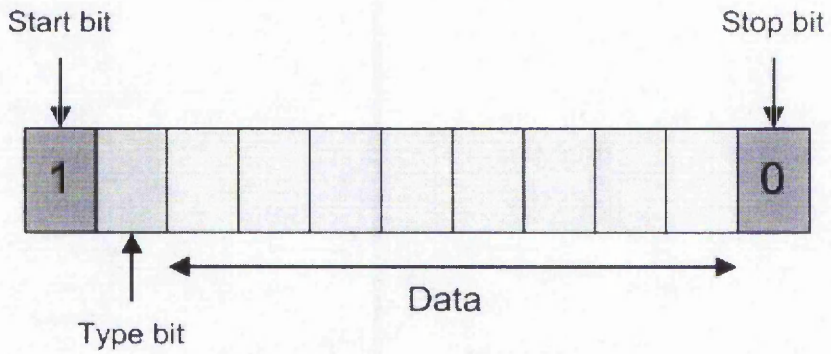


Figure 2-5: Token format for OS-Link protocol.

The OS-link is an oversampling serial link communication protocol, originally designed for the Transputer and is used by the early routers designed by the NTU research group. It is a bi-directional full-duplex communication protocol with data transferred over the link using tokens. The oversampling technique was capable of achieving a maximum bit rate of 44 Mbit/s. To provide more efficient support for higher level protocols, the size of the token is kept as small as possible. Figure 2-5 shows the format of a token. Each token was marked by logic '1' as a Start bit, and ends with a Stop bit, logic '0'. Logic '1' in the Type bit will indicate the following 8 bits were a byte of data; logic '0' in the Type bit will indicate that the following 8 bits were control information.

2.3.3.1 The ICR C416 Message Router

The ICR C416 [27] is a hardware routing switch developed by the NTU research group. It was marketed by IC Routing Ltd (now dormant) for use as interconnection

between the first generation Transputers[5] in embedded control applications[75]. The 16 channel dynamic router switch architecture [76] allows up to 8 bi-directional communications simultaneously when there is no output contention. It is an asynchronous serial communication network device with each channel consisting of a pair of full duplex lines transferring data at rates of either 10 or 20 Mbit/s. The ICR C416 uses an Over-Sampling technique for data recovery at the receiver. Therefore no clock information is encoded into the data stream.

Data is transmitted in the form of tokens and each token consists of 11 bits. The resultant routing switch has the maximum theoretical unidirectional data throughput of 14.55 Mbit/s when it is configure to run at data rate of 20 Mbit/s[77]. The credit-based flow control was implemented in this hardware routing switch, an acknowledgement of every token is required for every token sent. The acknowledgement token consists of just 2 bits, a Start bit and a Stop bit. Due to the requirement of an acknowledge token the actual conveyed bits per token data (or control) byte in the bi-directional communication is 13 bits. Taking this into account, the credit-based flow control gives a theoretical maximum bi-directional data throughput of up to 12.31 Mbit/s when operating at data rate of 20 Mbit/s. However, practically, the data rate is lower due to the factors such as transmission length, network traffic and the receiver buffer status.

Messages in the ICR C416 network are divided into 256 bytes per packet and the maximum message length is 64 kBytes. A packet consists of a Header section, a Length section, and finally the payload. The Header section contains two bytes of output link information and one message identity header for a particular message. The header bytes will be stripped, one by one, as they pass through the routers, and the message identity with the most significant bit of '0' will be identified at the destination node. It will followed by the length information decoding. The OS-Link based network utilises wormhole routing to minimise buffering requirements.

The ICR C416 is designed to provide a simple, flexible and low cost solution to interconnect multiple processing nodes in an irregular embedded network. Although it

operated at a lower link speed than its current network interconnects devices, its generic format suited many applications because it provided direct communication between processors and it could be easily cascaded to form a larger embedded network. The features of minimal wiring and low pin count are other advantages offered in a physically distributed network. When differential transceiver circuit are implemented on the board, it was capable of operating at a data rate of 44 Mbit/s over 100m of CAT 5 unshielded twisted pair cable[54].

2.3.3.2 The NTR-FTM08 Message Router

The FT-PCI-Li[42] interface development board had an NTR-FTM08 routing switch with enhanced tolerance to faults features and an OS-Link PCI interface. The FT-PCI-Li was implemented in a PLD (Altera's APEK20KC[43] device) which offered a high density of logic functions with programmable features, large amounts of programmable embedded memory (which support Content Addressable Memory[78] (CAM)), and a high speed I/O interface. Enhanced features implemented were the distributed fault detection, isolation and recovery mechanism, and the utilisation of CAM[79] as hardware virtual channel in order to reduce processor intervention. The CAM was used to store up to 16 expected message IDs with pre-allocated memory addresses which can be pre-loaded at any time. The use of CAM reduces the need of the processor to allocate the memory address in the DMA channel in the event of the arrival of new message.

The FT-PCI-OSLi was clocked at maximum sample clock frequency of 66 MHz, which gives a 42 Mbit/s data rate; NTR-FTM08 was clocked at the same sample clock frequency, 66 MHz, giving 42 Mbit/s data rate for each channel. Therefore, with 8 channels, NTR-FTM08 was able to give approximately 336 Mbit/s when 8 channels operate simultaneously with no output port contention. The StrongARM and PCI interface allowed the RISC and general purpose processors to operate simultaneously in the same network as processing nodes. Using the NTR-FTM08 allowed up to 8

processing nodes to be interconnected or using extra NTR-FTM08 to cascade ands increase the size of the scalable network.

The NTR-FTM08 also supports the Group Adaptive Routing technique. The Group Adaptive Routing technique allows different output ports of the routing device, which connect to the same destination node, to be grouped together. When one channel was busy with message passing and a new message was designated to the same output port, instead of waiting for the first one to finish, the routing device can provide an alternative path, which will reach the same designated node (the designated node or following routing device must have 2 or more channels interconnected).

FT-PCI-Li utilised a Permission Based or 'STOP/GO' mechanism for data flow control. This method is also used by Myrinet for control of message passing within it's Massively-Parallel Processor (MPP) system[80]. The 'STOP/GO' (Xon/Xoff) simply means message transmission will either stop or continue depending on the status of the receiver's buffer. Figure 2-6 shows the data flow control between processors in the communication network using this adapted protocol (adapted from OS links).

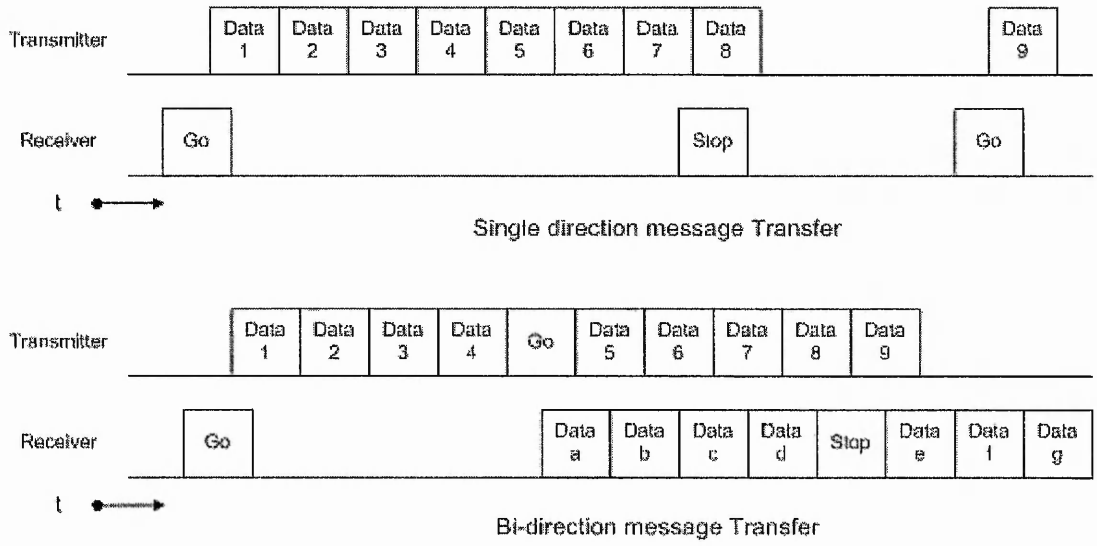


Figure 2-6 Flow control of the adapted OS-Link based system

Initially a GO/Xon is transmitted by the receiver node to indicate that the receiver node is ready to receive data. The transmitting node will start to transmit data as soon as a Xon token is received. The received data token will start to fill up the receiver node's token buffer waiting to be de-packetised. The receiver side will send a STOP/Xoff token when the receiver's buffer has reached a pre-determined level, referred to as 'Almost Full', as shown in Figure 2-7. The Xoff/STOP token will stop the transmitting node from sending any more data until the next Xon/GO token is received. This will give the receiver node time to clear its buffer before it overflows. The receiver node will send the Xon/GO when the buffer level reaches the pre-determined level, referred to as 'Almost Empty.'

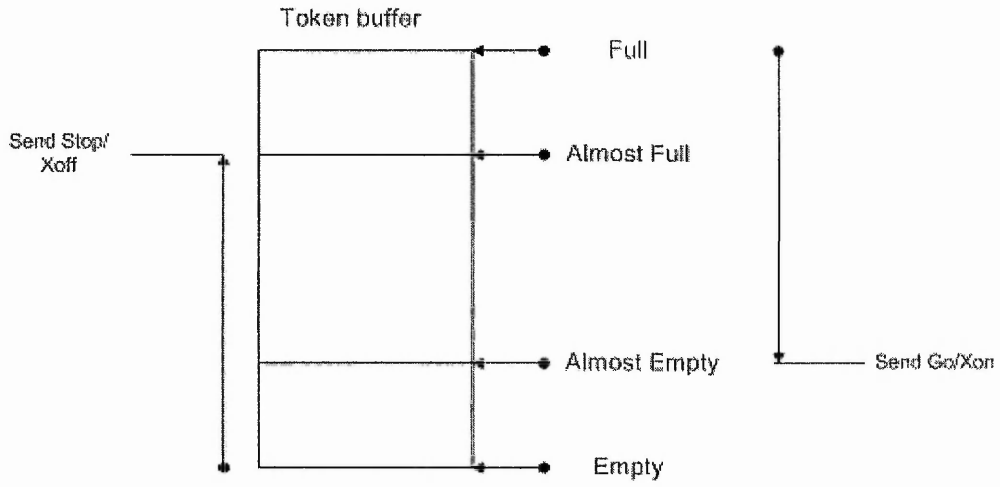


Figure 2-7 Receiver buffering control

The buffer gap between the 'Almost Full' and 'Full' was used to fill up the tokens that were sent by the transmitting node before the Stop/Xoff was received; Go/Xon was sent when it reach 'Almost Empty' to resume the transmission before the buffer was actually empty. This will provide more efficient use of the communication bandwidth between the two processing node. The End of Message token (EOM) and End of Packet token (EOP) are also used to indicate the termination of message and termination of packet respectively. Finally the Bad End of Packet (BEOP) token is used to indicate when a fault condition occurs.

3 Multiprocessor Platform

FPGA devices have emerged as prototyping tool alternative in digital design in many markets. It can also be used to prototype custom ASIC designs before manufacturing. Evolution of FPGA technology has changed rapidly and greatly increases its programmable element capacity. FPGA devices' are starting to be integrated into SoC because they can now implement most (or all) of the functions of a complete electronic system. The system may contain memory, processors, specialized logic, communication buses and other digital functions.

Multiprocessor SoC (MPSoC) design is a complex process that involves different steps at different abstraction levels. It can be grouped into two major tasks: design space exploration and architectural design. Design space exploration involves hardware or software partitioning, selection of architectural platform and components; architectural design refers to design of components, for example hardware and software interface design.

3.1 Digital System Implementation

An overview of some of the issues regarding digital design prototyping and implementation solution is offered in this section. This section is not intended to be a definite discussion on this open-ended subject, it focuses mainly on solutions utilised in the development of the OSL-ST2.

The FPGA is one type of Programmable Logic Devices (PLD). It is a semiconductor device containing programmable logic components and programmable interconnects. It allows the implementation of custom digital designs tailored specifically to applications. Early FPGA devices included gate numbers in the order of tens or hundreds and were mainly used to implement 'glue' logic. The growing sophistication of

FPGAs has had a great impact on digital design. The number of transistors and logic gates that can be implemented on a single chip has increased dramatically. FPGA technology continues to advance and mature, making a SoC possible. This is due to the increase of clock speed, reduced power consumption and reduced cost in contrast with an increase of logic element count. The hardware capacity offered by modern FPGA devices is a compelling proposition and gives designers the opportunity to integrate more functions in a single chip.

Application Specific Integrated Circuits (ASICs) are full custom device usually produced in the final stage of mass production. For very high volume production, per unit costs become very low. However for prototyping and small production runs the FPGA is a better solution, it requires less investment in terms of time (laying the new design out on silicon when redesigning due to error(s)) and money (cost of die production). The programmability and flexible nature of FPGAs allows them to become an alternative when prototyping a design. Upon the occurrence of design errors or the need of re-design, the same FPGA can be reprogrammed for verification (until error free) or provide a means for design optimisation before the product reaches the manufacturing stage.

A lot of designs are built on previous designs, common features and components being retained. Recycling these reusable parts of the design, called Intellectual Property (IP)[81], can significantly reduce the design time or cycle and thus increasing the cost effectiveness. IPs are now also functions offered by FPGA vendors. These IP cores are pre-verified functions that can be easily plugged in, easing the design effort.

These IP cores come with either source code or a net list. They have been optimised to work on a manufacturer's architecture. By providing components in IP form, designers can choose the components that are required in their design up to the capacity of the FPGA device used. Some of the IP cores provided are industrial standard functions, such as: PCI[34], PCI Express[82] and Double Data Rate Synchronous Dynamic Random Access Memory[83] (DDR SDRAM) Controller. These IP cores help accelerate the design process.

Later FPGA devices came with on-chip embedded processors. There were two kinds of embedded processor available: the 'hard' embedded processor and the 'soft' embedded processor. An example of an FPGA that comes with a 'hard' processor is the Excalibur device family from Altera. The Excalibur device family is the combination of an industrial standard 32 bit ARM™922T RISC processor and APEX20™KE-like programmable logic. An Excalibur chip was divided into two partitions, the embedded processor stripe and the programmable logic partition. The embedded processor stripe consists of an ARM™922T RISC processor and other basic components including a Phase Lock Loop (PLL), a Timer, and an interrupt controller. Further details can be found in Chapter 4. Meanwhile, examples of the 'soft' processors are the NIOS[84], NIOS II[46] from Altera and the MicroBlaze™[85] from Xilinx.

3.2 Processor Choice for an Embedded SoC

SoC designs are powered by one or more general processors, digital signal processors, or fixed-function co-processors. Understanding of processor architecture provides a context for choosing the right processor(s) for the SoC. Embedded processors are general purpose in a different sense than the high performance processors used in personal computers. A personal computer is expected or mostly used to run arbitrary software: word processing, computer aided design (CAD), games, multimedia, and the Operating System itself; whereas a closed embedded system runs a fixed set of tasks. Embedded processors also can have extra application specific instructions implemented.

The micro-architecture of a RISC microprocessor reflects the nature of the instruction fetch/execute cycle. The instruction execution is divided into multiple stages, forming an instruction pipeline. An example of a simple pipeline is the five-stage pipeline shown in Figure 3-1. The first stage is the instruction fetch from the memory (IF). The next stage is the Instruction Decode (ID); the instruction is decoded and its operands are read from register file. Then the instruction is executed (EX stage) using the operands. In the next stage, memory access (MEM) to retrieve or to store a value, for load and store

command respectively, using the computed address. Finally, register update as a result of instruction execution (write-back WB). The Pipeline increases the performance by simultaneously processing multiple instructions in different stages. It takes 5 cycles to fill the pipeline assuming that the pipeline can be kept flowing smoothly. There are conditions, such as execution of the branch with conditions, which will stall the pipeline and techniques to minimize the stalls. These techniques are the characteristic of the high performance microprocessor. The difference between the embedded and high performance processor lies in their stage of evolution.

In this section, architecture of ARM922T processor from ARM, MicroBlaze™ from Xilinx and NIOS II from Altera were surveyed.

IF -> ID -> EX -> MEM -> WB

Figure 3-1: Simple Five-Stage Pipeline

3.2.1 The ARM922T

The ARM922T [86] is member of ARM9TDMI family of general purpose microprocessors. It is a high performance RISC processor, which supports both 32 bit ARM and 16 bit Thumb instruction sets, allowing the user to trade off between high performance and high code density. The processor is targeted at multimedia applications, such as smart phones, Personal Data Assistants (PDA), digital cameras, and set-top boxes; also at other embedded applications, for example as communicator, audio and video decoding and platform OS based device.

It has a Harvard architecture (separate instruction and data path), implemented using a five stage pipeline, consisting of Fetch, Decode, Execute, Memory and Write stages. It also has a trace interface port that allows the use of trace hardware and tools for real-time instruction and data tracing (for the ease of debugging and the development of application software, operating system and hardware).

The ARM922T has been designed to work with the Advanced Microcontroller Bus Architecture[89] (AMBA) unidirectional Advanced System Bus (ASB) interface. It also contains the necessary extra control signals to enable the implementation of both the Advanced High Performance Bus (AHB) and ASB interfaces of AMBA. The ARM922T can be used as single master with no additional logic, or used with multiple bus masters with only the granted master controls to drive the bus system according to AMBA specification version 2.0.

3.2.2 NIOS II

The NIOS II[46] is a 32 bit general purpose RISC processor introduced by Altera Ltd. NIOS II is a configurable softcore processor. Features can be added or removed in order to meet the performance and price requirements. The concept of 'softcore' is related to flexibility for the hardware engineer to optimise the system implementation according to the design requirements, as the processor core is offered in a soft design form instead of a chip or FPGA 'stripe'. It is to be implemented onto NIOS II supported FPGA families, together with other peripherals.

The configurable nature of the NIOS II processor also allows the integration of custom instructions directly into the Arithmetic Logic Unit (ALU). System performance can be increased by offloading portions of the software code to hardware functions, extending the CPU instruction set to accelerate time-critical software.

Currently there are three NIOS II processor cores available that implement a common instruction set architecture, each optimised for specific price and performance requirements. The processor cores are in standard form, economy form and fast form[87]. Table 1 shows some of the differences between the NIOS II processors.

Features		Core		
		Economy	Standard	Fast
Description		Minimal core size	Small core size	Fast Execution Speed core
Performance	Max DMIPS	31	127	218
	F_{max} (MHz)	200	165	185
Area		<700 LEs; <350 ALMs	<1400 LEs; <700 ALMs	<1800 LEs; <900 ALMs
Pipeline	stage	1	5	6
Instruction Bus	Cache	-	512-64 kBytes	512-64 kBytes
	Pipeline memory access	-	Yes	Yes
	Tightly Couple Memory	-	Optional	Optional
Arithmetic Logic Unit	Hardware Multiply	-	3 cycle	1 cycle
	Hardware Divide	-	Optional	Optional

Table 1: Some of NIOS II Processor cores features

Note1: DMIPS: Dhrystone Million Instruction per Second is a synthetic benchmark for general processors (CPU).

Note2: LE means Logic Element

Note3: ALM means Adaptive Logic Module

The NIOS II economy core is the smallest core of the three. It was designed in such a way that it consumes the least resource among the three while remain compatible with NIOS II instruction set architecture; the NIOS II fast core consumed the most resource as it was designed for high execution performance; the NIOS II standard core was optimal for cost sensitive, medium performance applications. The standard core has medium performance and medium resource consumption compared to the other two cores.

3.2.3 MicroBlaze

The MicroBlaze[85] is a softcore processor from Xilinx. It is a 32 bit RISC processor optimised for implementation in Xilinx FPGA devices. It is a highly configurable

processor as it allows the user to include specific sets of optional features to suit the design requirements. There are a few fixed features in the processor: thirty two 32 bit general purpose registers, 32 bit instruction word with three operands and two addressing modes, 32 bit address bus and a single issue pipeline. Some of the optional features are: Hardware Barrel Shifter, Multiplier, Divider, Floating Point Unit (FPU) and Fast Simple Link (FSL) Interface.

The MicroBlaze supports up to three interfaces for memory access. The Local Memory Bus (LMB) provides single cycle access to an on-chip dual-port RAM block. The IBM On-chip Peripheral Bus[88] (OPB) was used to interface on-chip and off-chip peripheral and memory. The CacheLink Interface is for use with a specialised external memory controller. The system designer can chose a suitable memory access interface to suit and optimise the utilisation of the implemented memory modules.

The MicroBlaze supports up to 8 FSL channels. The FSL channels are dedicated uni-directional point-to-point data streaming interface between and output First-In-First-Out (FIFO) and input FIFO. This can be used for fast transfer of data between master and slave that utilise FSL. The FSL can be used with custom hardware acceleration functions. This is similar to implementing a custom instruction, with the benefit of not making the overall speed of the processor pipeline dependant on the custom function.

3.3 System Bus Architecture

The system bus architecture has great influence on the overall performance of the system because each system has a different combination of components and requirements. In a complex embedded system, not all implemented slave devices are high speed, low latency devices such as Universal Asynchronies Receiver and Transmitter (UART) and Programmable Input/Output (PIO) device; and also a large number of the embedded systems have more than one bus master. There are a few possible bus architectures

available and each offers different arbitration schemes and benefits to suit the requirements of the target system.

3.3.1 The AMBA Bus

The Advanced Microcontroller Bus Architecture[65] [89] (AMBA) is one of the commonly used interface bus types in embedded systems. It is an open standard bus used to interconnect the processor and all other implemented functional blocks. It is designed for the ease of development of embedded microcontroller products with one or more processors. An important feature of AMBA is that it is technology-independent, which ensures that highly reusable peripheral and system macrocells can be migrated across a different range of IC processes and be appropriate for full-custom, standard cell and gate-array technologies.

The AMBA consists of a high performance backbone bus, AHB or ASB, which are used to interconnect high performance functional units such as microprocessor, high speed on-chip or off-chip memory and other Direct Memory Access featured bus masters. ASB is the first generation AMBA. It is a multiple bus master system bus that supports burst transfer and pipeline transfer operation. AHB is the second generation AMBA. It inherits the advantages of ASB with new features added to support the latest embedded system designs. The new included features in AHB are: split transfer protocol, single-cycle bus master handover, and wider data bus support of up to 1024 bits. The AMBA specification 2.0 recommended minimum bus width is 32 bits and it is expected that a maximum of 256 bits will be adequate for almost all applications.

The Advance Peripheral Bus (APB) is the secondary local bus that is designed to interface devices that do not require too high performance, of a pipeline bus interface. A 'bridge' is required to convert AHB or ASB transfer into more suitable format for the slave devices attached in APB. The bridge will latch all address, data and control signal,

and then decode the address to select the appropriate slave device before performing the data transfer operation. Figure 3-2 shows a typical AMBA system.

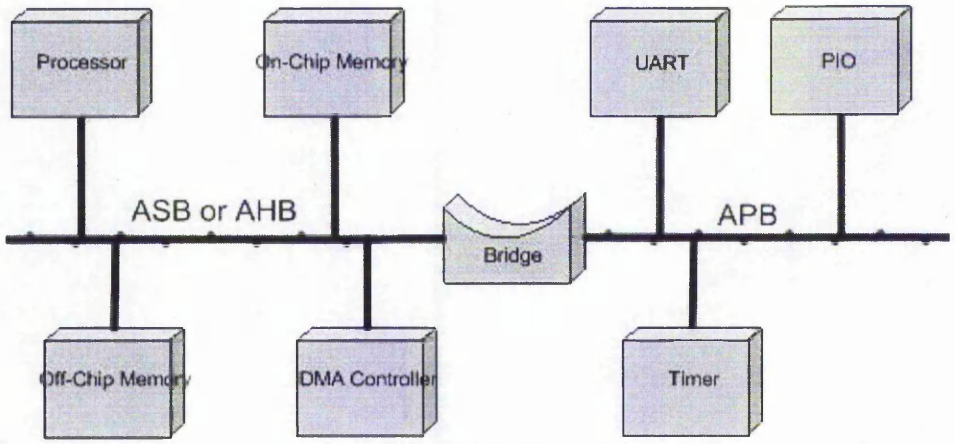


Figure 3-2: Typical AMBA System

In split transaction architecture[90], as implemented in AMBA, the operation of the bus master providing the address to the slave (when requesting an access) and the response operation of the slave to the bus master is separated. The bus master will only be allowed to perform data transfers when the slave device is ready. If the slave device is not ready, the bus master will have to retry again later or wait. Meanwhile, the bus ownership will be granted to other bus masters. This bus architecture can support multiple outstanding transactions, but it comes with the price of more complex design of bus master and bus interface.

3.3.2 The Avalon Switch Fabric

The Avalon Switch Fabric[91] is a specialised system interconnects technology from Altera. It is generated automatically by SOPC Builder, which is part of the Quartus II software.

The Avalon Switch Fabric includes chip select signals for each slave device. This provides an easy to interface to custom on-chip peripheral. It has a dynamic bus sizing feature to enhance the data transfer between a bus master and slave of different data bus widths. For example, the dynamic bus sizing logic will execute multiple bus cycles, to perform re-sizing and alignment, to fetch data value from peripheral with a narrower data bus size. For a system that has peripherals running in different frequencies, the switch fabric will add special circuitry to support the transaction between peripherals.

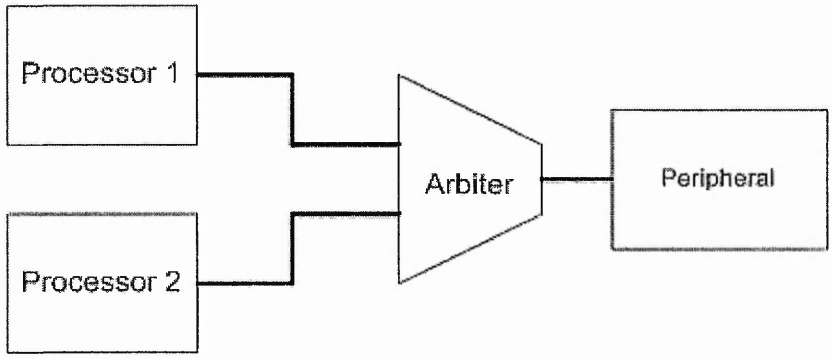


Figure 3-3: Slave Side Arbitration Technique

The Avalon Switch fabric also supports the simultaneous multiple bus masters. The Avalon masters and slaves interact with each other based on a slave-side arbitration technique, as shown in Figure 3-3. Each bus slave will have an arbiter. The arbiter will determine which interconnected bus master will gain access to that slave device when there are multiple masters attempting to access the same slave. This technique offers two advantages:

- Firstly, each bus master interfaces to the Avalon bus as if it was the only bus master on the bus, because the detail of the arbitration are encapsulated inside the Avalon bus. Therefore, the master and slave interface are consistent regardless of the number of masters or slaves implemented on the same system.
- Secondly, multiple masters can perform data transactions simultaneously, provided they are not accessing the same slave device in the same time, as shown in Figure 3-4.

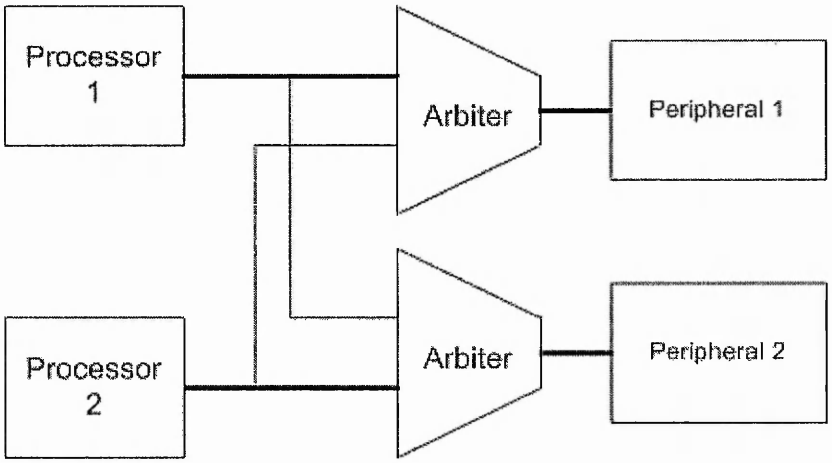


Figure 3-4 : Simultaneous multiple Bus master data Transactions.

The Avalon Bus Interface[92] is a simple bus interface implemented into designs to interact with the Avalon Switch Fabrics Interconnect. The principal design goals of the Avalon Bus interface were to provide a simple and easy to understand protocol; to optimise the resource utilization for bus logic; and synchronise with all other user logic implemented on the same PLD, while avoiding complex timing analysis. The Avalon bus specification supports different bus transfer types between the bus master and slave pair. The fundamental transfer types stated in Avalon Bus Specification version 2.3[93] are the transfer with fixed or variable latency type and the streaming transfer type. The latest Avalon Bus Specification[92] (renamed to Avalon Memory-Mapped Interfaces Specification) has included new transfer types to support different design requirements such as: burst transfer mode, pipeline transfer mode, and transfer with tristate property.

3.4 Modern Solutions in Multiprocessor Systems

Multiprocessor SoC design use complex on-chip network and interconnection to integrate multiple programmable processor cores, specialised memories and other intellectual property (IP) components on a single chip.

3.4.1 Cell Based Systems

Cell architecture[94], also called Cell Broadband Engine Architecture, was based on research conducted by IBM, Sony and Toshiba to provide a power-efficient, cost-effective high performance for wide range of applications such as gaming consoles, scientific calculation and data processing. It is a heterogeneous multiprocessor system that consists of one IBM 64 bits Power Architecture ™ core and eight specialized co-processors called Synergistic Processor Units (SPU). The system was integrated by a coherent on-chip bus as shown in Figure 3-5. It's design can be classified as SMP. All processing units rely on their DMA engines to access the memory resources for instruction and data fetching. A DMA engine has become an important component in modern multiprocessor systems as it consumes the idle slots in the interconnection bus to fetch instruction and data to prevent starvation of the processing unit. It can help to improve the efficiency of the overall performance of the system by keeping all the processing units busy by feeding enough data to it.

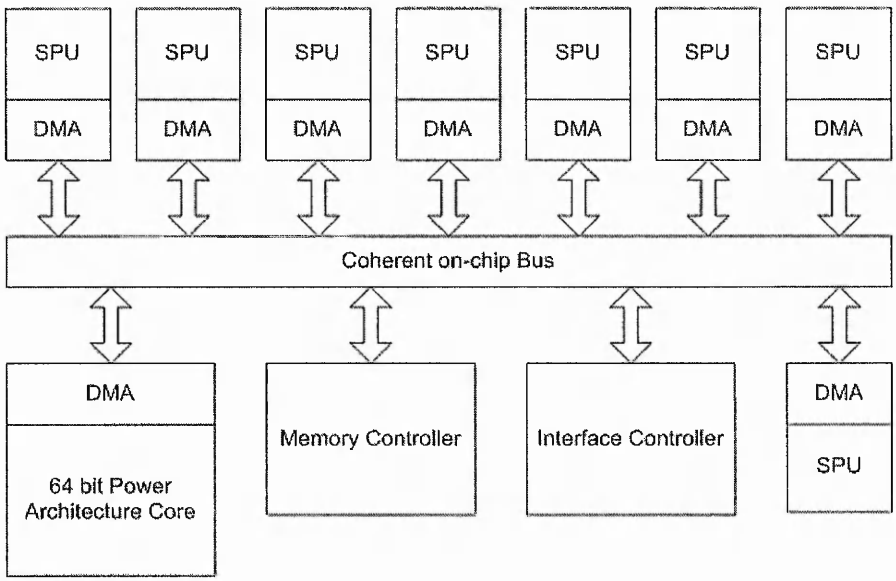


Figure 3-5 Cell architecture block diagram.

3.4.2 The XA10 System

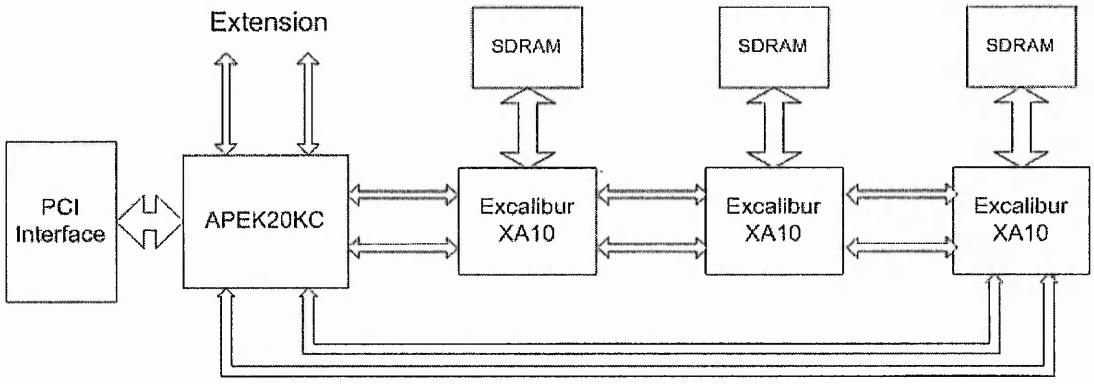


Figure 3-6: XA10 System Block Diagram.

The XA10[95] system was a distributed multiprocessor embedded system developed by IC Routing Ltd. It consists of three XA10 chips and an APEK20K chip. The three XA10 chips form three processing nodes, each processing node has its own dedicated on-chip RAM and external SDRAM memory resource. The APEK20K chip consists of an OS-Link Router and OS-Link based Network Interface Controller with PCI interface. Four chips were interconnected by OS-Link based network and they were linked together in a daisy-chain connection, as shown in Figure 3-6. Inter-processor communications were performed via message passing through the OS-Link network.

The PCI interface in the XA10 system enables it to be plugged into any system with PCI slots, such as a Host PC. In the case of plugging it to a Host PC, the Host PC can be used as a main control processing node which dedicates tasks to the three or more processing nodes[96] (the system can be cascaded by connecting it to more processing nodes via the OS-Link network). The detail of the XA10 designs will be covered in more detail in Chapter 4.

4 The XA1 System Prototype Board

4.1 Introduction

This chapter details the XA1 board, which was a prototype of OS-Link based networked distributed multiprocessor embedded system. The OS-Link network interconnects three processing nodes and a host PC. This chapter also details the alterations made to the protocol and the network interface controller in order to reduce processor intervention when passing messages.

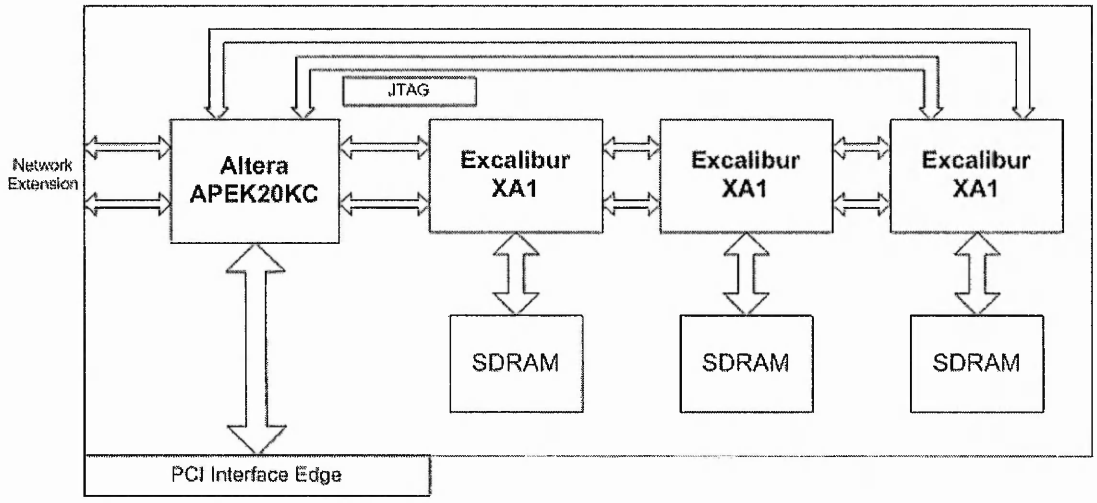


Figure 4-1 The XA1's Hardware layout

Figure 4-1 shows the hardware layout of the XA1 board including an APEX20KC chip, and three Excalibur XA1[44] chips. The three XA1 chips each provided one processing node. Each processing node had an in-built processor and an external 128 Mbytes SDRAM. The APEX20KC device consisted of a PCI bus interface, an OS-link Interface and an 8 port router. Figure 4-2 shows the architecture of the Excalibur XA1 chip with the OS-link Interface and NTR-FTM05 router. The NTR-FTM05 router was adapted from an NTR-FTM08 router, and modified (from 8 ports to 5 ports), to fit into

the XA1 chip (because XA1 chip had lower logic element counts than APEK20KC devices) and connected to form the chosen topology.

4.2 The XA1 chip

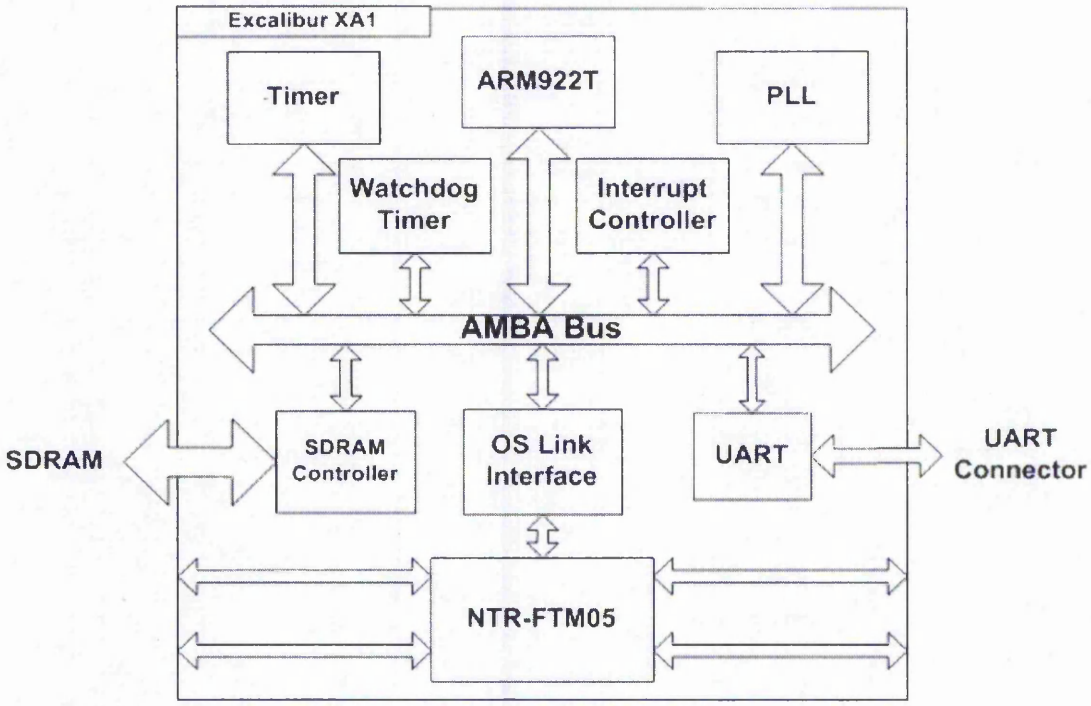


Figure 4-2: Excalibur XA1 device with the OS-link Interface and a 5 port router.

The Excalibur devices[44] are a combination of a RISC core processor and programmable logic in a single device. It integrates an industrial standard ARM922T processor, on-chip memory, and peripheral with APEK20KE device-like architecture. The ARM922T[45] processor and peripheral such as PLL, Watchdog Timer, Timer, Interrupt controller, UART, and SRAM were in-built to the 'Embedded Stripe'.

The OS-Link Network Interface Controller and NTR-FTM05 router were implemented in the APEK20KE device-like architecture part of the XA1. The OS-Link Network Interface Controller formed the interconnection between the processor and the

OS-Link embedded network. The NTR-FTM05 router was used to interconnect that processing node to other adjacent processing nodes.

4.2.1 XA1 Internal System design

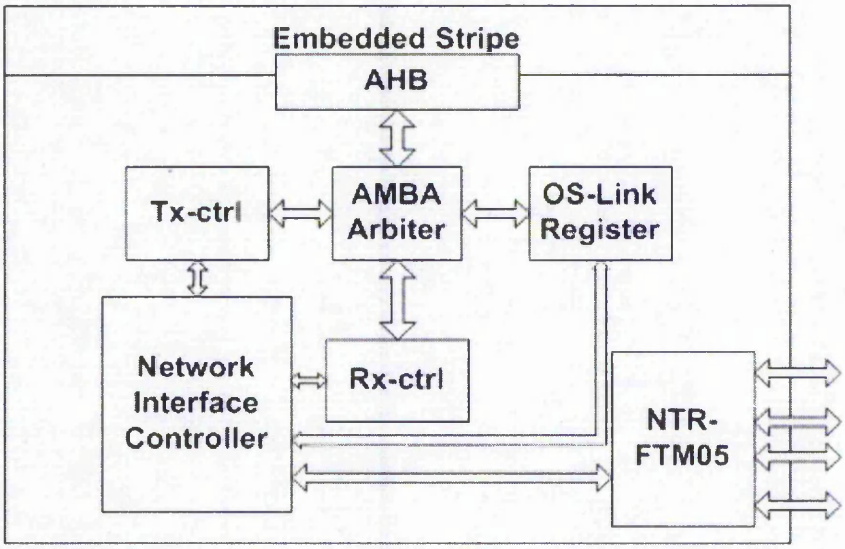


Figure 4-3: Block Diagram for Component Interconnection in PLD area.

Figure 4-3 shows the block diagram of the design modules implemented in the APEK20KE device-like architecture part of the Excalibur Chip. The AMBA bus arbiter was in charge of deciding which bus master granted the permission to drive the bus to access a memory-mapped bus slave. There were three bus masters in each XA1 device. They were the ARM922T Processor, the Transmitter DMA Controller (Tx-ctrl) and the Receiver DMA Controller (Rx-ctrl). The data transfer between the Embedded Stripe and the PLD was performed via the AHB Bridge.

Transmitter DMA Controller

The Transmitter DMA Controller module was the AMBA interfaced DMA controller. The DMA was initiated by the processor by loading the target memory address,

the header information and the length of the payload in order. Then the Transmitter DMA Controller would request memory access. The Transmitter DMA Controllers accessed target memory as soon as permission was grant by AMBA arbiter, acquired data would then be stored in the DMA buffer (FIFO). The value in the address register indicating the target location was incremented by 4 (because it is a 32 bit data transfer) each time a valid data was sampled from the data bus, while the value in the Length register was decremented. When the value in the Length register reached zero, the Transmitter DMA Controller terminated the DMA operation. The Transmitter DMA Controller paused and released the DMA access if the DMA Buffer signalled that it was full. It would request memory access again to fetch the remainder data when a further 'kick-start' was asserted, when the DMA buffer was almost empty.

Receiver DMA Controller

The de-packetised data was stored in the receiver DMA buffer (FIFO). The Receiver DMA Controller started the DMA operation when the DMA buffer was nearly full and a 'kick-start' signal was asserted. The Receiver DMA Controller first made a memory access request to the address input into the receiver channel's Address Register. When the request was granted it started fetching data from DMA Buffer and writing it to the memory module via the data bus. The Address register was incremented by 4 for each 32 bit data written to the memory module. The Receiver DMA Controller released the DMA when the DMA Buffer was emptied. If the message was unfinished, a 'kick-start' signal was asserted again when the DMA Buffer is filled up.

DMA Register

The DMA Register was a list of registers associated with the network interface controller. It had the control registers which were used to kick-start DMA operations and also the status registers used for monitoring purposes. It was an AMBA bus slave module that's accessed by the processor in the Excalibur chip. To transmit a message, the processor inputted the address location where the message was stored to the transmitter's

address register, followed by the length of the message into the Length register (writing to the Length register will kick-start the DMA). To prepare the receiver to start a DMA transfer, the address allocated to the new message was inputted into the receiver's Address register, followed by the expected length into the Length register. The status registers showed information about the network interface controller, such as the device's ID, current operation, and received header information. This module was particularly useful when debugging the network interface controller because signal information from other module could be sampled and accessed by the processor.

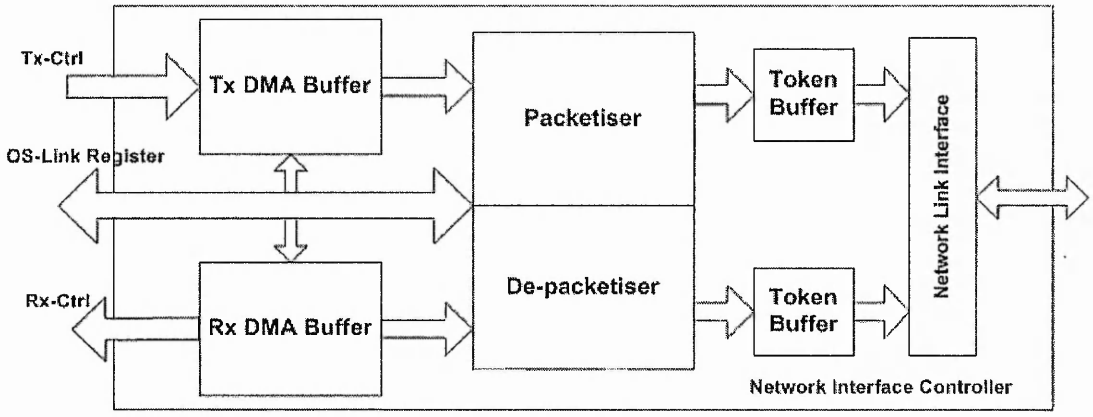


Figure 4-4: Block diagram for design modules in Network Interface Controller of Excalibur chip.

The Network Interface Controller was the interconnection between the processing node and the OS-Link network. Figure 4-4 illustrates the main modules implemented in the Network Interface Controller.

DMA buffer

There was one DMA buffer in each direction. The DMA buffers were in the form of FIFOs (the first data written in will be the first data read out). Each was 32 bits wide and 64 words depth. They acted like an intermediate storage, to buffer the data transferred from memory to be packetised for the transmit operation, and vice versa.

The DMA buffer at the transmitter channel stored the data fetched from the memory by the Transmitter controller. The data stored here was read by the packetiser to convert the 32 bit data into OS-link protocol tokens. The receiver channel's DMA would have the reverse function, to buffer the de-packetised data. The storage was accumulated so that there was sufficient data to make an efficient DMA transfer to the memory. The Receiver controller asserted a memory write request when the receiver channel's DMA buffer was almost full or when it was the end of message.

Packetiser/ Depacketiser

The Packetiser fetched the buffered data from the transmitter channel's DMA buffer. The 32 bit data was split and formatted into 9 bit tokens. After that, the tokens were buffered in the Token buffer. The Packetiser had three stages to packetise a message. The first stage was to identify all the message headers, format it and then store all the headers into the Token buffer; the second stage formated the remainder of the data into tokens and stored them in the Token Buffer; the final stage inserted a message termination token to the tail of the message. The Packetiser buffered the tokens into the Token Buffer as long as the 'Almost Full' flag was not set, otherwise it paused and waited.

The Depacketiser was responsible to de-format every four 9 bit data tokens from the receiver's Token Buffer and combined them into 32 bit data. First it identified the message's header ID then stored the message ID into the Received Header register in the DMA Register module. Next, every four data tokens were de-formatted (removing the type bit) then re-arranged in the order of the original 32 bit data before being buffered into the receiver channel's DMA Buffer.

Data was packetised and de-packetised in hardware in order to reduce software overheads. Therefore the software just had to input the address location of the first 32 bit data and input the length of the message.

Token Buffer

There were two token buffers in the network interface controller, one in each direction. The Token Buffers were 9 bits wide and 32 token in depth FIFO. The least significant bit of the token was the token type bit to indicate the token type (bit '1' for a data token, bit '0' for a message termination control token). Both functioned as temporary storage for the formatted token and received token from the network for the transmitter channel and the receiver channel, respectively.

The transmitter channel was the intermediate buffer for the packetised tokens prior to being transmitted out to the network. It had an 'Almost Full' flag to signal the Packtiser, so that it held further packetising operations until the buffer was cleared down to a pre-defined 'Almost Empty' level, to avoid FIFO overflow. When it reached the 'Almost Empty' level, the FIFO set the 'Almost Empty' flag to permit the Packetiser to resume the packetising operation.

The receiver channel's Token Buffer stored the valid incoming token, sampled by the Network Link Interface module, waiting to be de-packetised. It also used the 'Almost Full' flag and 'Almost Empty' flags to control the buffering space availability. When the buffer reached the pre-defined 'Almost Full' level, the 'Almost Full' flag was set. This signal caused the Network Link Interface to transmit an 'Xoff' token to the upstream node to pause the transmission until the buffer was cleared. The 'Xon' was transmitted to permit the upstream node to resume transmission when the buffer level dropped to a pre-defined 'Almost Empty' level ('Almost Empty' flag set).

Network Link interface

The Network Link Interface was the module that interfaced the processing node to OS-Link network. The Transmitter first fetched tokens buffered in the transmit channel's Token Buffer. The tokens were transmitted out in a serial bit stream, alongside the Start bit and Stop bit (refer to 2.3.3). In the case when the receiver channel's 'Almost Full' flag

was set, the Transmitter transmitted an 'Xoff' token so that the upstream node paused the transmission operation. It later transmitted an 'Xoff' token to permit the upstream node to continue to transmit the remaining tokens.

The Receiver sampled and validated the received signal. Then later stored the valid tokens from the network into the receiver channel's Token Buffer. Only data tokens and message termination tokens were stored into Token Buffer because the rest of the network control token were invisible control tokens, used between the Network Link Interface and OS-Link, for flow control. If an 'Xoff' token was received, the Network Link Interface stopped the Transmitter's operation until an 'Xon' is received.

4.2.2 XA1 External communication

The UART of each XA1 system was connected to the serial port of the host PC through a selector circuit. This enabled the host PC to monitor and access the mapped register or memory location of each XA1 system when tests were run on it. A boot program was developed by the research group[97] and run in the Excalibur device. It enabled the remote map registers or memory location access from the Host PC via a UART connection.

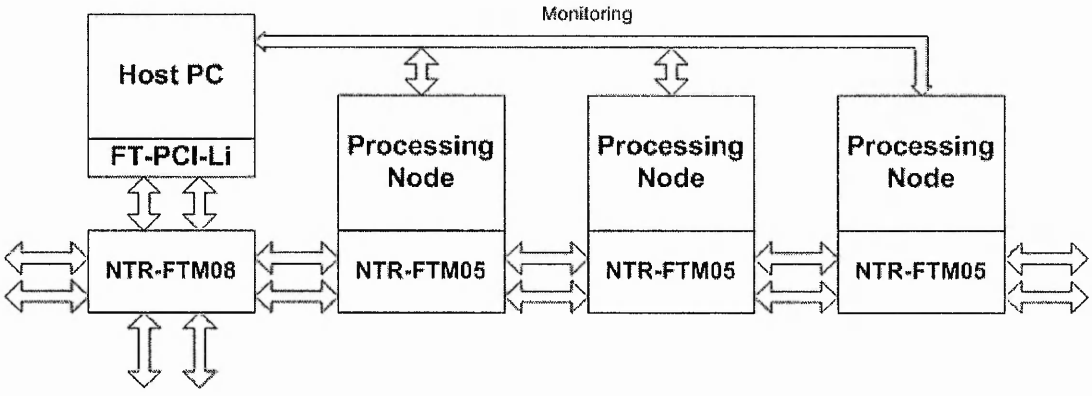


Figure 4-5: Multi processor embedded system.

Figure 4-5 is an example of topology of a distributed multiprocessor embedded system that could be formed by the XA1 prototype system, a daisy chain topology. In the example the host PC could act as the main control. The Host PC could be used to configure all the routers, boot up the all the processing nodes and dedicate a task to each processor. All the communications between the processors (including the host PC) are via the routers. Every router will have two channels connected to each neighbouring router[95].

4.2.3 Group Adaptive Routing in XA1 Prototype Board

The use of the NTR-FTM05 routing device has inherited the Group Adaptive Routing features from the NTR-FTM08 routing device. Therefore the 4 OS-Link channels, which were to be connected to other processing nodes, can be configured and grouped to form the target network topology.

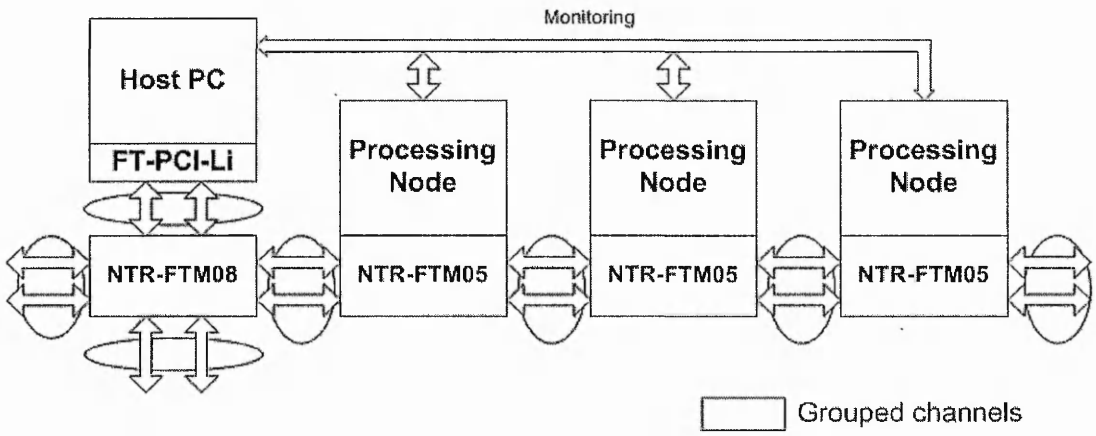


Figure 4-6: Grouped Adaptive Routing utilisation in XA1 system

An example of the utilisation of Group Adaptive Routing feature in the XA1 system is illustrated in Figure 4-6. The communication channels between two adjacent NTR-FTM05 routing devices were grouped together by configuring both routing devices. If two message were to be passed to the same destination NTR-FTM05 routing device but both had the same output port header information, one of the messages would be passed

via an alternative channel (if that channel was free), when the communication channels were grouped in advance.

4.3 The APEK20KC chip

The designs implemented in the APEK20KC were inherited from the FT-PCI-Li interface development board (see section 2.3.3.2). It contained a PCI based OS-Link Network interface controller (FT-PCI-Li), and an NTR-FTM08 router. The PCI interface, illustrated in Figure 4-1, could be plugged into a Host PC to include the Host PC to the embedded network.

The FT-PCI-Li had two bi-directional communication channels connected to the NTR-FTM08 router. This allowed multiple message passing using two bi-directional channels. Both communication channels could be grouped together by configuring the router to have more efficient use of the communication bandwidth between the FT-PCI-Li and the NTR-FTM08 router. The NTR-FTM08 router had 8 communication channels. Two of the communication channels were connected to the first XA1 processing node, another two could be either connected to the third XA1 or to an extended network (by changing the jumper setting on the XA1 prototype board), the remainder of the channels could be used to extend the embedded network.

The Host PC could be used to boot up all processing nodes in the network, to configure all the implemented routers (grouping) and pass messages to the processing nodes in the network via the FT-PCI-Li interface. During the investigation and design optimisation, the Host PC was used to transmit messages to the processing nodes to verify the functionality of designs.

4.4 Design Optimisation

The network interface controller was adapted and modified to reduce processor intervention when passing messages. The previous design required the processor to input the header information into the DMA register, prior to kick-starting the Transmitter DMA Controller. It needed three write operations to transmit a message: input the address, input the header and input the length of message. Therefore the header information was included in the message (in memory) and header information was identified when the message was packetised. This resulted in only two write operations, to input the address and message's length into the DMA register, to 'kick-start' a DMA when transmitting a message.

4.4.1 New message Structure

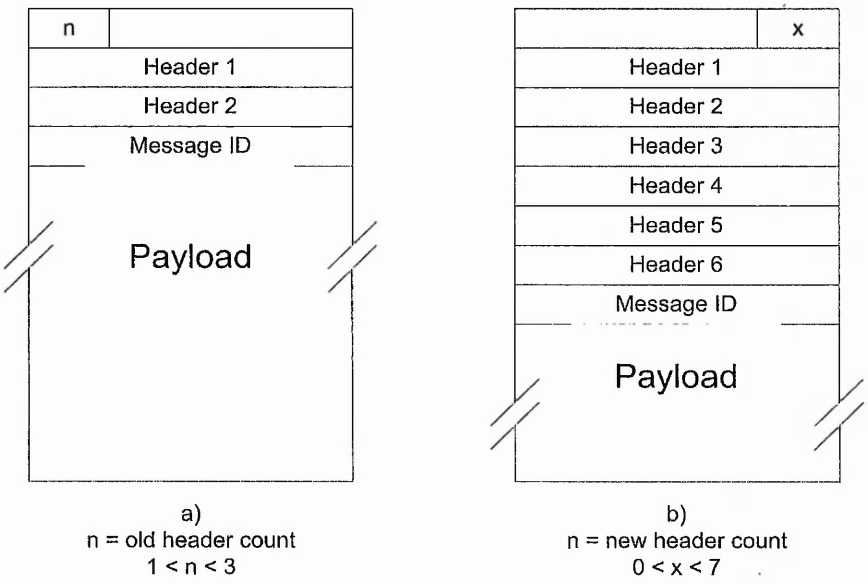


Figure 4-7: a) Old message structure. b) New message structure.

The optimisation resulted in a new message structure in memory. Each header byte informed the downstream router which output port the message was designated to and the

Message ID used by the processing node to identify the new incoming message. The original design supported up to three header bytes, including the Message ID byte, because the Header register in the DMA Register module was only 32 bit. Therefore the message was limited to pass through only 2 routers. By including the header information in the message in memory, it enabled the header information to have a higher capacity, to support up to seven header bytes including the Message ID.

Referring to Figure 4-7, the least significant 3 bits of the first most significant bytes of the message structure was the header count. It indicated the number of header bytes in the header section of the message, including the Message ID byte when the message was being packetised. The headers were packetised in the sequence from the most significant byte to the least significant byte. Higher capacity in the header section enabled the message to travel via more routers, thus increased the network's scalability. This also offered an option of using a Zero Header, which included all the headers into the message payload to support more than 7 headers. If a zero header count was used, then the header information must be re-arranged because the payload was packetised from least significant byte to most significant byte for every 32 bit of data.

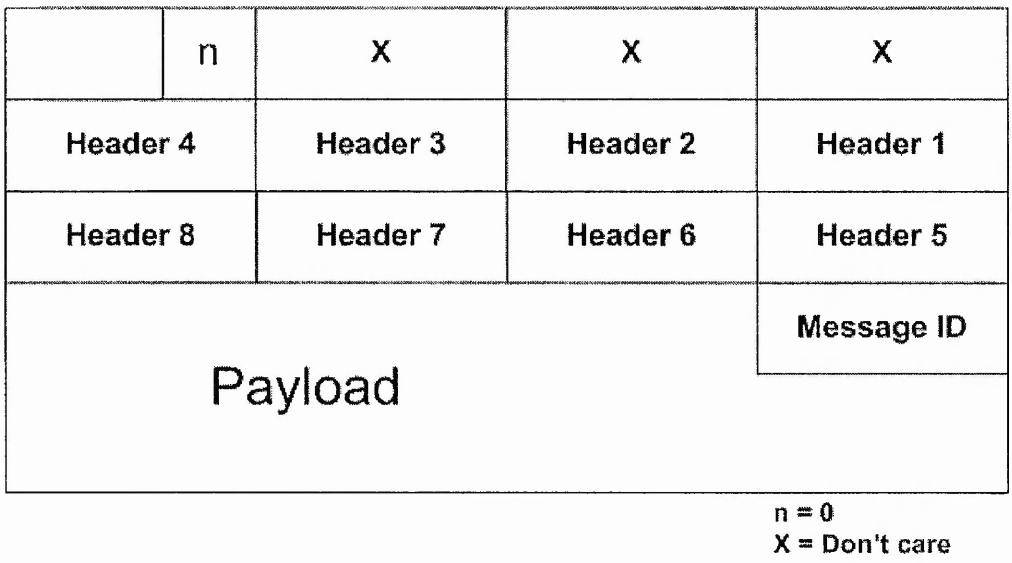


Figure 4-8: Message Structure in Memory when Zero Header was used.

4.5 Technology Migration

At the end of the design optimisation, the research group decided to migrate to a new technology because improved SoC options were available.

The initial XA1 prototype board consisted of three Excalibur XA1 chips and each had an in-built 'hard' core ARM922T processor. Although the ARM922T could operate at high clock frequencies, up to 200 MHz, comparing to the NIOS II and MicroBlaze softcore processors, but because it is an 'in-built' processor, it is lack of reusability. It cannot be replicated in the same chip to form multiprocessor SoC. The NIOS II and the MicroBlaze softcore processor had the potential of implementing multiple processors in a single programmable chip, as softcore processors were provided in the form of IP, therefore they could be replicated as many time as possible within the available programmable elements in the FPGA device.

In term of flexibility, the configurability of a softcore processor allows designers to include only the optional features that are required for their design. Hence, the programmable elements in the FPGA would not be wasted and more functions or IP could be implemented. Designers could also construct different kinds of systems to suit their application. Features of the 'Embedded Stripe' in Excalibur devices were in-built: silicon area has been consumed whether the application required the usage of the in-built features or not.

Most of the components were custom built or provided by the FPGA vendor in the form of soft IP (VHDL or Verilog HDL code). Therefore configurations of components could be changed to suit the target application. Most importantly the most updated component designs can be obtained from the FPGA vendor.

The capacity of the FPGA's programmable elements was also a crucial decision factor when migrating to the new technology. Comparing the specification of FPGA devices of Excalibur XA1, Stratix II devices[47], and Virtex-4 family (in Appendix B) the Excalibur devices had the lowest LE count. Therefore the Stratix II device shows a higher capability of including multiple processing nodes.

Using the Excalibur devices to construct an embedded distributed multiprocessor system on a PCB involved more hardware design and tracking. By using a higher programmable element FPGA to implement multiple processing nodes, the risk of hardware tracking errors, which could result in investment in new PCB designs, could be reduced.

The NIOS II processor (with Stratix II device) and the MicroBlaze (with Virtex device) showed more potential of improvement in OS-Link network implementation when comparing the FPGA's specification and their capacity with the Excalibur devices. After taking consideration, the research group reached a decision to migrate the designs to a Stratix II device. This was partly due to the reason that the research group had many years of experience with Altera devices and partly due to the flexibility of NIOS II processor.

5 Design Structure for NIOS Based SoC

5.1 OS-Link Network Interface Module

The OS-Link network interface controller (NIOSNIC) was built as the communication medium between the processing node and the interconnected OS-Link-based network. The design was divided into two sections, as shown in Figure 5-1. The main contribution was in the front end section. The front-end section was the custom build Avalon bus interface for NIOSNIC, which was designed to interconnect the system and the back-end section of NIOSNIC. Both of the DMA Buffers and Control/Status register module were also modified to facilitate the Avalon Bus Interface built. This front-end was responsible for initiating the Direct Memory Access transfer to move the received data to the designated memory location. The back-end section was the Message Packetiser/ Depacketiser and the network link interface, to transmit and receive messages to and from the interconnected OS-Link-based network, and to encode and decode the control tokens.

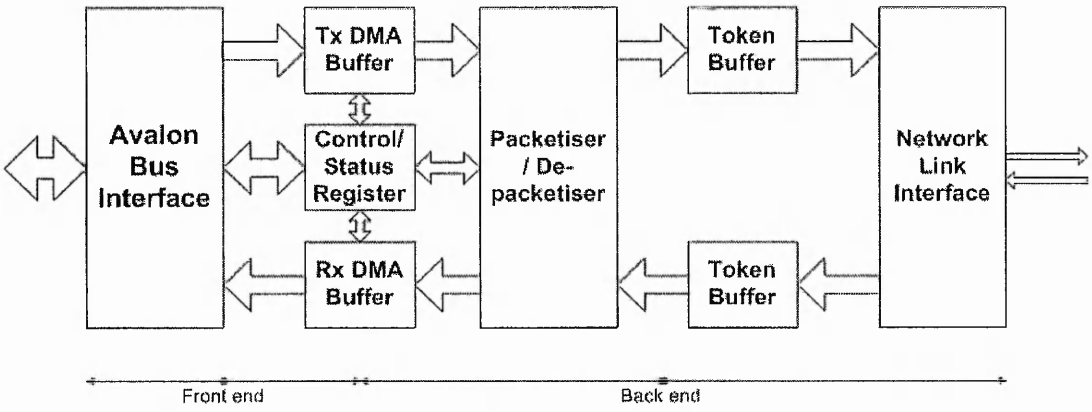


Figure 5-1: Basic construction of OS-Link Network Interface controller block diagram.

From Figure 5-1, the front-end section consists of an Avalon bus interface module and it was separated from the back end section by two FIFO buffers, one for the transmit channel and one for the receive channel. The entire module was synchronised to the

system clock except the Network Link Interface which was synchronised by the Over Sampling (OS) clock. The token buffers facilitated the data flow between asynchronous serial link of the OS-Link network and the Packetiser/ Depacketiser. The developed OS-Link network interface controller was adapted from the optimised AMBA bus-based design, with modules redesigned to interface to the Avalon Bus in the new distributed embedded multiprocessors system.

5.1.1 Avalon Bus Interface Module

The Avalon Bus interface module in the OS-Link network interface controller was designed to initialise and to kick-start the data transfer between the OS-Link network interface controller and the memory module in a processing node. Besides that, it also contained registers to indicate the OS-Link network interface controller's status. The Avalon Bus interface was divided into three main modules, the Transmitter module, the Receiver module and the OS-Link Register module.

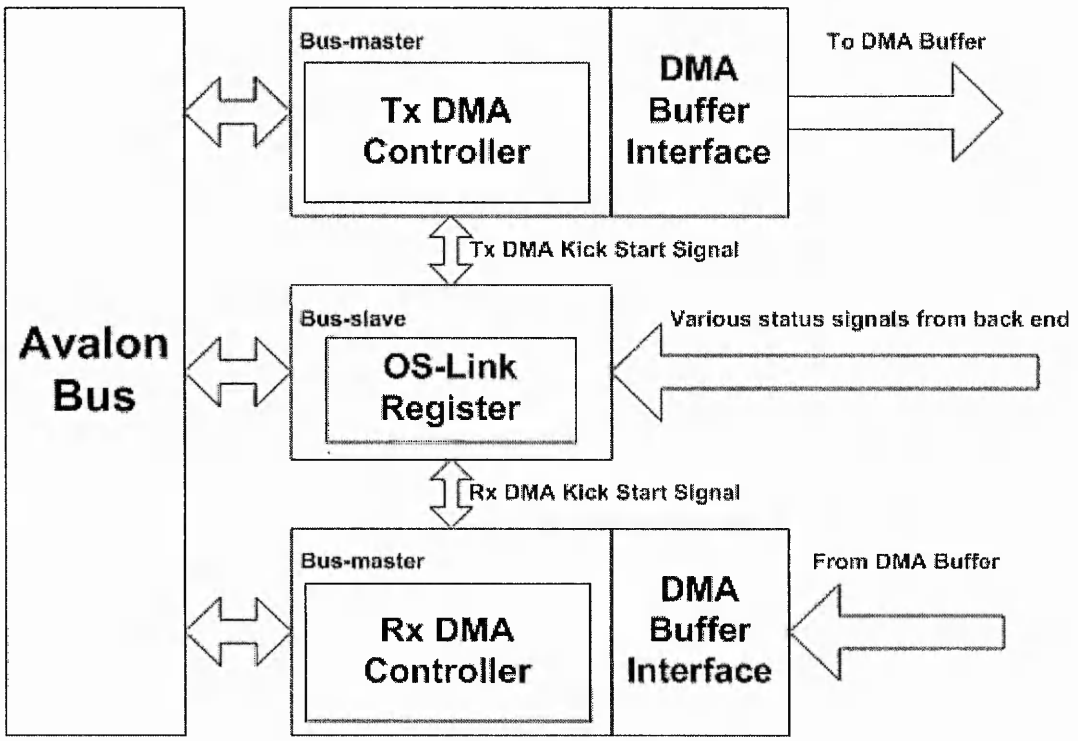


Figure 5-2: Block Diagram for the Avalon Bus interface section and its associated signals.

Referring to Figure 5-2, both the Transmitter and Receiver modules were designed as Avalon bus masters while the OS-Link Register module was designed as the Avalon Slave device in a processing node. The Transmitter bus master (Tx master) was responsible to initialise DMA operation to transfer the message from a memory module then store the message in the 32 bit transmitter channel DMA buffer, waiting for packetisation before being transmitted out from the NIOSNIC. The Receiver bus master (Rx Master) was responsible for transfer of the de-packetised message from the 32 bit receiver channel's DMA buffer to a memory module. The OS-Link Register was designed to store and indicate the current status of the OS-Link network interface controller. The OS-Link Register was also used to configure both Transmitter and Receiver bus masters to kick-start a DMA transfer.

The Avalon bus interface modules were designed as separate bus masters so that both bus masters can operate simultaneously. The Avalon Bus allows multiple bus masters to drive the system bus, providing that the bus masters were not accessing the same slave device[93]. For example, when the processing node was transmitting a message, data was transferred from SDRAM via the Transmitter module to the transmit channel DMA buffer. During this time, if a new message was received and the destination address where the received data was going to be stored at another memory module on that processing node, say the on-chip RAM, then both transmit and receive operations would operate simultaneously. In the worst case scenario when both bus masters were accessing the same memory module, then the decision would be made by the memory module's arbitration, to choose which of the bus masters would gain access.

5.1.1.1 The Bus Masters and Bus Slave Modules

Both Transmitter and Receiver bus master modules initialised and monitored the DMA operation. They were also connected directly to their target memory arbitration

module by the Avalon Bus. Each bus master module, shown in Figure 5-2, consisted of an address generator and DMA size counter. The address generator and size counter were controlled by a state-machine.

Avalon Bus Ownership request

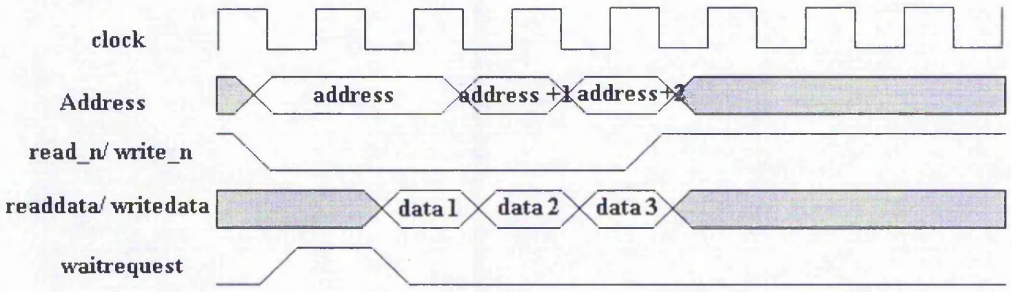


Figure 5-3: Timing diagram for bus access or memory read/ write operation.

To access a slave device, such as a memory module, the target memory address was asserted to the address bus as the same time as asserting the active low read request (signal 'read_n') or active low write request (signal 'write_n') for memory read and memory write operations respectively.

The Avalon Bus used slave-side arbitration where the access arbitration was handled by the slave device's arbitrator (see section 3.3.2). The way to determine that the bus master had successfully gained the bus ownership was by sampling the active high 'waitrequest' signal. For example, referring to Figure 5-3, if the 'waitrequest' was set high at the following rising clock after the read or write signal was asserted, then the read or write operation had to be stalled until the 'waitrequest' signal was de-asserted. When the 'waitrequest' signal was de-asserted, the data transaction could take place at the following rising clock edge.

Bus master controller

The bus master operation was controlled by a state-machine. The state-machine was responsible for the start and halt of data transaction operation (DMA operation), address generation and DMA size counting.

Transmitter Bus master

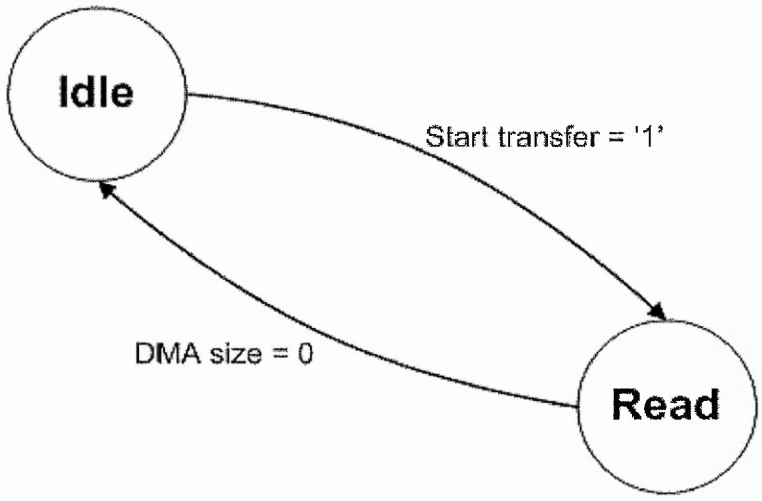


Figure 5-4: Transmitter bus master state-machine operation.

Figure 5-4 is a diagram that shows the state-machine of the Transmitter Bus master controller. Upon reset, the state-machine would be in the Idle State. When the Address and DMA size was loaded to kick start the DMA operation, the state-machine would enter the Read State. At this state, the active low read signal (read_n) would be set LOW to request for memory access permission. For each rising clock edge, if the 'waitrequest' was de-asserted, the data present at the data bus would be written to the 32 bit DMA buffer, the address generator would increase the address by one and the DMA size counter would decrease by one. If the 'waitrequest' was asserted then all operations would be stalled until it was de-asserted. The read operation would be repeated until the DMA size decreased to zero and the state-machine would return to the Idle State, waiting for next memory read operation. However, during the DMA read operation, if the

Transmitter DMA buffer was full before the DMA size decreased to zero, the state-machine would return to the Idle State to wait for the DMA buffer to be cleared to 'Almost Empty' level before resuming the DMA read operation.

Receiver Bus master

Part of the structure of the Receiver Bus master module was similar to the Transmitter Bus master module. It also had a state-machine to control and monitor the data transaction between the DMA buffer and target memory location via DMA write operation. Its operation is illustrated in Figure 5-5.

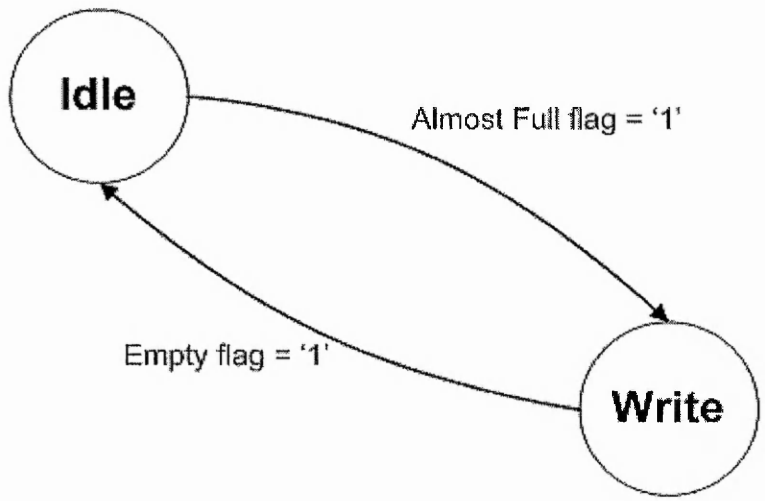


Figure 5-5: Receiver Bus master's state-machine.

Upon reset, it entered the Idle state and waited. When a message was received and the Receiver DMA buffer was filled to a level when it is almost full, the state-machine was notified and the state-machine entered the Write State. Therefore the DMA write operation was kick-started to transfer all the data from the Receiver DMA buffer to a designated memory location. During the data transaction operation, the transaction was only paused when the 'waitrequest' signal was set HIGH. At this stage, the 'write_n' signal remained LOW but the module stopped reading the Receiver DMA buffer until the 'waitrequest' signal was de-asserted. When the Receiver DMA buffer was empty, the

state-machine would return to the 'Idle' state from the Write State and wait for the next DMA write operation.

OS-Link Register Slave module

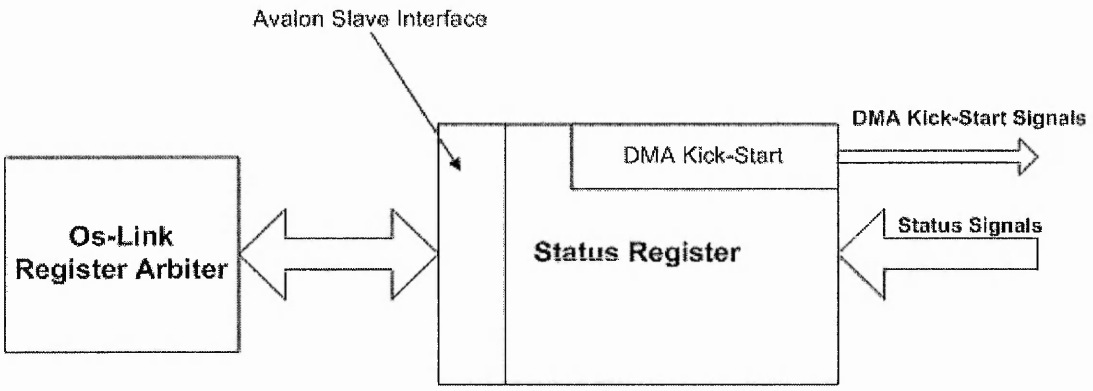


Figure 5-6: Block diagram for OS-Link network interface's Avalon Bus slave module.

Figure 5-6 illustrates the block diagram structure of the OS-Link Register module. It consisted of 3 main modules: the Avalon Slave Bus interface, DMA Operation Initialization Logic and the Status Registers. The processor could read or write data to the relevant registers. In the Avalon Specification [93], individual peripherals did not need to decode the address lines to generate chip-select signals because they were generated by the address decoding logic from the Avalon Bus. However the offset of the address, where registers would be accessed by the system, had to be decoded.

The Register module was the module where all the necessary registers, such as control registers and status registers, in the OS-Link Network Interface device were placed. There were twelve 32 bit registers contained in the OS-Link Register module, as listed in Appendix C. The registers were used to assist the development, testing and monitoring of the network interface.

The DMA Operation Initialization Logic was used to initialise and kick start the DMA read and write operations of the Transmitter Bus master and Receiver Bus master

respectively. To begin a DMA data transfer, either DMA read or DMA write, the address or the target memory location must be loaded into the address register, and then followed by the size of the message to be loaded into the message size register. When the size register was being loaded, a pulse would be sent to the OS-Link bus master to kick start the DMA operation and to get the back end OS-Link module ready.

DMA Buffers

From Figure 5-1, there are two DMA buffer modules, one buffer for each direction. The buffers operated based on FIFO principles. The Transmitter DMA buffer was the temporary storage for data obtained from the system's memory via the Transmitter Bus master before packetisation. The FIFO was allowed to store one 32bit data per clock cycle when the request to write to the FIFO was asserted.

The Receiver DMA buffer also operated based on FIFO principles. It provided the temporary storage for the depacketised 32 bit data from the Receiver Message De-Packetiser. The data stored here would be waiting for the DMA operation to be transferred to the system's memory via the Receiver Bus master. The storage capacity needed to be sufficient to provide a smooth data flow between the Receiver Bus master and the Receiver's Message DePacketiser, so that there was no need to stall read/write operations because the buffer was either empty or full. The FIFO was able to unload one 32bit data word per clock cycle to the data bus, provided the target memory module was ready to receive data with no access latency.

The Transmitter DMA buffer was a synchronous Legacy FIFO while the Receiver DMA buffer was a synchronous Show-ahead FIFO. For a synchronous Legacy FIFO, the data would only be present on the data bus at the next clock cycle after a read request was asserted; for the synchronous Show-ahead FIFO, the first data would be presented on the data bus immediately after the data was written into it. The read request was asserted for the next data to be presented on the data bus at the next clock cycle. The simplified timing diagrams to explain the operation of synchronous Legacy FIFO and Show-ahead

FIFO are shown in Figure 5-7 below. The use of Show-ahead FIFO as a Receiver DMA buffer was to optimise the DMA transfer in Avalon Bus, because the address, data, and write requests were asserted together in the address bus, data bus and write request signals respectively. If Legacy FIFO was used, then one clock cycle delay needed to be added in order to read a data from the buffer, for each DMA write operation.

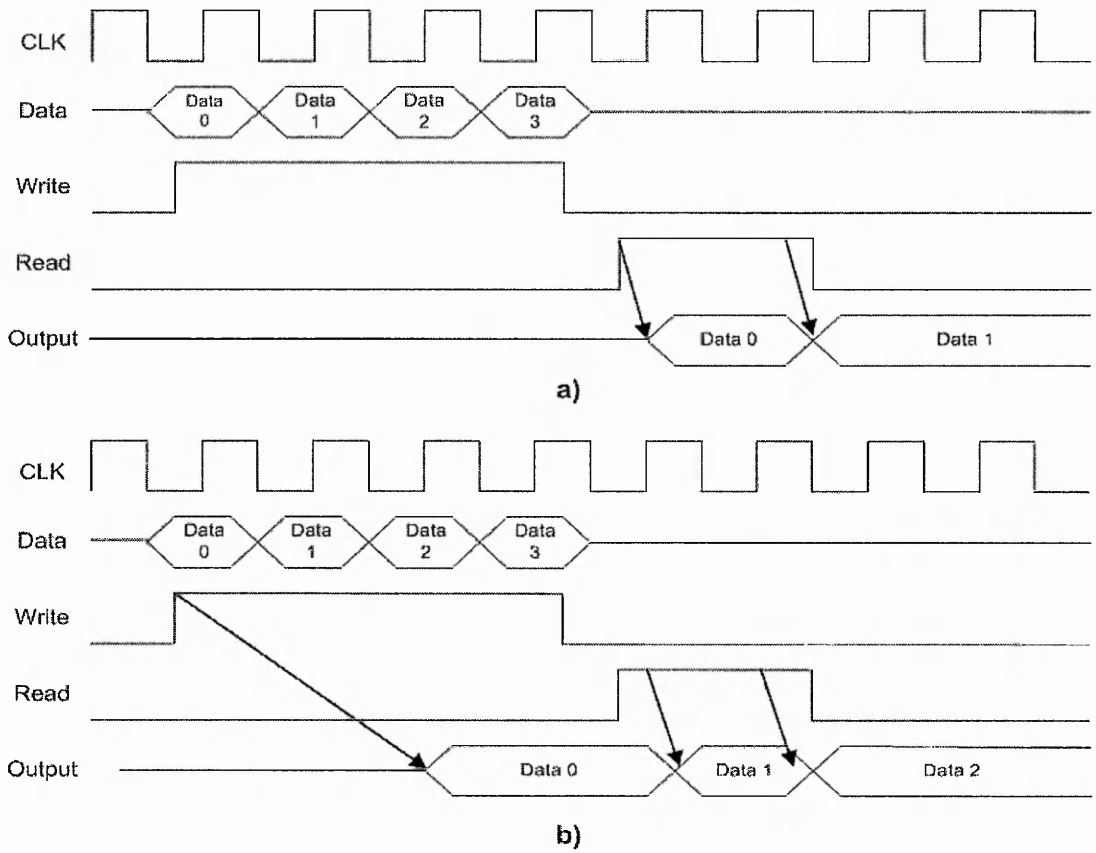


Figure 5-7: Timing diagram for a) synchronous Legacy FIFO and b) synchronous Show-ahead FIFO.

5.1.2 Back-end Module

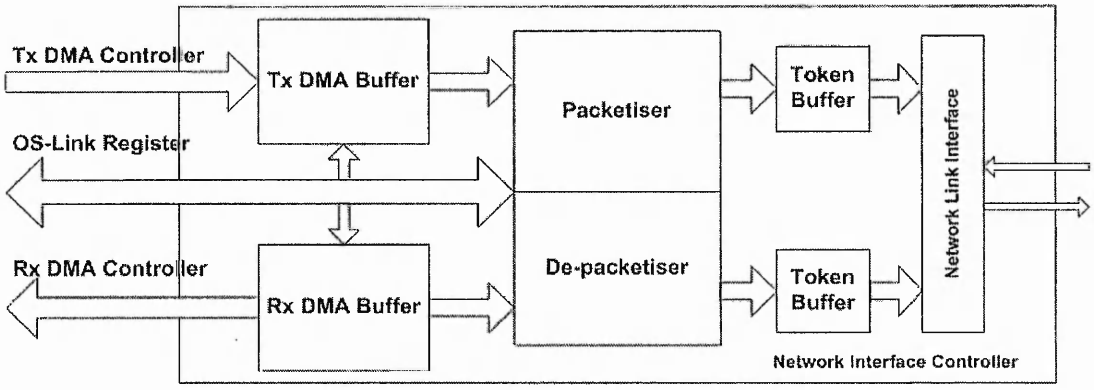


Figure 5-8: Block diagram for the back-end modules of the OS-Link Network Interface Controller.

Figure 5-8 is a block diagram that shows the important modules in the back-end section. They are the Message Packetiser, Depacketiser, Token Buffers and Network Link Interface. The functionality of these modules will be discussed next.

Transmitter Message Packetiser

The Transmitter Message Packetiser was to read data from the DMA buffer, packetise it, and then write it to the Transmitter Token buffer. The message packetisation was done in hardware in order to reduce software overhead. Each 32 bit data from DMA buffer was split into four 8 bits sections. A token type bit of logic '1' was added to each data section to form a 9 bit data token before it was transferred to the Token Buffer. The optimised OS-Link Network interface device supported up to seven header tokens. The header tokens included up to six routing information bytes including a message identity byte. The Header Token Number, which was located at the most significant byte of the first 32 bits header word, was to count the number of routing information bytes. However, if the network was larger and required more than 6 routing headers, the user can use the Zero Header mode. Each header byte also had a data type bit of logic '1'. The arrangements of headers and data tokens are illustrated in Figure 5-9.

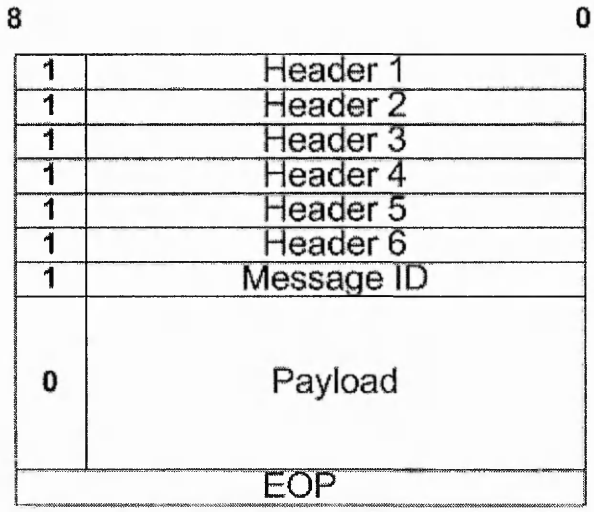


Figure 5-9: Header and data token arrangement.

The header routing bytes would be stripped one after the other, according to their position in the message stream, as the message traverses the OS-Link-based network. The obligatory message identity byte would be received by the destination processing node. The packetising process would end when Length Register in the DMA module was decreased to zero and the DMA buffer was emptied. At the end of message, a control token, End Of Message (EOM), with data type bit logic '0', would be appended.

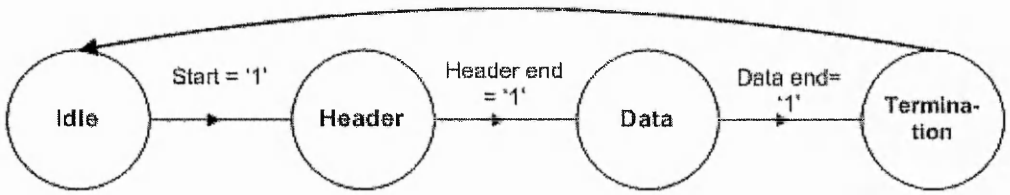


Figure 5-10: Message state-machine.

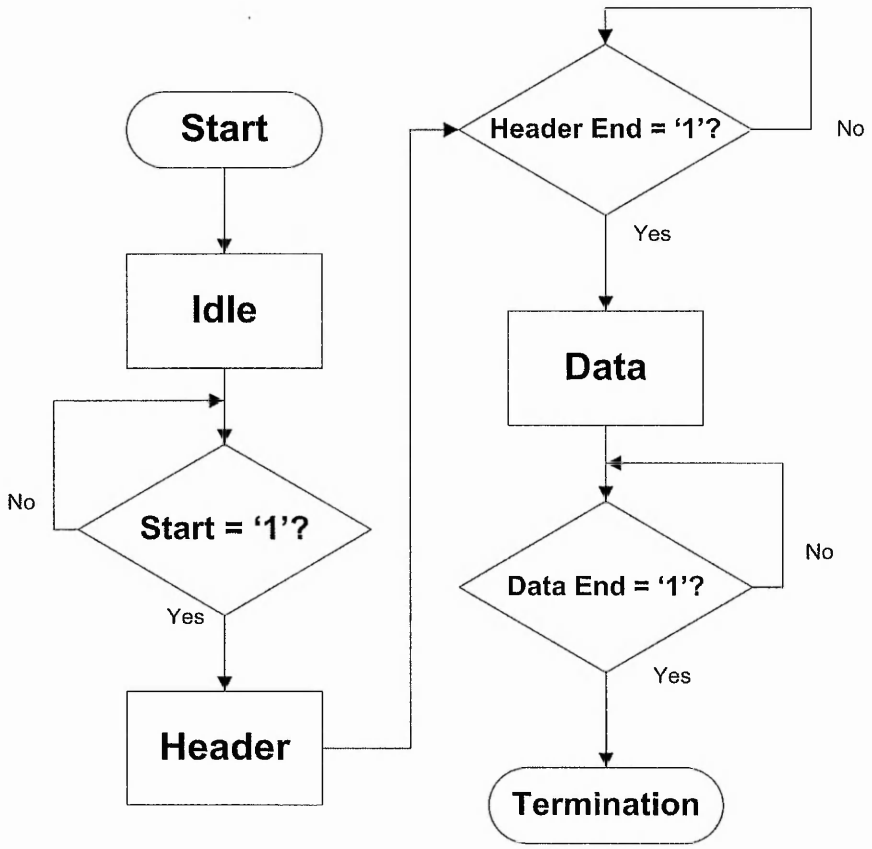


Figure 5-11: Message state-machine flow chart.

The Transmitter Message Packetiser contained two state-machines. The Message state-machine was to control and monitor the data packetising process. The Message state-machine had four states; 'Idle', 'Header', 'Data', and the 'Termination', as shown in Figure 5-10. Each stage reflected the type of data being processed and transferred. Referring to Figure 5-11, a message was started in the 'Idle' state when the processing node was booted or reset. It would enter the 'Header' state when the start message command pulse was received from the OS-Link Register module's control logic. This command pulse occurred when the message's length was loaded into the message size register in OS-Link Register module. At this stage, the header information would be loaded into the Token Buffer with the type bit. When all the header information was loaded, the state-machine would enter the 'Data' state. However if the Zero Header mode was used, the Header Token Number was made 'Zero'. This means there was no header

information as all the header information was included into the payload, no header count was necessary and forced the state-machine to move into the 'Data' state immediately. In the 'Data' state, the payload of the message was loaded into the Token Buffer. After the data packetisation process was finished, the state-machine would move to the 'Termination' state before returning to the 'Idle' state. At 'Termination' state, the EOM token was loaded into the Token Buffer. After returning to the 'Idle' state, the state-machine would wait for the next message transmits to begin.

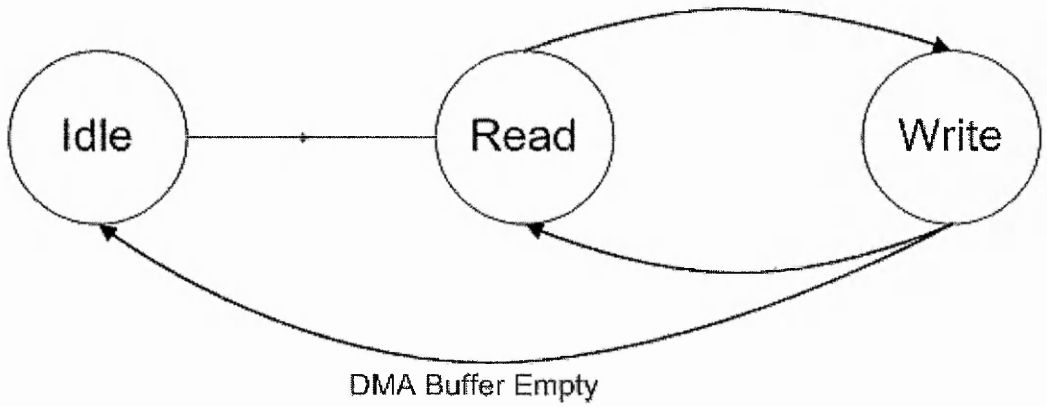


Figure 5-12: Buffer Read-Write controller State-machine

The second state-machine was the Buffer Read Write controller State-machine. As illustrated in Figure 5-12, it had three states; 'Idle', 'Read', and 'Write'. It would start operating, moving from 'Idle' to 'Read' when the Message state-machine entered the 'Header Stage'. When in the 'Read' state, it might have remained in the current state or advanced to one of the other two states, depending on the condition status:

- Condition 1: If a 32bit data was successfully read from the DMA buffer, it would advance to the 'Write' state.
- Condition 2: If the DMA buffer was empty and the message transmission is not ended yet, it would remain in the 'Read' state since there was no data to be acquired from the DMA buffer.

- Condition 3: If the DMA buffer was empty and the message transmission is ended, then it advanced to the 'Idle' state.

When in the 'Write' state, the obtained 32bit data from DMA buffer would be split into four 8 bit sections. Each data section would be formatted into a 9 bit data token by appending a logic bit '1' as a type bit before being written into the token buffer. After writing the data token into Token Buffer, the state-machine would return to the 'Read' state immediately. The state-machine would go back and forward between the 'Read' and 'Write' states until the all the data in the DMA buffer was emptied, it would return to the 'Idle' state waiting for the next transaction.

Receiver Message Depacketiser.

The receiver message Depacketiser transferred and depacketised the received data tokens from the Receiver Token Buffer (to the Receiver DMA buffer). The 9 bit data tokens were converted back to 8 bit data sections, by removing the data type bit. Every four data sections were combined to form a 32 bit data word, before it was written to the Receiver DMA buffer. Upon receiving a message before the depacketising process, the type bit of each token was checked. Before the received tokens were stored into the Receiver Token Buffer, all the control tokens have been removed, except the EOP, EOM and BEOP tokens.

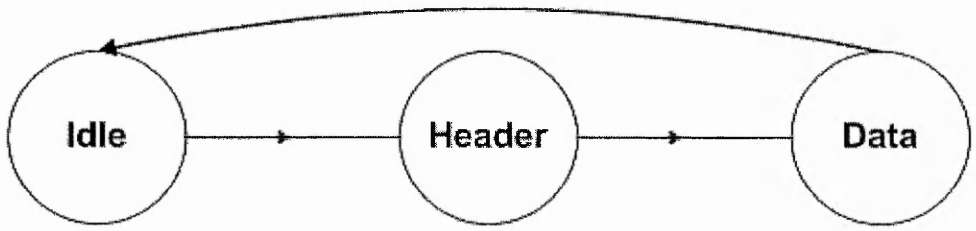


Figure 5-13: Depacketiser's Message state-machine.

The whole de-packetising process was also controlled and monitored by two state-machines; the Message state-machine and the Read-Write state-machine. The Message

state-machine had three states; the 'Idle' state, the 'Header' state and the 'Data' state. The states of the Message state-machine are shown in Figure 5-13. Upon reset, the Message state would be in the 'Idle' state waiting for a kick start. From the 'Idle' state, it would move to the 'Header' state, as the address of the target memory location and then the length of the message has been loaded ('Rx_pkt_strt' pulse received). In the 'Header' state, as soon as the Message ID was written into the Receiver Token buffer, the data would be de-packetised and then written into the Header register in OS-Link Register module, indicating the new received header from the expected message. After that, the state-machine entered the 'Data' state. In the 'Data' state, the data tokens were extracted from the Receiver Token buffer and transferred to the Receiver DMA buffer.

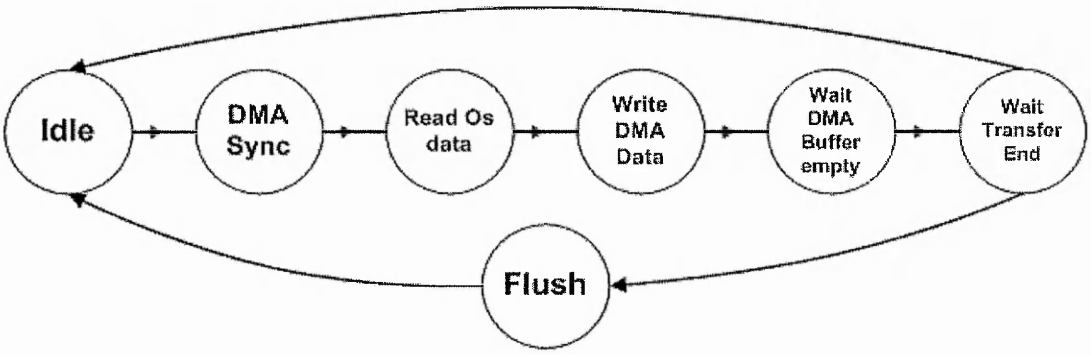


Figure 5-14: State diagram for De-packetiser's Read-Write State-machine.

The second state-machine was responsible for the Read-Write operation, as shown in Figure 5-14. Starting from reset, it entered the 'Idle' state. As the 'Rx_pkt_strt' pulse was received, it would move to the 'DMA Sync Stage' and wait until the Message state-machine moved from the 'Header' state into the 'Data' state. When the Message state was in the 'Data' state, it would advance into 'Read OS Data' to read from the Token Buffer while extracting the type bit from the data token. Each time a data token was read, the Message Size register would be decremented by one. The Read-Write State-machine would have read the Token Buffer four times to obtain a four 8 bit data bytes, after the type bit was extracted, and combined them to form a 32 bit data word.

After four reads, the Read-Write State would enter the 'Write DMA Data' state to transfer the 32bit data word into the Receiver DMA buffer. The four reads and one write procedure would repeat until the whole message was received (when the value in the message size register was decrement to zero), or the receiving operation was terminated due to the unexpected control token; EOP, EOM or BEOP. If the message was fully received or the control token was received before the Message Size register decreased to zero (early termination) the Read-Write State-machine would proceed to 'Wait DMA Buffer Empty' state. In the case of early termination, the Early Termination Flag would be set in the status register. As soon as the DMA buffer was emptied by the Receiver Bus master, the state-machine would move to the 'Wait Transfer End' state to confirm that the Receiver DMA buffer was empty and the control token was received. If both conditions were fulfilled, the state-machine would return to the 'Idle' state. However, at this point if the control token was still not received but the Message Size Register had already reduced to zero, a Late Termination Flag would be set in the status register, and the state-machine will proceed to the 'Flush' state where the unexpected extra data token would be flushed until a control token was received, before returning to the 'Idle' state.

Token Buffers

The Token Buffers were the temporary storage for the tokens waiting to be transmitted or de-packetised for the transmitter channel and receiver channel respectively. It was a FIFO of 9 bits wide and 32 tokens depth. The type bit to indicate the token type (data token or message termination control token) was located at the least significant bit of each token.

For the transmitter channel, the data read and data write operations were synchronised to the OS clock and sample clock respectively. The transmitter channel's Token Buffer stored the packetised tokens from the Packetiser. The transmitter channel's Token Buffer had a 'Full' flag to prevent data storage saturation (to pause the data write operation from the Packetise) and an 'Almost Empty' flag to signal the Packetiser to resume packetising operation, to prevent data starvation.

The receiver channel's Token Buffer worked initially in the reverse direction. The receiver channel's Token Buffer was used to store the received token from the network, via the Network Link Interface. Instead of using a 'Full' flag and an 'Empty' flag, the receiver Token Buffer used 'Almost Full' and 'Almost Empty' flags for data flow control. The 'Almost Full' flag would set if the stored token was more than four tokens below the full value. The 'Almost Full' flag would signal the Network Link Interface to transmit an 'Xoff' to stop the upstream node from transmitting any more data tokens until an 'Xon' was sent. For Almost Empty flag, it would set when the Token Buffer storage level dropped to four or less tokens before empty, so that the Network Link Interface could sent an 'Xon' to permit the upstream node to continue sending data tokens.

Network Link Interface

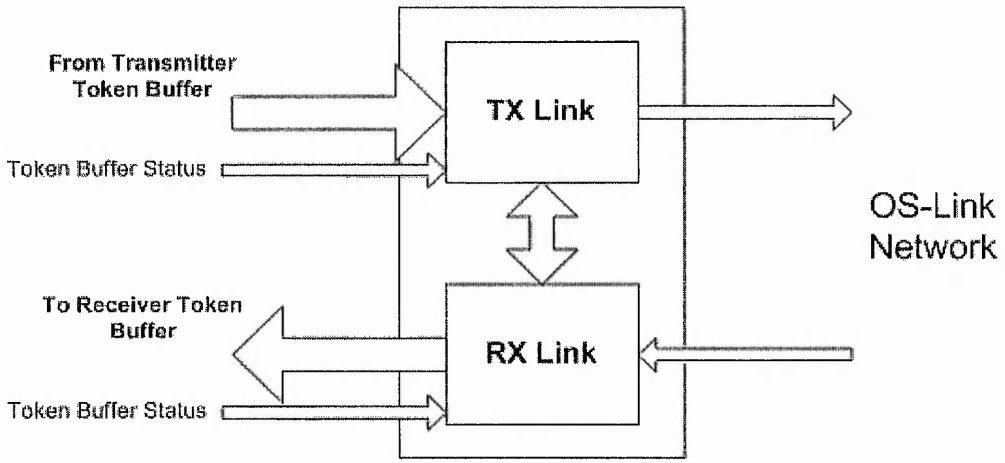


Figure 5-15: Block diagram for Network Link Interface.

Figure 5-15 shows the different modules in the Network Link Interface. It consisted of a TX Link module and an RX Link module. The 9 bit tokens were read from the Token Buffer and converted into a serial bit stream when transmitting a message to the network and vice versa. The Start bit and stop bit were attached at the beginning and ending of each token respectively before being transmitted out of the processing node, and both would be removed immediately upon receiving each token.

The Network Link Interface also monitored the data flow and the link connection status. Control tokens, such as 'Xoff' and 'Xon' tokens, were inserted into the bit stream between the data tokens or removed when necessary. The 'Xoff' token was transmitted when the receiver channel's Token Buffer reached the 'Almost Full' level, to signal the upstream node to hold the transmission process; 'Xon' token were sent to give permission to the upstream node to resume the unfinished data transmission. Among the tokens used in the OS-Link network, only the termination token was allowed to progress into the token buffer to end the de-packetisation process of that message, other tokens would be removed at the RX Link module.

The link connection status was monitored to maintain and verify the connectivity between upstream and downstream nodes. A 'Heartbeat' was asserted every 128 OS clock cycles and the upstream node would require the transmission of flow control token. When the communication link was idle, i.e. no data transmission took place, a flow control token was sent to the receiver to verify that the link is still connected and operational. Link verification was made periodic to reduce the signal activity and power consumption. When a flow control token was received, a flag named 'got_token' was set. The 'Heartbeat' assertion was to send flow control tokens to verify link status. The 'Checkpulse' procedure was to check and clear the 'got_token' flag every 512 OS clock cycles. If the 'got_token' flag was not set when the 'Checkpulse' was asserted, a link disconnection was detected (because flow control tokens should have been received before the 'Checkpulse' was asserted). This not only allowed link status to be updated regularly, but also to avoid data loss due to unknown link disconnection. Triggering the link disconnection flag would disable the data transmission until the communication link was correctly re-initialised.

Link dormancy was one of the configurable features implemented in the adapted OS-Link protocol, working alongside the NTR-FTM08 router. It allowed a link to go into a 'sleeping' mode after a pre-defined period of link inactivity. No flow control tokens were required and the links remained silent when the link was in sleeping mode. In the event of transmitting a message when the link was in sleeping mode, the link was reset and awoken through a handshaking process (sending control tokens) between the upstream and downstream nodes. Power consumption was reduced when the link was in sleeping mode as no flow control tokens were required to be transmitted.

5.2 The Stratix II Subsystem (ST2SS) Module Description

This section describes the overview of the subsystem (processing node) of the Distributed Embedded Multiprocessor System built on the Stratix II development board.

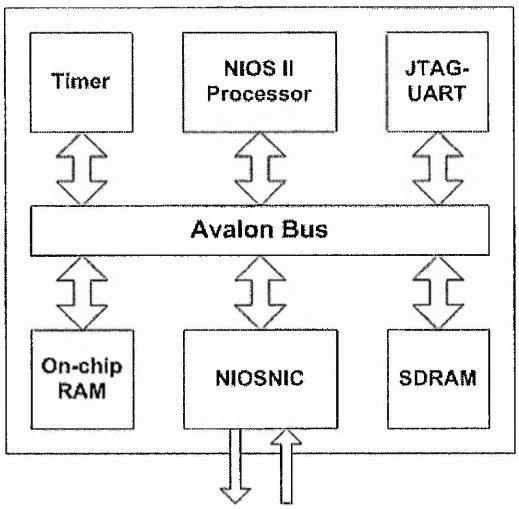


Figure 5-16: Block diagram of a subsystem design.

The subsystem module is illustrated in Figure 5-16 and consists of six main components: the NIOS II processor, a Timer, on-chip RAM, SDRAM, a Joint Test Action Group-Universal Asynchronies Receiver and Transmitter (JTAG-UART) and an OS-Link Network Interface module.

The ST2SS was adapted from the XA1 ARM-based platform design, with a novel OS-Link Network Interface module design; a new system bus interface with an improved DMA controller was added.

5.2.1 NIOS II processor

The NIOS II processor was the main processing unit in the ST2SS. This was the component where calculation or read/write operation control took place. Task and application program codes were executed there. It fetched instructions from the memory to be executed and returned the results.

5.2.2 Avalon Bus

The Avalon Bus was the communication medium between all the Bus masters and Bus slaves. There were 3 bus masters (the NIOS II processor, and the Transmitter bus master and the Receiver bus master from the OS-Link Network Interface module) and 5 bus slaves (Timer, on-chip RAM, SDRAM, JTAG-UART and OS-Link Register). The SDRAM controller allowed addressing of up to 16 Mbytes of SDRAM (pre-fixed on the development board). The implemented on-chip RAM allowed up to 50 kBytes of addressable memory. The OS-Link Register consisted of 128 bytes of the OS-Link Network Interface module configurable registers.

5.2.2.1 Avalon Bus Arbitrator

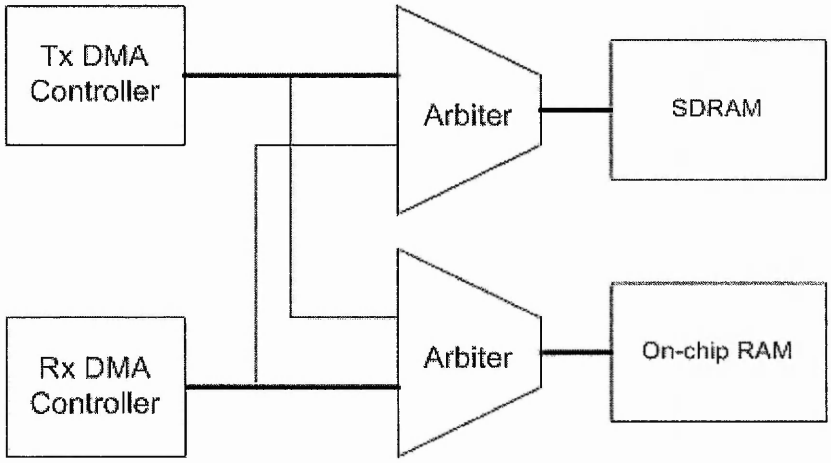


Figure 5-17: Slave-side arbitration.

The Avalon bus was a slave-side arbitration bus architecture, so each bus slave device was only associated with its own arbitrator. All the bus masters that had access to that particular bus slave would have to access requests via the slave arbitrator. Figure 5-17 illustrates an example of the Slave-side arbitration strategy arrangement.

The Slave Arbitrator received slave device access requests from interconnected bus masters. It decided which bus master gained the access permission, based on its hidden arbitration scheme, when there were multiple bus masters access requests. The architecture of the Avalon Bus also enabled multiple bus masters to access the system bus, providing the target device that each bus master accesses is different. For example, if the Transmitter bus master was accessing the SDRAM to fetch a message and meanwhile the Receiver Bus master kick-started a DMA transfer to the on-chip RAM to store the received message, both processes could proceed simultaneously because both were accessing different slave memory module. This was an advantage offered by the slave-side bus architecture compared to a single master driven bus architecture.

The memory location was decoded by the arbitrator and thus the chip select was generated by the target device's arbitrator. The target device was activated to commence bus transaction after its latency had elapsed. The arbitrator asserted the wait request signal to the bus master until the slave device was ready to commence the data transaction.

5.2.3 SDRAM controller interface and On-Chip memory

In the development board, the SDRAM controller was to interact with the off-chip pre-connected 16 Mbytes SDRAM.

5.2.4 On-chip RAM

There were three different kinds of embedded RAM blocks in the Stratix II FPGA device: M512, M4K, and M-RAM. Each memory block type provided different performance specifications and configurations to support different applications in FPGA designs; some of the specifications are shown in Appendix B. The M4K type memory was used in the processing node design as 'distributed memory' because it was easy to configure and include into the design in SOPC. 50 kBytes of on-chip RAM were allocated for each processing node.

5.2.5 Timer Core

The implemented Timer core was a 32 bit interval timer for ST2SS. It could be configured to function in different modes. When it was configured as a counter, it could be controlled to start, stop and to reset. It counted in two modes; count down once or continuous count down. The timer was equipped with a maskable interrupt request where an interrupt was asserted by the counter upon reaching zero. With this feature, it could also be configured as a watchdog timer, or periodic pulse generator, depending on

application requirements. Using it as a time-stamp driver, the time taken to perform or to run a function could be measured.

5.2.6 JTAG-UART

The JTAG Universal Asynchronous Receiver/Transmitter (UART) was implemented as a serial character stream communication between the host PC and the target processing node on the FPGA. The JTAG-UART used the JTAG cable to interact with the NIOS II Embedded Design Suite (EDS)[98] [99]. NIOS II EDS was software with a collection of tools, utilities, drivers and libraries that was used to develop the embedded software for the NIOS II processor. By using the NIOS II EDS, software could be written to display the required information on the host PC screen for development and debugging purposes.

5.2.7 NIOSNIC

This module contained two bus masters and a bus slave. This OS-Link Register, bus slave module, was used to monitor the status of the back-end designs. It also created the control signal to configure both of the bus masters for operation. The bus masters operation was covered in section 5.1.1.1.

The OS-Link Network Interface was responsible for the incoming and outgoing messages between the subsystem and the interconnected network. It formed full duplex bi-directional communication links. The incoming and outgoing messages were passed between the OS-Link Network Interface device and memory modules (either the SDRAM or on-chip RAM) via the system bus. The OS-Link Network Interface device's bus master modules were accessing the memory of the processing node using the DMA cycle stealing technique. Both were able to transfer one 32 bits data word each time when the target memory module was not accessing the processor. The nature of the Avalon Bus allowed the processor to continue to drive the system and the OS-Link Network Interface

device to access its target memory module simultaneously as long as both of them were not accessing the same memory module. This improved the overall performance of the processing node.

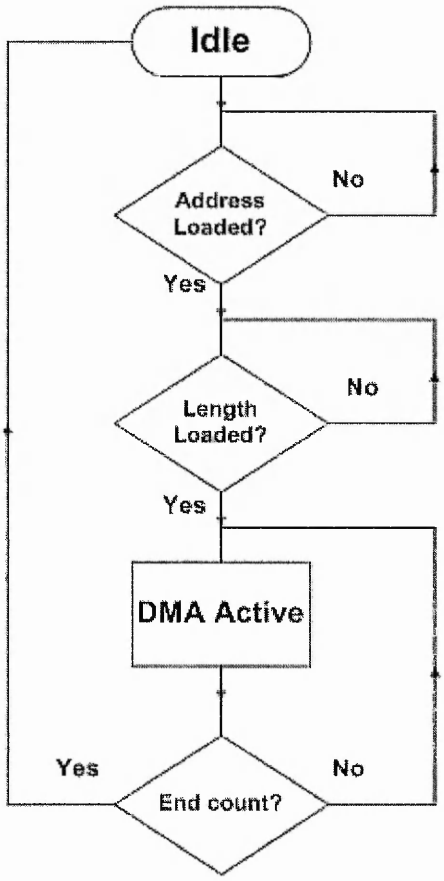


Figure 5-18: DMA Channel operational Flow diagram.

Both DMA channels operated on the same procedure. Figure 5-18 shows the DMA channel operational flow diagram. It started in the Idle State as no message passing takes place. To kick start a DMA transfer, the address and message length parameter was loaded into the OS-Link Register module where the message length information was the last one to be loaded. This was because loading the message length information sent a command pulse to the particular bus master to start the DMA operation. Hence the DMA channel was in the Active state where data transaction between the bus master and

memory module takes place. The DMA data transaction was subject to the resource status of the DMA buffer of that channel. The message transmit process was considered ended when the message length count was decreased to zero and the packetisation/de-packetisation was finished with no more data in the buffers. This would put an end to the message transfer and the channel would return to the idle status to wait for the next DMA operation to begin.

5.3 Embedded Distributed Multiprocessor prototype platform, OSL-ST2

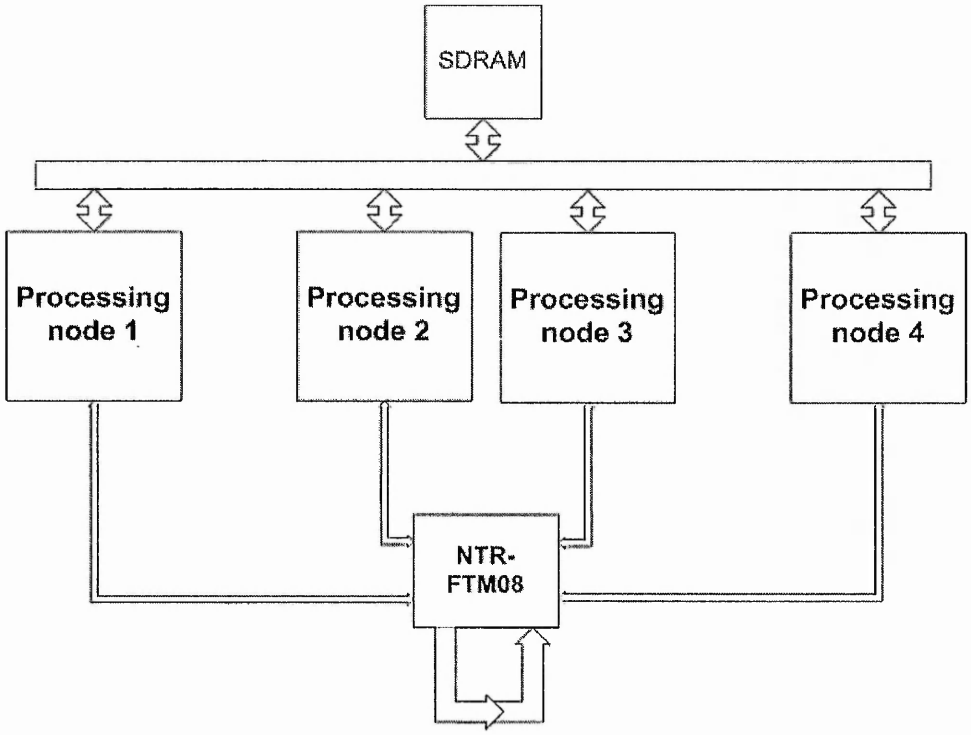


Figure 5-19: Embedded Distributed Multiprocessor platform with 4 processing nodes setup diagram.

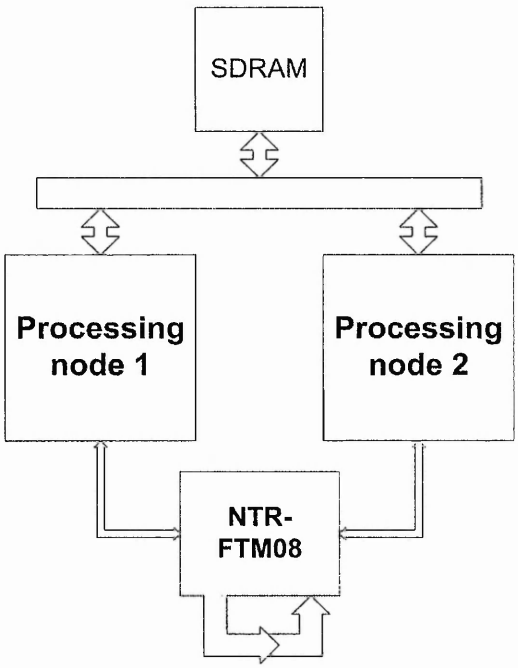


Figure 5-20: Embedded Distributed Multiprocessor prototype platform setup diagram.

Figure 5-19 shows the possible setup and connection of the Embedded Distributed Multiprocessor prototype platform, with the OS-Link embedded network, to be fit into the Stratix II 2S60 development board[100]. However two nodes prototype (the OSL-ST2), as shown in Figure 5-20, was realised in order to prove the principle. There were two processing nodes and a router implemented on the chip. Processing node 1 consisted of a NIOS II processor, 50 kBytes of on-chip RAM, a JTAG-UART and an OS-Link Network Interface Controller. Processing node 2 consisted of a NIOS II processor, 50 kBytes on-chip RAM and an OS-Link Network Interface Controller. Besides a 'dedicated' memory module for data storage in each processing node, both of the processing nodes shared the off-chip SDRAM. The SDRAM was used to store the program codes. The other reason SDRAM was shared, was to make the hardware testing easier, further explained in the next chapter.

A Bus based interconnection system is a well understood and widely used architecture in embedded systems. However its scalability is seriously limited. This bus-

based structure is still convenient for a SoC that integrates less than 5 processors [101] and rarely more than 10 bus masters. It is not suitable for long distance communications. Therefore in OSL-ST2, the interconnection of components within a subsystem utilised a bus based interconnection for the ease of data transaction between the bus master and bus slave pair, while serial communications (via routers) were used between processing nodes for message and data passing, just like a distributed system. This improved the performance of the processing node and increased the scalability of the overall system.

The router used was the OS-Link based Routing switch, NTR-FTM08. The router was connected to the constructed processing nodes as shown in Figure 5-19. All the remaining communication ports were temporary made 'loop back' (Transmit link connected directly to the Receive link of the same port) since only two processing nodes were implemented at this stage. All the 'loop-backed' connection could in future be modified to connect to more processing nodes or NTR-FTM08 routers.

6 Tests and Results

This chapter reports on the hardware tests performed on the design implemented on the Stratix II development platform. The aim of running the hardware tests was to ensure the functionality of the design and evaluate its success compared to previous implementations. The test results were also compared to the theoretical performance of the OS-Link Network Interface device.

The tests included investigation of the correct operation of the OS-Link Network Interface device i.e. transmission and reception of correct data; the DMA modules were able to perform various sizes of DMA transfer and to control the OS-Link Network Interface device.

6.1 Test Setup

6.1.1 Memory Distribution and Sharing

As all the off-chip memory and user I/O pins were fixed in the development board, in order to test the functionality of the developed multiprocessor system, only certain arrangements of memory mapping were possible. The OSL-ST2 was designed in such a way that some memory resources were shared, as shown in Figure 6-1.

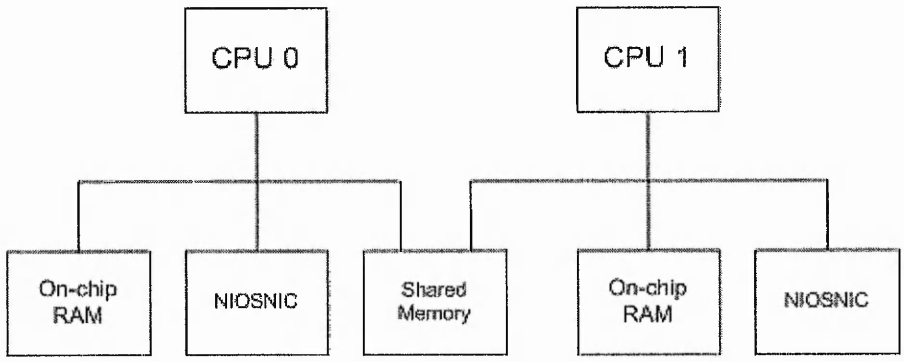


Figure 6-1: Shared and Distributed memory setup for hardware testing.

Each processing node has its own 'distributed' memory resource, the dedicated on-chip RAM, the SDRAM was shared by all processing nodes. The reasons that the SDRAM was shared were:

- Only one SDRAM chip was available on the development board. It required extra design effort to develop an extra SDRAM chip interface. Since all the hardware wiring was already fixed in the development board, only a few I/O pins were available for user interface, the remaining was unused as they are unconnected. Therefore the on-board SDRAM was either shared among the processing nodes or dedicated to one processing node only.
- The development board contained a 16 Mbytes SDRAM. It had enough memory space to store the program code for all the processing nodes in developed multiprocessor system, and it also had enough storage for other data.
- Testing was made easier because the original message could be setup in SDRAM or dedicated on-chip RAM in the first processing node; meanwhile the received message could be stored in an unused region in the SDRAM by the second processing node. The first processing node, which sent the message, would then compare the message received by the second processing node by accessing the dedicated memory region in the SDRAM.

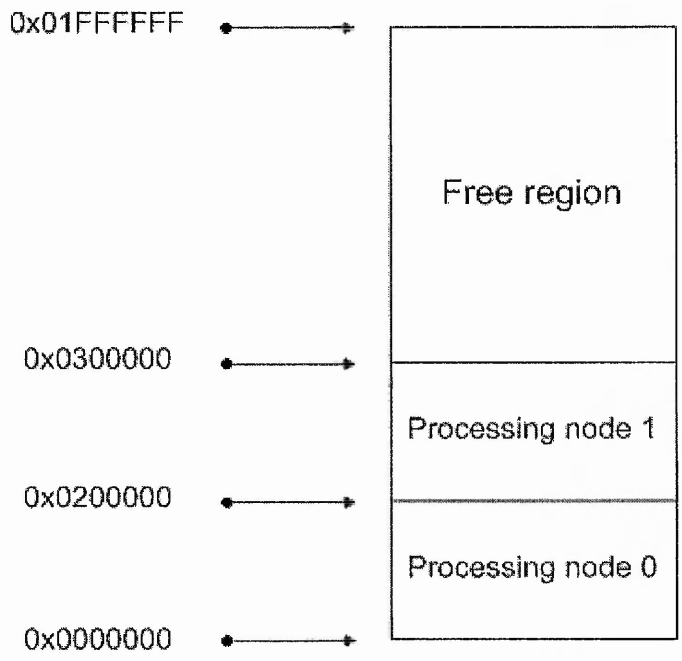


Figure 6-2: SDRAM utilisation in the test.

As shown in Figure 6-2, the address range starting from 0x000000 to 0x1FFFFF was dedicated to the program code of the first processing node; the address range from 0x200000 to 0x2FFFFF was dedicated for program code of the second processing node. The rest of the unused region was free and could be accessed by any processing node at any time.

Because it was a shared memory resource, the read and write operations had to be planned carefully, avoiding writing to any of the processing node's program code region, potentially causing that particular processing node to crash.

6.1.2 Pulse Generator

A pulse generator module was created to enable timing measurements. It had three input signals (reset, trigger and stop) and an output signal. The reset was an asynchronous reset used to reset the pulse generator in case of system failure; the trigger signal caused the pulse generator to start outputting a 'High' signal (when the stop signal was asserted, it will return to 'Low'). The output signal was connected to a user I/O pin where an oscilloscope was connected, as illustrated in Figure 6-3.

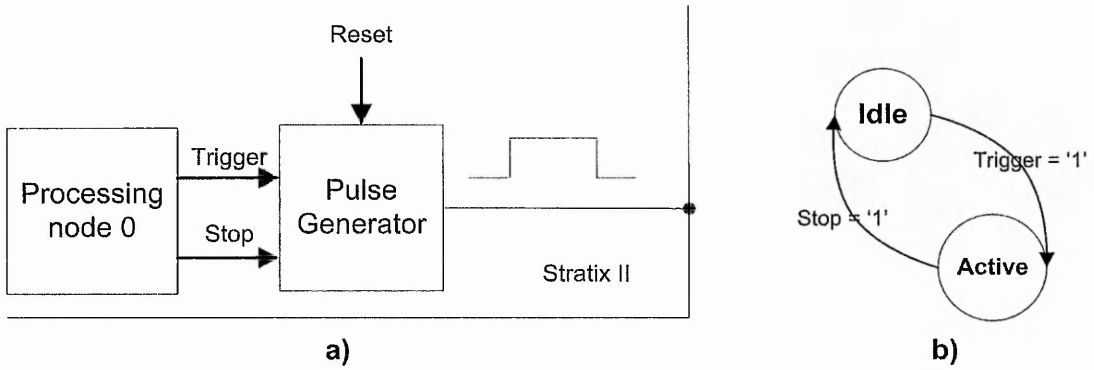


Figure 6-3: a) Connection of the pulse generator in timing measurement test. b) State machine in the Pulse Generator.

In the pulse generator module there was a state-machine, as shown in Figure 6-3b, which switched between the idle state and the active state. It started in the idle state upon reset. In the case when a trigger signal was asserted, it immediately proceeded to an active state and the output went high. As long as the stop signal was not asserted, the high continued, as shown in the timing diagram in Figure 6-4.

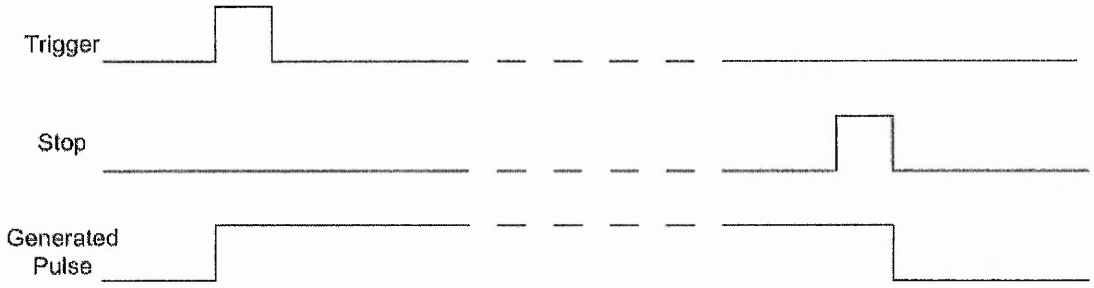


Figure 6-4: Timing diagram for pulse generator.

6.2 The NIOSNIC Test program

The test program was written in C and cross-compiled into binary code by NIOS II EDS software, before downloading into the SDRAM for real time testing. As the focus of this project was on the construction of the hardware of an Embedded Distributed Multiprocessor prototype platform, the software was designed purely to achieve the objectives of hardware testing. No operating system was implemented. The written program interacted with the host PC via the JTAG-UART. NIOS II EDS allows the user to interact with the developed embedded system on the development board via a console window, as shown in Figure 6-5. Through the console window, information or data can be displayed and data can be inputted and transferred to the embedded system.

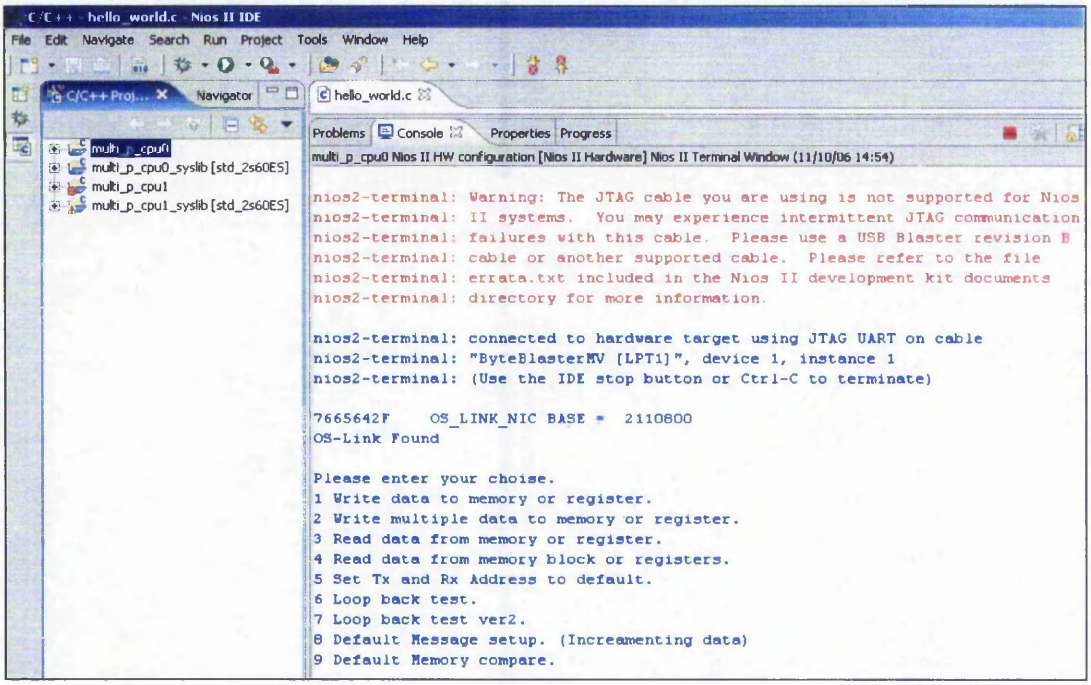


Figure 6-5: Console window in NIOS II EDS.

One program was written for each processing node. For the first processing node, the functions that were implemented in the test program included:

1) Single memory/register write.
2) Block memory write.
3) Single memory/register read.
4) Block memory read.
5) Message setup.
6) Self loop-back test.
7) Processor 1 transmit, Processor 2 receive test.
8) Memory block compare.

The first five functions were to perform single or block read/write operations. A user's 32 bits of data was written to a specific memory location or block of memory by using function numbers 1 or 2 respectively. Memory can be read and displayed on the

host PC's screen by calling Function 3 for a single location or 4 for a block of memory. These functions were used to create and manipulate data contained in the free memory region. They could also be used to configure or access the OS-Link Network Interface Controller device registers.

Function numbers 5 to 7 were used to test the OS-Link Network Interface Controller device via the processing node. Function 5 was used to create a message with different data patterns. There are 2^{32} possible data patterns for a 32 bit data word; however, it is very difficult to run hardware tests for all possible patterns. Therefore only selected patterns were used in the hardware test. The user could either perform a loop-back test using Function 6, or processing node to processing node message passing using Function 7.

To use Function 7, the second processing node had to be pre-loaded with a fixed task to load the Receiver Address register and the Message Length register in advance or reload both after a new message has been fully received. In other words, the receiver of the second processing node was always be ready to receive a new message. The Host PC had neither access to the second processing node (to manipulate the relevant registers to prepare the NIOSNIC for new messages) nor was able to display any messages from the second processing node. Therefore the user was not able to interact with both of the processing nodes simultaneously via the same JTAGUART.

6.3 Hardware Tests

The hardware tests were divided into three stages. The first stage was run when the first processing node was successfully constructed. The setup of the first stage of the experiment was shown in Figure 6-6, section a. The communication link at this stage was made a 'loop-back'. The second stage was to connect both processing nodes directly so that one processing node sent messages and the other one received messages as shown in

Figure 6-6, section b. Finally the last stage, illustrated in diagram c) of Figure 6-6, was to interconnect both processing nodes via the Router.

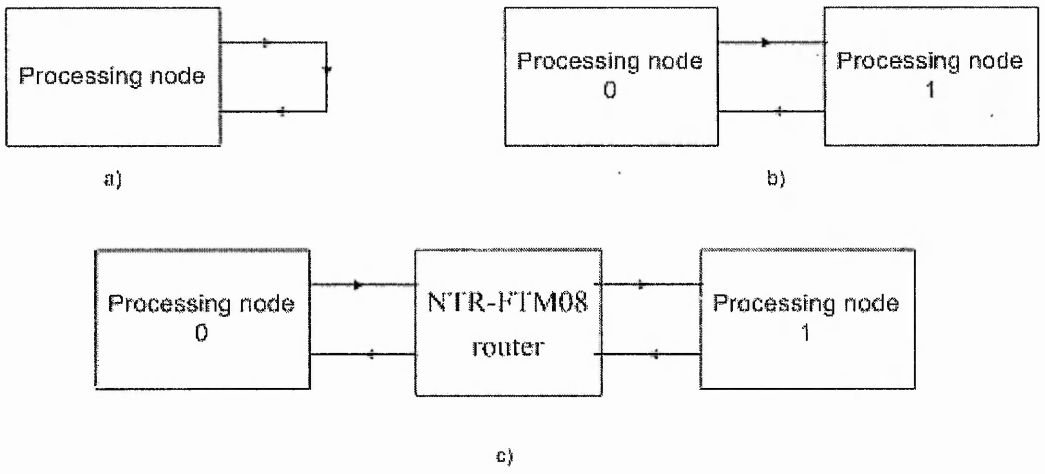


Figure 6-6: Tests setup a) Self Loop back of one processing node. b) Direct connection between 2 processing nodes. c) Interconnection of 2 processing nodes via OS-Link Router.

The 'loop-back' communication test involved message fetching from the target memory location via DMA operation, and then processing the message in the OS-Link Network Interface device. The message was then transmitted by the OS-Link Network Interface device. Because it was all built on-chip, the receiver would pick up the message almost immediately after the message left the transmitter. The receiver reassembled the data tokens into 32 bit data and then transferred the data to the target memory location via DMA transfer.

This 'loop-back' test ensured the functionality of the constructed OS-Link Network Interface device and assessed its bi-directional communication capability. The clock source was from the Phase Lock-Loop (PLL) core. The PLL was used to generate the system clock and the OS clock. The Avalon Bus was operating at 50 MHz (system clock). The OS clock was altered from 50 MHz to the maximum operational frequency for the OS-Link Network Interface device, within the PLL specification. Only results from the maximum data rate are detailed here.

Due to lack of software drivers developed for the new Embedded Distributed Multiprocessor system, all the read/write, transmit and receiver operations were performed through the test program written, compiled, downloaded and interacted with via the NIOS II EDS. Parameters to initialise the DMA operation of the OS-Link Network Interface device were loaded manually via the functions in the test program.

6.3.1 Avalon Bus Access Testing

The Avalon Bus access procedure began with the address and read or write signal asserted. Data was sampled from the data bus at the rising synchronising clock after the read or write request was made, if the 'waitrequest' was not asserted. The switch fabric in the Avalon bus used round robin scheduling[102]. The bandwidth allocated for each bus master's transfer was determined by the number of bus masters that were attempting to access the same memory module and the transfer type used by bus masters involved. There were 3 reasons that 'waitrequest' would be asserted by the arbiter: a) Target slave already being accessed by another bus master; b) Latency required for the target slave module to produce valid data; c) The slave module was not ready for any transaction yet. If 'waitrequest' was asserted, the bus master had to wait until 'waitrequest' was de-asserted to perform a valid read/write operation. Figure 6-7 shows a sample of timing diagram of a DMA transfer by the Transmitter Bus master. Figure 6-8 shows a captured timing diagram for SDRAM arbiter when accessed by the Transmitter Bus Master.

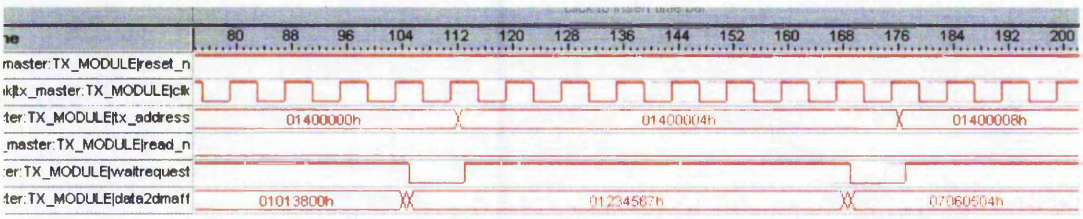


Figure 6-7: Captured Timing diagram for a fundamental Avalon bus transfer at 100 MHz system clock.

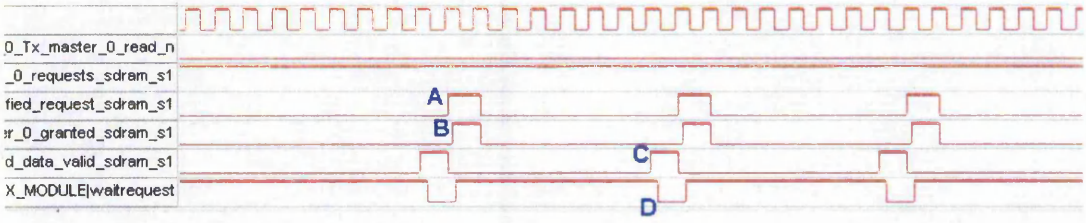


Figure 6-8: Captured Timing diagram for SDRAM arbiter at 100 MHz system clock.

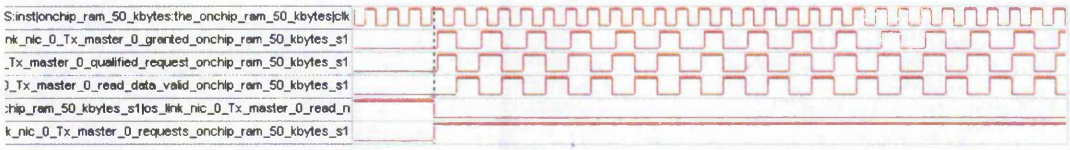


Figure 6-9: Captured Timing diagram for on-chip RAM arbiter at 100 MHz system clock.

Both Figure 6-7 and Figure 6-8 were diagrams captured simultaneously by tapping some of the signals from Transmitter DMA module of the NIOSNIC bus master and the SDRAM arbiter (respectively) using the Signal Tap function from the Quartus II software. Both timing diagrams were captured when the Transmitter bus master was accessing the SDRAM to read data. Figure 6-9 shows the captured timing diagram for activity in the on-chip RAM arbiter when the Transmitter DMA module of NIOSNIC bus master was accessing the on-chip RAM to read data.

The following explains the sequence of events captured when accessing the SDRAM:

- Point A - The bus master asserted 'read_n'.
- Point B - Valid access request received by the SDRAM arbiter.
- Point C - Valid data was available in data bus.
- Point D - 'Waitrequest' de-asserted so that bus master could sample the data from the data bus at the first raising clock edge after 'waitrequest' was de-asserted.

Observing the signal pattern of the SDRAM arbiter in Figure 6-8 and Figure 6-9, the timing diagram suggests that each transaction of 32 bit data was treated as a single memory access request because the memory request was repeated after each valid data was presented on the data bus. The difference between both timing diagrams is that the SDRAM incurred a higher latency cycle than the on-chip RAM. On average, the SDRAM and on-chip RAM have 8 latency cycles and 1 latency cycle respectively. This will give a data throughput of 22.2 MByte/s and 100 MByte/s for SDRAM and on-chip RAM respectively, using the equation below:

$$data\_throughput = \frac{transaction\_length \cdot system\_clock}{no\_of\_latency\_cycles + 1 \cdot transaction\_cycle}$$

Where transaction_length = 4 Bytes

Equation 1: Data Throughput for DMA.

6.3.1.1 Clock Cycle Efficiency Testing

The clock cycle efficiency of the OS-Link Network Interface device Bus master's DMA transfer was calculated. Referring to Figure 6-8, the arbiter treated each assertion of read or write request as a single transaction request. The formula that was used to calculate its efficiency was as below:

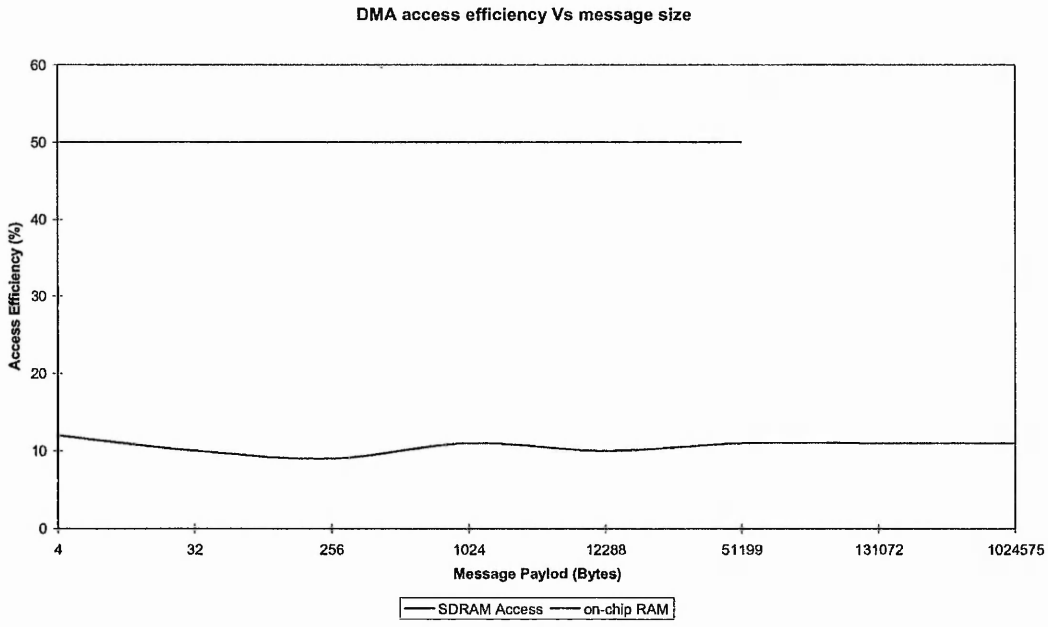
$$efficiency = \frac{number\_of\_transactions}{setup\_latency + number\_of\_transactions} \cdot 100$$

Transaction = Data exchange between a bus master and slave pair.

Equation 2: Clock Cycle Efficiency equation.

The efficiency of the DMA transfer of the OS-Link Network Interface Device was calculated based on the equation above. To calculate the efficiency of the DMA module, two counters were implemented in the Transmitter bus master module. One to count the

number of successive transaction, and the other was to count the number clock cycles the DMA module has to wait for valid data. Graph 1 shows the results of the calculated Avalon Bus access efficiency for OS-Link Network Interface device's Bus Master verses the message payload length.



Graph 1: Graph of the Efficiency of the OS-Link Network Interface device's Bus Master versus message size.

The message size was varied for the SDRAM tested from 1 byte up to 1 MByte and the on-chip RAM up to 50 Kbytes (50 kBytes was dedicated). Message length restrictions being due to the limitation of the message length register (20 bits) and the size of the distributed on-chip RAM. The DMA module shows higher efficiency when accessing the on-chip RAM compared to the efficiency of accessing the SDRAM. The efficiency of on-chip RAM access was constantly 50%, with only 1 latency cycle incurred by the arbiter per successive transaction. The efficiency of the SDRAM access was about 10% on average. The result of the SDRAM access efficiency was not constantly 10% and different tests may give a different variation (with the same range of deviation) because the memory was accessed by the processor for instruction fetching from time to time and

also the SDRAM needed to be refreshed periodically. Therefore the efficiency difference between the SDRAM access and on-chip RAM was mainly due to the access latency of the memory modules.

6.3.2 Data Transfer Tests

The data transfer test was the measurement of the message duration for a complete transmission and reception of a message. The measurement was performed on messages of various sizes, from 1 byte up to 1 MByte. This was because the maximum allowed message size for the Message Length Register was set to 20 bits, i.e. 1 MByte.

The time duration was measured from the moment the Packetiser module started to read data from the DMA buffer of the transmitter until the message has fully received and stored in the DMA buffer of the receiver. The duration of DMA operation was not included due to the reason that the 'waitrequest' signal could be asserted unpredictably (anytime when there was another bus master intending to access the same memory device). The setup of the message transfer duration test is shown in Figure 6-10.

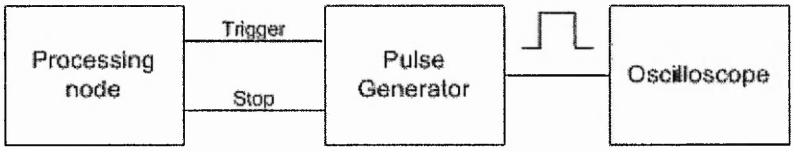
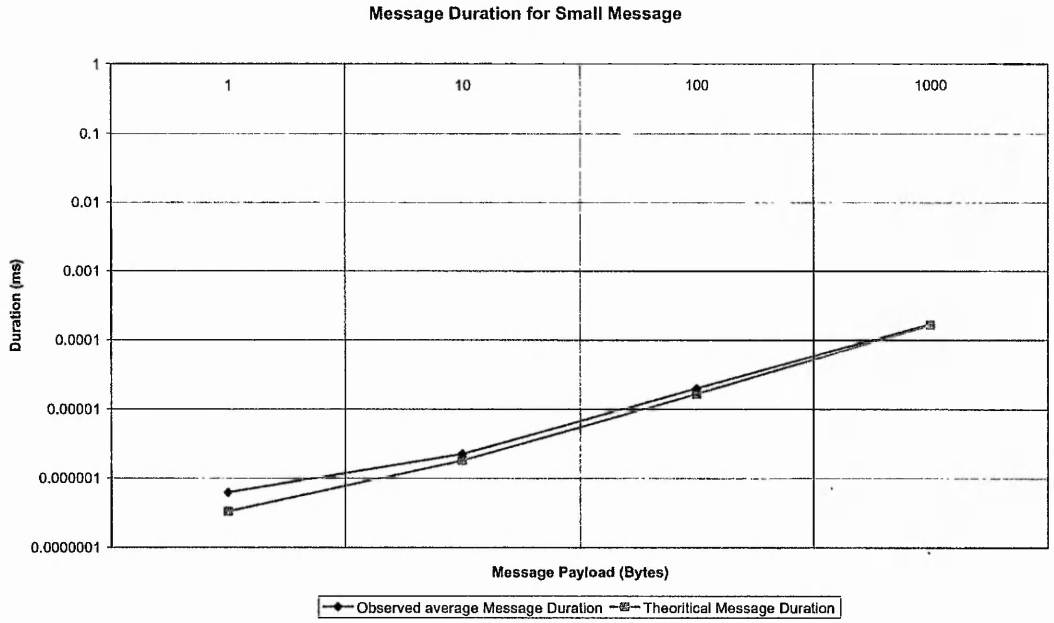


Figure 6-10: Block diagram of message transfer duration test setup.

Two signals from the OS-Link Network Interface Controller, a trigger and a stop signal, will be connected to a pulse generator module (refer to section 6.1.2). The OS-Link Network Interface Controller asserted the trigger signal when the Packetiser module started to access the DMA buffer to extract data; the stop signal was asserted when the receiver received the whole message and the last byte was written to the receiver's DMA buffer. In the pulse generator module, there was a state-machine. The state-machine was in the idle-state upon start-up or reset. When a trigger signal was asserted, the state-

machine moved from the idle-state to the active-state. The state-machine only returned to the idle-state when the stop signal or reset signal was asserted. During the active-state the pulse generator outputted a 'high', which was sampled by the externally connected oscilloscope and the duration or the length of the pulse determined the duration of the message transfer.

6.3.2.1 Small Message Transfer Duration Tests



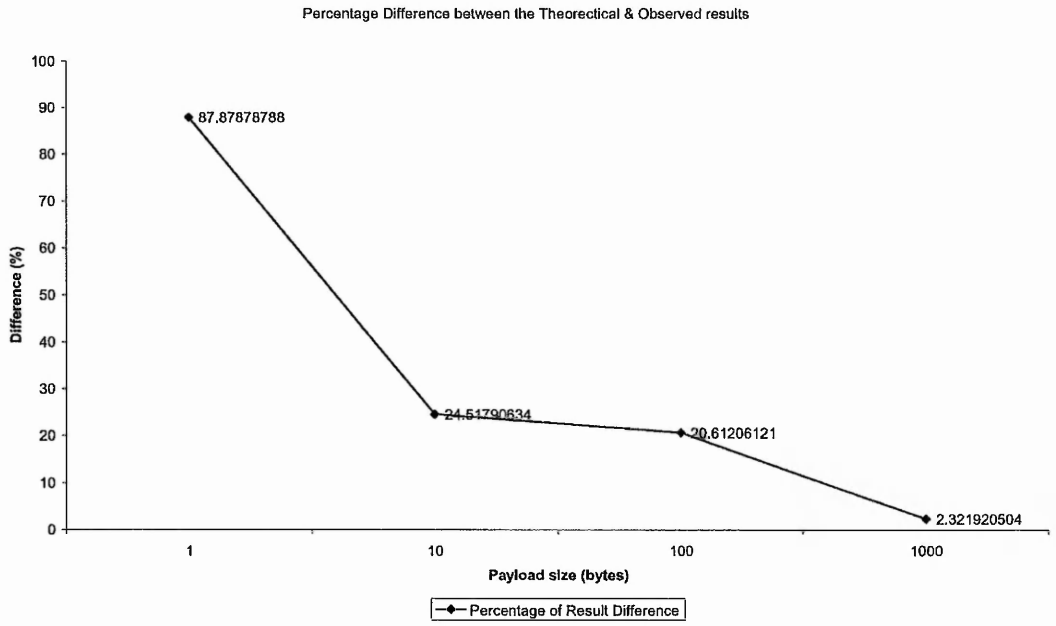
Graph 2: Message duration result for small message up to 1 kBytes at 48.5 Mbit/s Data Rate

Graph 2 shows the performance of the OS-Link with a small message payload, i.e. up to 1 kBytes. The increase in message size resulted in a linear increase in message duration. It is noticeable that there is a gap between the theoretical results and the observed results. This was because the theoretical results only take into account the length of time taken to transmit the message across the medium, factors such as message packetisation or depacketisation delay, FIFO access and token serialization were ignored. The lower the message size then such information as message headers, type bit and synchronization bits are more significant compared to the actual data. For example, the message with 1 byte of header and 1 byte of data will consist of 14 redundancy bits and 8 data bits: only about 36% of this message will be the actual transmitted data. Comparing both results from the graph, as the message size increases the slope of both curves is almost the same. It is suggested that the difference between the two was due to initial

start-up latency. However, it becomes insignificant as the message size increased, as shown in Graph 3.

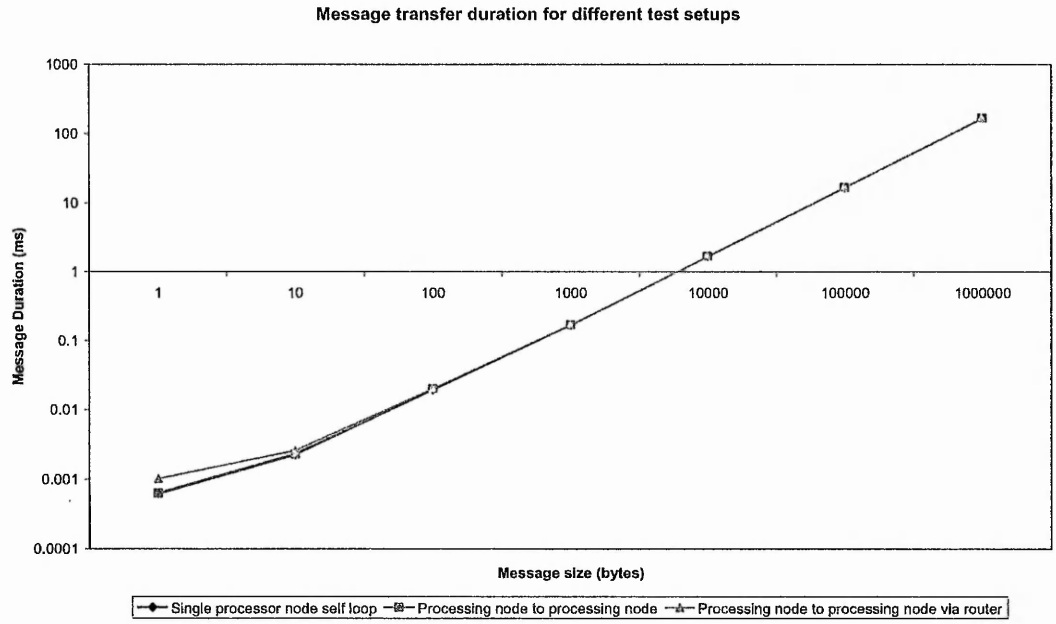
$$Percentage\_difference = \frac{Observed\_value - Theoretical\_value}{Theoretical\_value} \cdot 100$$

Equation 3: Percentage difference of the Observed Data Rate and Theoretical Data Rate for small message.



Graph 3: Percentage Difference between the Theoretical and Observed Results for small messages.

6.3.2.2 Message Passing Duration Tests



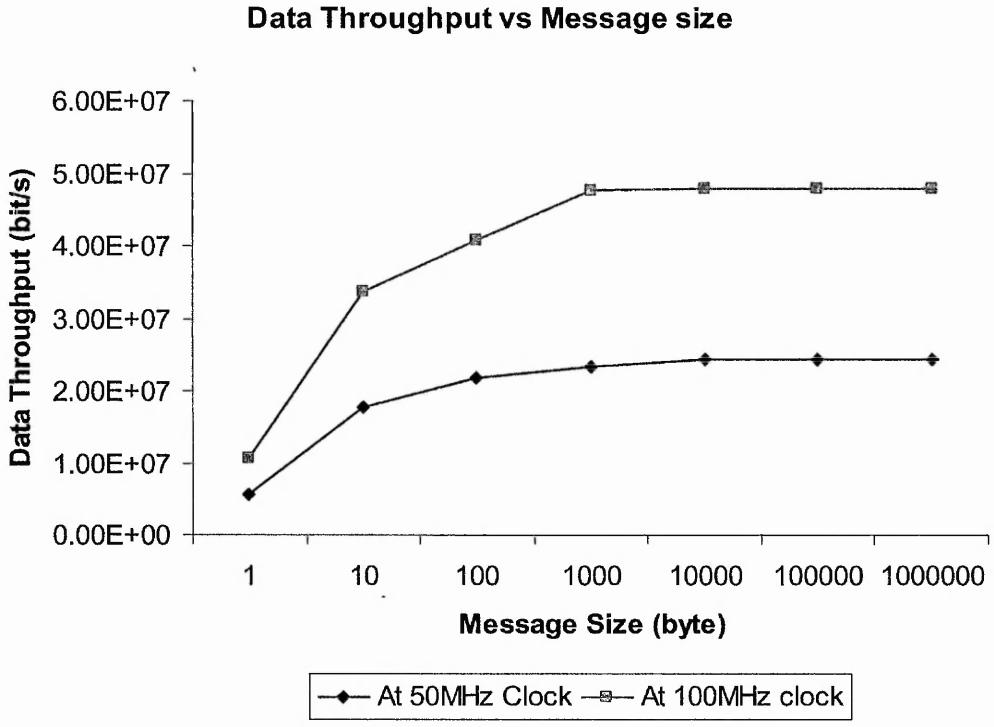
Graph 4: Message round trip duration measurement for different test setups.

Graph 4 was the measurement of round trip duration for messages of various sizes. The transfer duration measures the length of time taken for round trip travel of a message being transmitted and received. It ignores the delay factor for DMA operation due to the unpredictable wait assertion.

The duration of the round trip message transfer for the single processing node loop back and the processing node to processing node setup are nearly the same. When referring to Table A-5, the difference of the time duration for these two set of tests was so small that it is approximately 1.8%. It is suggested that any extra time is due the additional transition time required by the message to travel from one processing node to the other, instead of the shorter immediate loop-back. Looking at the third test set up, which was the message transfer from one processing node to another via a router, at the

lower message size, it seems to take approximately 0.41 μs extra for each message. The time difference was more obvious for lower message payload sizes below 1 kByte. The extra time was consumed by the router decoding the received header flit and directing the remaining message to the designated output port. Beyond message payload sizes of 1 kByte, the message passing becomes relatively constant at around 47.3 Mbit/s.

6.3.2.3 Data Throughput Tests



Graph 5: Average Data throughput of OS-Link network at various message size and system clock

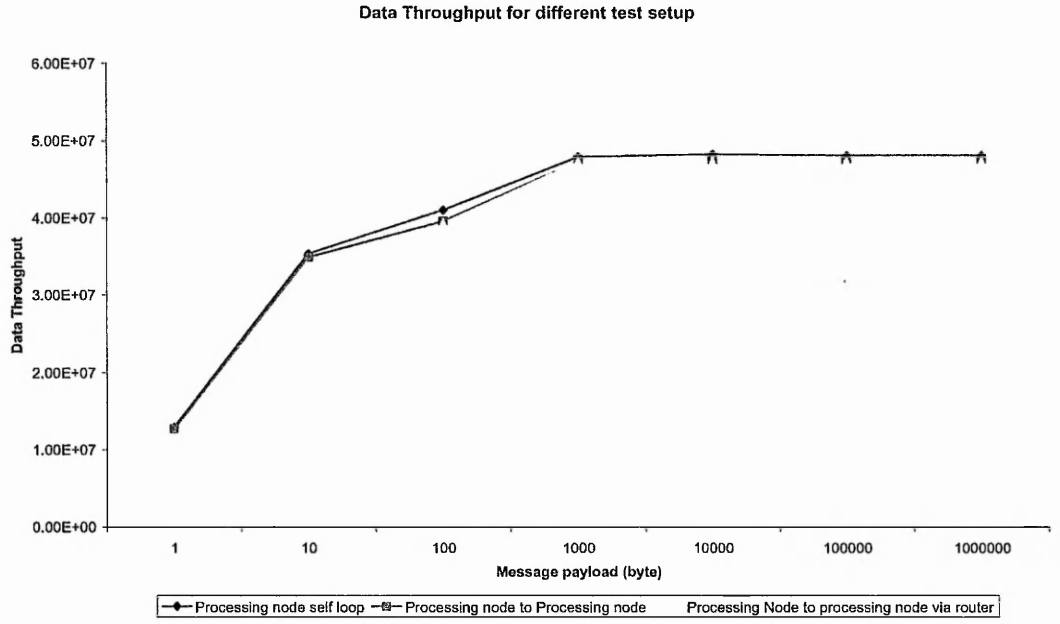
$$data\_throughput = \frac{2 \cdot clock \cdot data\_bit}{3 \cdot token\_bits}$$

Equation 4: Data Throughput equation

The Sampling frequency of 1.5 times of the link rate was applied in the OS-Link based network, therefore by substituting data bit (8 bit), token bit (11 bits) and the clock with the sampling frequency of 50 MHz and 100 MHz, the resultant theoretical data throughput would be 24.4 Mbit/s and 48.5 Mbit/s respectively. The 1.5 factor arises from 3 times oversampling using both edges of the sampling clock[103].

Graph 5 represents the average data throughput for the first and second test setup, see Figure 6-6. The measurement of data throughput was at an OS clock frequency of 50 MHz or a clock frequency of 100 MHz. The highest frequency that the OS-Link Network Interface Device can operate was tested at about 100 MHz. According to the graph, increasing the OS-clock frequency successfully increased the data throughput.

It can be observed that a message with payload lower than 1000 bytes has lower throughput. The reasons for this were similar to the analysis in section 6.3.2.1. As the payload size increases, the ratio of redundancy to the actual data will become less significant. At a message size of 1 Mbyte, approximately 72% of the message was actual data. When message payload size was beyond 1 Mbytes the data throughput achieved was about the theoretical value, 48.5 Mbit/s.



Graph 6: Data Throughput Comparison between Different experiments setup.

Graph 6 shows the data throughput achievement of three of the experiments setup. The results were calculated based on the message payload size and the duration of the message passing. The theoretical value of the OS-Link data throughput is 48.5 Mbit/s, according to Equation 4. The highest data throughput achieved by a single processing node loop-back, processing-node to processing-node and processing nodes via a router were 48.0 Mbit/s, 47.3 Mbit/s and 47.3 Mbit/s respectively, see Table A-4. All three maximum achieved results that were close to the theoretical value. The data throughput for the processing node to processing node via router setup was lower for message payloads below 1 kByte. This was due to the overhead added to the data in the data tokens (such as: header byte, type bits, Start bit and stop bit). At lower message payloads the ratio was higher comparing overhead to actual data per message.

6.3.2.4 Correct Message Passing Test

The implemented network was also tested for correct operation i.e. that the transmitted and received messages are exactly the same. For every 32bits of data, there are 2^{32}(4294967296) binary permutations, not to mention the possible permutations available for the whole message, so it was too time consuming and difficult to test all the possible permutations. Therefore only a few combinations of patterns were selected in order to test the correct operation of the message passing interface and medium. The chosen patterns were as follow:

Pattern (hex form)
00 00 00 00
FF FF FF FF
00 01 02 03(incrementing)
AA AA AA AA
55 55 55 55
80 80 80 80
08 08 08 08

Figure 6-11: Chosen bit patterns in Hexadecimal for message passing tests.

Each message pattern was transmitted, received and compared for all 3 test setups i.e. loop back, processor to processor and processor to processor via a router. The result of pattern comparison was 100% matched. The logged result is available in Appendix A.

6.4 Resource Utilisation Report

The entire design was implemented in a EP2S60F672C5ES of the Stratix II Family. OSL-ST2 and consumed 24264 ALUTs (50% of the total available on the chip), 928064 memory bits (36% of the total available), 17556 registers and finally 126 I/O pins were used (25% of 493 I/O pins). Table 2 below is the simplified Resource Usage Table for the entire design. It was obtained after the compilation and fitting process was completed. It shows the resource utilisation for each module in terms of Logic Cell Combination, Logic Cell Register and memory bits usage.

Main Module	Sub Module layer 1	Sub Module layer 2	LC Combination	LC Register	Memory bits
Top Level	OSL-ST2		20771	17744	928064
Router			2057	1930	3200
CPU1			2064	1592	76032
JTAGUART			122	106	1024
On-Chip RAM			80	4	409600
OS-link NIC	Tx Master(Controller)		131	120	
	Tx DMA buffer		2145	2144	
	Rx Master(Controler)		31	32	
	Rx DMA buffer		2157	2119	
	OS Link Register		67		
	Message Interface	Packetiser	59	20	
		Depacketiser	94	82	
	Link Interface	Tx OS-Link Buffer	146	324	
		Rx OS-Link Buffer	151	324	
		Transmitter	36	34	
		Receiver	28	24	
CPU2			2064	1592	76032
JTAGUART			122	106	1024
On-Chip RAM			80	4	409600
OS-link NIC	Tx Master(Controller)		131	120	
	Tx DMA buffer		2145	2144	
	Rx Master(Controler)		31	32	
	Rx DMA buffer		2157	2119	
	OS Link Register		67		
	Message Interface	Packetiser	59	20	

		Depacketiser	94	82	
	Link Interface	Tx OS-Link Buffer	146	324	
		Rx OS-Link Buffer	151	324	
		Transmitter	36	34	
		Receiver	28	24	

Table 2: Simplified Table of Resource Usage in OSL-ST2 System.

6.5 Power Consumption Report

Embedded systems are often used in battery powered applications, for convenience or due to their application requirements. Power consumption is one of the crucial factors that will determine the length of time that the system can operate.

The internal architecture of the target FPGA allows the synthesis tool to estimate the logic cell utilisation after fitting the design to the target device, including configuration details such as clock frequency the software is able to estimate power consumption. Highly accurate power consumption assessment can be made by the Quartus II software by activating the PowerPlay Power Analyzer. Assumptions were made such as ambient temperature was 25 °C and the cooling solution was a 23 mm heat sink with 200 LFpM Airflow. An alternative power consumption calculator[104] is also available for the Stratix II Family.

	Operational Clock Frequency	
Module	50 MHz	100 MHz
NIOSNIC	110.02 mW	121.51 mW
NTR-FTM08	60.17 mW	120.32 mW

Table 3: Power Consumption of NIOSNIC and NTR-FTM08 at two different Operational Clock Frequencies.

Table 3 shows the power consumption results of the NIOSNIC and the NTR-FTM08 router operating at operational clock frequency of 50 MHz and 100 MHz obtained from the power consumption report generated by the PowerPlay Power

Analyzer. The table shows only about 10 mW increase of power consumption in NIOSNIC while the power consumption of the NTR-FTM08 doubled as the operational clock frequency doubled. This is because only part of the NIOSNIC was synchronised by OS-Clock (see section 5.1) while the NTR-FTM router was synchronised by the OS-Clock. The increase of power consumption was due to fast-switching of transistors when operating at higher frequency[105].

The total power consumption for the ST2-OSL was 1532.57 mW while using a 50 MHz System Clock frequency and a 100 MHz OS-Clock frequency.

6.6 Summary and Discussion

Section 6.2 first explains the setup of the test bed; each processing node has its own dedicated memory resource i.e. on-chip RAM, and shared memory resource i.e. SDRAM. The memory resource was distributed and shared in such a way that it makes the hardware tests easier. This section also included the explanation about the pulse generator, which was used for duration measurement. The duration of the generated pulse was the period of time used in the process of transmitting a message until the message was fully received. This pulse was captured by an externally connected oscilloscope via a user I/O pin for measurement.

Section 6.2 explains the available functions in the test program that were used to access the addressable memory of each component. This test program was also used to setup a message, initiate message passing, and finally compare the transmitted and received message.

The result of the hardware test was explained in Section 6.3. The DMA module's memory access was tested on both SDRAM and on-chip RAM. The DMA modules were using the Avalon bus fundamental transfer type when accessing target memory modules. The observed timing diagram shows that each 32 bit transaction between a bus master

and the target memory module was treated as single memory access request. The transfer overhead of the DMA transfer, of the NIOSNIC, relies on the type of Avalon bus transfer used and the timing property of the target memory module accessed, thus different clock efficiencies and data throughput are obtained dependant on the exact type of RAM. It is possible to increase the DMA module's data throughput by using other Avalon bus transfer types, such as Burst Transfer or Pipeline Transfer. The various properties of each transfer type are explained in the Avalon Memory-Mapped Interface Specification version 3.2[92]. The other types of transfer must be implemented with precaution as each type will have a different effect on the performance of the processor and NIOSNIC.

The hardware tests also demonstrated a significant improvement in the data throughput of an OS-Link based network when implemented in a single programmable chip, especially when compared to the XA1 prototype. The operating clock frequency has been successfully increased from originally 66 MHz in the XA1 prototype to 100 MHz in the Stratix II chip. Each OS-Link channel can give unidirectional data throughput of up to 48 Mbit/s.

The data throughput of the DMA module was higher than the data throughput of the OS-Link network (see section 6.3.1). The DMA buffer was filled up faster than the NIOSNIC can transmit the message. Therefore multiple DMA requests were required for message sizes larger than 300 bytes. This is due to the DMA modules and the Packetiser being synchronised by the same clock frequency. It took 1 clock cycle to store 32 bit data into the DMA buffer but 5 clock cycles to packetise (1 clock cycle to read a 32 bit data from the DMA buffer, 4 clock cycles to format the 32 bit data into four 9 bit tokens) and buffer the tokens into the Token buffer.

Section 6.4 is the report of resource utilisation of each module in OSL-ST2 on the Stratix II chip. For each module it states the Logic Cell (either combinational or register) as well as the memory bits used. After implementing two processing nodes and a NTR-FTM08 router, only approximately 50% of the ALUT and 36% of the memory bits were utilised. The Stratix II chip resource utilisation has suggested the potential of

implementing up to four processing nodes in it. However, to implement more processing nodes than the initial design will involve the distribution of on-chip RAM to be reconsidered. This is because distributing 50 kBytes of on-chip RAM to each of the two processing nodes initially used up all the M4K type RAM in the current Stratix II chip. To use the other two types of RAM resource available will involve a different configuration of the system generation in the SOPC and a redesign of the memory interface logic.

Section 6.5 is the summarised power consumption analysis report. The report shown in Table 3 shows the increase of power consumption due to the increase of the operational clock frequency. Although operating at higher clock frequency theoretically will give higher data rate, the increase of operational frequency causes higher consumption of power. Network interface controllers such as Ethernet, which can operate at multi gigabit per second speed, could result in power wastage[106]. A large number of applications run at higher speed than they need to and are also left 'On' 24x7 with low utilisation. A white paper provided by Ethernet Aliance[107] stated that ADSL with data rate of 24 Mbit/s has power consumption of about 2 W when operating in full data rate. Even when it is in 'Low' power and 'Off' state the power consumption are 0.75 W and 0.3 W respectively[108]. Therefore high power consumption communication systems may not be suitable for embedded systems when power consumption is a crucial requirement, taking factors such as the distance, data rate requirement, and power supply use into consideration. The power consumption of the test FPGA was approx. 1532.57 mW for a data rate of 48.5 Mbps, excluding the transceiver that would be needed for longer distances. The design has not been optimised for power consumption and could offer possibilities for further work in this area.

7 Conclusions and Future Work

7.1 Conclusions

This thesis has documented research into a novel distributed multiprocessor system on a single programmable chip. This involved the construction of custom Avalon-based OS-Link Network Interface controller. The network interface controller was then implemented as part of a processing node, which consisted of a NIOS II processor and memory modules. These were designed to be used as serial communication building block in a NTR-FTM08 router based embedded network. The OS-Link embedded network is targeted at real-time, distributed, embedded multiprocessor applications. The interface device could be utilised to produce a communication network linking multiple processors either on-chip or off-chip. The OS-Link embedded network would allow multiple RISC processors to operate as processing nodes in the same network, thus increasing system flexibility and applications. Inter-processor bi-directional throughput was increased by implementing multiple processing nodes on the same chip (when comparing to previous systems in the research group).

The network interface device was designed to interconnect the NIOS II processor to the router network which utilises an adapted OS-Link protocol. The network interface controller was named NIOSNIC. It will read the messages from memory, via a DMA operation, and convert them to router tokens before transmission (and vice versa). The network interface controller for the processing node was built following on from work based on the previous XA1 prototype design; adapted, in amongst other respects, to reduce the processor interval required to initiate the DMA operation when message passing (so that the processor can return to its dedicated task faster). The NIOS II system uses the Avalon Bus interface; therefore the NIOSNIC was designed for this interface.

Two processing nodes, sub-systems within the OSL-ST2 SoC, were constructed: each has a NIOS II processor, 50 kBytes of on-chip RAM, and a NIOSNIC module. They

were generated by using SOPC software in VHDL format, where each module was included and connected via a graphical user interface provided by the Quartus II software.

The custom NTR-FTM08 router was used as inter-processor communication medium in this embedded multiprocessor network. It is an 8-channel off-the-shelf hardware message router, which was developed by previous researchers in the NTU research group. It allowed up to 8 processing nodes to be connected, or to form a larger network by linking to additional NTR-FTM08 routers. Using the routing device, different network topologies can be formed to suit the application, such as: star, daisy chain, irregular or even hybrid networks.

Together with the processing nodes, the entire system was compiled and synthesised using the Quartus II software, before being incorporated into ASIC design. After synthesising the designs, the designs were downloaded onto the FPGA chip through the JTAG so that it can be tested. The test program was also downloaded into the dedicated memory region in the same manner, via the JTAG. The testing and debugging were aided and performed on the development board with re-programmable SRAM based FPGA, Stratix II 2S60 chip. The FPGA can be programmed/re-programmed allowing modifications and experimentation without the need to invest in new hardware when design errors occur. The functionality and performance of the NIOSNIC and the router was verified via real-time hardware tests.

There were two communication architectures implemented in the designs. Within a processing node, a shared bus or bus based topology was used for the reasons discussed in section 3.3. The system bus used in the design was the Avalon Bus. As described in section 3.3, the Avalon Bus is a simple bus architecture used by Altera in its SOPC design. It provides a simple and easy to understand protocol that will help to keep the interface design as simple as possible (thus reducing the use of Logic Cells). It also provides an interesting arbitration technique, which is the Slave-side arbitration where each bus slave will have its individual arbitrator, which controls the access of the bus masters that interconnected to it. The implementation of Avalon bus allows multiple bus

masters to access the interconnected bus slaves simultaneously, provided they are not accessing the same bus slave. This enables the processor to run its task, while the NIOSNIC is accessing the memory module for data transfer. This will help to improve the overall processing node performance and speed up the task execution time, because one bus master does not have to wait for another to release the bus access in order to access the target bus slave as in central arbitration system.

One of the processing nodes has a JTAGUART. It is a special UART core that can transfer data over the JTAG connection. The JTAGUART was used to interact and communicated with the Host PC. The use of the JTAG-UART has eliminated the need for separate RS232 connection to host PC for communication compared to the XA1 prototype or previous group designs. Through the JTAGUART, messages or data from that processing node can be displayed on the console window of the NIOS II EDS software; also via the console window, data can be inputted to that processing node.

During the development process, it was realised that the amount of memory resource on chip is a crucial issue that must taken into consideration when designing this SoC. The on-chip RAM for the processing cores is divided with other peripherals or modules that require storage. There were a few factors that determined how the memory resource was distributed:

- The type of system to be developed. A Symmetric Multiprocessor (SMP) system will be easier because the memory resources are shared by all bus masters. However the Massively Parallel (MPP) system, or distributed system, has a more complex distribution of the on-chip RAM. The decision of the amount of memory to be distributed to one processor will depend on the requirement of the application or task to be dedicated to that processor.
- The requirement of memory by other peripheral or modules. Besides the processors, some of the peripherals or other modules (for example DMA machine, UART, and other custom designs) require storage; to store data before they are transferred to other locations or processed.

In the case of OSL-ST2, with 2 processing nodes, each allocated 50 kBytes of M4K type On-Chip RAM with 32bits data bus, only 36% of the total memory resources (of approximately 9 Mbits availability) were consumed. The distribution of 100 kBytes to both processing nodes has fully consumed all the M4k type On-Chip RAM. However, adding extra processing nodes is still possible by using other (slower accessed) memory. The distribution of the On-Chip RAM will be different depending on the requirement of the application. In the case where more memory resources are required; there are two options to overcome the lack of memory resource on-chip. Firstly, use the logic element or logic cell to construct storage (sacrificing these); secondly, relying on additional off-chip memory resources. Adding off-chip memory will require the use of I/O pins. In this case the loss of I/O pins available for the design and the type of I/O pins to be used will be a consideration.

The key conclusions resulting from the hardware tests of the designs are presented below:

- The Over Sampling technique used in the OS-Link based network proved to be capable of operating at sampling frequencies as high as 100 MHz in the network on-chip. The Phase Lock Loop (PLL) core enables the alteration of sampling frequency. Therefore the designs can be tested on different sampling frequencies. The embedded network was operating with the sampling frequency of 100 MHz compared to the previous prototype XA1 system, which was running at a maximum sampling frequency of 50 MHz.
- By including the message headers into the message payload, the intervention of the processor in order to initiate a message transmitted has been reduced by 33%. The previous prototype design needed to perform three write operations to initiate a transfer: to input the address where the message payload was stored, to input the message header, and finally to input the message length into the registers. By including the message headers into the message in advance, the processor

required only to write to the address register and the message length register to 'kick start' a message transmit.

- Alteration of the Packetiser has increased the capacity of the header information supported in the NIOSNIC from three to seven header bytes. This enables a message to pass though as many as six routers.
- Building a SoC can improve the performance of the OS-Link network because the propagation delay from a transmitter to the receiver has been reduced. This has enabled the data throughputs of 48 MBit/s and 47 MBit/s for single processing node loop back and processing node-to-processing node message passing respectively, which is very near to theoretical value of 48.48 Mbit/s.
- The use of NIOS II 'softcore' processor enables the construction of a multiprocessor embedded system on a single programmable chip. With current design and configuration of a processing node, the Stratix II EP2S60F672C5ES has the potential capacity to include four processing nodes and a NTR-FTM08 routing device alongside ancillary devices such as memory and I/O.

7.2 Future Work

7.2.1 Enhanced NIONIC's DMA Modules

The latest version of the Avalon bus specification[92] has suggested that the DMA module can be further improved and enhanced by supporting "Burst transfer". The Burst transfer executes multiple transfers as a unit, instead of treating every unit of data as an independent transfer as in the basic Avalon transfer. Using Burst transfer, the Avalon bus will guarantee an uninterrupted access to the target device, implemented on the same bus, for the duration of the burst. This will maximise the data throughput for each transfer between the involved bus master and slave.

Increasing the DMA buffer's data width, from 32bits to 64bits or 128bits, will improve the data throughput of the DMA modules. The Avalon switch fabric provides the dynamic bus sizing feature that manages the data transfer between master and slave ports

with different data bus sizes. All data will be aligned in contiguous bytes in the bus master's address space when dynamic bus sizing property is addressed. Having the data bus width increased, say 128bits, four times more data bytes can be accessed per transfer. Hence, the depth of the DMA buffer will need to be reduced to match the original capacity, with fewer transactions to fill the buffer. The shorter the memory access burst period, the faster the access request can be freed for other bus masters gain their request to the same memory module. Altering the DMA buffer's width will involve alterations on the Packetiser and Depacketiser modules to suit the new DMA buffer width. This will have the benefit that the packetiser/depacketised will work more efficiently. When transmitting a message for example, in 32bits bus DMA buffer, there will be one read to the DMA buffer and four writes to the Token buffer; using a 128bits DMA buffer, there will be one read to the DMA buffer and sixteen writes to the Token buffer.

7.2.2 Group Adaptive Routing

The NTR-FTM08 routing device provides the Group Adaptive Routing communication feature. It allows a processing node with more than one OS-Link based network interface controller to be connected to it. The group adaptive communication offers the benefit of an alternative path to the same destination when the dedicated path is busy or down. This will increase the communication bandwidth per processing node for nodes with more than one network interface controller. It is important to investigate the potential efficiency gains and influence on the traffic in the network.

7.2.3 Additional Communication Channels

To increase the available bandwidth for communication, additional communication channels can be added. Utilisation of the NTR-FTM08 Routing Device allows additional channels to be added to each processing node, as the Group Adaptive Routing Technique was implemented in NTR-FTM08 Routing Device; all the channels that interconnect the same processing node and router can be grouped. This will improve the bandwidth

utilisation for each processing node (router pair), allowing simultaneous multiple messages passing as well as improved tolerance to faults.

Additional communication channels for each processing node are possible without monopolising the Avalon Bus if the implementation of memory modules is planned carefully. For example, having two 25 kBytes on-chip RAM instead of a single 50 kBytes on-chip RAM, for the Avalon Bus would allow simultaneous access for multiple bus masters, provided they are not competing for the same target slave device. Of course, this would mean the use of extra logic cells to construct the arbiter for the extra memory module.

The overall performance of a processing node has to be re-evaluated when more communication channels are added. This is because additional communication channels mean an increase in the number of bus masters in the processing node that will compete for memory access.

7.2.4 Virtual Channels for NIOSNIC

Virtual Channels refers to automatic message channel allocation upon the arrival of a message. Devolving the memory address allocation function to the network interface controller allows the messages, with known message ID, to be allocated their storage address automatically (without involving the processor to suspend its task to process the new arrival). This operation, however, requires that the message ID and target memory address are pre-loaded into Context-Addressable-Memory (CAM). Virtual Channels are useful to handle situations when two or more different incoming messages arrive at the same time[109].

The FT-PCI-OSLI design was implemented on an APEK20K chip, which supported CAM. CAM was used as a 'search engine' which outputs an address for a message ID when the given message ID is matched to the incoming header. However, the

Stratix II family chip does not support the use of CAM; therefore Virtual Channels were abandoned in this design. The function of CAM might possibly be re-constructed with available registers and a small amount of RAM.

7.2.5 Realisation of the Embedded Distributed Multiprocessor System

The research concluded with the design and synthesis of the OSL-ST2, and then hardware was implemented on the Stratix II 2S60 development board. The OSL-ST2 was tested using loop back tests and message passing between processors, with and without the routing device. To enable the construction of an embedded distributed network, controlled by a Host PC, OSL-ST2 should be integrated together with FT-PCI-OSLi on a custom developed board. This would allow further research to experiment and access the effectiveness of the network as a whole and identify the possibility for further improvement. The OSL-ST2 design is ready for hardware implementation, the FT-PCI-OSLi and NTR-FTM08, have been successfully implemented in hardware in the previous research.

Another focus of effort in order to construct the embedded distributed multiprocessor system will be the software aspects of the system. Software drivers for use in the processing node are required. The software driver that supports the network interface should be simple and minimised to reduce the intervention of the processor, thus minimising the overhead and maximising the computing ability of each processing node of the parallel system.

Hardware realisation would also permit research into multi-router networks for various network topologies. Such experiments will show how network topology could influence traffic patterns and the overall performance of the parallel system. And finally it will also help to seek 'room for improvement' to suit any applications in the future.

7.2.6 Power efficiency investigation

Power considerations are often crucial for embedded system. The current design has not been optimised for power consumption and this could offer possibilities for further work.

Appendix A: Test Result

Single processor Test	number of loop	Byte	Pattern 1	Pattern 2	Pattern 3	Pattern 4	Pattern 5	Pattern 6	Pattern 7	Success
SDRAM read/write										
Write		4	x	x	x	x	x	x	x	x
		128	x	x	x	x	x	x	x	x
		512	x	x	x	x	x	x	x	x
		1 M	x	x	x	x	x	x	x	x
Read double Word		4	x	x	x	x	x	x	x	x
		128	x	x	x	x	x	x	x	x
Loop back Test										
Default message size	1	1	x	x	x	x	x	x	x	x
	10	1	x	x	x	x	x	x	x	x
	100	1	x	x	x	x	x	x	x	x
	1000	1	x	x	x	x	x	x	x	x
	10000	1	x	x	x	x	x	x	x	x
	100000	1	x	x	x	x	x	x	x	x
	1	256	x	x	x	x	x	x	x	x
	10	256	x	x	x	x	x	x	x	x
	100	256	x	x	x	x	x	x	x	x
	1000	256	x	x	x	x	x	x	x	x
	10000	256	x	x	x	x	x	x	x	x
	100000	256	x	x	x	x	x	x	x	x
	1	10000	x	x	x	x	x	x	x	x
	10	10000	x	x	x	x	x	x	x	x
	100	10000	x	x	x	x	x	x	x	x
	1000	10000	x	x	x	x	x	x	x	x
	10000	10000	x	x	x	x	x	x	x	x
	100000	10000	x	x	x	x	x	x	x	x
	1	100000	x	x	x	x	x	x	x	x
	10	100000	x	x	x	x	x	x	x	x
	100	100000	x	x	x	x	x	x	x	x
	1000	100000	x	x	x	x	x	x	x	x
	10000	100000	x	x	x	x	x	x	x	x
	100000	100000	x	x	x	x	x	x	x	x
	1	1000000	x	x	x	x	x	x	x	x
	10	1000000	x	x	x	x	x	x	x	x
	100	1000000	x	x	x	x	x	x	x	x
	1000	1000000	x	x	x	x	x	x	x	x
	10000	1000000	x	x	x	x	x	x	x	x
	100000	1000000	x	x	x	x	x	x	x	x

Table A-1: Single processor loop back message test with chosen test patterns.

Test PN-to-PN	number of loop	Byte	Pattern 1	Pattern 2	Pattern 3	Pattern 4	Pattern 5	Pattern 6	Pattern 7	Success
Default message size	1	1	x	x	x	x	x	x	x	x
	10	1	x	x	x	x	x	x	x	x
	100	1	x	x	x	x	x	x	x	x
	1000	1	x	x	x	x	x	x	x	x
	10000	1	x	x	x	x	x	x	x	x
	100000	1	x	x	x	x	x	x	x	x
	1	256	x	x	x	x	x	x	x	x
	10	256	x	x	x	x	x	x	x	x
	100	256	x	x	x	x	x	x	x	x
	1000	256	x	x	x	x	x	x	x	x
	10000	256	x	x	x	x	x	x	x	x
	100000	256	x	x	x	x	x	x	x	x
	1	10000	x	x	x	x	x	x	x	x
	10	10000	x	x	x	x	x	x	x	x
	100	10000	x	x	x	x	x	x	x	x
	1000	10000	x	x	x	x	x	x	x	x
	10000	10000	x	x	x	x	x	x	x	x
	100000	10000	x	x	x	x	x	x	x	x
	1	100000	x	x	x	x	x	x	x	x
	10	100000	x	x	x	x	x	x	x	x
	100	100000	x	x	x	x	x	x	x	x
	1000	100000	x	x	x	x	x	x	x	x
	10000	100000	x	x	x	x	x	x	x	x
	100000	100000	x	x	x	x	x	x	x	x
	1	1000000	x	x	x	x	x	x	x	x
	10	1000000	x	x	x	x	x	x	x	x
	100	1000000	x	x	x	x	x	x	x	x
	1000	1000000	x	x	x	x	x	x	x	x
	10000	1000000	x	x	x	x	x	x	x	x
	100000	1000000	x	x	x	x	x	x	x	x

Table A-2: Processing node to processing node message passing test with chosen test patterns.

Test Message Passing Test via router	number of loop	Byte	Pattern 1	Pattern 2	Pattern 3	Pattern 4	Pattern 5	Pattern 6	Pattern 7	Success
Default	1	1	x	x	x	x	x	x	x	x
message	10	1	x	x	x	x	x	x	x	x
size										
	100	1	x	x	x	x	x	x	x	x
	1000	1	x	x	x	x	x	x	x	x
	10000	1	x	x	x	x	x	x	x	x
	100000	1	x	x	x	x	x	x	x	x
	1	256	x	x	x	x	x	x	x	x
	10	256	x	x	x	x	x	x	x	x
	100	256	x	x	x	x	x	x	x	x
	1000	256	x	x	x	x	x	x	x	x
	10000	256	x	x	x	x	x	x	x	x
	100000	256	x	x	x	x	x	x	x	x
	1	10000	x	x	x	x	x	x	x	x
	10	10000	x	x	x	x	x	x	x	x
	100	10000	x	x	x	x	x	x	x	x
	1000	10000	x	x	x	x	x	x	x	x
	10000	10000	x	x	x	x	x	x	x	x
	100000	10000	x	x	x	x	x	x	x	x
	1	100000	x	x	x	x	x	x	x	x
	10	100000	x	x	x	x	x	x	x	x
	100	100000	x	x	x	x	x	x	x	x
	1000	100000	x	x	x	x	x	x	x	x
	10000	100000	x	x	x	x	x	x	x	x
	100000	100000	x	x	x	x	x	x	x	x
	1	1000000	x	x	x	x	x	x	x	x
	10	1000000	x	x	x	x	x	x	x	x
	100	1000000	x	x	x	x	x	x	x	x
	1000	1000000	x	x	x	x	x	x	x	x
	10000	1000000	x	x	x	x	x	x	x	x
	100000	1000000	x	x	x	x	x	x	x	x

Table A-3: Message Passing Test via Router with chosen test patterns.

	Single Processor Loop	Percentage Difference between Single Processor & Processor-to-processor	Processor to Processor	Percentage Difference between Processor-to-processor with and without router	Processor to Processor via Router
Byte(s)	Data Throughput(bit/s)	%	Data Throughput(bit/s)	%	Data Throughput(bit/s)
1	1.29E+07	1.6	1.27E+07	38	7.84E+06
10	3.54E+07	1.4	3.49E+07	11	3.09E+07
100	4.10E+07	3.4	3.96E+07	0.5	3.94E+07
1000	4.79E+07	1.3	4.73E+07	0	4.73E+07
10000	4.82E+07	1.9	4.73E+07	0	4.73E+07
100000	4.80E+07	1.5	4.73E+07	0	4.73E+07
1000000	4.80E+07	1.5	4.73E+07	0	4.73E+07

Table A-4: Data Throughput measurement for different test setup.

	Single Processor Loop	Processor to Processor	Processor to Processor via Router
byte	round trip time (ms)		
1	0.00062	0.00063	0.00102
10	0.00226	0.00229	0.00259
100	0.0195	0.0202	0.0203
1000	0.167	0.169	0.169
10000	1.66	1.69	1.69
100000	16.65	16.9	16.9
1000000	166.5	169	169

Table A-5: Result of Message Duration measurement for different test setup.

Payload	SDRAM		on-chip RAM	
	Transaction	Wait cycle	Transaction	Wait cycle
1	2	20	2	2
10	4	34	4	4
100	26	336	26	26
1000	251	2861	251	251
10000	2501	20414	2501	2501
50000	12501	95027	12501	12501
100000	25001	192281	-	-
1000000	250001	3579901	-	-

Table A-6: Clock Cycle Efficiency Test.

Appendix B: FPGA device Specification

Altera Stratix II family

This section shows part of the specification of Stratix II family devices. This Stratix II 2S60 is the target FPGA category used to develop ST2-OSL system. Further detail can be obtained from reference 47.

Feature	Device					
	EP2S15	EP2S30	EP2S60	EP2S90	EP2S130	EP2S180
Adaptive Logic Modules (ALMs)	6,240	13,552	24,176	36,384	53,016	71,760
Equivalent Logic Elements (LEs)	15,600	33,880	60,440	90,960	132,540	179,400
M512 RAM Blocks	104	202	329	488	699	930
M4K RAM Blocks	78	144	255	408	609	768
M-RAM Blocks	0	1	2	4	6	9
Total RAM bits	419,328	1,369,728	2,544,192	4,520,448	6,747,840	9,383,040
Phase-Locked Loops (PLLs)	6	6	12	12	12	12
Maximum User I/O Pins	366	500	718	902	1,126	1,170

Table B-1: Stratix II device family specification.

Note: Each ALM is equivalent to 2.5 Les.

On-chip RAM properties of Stratix II device

Feature	M512 block	M4K block	M-RAM block
Performance (MHz)	319	290	287
Total RAM bits (including parity bits)	576	4608	589824
Configuration	512 x 1	4K x 1	64K x 8
	256 x 2	2K x 2	64K x 9
	128 x 4	1K x 4	32K x 16
	64 x 8	512 x 8	32K x 18
	64 x 9	512 x 9	16K x 32
	32 x 16	256 x 16	16K x 36
	32 x 18	256 x 18	8K x 64
		128 x 32	8K x 72
		128 x 36	4K x 128
			4K x 144
Single-port memory	x	x	x
Simple dual-port memory	x	x	x
True dual-port memory		x	x
Memory initialization file (.mif)	x	x	
Mixed-clock mode	x	x	x

Table B-2: Specification of three supported on-chip RAM type in Stratix II device Family.

Excalibur device family

This section shows part of the specification of Excalibur devices family. The EPXA1 was used in XA1 prototype board. Further detail can be obtained from reference 44.

Feature	Excalibur		
	EPXA1	EPXA4	EPXA10
Processor	ARM922T	ARM922T	ARM922T
Maximum Operating Frequency (MHz)	200	200	200
Single-port SRAM (kBytes)	32	128	256
Dual-port SRAM (kBytes)	16	64	128
Typical gates	100000	400000	1000000
Logic Elements (LEs)	4160	16640	38400
Embedded System Blocks (ESBs)	26	104	160
Maximum User I/O	246	488	711

Table B-3: Excalibur devices Family specification.

Appendix C: Register of NIOSNIC

The following gives the descriptions for the NIOSNIC memory-mapped, 32 bit registers. The 'Base' is the base address assigned to the NIOSNIC in the processing node, which it is implemented.

Base + 0x00 Device ID

Bit	Read/Write	Description
31...0	R	This 32 bit register is an ID for NIOSNIC. The device ID is label as '0x2560E001'

Base + 0x04 Status Register

Bit	Read/Write	Description
31...2	R	Unused. Hardwired to '0'.
1	R	Label: Tx_mssg_end This bit indicate that the message has been transmitted
0	R	Label: Rx_mssg_end This bit indicates that the message has been received.

Base + 0x08 Received Header

Bit	Read/Write	Description
31...8	R	Unused. Hardwired to '0'.
7...0	R	Label: rx_rcvd_hdr This byte indicates the received Message ID from the new messge.

Base + 0x10 Receiver address

Bit	Read/Write	Description
31...2	R/W	Label: rx_addr_in These bits stores the address of the current memory location the device is pointing for memory write operation. The value increment by 1 after each successful data transfer.
1...0	R	Unused. Hardwired to '0'.

Base + 0x14 Receiver Length Register

Bit	Read/Write	Description
31...20	R	Unused. Hardwired to '0'.
19...0	R/W	Label: rx_mssg_lgth_out These bits store the length of the current DMA transfer. The value decreased by 4 after each successful data transfer. Writing to this bits will trigger the DMA receiver bus master DMA operation.

Base + 0x18 Receiver DMA Buffer Status

Bit	Read/Write	Description
31	R	Label: rx_dmaff_full 1' Indicating the Receiver DMA buffer is full. 0' Indicating the Receiver DMA buffer is not full.
30	R	Label: rx_dmaff_emp 1' Indicating the Receiver DMA Buffer is empty. 0' Indicating the Receiver DMA Buffer is not empty.
29...6	R	Unused. Hardwired to '0'.
5...0	R	Label: rx_dmaff_usedw These bits store the number of used word in the Receiver's DMA buffer.

Base + 0x1C Receiver Token Buffer Status

Bit	Read/Write	Description
31	R	Label: rx_osff_full 1' Indicating the Receiver Token Buffer is full. 0' Indicating the Receiver Token Buffer is not full.
30	R	Label: rx_osff_emp 1' Indicating the Receiver Token Buffer is empty. 0' Indicating the Receiver Token Buffer is not empty.
29...5	R	Unused. Hardwired to '0'.
4...0	R	Label: rx_dmaff_usedw These bits store the number of used word in the Receiver's Token Buffer.

Base + 0x10 Transmitter address

Bit	Read/Write	Description
31...2	R/W	Label: tx_addr_in These bits stores the address of the current memory location the device is pointing for memory read operation. The value increment by 1 after each successful data transfer.
1...0	R	Unused. Hardwired to '0'.

Base + 0x14 Transmitter Length Register

Bit	Read/Write	Description
31...20	R	Unused. Hardwired to '0'.
19...0	R/W	Label: tx_mssg_lgth These bits store the length of the current DMA transfer. The value decreased by 4 after each successful data transfer. Writing to this bits will trigger the DMA Transmitter bus master DMA operation.

Base + 0x18 Transmitter DMA Buffer Status

Bit	Read/Write	Description
31	R	Label: tx_dmaff_full 1' Indicating the Transmitter DMA buffer is full. 0' Indicating the Transmitter DMA buffer is not full.
30	R	Label: rx_dmaff_emp 1' Indicating the Transmitter DMA Buffer is empty. 0' Indicating the Transmitter DMA Buffer is not empty.
29...6	R	Unused. Hardwired to '0'.
5...0	R	Label: rx_dmaff_usedw These bits store the number of used word in the Transmitter DMA buffer.

Base + 0x1C Transmitter Token Buffer Status

Bit	Read/Write	Description
31	R	Label: tx_osff_full 1' Indicating the Transmitter Token Buffer is full. 0' Indicating the Transmitter Token Buffer is not full.
30	R	Label: rx_osff_emp 1' Indicating the Transmitter Token Buffer is empty. 0' Indicating the Transmitter Token Buffer is not empty.
29...5	R	Unused. Hardwired to '0'.
4...0	R	Label: rx_dmaff_usedw These bits store the number of used word in the Transmitter Token Buffer.

Appendix D: Avalon Bus Signal Descriptions for the NIOSNIC

Tx Bus Master

Signal	Width	Direction	Description
reset_n	1	in	Reset signal. When asserted, bus master must enter the deterministic reset state.
clk	1	in	Synchronising clock for Avalon bus master Interface.
address	32	out	Address lines from bus master to Avalon Switch Fabric
read_n	1	out	Read request signal from master port.
readdata	32	in	Datalines from Avalon Switch Fabric
waitrequest	1	in	Signal to force the bus master to wait until the Avalon Switch Fabric is ready to transfer data

Rx Bus Master

Signal	Width	Direction	Description
reset_n	1	in	Reset signal. When asserted, bus master must enter the deterministic reset state.
clk	1	in	Synchronising clock for Avalon bus master Interface.
address	32	out	Address lines from bus master to Avalon Switch Fabric
write_n	1	out	Write request signal from master port.
writedata	32	out	Data lines to Avalon Switch Fabric
waitrequest	1	in	Signal to force the bus master to wait until the Avalon Switch Fabric is ready to transfer data

OS-Link Bus Slave

Signal	Width	Direction	Description
reset_n	1	in	Reset signal. When asserted, bus master must enter the deterministic reset state.
clk	1	in	Synchronising clock for Avalon bus master Interface.
chipselect	1	in	Chipselect signal to the slave port. The bus slave will ignore other Avalon signal input unless chipselect is asserted.
address	7	in	Address lines from Avalon Switch Fabric. Specifies a word offset into the slave address space.
read_n	1	in	Read request signal from Avalon Switch Fabric.
readdata	32	out	Data lines to Avalon Switch Fabric for read transfer.
write_n	1	in	Write request signal from Avalon Switch Fabric.
writedata	32	in	Data lines from Avalon Switch Fabric for write transfer.

References

[1] Valiant, L.G., *General Purpose Parallel Architecture*, Technical report TR-07-89, April 1989, Aiken computation Laboratory, Harvard University, Prentice Hall.

[2] Jin, L, *Parallel Processing- Exploring the Architectures' and Algorithms' close relationship*, Potential IEEE Volume 13, Issue 5, Dec 1994- Jan 1995, Page 17-20.

[3] Fox, G.C., Johnson, M.A., Lyzenga, G.A., Otto, S.W., Salmon, J.K., Walker, D.W., *Solving Problems on Concurrent Processor, Volume 1 General Techniques and Regular Problems*, USA, Prentice Hall International Inc., 1988, Page 17-38.

[4] Richard M. Russell, *The CRAY-1 Computer System*, Communications of the ACM, January 1978, Pages 63-72.

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

[6] Rob I., *IP Delivery Key to Unlocking FPGA Potential*, ESE Magazine Nov/Dec 2005, page 36.

[7] ARM Ltd, *ARM Mobile*, available at http,//www.arm.com/markets/mobile_solutions Last Visit February 2007.

[8] Kunimatsu, A., et al, *Vector Unit Architecture for Emotion Synthesis*, IEEE Micro, Volume 20, Issue 2, March-April 2000, Page 40-47.

[9] Patterson D., *Reduced Instruction Set Computer*, Communication of the ACM, Vol.28, No.1, January 1985, Page 9-21.

[10] Jerraya, A., Wolf, W., *Multiprocessor System-On-Chips*, Elsevier Inc. Year 2005. ISBN: 0-12-385251-X.

[11] Jen-Hao, T., Chin-Yuan, T., Yu-Hung, C., *Integration of Network Embedded Systems into Power Equipment, Remote Control and Monitoring*, TENCON 2004. 2004 IEEE Region 10 Conference Volume C, 21-24 Nov 2004, Page(s) 566 - 569 Vol. 3.

[12] R.H. Day, R. Germon, and B. C. O'Neill, *A Pulse Compression Radar Signal Processor*, Conference on DSP Chips in real-time instrumentation and display system, IEE Colloquium, September 1997, pp4/1-4/5

[13] NTU, IC-Routing Ltd, *Electronic System Design and Parallel Processing Group*, available at http.//www.eee.ntu.ac.uk/research/parallel/index.html Last Visit February 2007.

[14] Wong K.L., *A Message Controller for Distributed Processing System*, PhD Thesis, Nottingham Trent University, April 2000.

[15] Robin Hotchkiss, *Integrated Fault Tolerance for Packet-Switch Networks*, PhD Thesis, Nottingham Trent University, October 2000.

[16] Hinton, J., Pinder, A., *Transputer Hardware and System Design*, Prentice Hall International (UK) Ltd, Year 1993, ISBN 0-13-953001-0(pbk), page(s) 1-17, page(s). 142-154.

[17] Ruth Ivimey-cook, *Legacy of the Transputer*, Architecture, Language and Techniques for Concurrent Systems; WoTUG-22 Proceeding of the 22nd World Occam and Transputer User Group Technical Meeting 11 -14 April1999, Keele, UK.

[18] Hoare, C.A.R., *Communication Sequential Processes*, Hemel Hempstead, Prentice Hall International, 1985.

[19] Foo, Y. W., Chong Y. K., *Performance Analysis of Parallel Processing in Local Area Network*, International Conference on Information, Communication and Signal Processing ICICS'97, Singapore. September 1997.

[20] Da Qing Zhang, Carlo Cecati and Enzo Chiricozzi, *Some Practical Issues of the Transputer Based Real-Time Systems*, Industrial Electronics, Control, Instrumentation, and Automation, 1992. 'Power Electronics and Motion Control', Proceedings of the 1992 International Conference on9-13 Nov. 1992, Volume .3Page(s) 1403 - 1407.

[21] Colin W.S., INMOS Limited, *Transputers- Past, Present and Future*, IEEE Micro DEC 1990.

[22] J. W. Ellis, B. C. O'Neill and S. Clark, *Performance of the NTR08 Routing Device in Transputers Networks*, - Transputer Applications and Systems '94, ed. A De Gloria, M R Jane and D Marani, IOS Press, 1994, ISSN 0925-4986, Vol. 41, pp958-965.

[23] K. M. Curtis, S. Wilde, B. C. O'Neill, J. W. Ellis, I. Jelly and D Lloyd, *A Dynamic Routing Strategy for Transputer Networks*, Transputer Applications and Systems '94, ed. A De Gloria, M R Jane and D Marani, IOS Press, 1994, ISSN 0925-4986, Vol. 41, pp235-246.

[24] J. W. Ellis, B. C. O'Neill and S. Clark, *A Router Design for T800 Compatible Transputer Arrays*, Transputer Applications, 1993, Pub. The Transputer Consortium, ISSN 0969-9341 Vol. 1(2) pp12-18

[25] J. W. Ellis, B. C. O'Neill and S. Clark, *The Realisation of a Hardware Routing Device for Transputer Arrays*, Fourth EUROCHIP Workshop on VLSI Design Training, Toledo, Spain, Sept. 1993 pp230-235.

[26] J. W. Ellis, B. C. O'Neill and S Clark, *Performance of a Routing Device for First Generation Transputers*, Poster presentation to The World Transputer Congress, Aachen, Germany, Sept. 1993.

[27] IC Routing Ltd., *16 Port Dynamic Routing Switch for Transputer Link*, Data Sheet Version 1.3, 1996, Page 1-3.

[28] E. W. K. Liew, D. Kaye, B. C. O'Neill and S. Clark, *Operating System Support for StrongARM Multi-Processor Communications*, Proceedings of the ISCA 12th International Conference on Parallel and Distributed Computing Systems, ISBN 1 880843 34 X, Aug 2000, pp 334-339.

[29] R Hotchkiss, B. C. O'Neill and S. Clark, *A Fault Tolerant Router for Parallel Network*, PREP 2000, ISBN 0 86341 3218, April 2000, IEE, pp 19-24.

[30] B. C. O'Neill, K. L. Wong, G. C. Coulson, R. Hotchkiss, J. H. Ng, S. Clark, P. D. Thomas and A. Cawley, *A Distributed Parallel Processing System for the StrongARM Microprocessor,* Concurrent Systems Engineering Vol. 52, ISBN 90-5199-391-9, April 1998, pp 39-48.

[31] IC-Routing LTD, *ICR C416 – 16 Port Packet Routing Switch for 20 MBit/s Serial Link Data Sheet*, Year 1996, available at http.//www.eee.ntu.ac.uk/research/parallel/docs/C416_DAT.DOC Last Visit February 2007.

[32] Coulson, G., *Optimisation of a Processing Farm Using Hardware* Routing, Transputer Application and Systems, IOS Press, 1995, Vol. 46, Page 70-77.

[33] T. Shanley, D. Anderson, *PCI System Architecture 3rd Edition*, Addison-Wesley Publishing Company, 1997, ISBN 0-201-40993-3.

[34] PCI Special Interest Group, *PCI Local Bus Specification Revision 2.2*, Dec 1998, available at http.//www.ics.uci.edu/~harris/ics216/pci/PCI_22.pdf Last visit February 2007.

[35] Digital Equipment Corporation, *SA-110 Microprocessor Technical Reference Manual*, Maynard, Massachusetts, USA, 1996.

[36] Kim, D., Managuli, R., Kim, Y., *Data cache and direct memory access in programming mediaprocessors*, Micro IEEE Volume 21, Issue 4, July-Aug. 2001 Page(s), 33 – 42.

[37] B. C. O'Neill, K. L. Wong, G. Coulson, R. Hotchkiss, J. H. Ng, S. Clark and P. D. Thomas, *An Interface Device to Support a Distributed Parallel System for the StrongARM Microprocessor*, High performance Computer Networks 98, Amsterdam April 1998 pp 1047-1050.

[38] R. Hotchkiss, B. C. O'Neill and S. Clark, *Fault Tolerance for an Embedded Wormhole Switched Network*, Parelec'2000, IEEE Computer Society proceedings, ISBN 0 7685 0759 X Aug 2000, pp 79-83.

[39] K. W. K. Liew, B. C. O'Neill, K. L. Wong, S. Clark, P. D. Thomas and R. Cant, *A Proposal for an Operating System for a Multi-Processor StrongARM System*, Concurrent Systems Engineering, ISBN 90-5199-480-X, Vol. 57, April, 1999, pp 37-47, April, 1999, pp 37-47.

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

[41] R. Hotchkiss, K. L. Wong, B. C. O'Neill, G. C. Coulson, S. Clark and P. D. Thomas, *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, Nevada, USA, ISBN 1-892512-07-6, July, 1998 pp1863-1870.

[42] Simon, T. *Interface for Embedded Parallel Multiprocessor Network*, PhD Thesis, Year 2002.

[43] Altera Ltd, *APEK20K device family Overview*, available at http,//www.altera.com/products/devices/apex/overview/apx-overview.html Last visit February 2007.

[44] Altera Ltd, *Excalibur Device Overview May 2002 Version 2.0*, available at http,//www.altera.com/literature/ds/ds_arm.pdf Last Visit February 2007.

[45] ARM Ltd, *ARM922TTM with AHB-System-on-chip Platform OS Processor product overview*, Year 2001, Rev 1, Document reference, ARMDVI0025B.

[46] Altera Ltd, *NIOS II Processor Reference Handbook*, Altera Ltd, document ID, NII5V1, last revised in November 2006, available at http,//www.altera.com/literature/hb/nios2/n2cpu_nii5v1.pdf Last visit January 2007.

[47] Altera Ltd, *Stratix II Device Family*, available at http://www.altera.com/products/devices/stratix2/st2-index.jsp Last Visit February 2007.

[48] IEEE, *VASG, VHDL Analysis and Standardisation Group*, available at http://www.eda.org/vhdl-200x/ Last update March 2003, Last Visit February 2007.

[49] Bhasker, J., *VHDL Primer*, 3rd Edition, Prentice-Hall, Inc. Year 1999.

[50] Altera Ltd, *Quartus II Software*, available at http://www.altera.com/products/software/products/quartus2/qts-index.html?f=cscsandk=t1 Last Visit February 2007.

[51] Altera Ltd, website available at http://www.altera.com/index.jsp Last Visit February 2007.

[52] Altera Ltd, *Quartus II Programmer Version 6.1*, available at https://www.altera.com/support/software/download/programming/quartus2/dnl-quartus2_programmer.jsp Last Visit February 2007.

[53] Altera Ltd, *NIOS II Integrated Development Environment*, available at http://www.altera.com/products/ip/processors/nios2/tools/ide/ni2-ide.html Last Visit February 2007.

[54] Brian C. O'Neil, Steve Clark and K. L. Wong, *Serial communication circuit with optimized skew characteristics*, IEEE Communications Letters, IEEE computer Society.

[55] *Myrinet Overview*, available at http://www.myricom.com/myrinet/overview Last Visit January 2007.

[56] Davide, B., Luca, B., *Xpipes: A Network-on-Chip Architecture for Gigascale Systems-on-Chip*, IEEE Circuits and System Magazine, Second Quarter 2004.

[57] B. C. O'Neill, P. W. Moore and S. Clark, *Inter-Processor Communications for a Distributed System*, Embedded System Show, May 2003, London.

[58] Wikipedia, *Symmetric Multiprocessing*, available at http://en.wikipedia.org/wiki/Symmetric_multiprocessing Last visit February 2007.

[59] William, Wong., *Basics of Design*, A supplement to Electronic Design, February 2006.

[60] Wikipedia, *Massive Parallelism*, available at http://en.wikipedia.org/wiki/Massively_parallel_processing Last visit February 2007.

[61] Arkadiy, M., Israel, C., Avinoam, Kolodny, and Ran, G., *Comparative Analysis of Serial Vs Parallel Links in NOC*, System-on-Chip, 2004. Proceedings 2004 International Symposium, 2004, Page(s) 185-188.

[62] Jianfeng, Z.,Nan C., Holm-Nielsen, P.V., Peucheret, C., Jeppesen, P., *Method for high-speed Manchester encoded optical signal generation*, Optical Fibre Communication Conference, 2004. OFC 2004, Volume 1, 23-27 Feb. 2004.

[63] Hotchkiss, R. *A Fault Tolerant Multicast Message Routing Switch for Interprocessor Communications*, PhD Thesis, The Nottingham Trent University, UK, September 2000, Page 14.

[64] Remarklus, W., *On-Chip Bus Structure for Custom Core Logic Design*, IEEE Wescon, Page 714, 1998. Digital Object Identifier 10.1109/WESCON.1998.716402

[65] Aldworth, P., *System-On-Chip Bus Architecture for Embedded Application*, IEEE International Conference on Computer Design, Page 297-298, 1999.

[66] Cordan, B., *An Efficient Bus Architecture for System-On-Chip Design*, IEEE Custom Integrated Circuit Conference, Page 623-626, 1999.

[67] Winegarden, S., *A Bus Architecture Centric Configurable Processor System*, IEEE Custom Integrated Circuit Conference, Page 627-630, 1999.

[68] Zhang, X, *Performance measurement and modelling to evaluate various effects on a shared memory multiprocessor*, Software Engineering, IEEE Transactions on, Volume 17, Issue 1, Jan. 1991 Page(s),87 – 93.

[69] Duato, J., Yalamanchili, S. and Ni, L., *Interconnection Networks, An Engineering Approach*, Morgan Kaufmann, 2003.

[70] P. Kermani, L. Kleinrock, *Virtual Cut-Through, A New Computer Communication Switching Technique*, Computer Network Vol. 3, page 267-286, September 1979.

[71] Mithuna, Thottethodi., Alvin, R., Shubhendu. S., *BLAM, A High-Performance Routing Algorithm for Virtual Cut-Through Networks*, Proceeding of the International Parallel and Distributed Processing Symposium (IPDPS'03) Year 2003.

[72] L. M. Ni., P. K. McKinley, *A Survey of Wormhole Routing Technique in Direct Networks*, IEEE Computer, 26(2) page 62-76, February 1993.

[73] Katevenis, M., *Buffer Requirements of Credit Based Flow Control when A Minimum Drainning Rate is Guanteed*, High Performance Communication System, 1997, The Forth IEEE Workshop on 23-25 June 1997 Page 168 – 178.

[74] Bashir, M., Al-Hashimi, *System-on-Chip, Next Generation Electronics*, The Institution of Electrical Engineers, London, 2006, Page 605 – 607, ISBN, 0-86341-552-0.

[75] B. C. O'Neill, *Parallel Network Solution, From Academia into Industry*, invited presentation on The Role of Physicists in Building the Internet conference, the Institute of Physics Annual Congress, March 2000, PBI.8.3.

[76] B. C. O'Neill, *Routing Hardware Design*, Invited presentation to ESA Spacewire working group, Netherlands, Feb 2000.

[77] K. L. Wong., *A Message Controller for Distribute Processing System*, PhD Thesis, Nottingham Trent University, UK, June 2000, Page 27.

[78] Altera Ltd, *Content Addressable Memory definition*, available at http://www.altera.com/support/software/nativelink/quartus2/glossary/def_cam.html Last visit December 2006.

[79] Altera Ltd, *Implementing High Speed Search Application with Altera CAM*, Application note 119, Version 2.1, July 2001 available at http://www.altera.com/literature/an/an119.pdf Last visit February 2007. Last visit February 2007.

[80] MYRICOM Inc., *Myrinet Link Specification*, Archived specification available from Myricom Inc. USA, at http://www.myri.com/scs/documentation/link/index.html Last visit October 2006.

[81] Bernie Perrin, Lattice Semiconductor, *Intellectual Property*, Embedded System Engineering, Volume 13, 8 Nov/Dec 2005.

[82] Jeff Moris, Andy Martwick, Brad Hostler, *PHY Interface for the PCI Express™ Architecture*, Draft Version 1.87 Intel Corporation, 2005, available at http://www.intel.com/technology/pciexpress/devnet/docs/pipe1_87.pdf Last visit February 2007 Last visit February 2007.

[83] JEDEC Standard, *Double Data Rate (DDR) SDRAM Specification*, JEDEC Solid State Technology Association, Revision JESD79E, May 2005, available at http://www.jedec.org/download/search/JESD79E.pdf Last visit February 2007 Last visit February 2007.

[84] Altera Ltd, *NIOS 3.0 CPU*, Version 2.2, October 2004, available at http://www.altera.com/literature/lit-nio.jsp Last visit January 2007.

[85] Xilinx, *MicroBlaze Processor Reference Guide – Embedded Development Kit EDK 8.2i*, Xilinx Inc, June 2006, available at http://www.xilinx.com/ise/embedded/mb_ref_guide.pdf Last Visit January 2007.

[86] ARM Ltd, *ARM922T Technical Reference Manual Rev 0*, ARM Limited 2000, available at http://www.arm.com/pdfs/DDI0184B_922T_TRM.pdf Last visit February 2007. Last visit February 2007.

[87] Altera Ltd, *NIOS II Cores Implementation Details*, Document reference NII51015-6.1.0, November 2006, available at http://www.altera.com/literature/hb/nios2/n2cpu_nii51015.pdf Last visit January 2007.

[88] IBM, *On-chip Peripheral Bus Architecture Specification Bus*, Version 2.1 Document reference SA-142528-02, available at http://www-306.ibm.com/chips/techlib/techlib.nsf/techdocs/9A7AFA74DAD200D087256AB30005F0C8/$file/OpbBus.pdf Last visit February 2007.

[89] ARM Ltd, *AMBA™ Specification Rev. 2.0.*, can be requested from, www.arm.com.

[90] Ackland, B. Anesko, A., Brinthaupt, D. *A Single-Chip, 1.6-Billion, 16-b MAC/s Multiprocessor DSP*, IEEE Journal of Solid-State Circuit, Vol. 35, No. 3 March 2000.

[91] Altera Ltd, *System Interconnect Fabric for Memory-mapped Interface*, Altera Ltd. Document reference, QII54003-6.1.0 available at http://www.altera.com/literature/hb/qts/qts_qii54003.pdf Last Visit, January 2007.

[92] Altera Ltd, *Avalon Memory-Mapped Interface Specification Ver3.2*, Altera Ltd. Document reference, MNLAVABUSREF-3.2. Available at http://www.altera.com/literature/manual/mnl_avalon_spec.pdf Last Visit January 2007.

[93] Altera Ltd, *Avalon Bus Specification Reference Manual Version 2.3*, July 2003, available at http://www.altera.com.cn/literature/manual/mnl_avalon_bus.pdf Last visit February 2007.

[94] IBM, Sony and Toshiba, *The Cell Architecture*, available at http://www.research.ibm.com/cell/home.html Last Visit 26 October 2006.

[95] E. W. K. Liew, B. C. O'Neill and S. Clark, *Porting Transputer Application to Multi-Processors StrongARM system,* Workshop on Parallel and Distributed

Computing in Image Processing, Video Processing, and Multimedia (PDIVM'2001) under the International Parallel and Distributed Processing Symposium (IPDPS 2001),San Francisco, US, April 23rd, 2001, sponsored by IEEE Computer Society.

[96] J. H. Ng, B. C. O'Neill and S. Clark, *A PC Interface Board for Parallel ARM Processor Network*, PREP 2000, ISBN 0 86341 3218, April 2000, IEE, page(s) 469-474.

[97] Fook, O., *Fault Detection for A Customised Multiprocessor Network*, MPhil Thesis, Year 2006.

[98] Altera Ltd, *Embedded Software Development Tool,* available at http://www.altera.com/products/ip/processors/nios2/tools/ni2-development_tools.html Last visit 15 December 2006.

[99] Altera Ltd, *NIOS II Software Developer's Handbook*, Version NII5V2-6.0. available at http://www.altera.com/literature/hb/nios2/n2sw_nii5v2.pdf Last visit February 2007.

[100] Altera Ltd, *NIOS Development Board Stratix II Edition Reference Manual*, available at http://www.altera.com/literature/manual/mnl_nios2_board_stratixII_2s60_rohs.pdf. Version 1.2, October 2006 Last visit 09/10/2006.

[101] Suh, J., Yoo, Hoi-Jun., *Arbitration Latency Analysis of the Shared Channel Architecture for High Performance Multi-Master SoC*, IEEE Asia-Pacific Conference on Advanced System Integrated Circuit, August 4 – 5, 2004.

[102] Altera Ltd, *Quartus II Version 6 Handbook Volume 4, SOPC Builder*, available at http://www.altera.com/literature/lit-qts.jsp Last visit 06 November 2006.

[103] J. W. Ellis., *A Hardware Routing device for Transputer Arrays*, PhD Thesis, year 1995.

[104] Altera Ltd, *Stratix II Power Play Early Power Estimator*, available at http://www.altera.com/support/devices/estimator/st2-estimator/st2-power-estimator.html Last Visit 12 Dec 2006.

[105] Allan Cantle, *Into The Fastlane*, IET Engineer and Technology, January 2007, Page 36 – 39.

[106] Bryan Betts, *Inefficient Ethernet Waste over $1 billion a Year*, available at http://www.theregister.co.uk/2007/02/05/ethernet_energy/ Last visit February 2007.

[107] Ethernet, http://www.ethernetalliance.org/home Last Visit February 2007.

[108] Mike, B., Ken, C., Bruce, N. *Improving the Energy Efficiency of Ethernet, Adaptive Link Rate Proposal*, Version 1.0, July 2006, Page 4 Available at http.//www.ethernetalliance.org/technology/white_papers/alr_v10.pdf Last visit February 2007.

[109] Simon, T., *Interface for Embedded Parallel Multiprocessor Network*, PhD Thesis, Year 2002. Page 81-82.