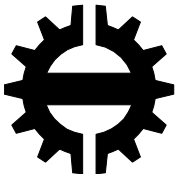


TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

ADNAN SALEEM

Implementation and Performance Analysis of Wishbone Shared Bus for Single Master-Multiple Slaves

Master of Science Thesis

Examiner (s):
Prof. Timo D. Hämäläinen
Erno Salminen

PREFACE

This Master of Science thesis work, "Implementation and Performance Analysis of Wishbone Shared Bus for Single Master-Multiple Slaves" has been written to complete my M.Sc. degree in Department of Pervasive Computing at Tampere University of Technology (TUT), Tampere, Finland.

First I would like to pay my deepest gratitude to my mentor and supervisor Manuel vor dem Brocke from TRUMPF Hüttinger GmbH + Co. KG. I want to thank him for his support and guidance in my thesis work at the company. I would also like to thank TRUMPF Hüttinger GmbH for hiring and trusting me for this thesis opportunity. Secondly, I would like to express my deep appreciation to my examiners Timo D. Hämäläinen and Erno Salmainen for their help in examining my work.

I would also like to thank my parents, siblings and fiancée. Their support, love and encouragement gave me strength and motivation to complete my studies.

Finally, I would also like to thank all my friends in Finland especially Aitzaz Haider Kazmi, Muhammad Ali Zaib, Hasib Raja, Mukesh Kumar, Naeem Tahir, Muhammad Qutab-ud-din, Syed Ameen-ur-Rehman, Usama Mazhar, Waqas Ansari and Zohaib Hassan for their encouragement and all the support. I would like to dedicate my thesis work to my family and friends.

Tampere, April, 2015
Adnan Saleem

ABSTRACT

ADNAN SALEEM: Implementation and Performance Analysis of Wishbone Shared Bus for Single Master-Multiple Slaves
TAMPERE UNVIERSITY OF TECHNOLOGY
Master of Science Thesis, 55 pages
Month and year of completion: April 2015
Master's Degree in Information Technology
Major: Digital and Computer Electronics
Examiner(s): Prof. Timo D. Hämäläinen, Erno Salminen

Keywords: Bus Architecture, FPGA, On-chip interconnection, On-chip interconnection comparison, SoC, Wishbone Shared Bus, Xilinx

System on Chip interconnections are gaining importance as many IP cores are being integrated on a single chip and interconnect is the bottleneck for design speed. In this paper an asynchronous design comprised of single master and multiple slaves connected via point-to-point topology is analysed. This design resulted in large multiplexer, poor timing closure and consumed large interconnect area in FPGA. The aim of the thesis is to evaluate the system on-chip interconnections and implement the system with the synchronous shared bus interconnection. Many system-on-chip interconnections are reviewed in the thesis, which includes study of major types of buses from different vendors. Synchronous shared bus system is proposed as solution for the interconnections between single master and multiple slaves. Shared bus for the single master and multiple slaves is implemented using WISHBONE architecture and protocols for shared bus system. A general model is designed and implemented which is flexible to be tested for single master and any number of slaves. Performance evaluation is done for the design in terms of resource utilization and timings performance.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

MPSoC	Multiprocessor System-on-Chip
IP	Intellectual Property
DMA	Direct Memory Access
FPGA	Field Programmable Gated Array
NoC	Network on Chip
SoC	System on Chip
VHDL	Hardware Descriptive Language
AMBA	Advance Microcontroller Bus Architecture
TDMA	Time division multiplexed access
RAM	Read Access Memory
LUT	Look up Table
CLB	Configurable Logic Block

TABLE OF CONTENTS

1. INTRODUCTION

Advancement in technology has made it possible to integrate millions of transistors on a single chip. To match the pace of integration, advancement and shrinking process technologies, designers and manufacturers are now trying to put the different Intellectual Property (IP) cores, general purpose processor cores, digital signal processor cores, on-chip memory and other application specific circuits on a single chip known as Multiprocessor System-on-Chip (MPSoC) for a specific application. [1] Figure *1-1* shows an example of a small MPSoC, which has two ARM9 microprocessors, on-chip memories, Direct Memory Access (DMA), peripherals and external interfaces which are interconnected by on-chip bus architecture consisting of multiples shared buses. [2]

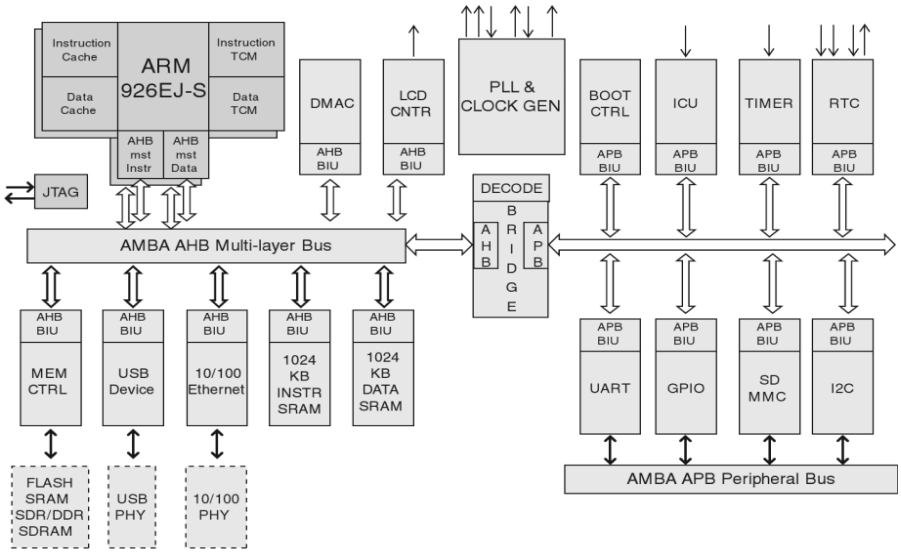


Figure 1-1 *Example of MPSoC [2]*

It is the responsibility of the on-chip architecture to make sure that data is routed correctly, reliably and efficiently from the source to the destination. It must also provide the required latency or bandwidth to meet the performance requirement. [2] These developments pose some challenges like Area/resource consumption, delay and power dissipation [3] but the significant challenge for the designers is on-chip communication architecture [4] amongst the modules. As the emphasis on design re-use is increasing, designing an interconnection is getting more complicated and complex [5].

FPGA has in abundance, Programmable switches and programmable interconnect like local and global wire segments. Interconnection architectures which are built on top of

these elements should use them efficiently to improve system performance and make the system scalable. [6]

Several methods like point-to-point communication, bus communication and Network on Chip (NoC) are used to provide interconnection solutions but each has its own advantages and disadvantages in terms of resource consumption, area, power dissipation, latency and throughput. On-chip communication is essential to provide high bandwidth and reliable data transfer between the processing modules and is directly related to the performance of the SOC. The bandwidth, latency and throughput plays significant role in deciding the overall performance of the system with increasing number of communicating components. [6] This increases the importance to improve the communication architectures for better performance.

Many leading manufacturers like IBM [7], ARM [8] and ALTERA [9] are providing bus based interconnect solutions.

1.1. Objective

The goal of the thesis is to replace the asynchronous point-to-point interconnection of the design with suitable synchronous on-chip shared bus system. It includes the analysis, implementation and performance evaluation of the shared bus system with single master and multiple slaves. Simplicity, scalability, resource consumption and throughput are the constraints of the system.

1.2. Thesis outline

Thesis is organized as follows Chapter 1 has introduction, objective and outline of the thesis. Chapter 2 contains the theoretical background of the topic which contains the three types of interconnections which are point-to-point, bus interconnection and Network on Chip. There advantages, disadvantages and types are discussed. Types of buses are explained in more detail as they are under main focus. Chapter 3 contains the analysis of the interconnections. Chapter 4 contains the implementation of WISHBONE bus [10] architecture for a single master into many slaves. WISHBONE architecture signals and rules are explained in detail. Chapter 5 contains the results and discussion. Results are taken from the implementation from Chapter 4. Finally the Chapter 6 contains the conclusion of the overall thesis. It explains the reasons for selecting, implementing and testing a particular architecture and also proposes the possible future work

.

2. THEORETICAL BACKGROUND

This chapter develops the foundations for the key concepts used in this thesis. It discusses the theoretical details of these concepts. This chapter gives in detail explanation of the on-chip interconnects like point-to-point interconnect, BUS interconnect, Network on Chip (NoC) interconnect. It also includes different types of buses by companies like ARM [8], IBM [7] and ALTERA [9]. The chapter also covers the basic types of NoC and its implementation details. At the end, the chapter contains the summary of all type of interconnections.

2.1. Field Programmable Gate Array (FPGA)

FPGA is a special kind of Integrated Circuit (ICs), which can be programmed after manufacturing. The design is implemented using Hardware Description Language (HDL) such as VHSIC (Very High Speed integrated Circuit) Hardware Description Language (VHDL) or Verilog. It doesn't contain any pre-determined functionality but it can be programmed to any digital design specific to the required application. It contains programmable logic blocks for implementing logic functions, programmable routing paths with switches to connect these logic blocks and I/Os as seen in Figure *2-1*. The logic blocks and interconnects are programmable interconnects. Current FPGAs also contain some Hard Cores [12] like memory, DSPs and processors. Processors can also be implemented as Soft Core [12] on FPGA fabric using logic blocks and making the design bigger and complex. [13]

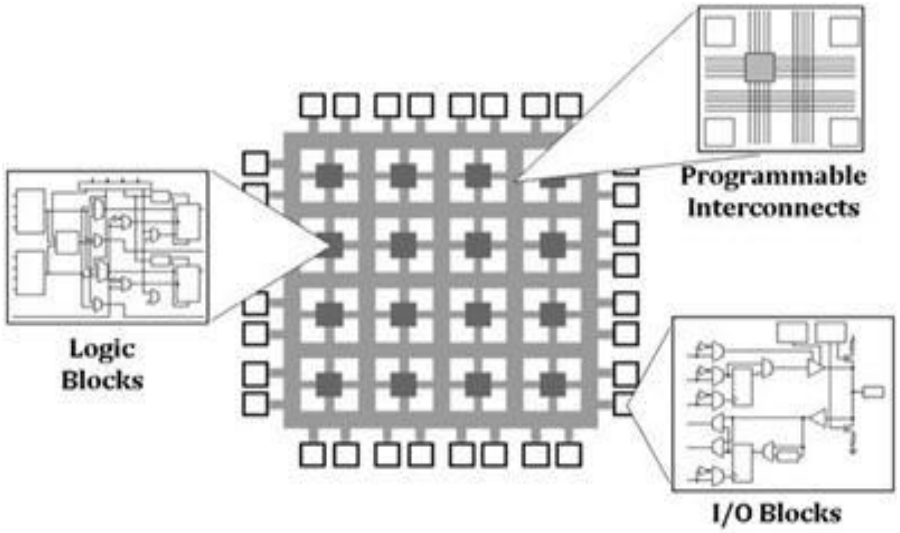


Figure 2-1 *FPGA Structure [13]*

2.1.1. FPGA Interconnection

Figure *2-2* shows the FPGA interconnection architecture. FPGA interconnects takes the 90% area of the total area while the logic blocks takes 10% of the total area. This interconnect plays major role in delay and area efficiency of the architecture. [14] Logic Blocks combine to form Intellectual Property cores (IP) like DSP, processor and memories. As the complexity of the circuit increases and more IP cores are put on FPGA, interconnect also becomes large and complex and the routing architecture becomes more important. Optimal use of this structure is important otherwise it will result in poor routing, speed or density and efficiency of the system decreases. [15] [16]

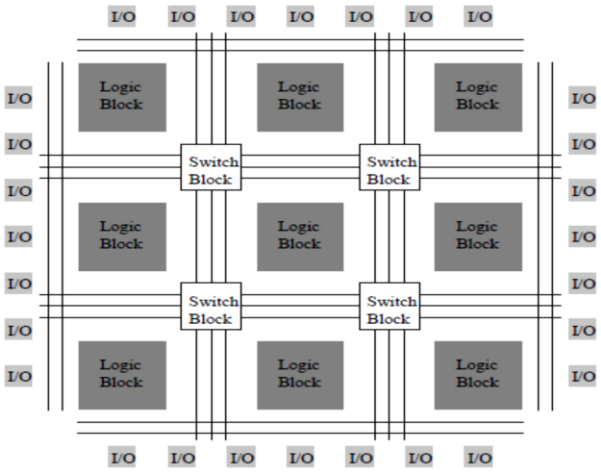


Figure 2-2 *FPGA Interconnect [13]*

2.2. On-Chip Interconnections

Communication architecture amongst the different processing elements on a chip is called on-chip interconnection. Interconnection is nothing but physical set of wires which can connect the source and destination modules to each other by following different protocols and topologies. Topology is the way in which processing elements are connected to each other and the protocol means the way in which they communicate with each other.

TYPES OF INTERCONNECTIONS

Typical on-chip interconnections are point-to-point interconnection, bus based interconnection and network-on-chip. Figure *2-3* shows the types of on chip interconnections. Each type is explained in detail below.

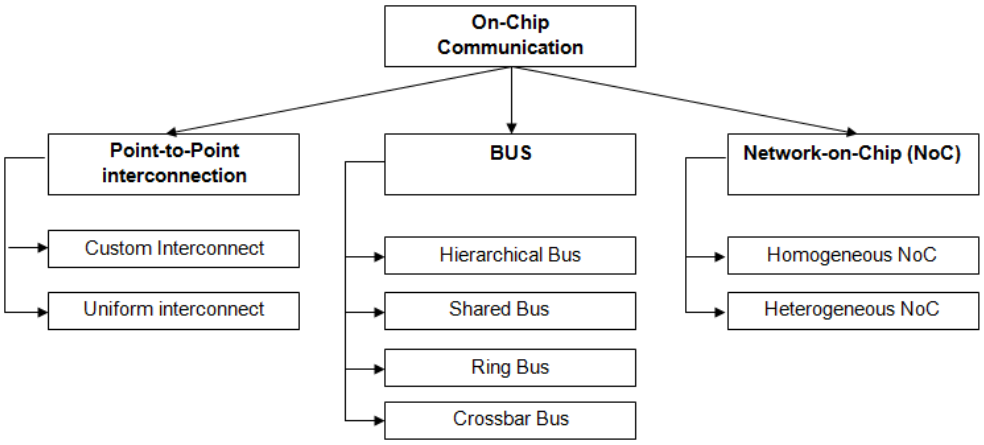


Figure 2-3 *Types of Interconnects*

2.2.1. Point-to-Point Interconnect

In Point-to-point interconnect, the two modules or IP cores communicate with each other over a dedicated communication channel. This interconnect needs a set of dedicated wires according to the application to connect the modules. Figure *2-4* shows a typical example of neural networks where the communication link between two processing elements is dedicated. There are two types of point-to-point interconnect; Custom interconnect and uniform Interconnect. In Custom interconnect, interconnect is designed and processing elements are connected according to the need of the application while the uniform interconnect is well defined interconnect which can be explained with graphs or equations.

Point-to-point communication stands amongst the simplest interconnection architectures and because of its simplicity it is deployed extensively in many applications worldwide [1]. Point to point communication for system having few components can run effectively with such light weight architecture. As there will be no sharing of the channel, its latency and performance can be calculated. [6]

The communication channel between two nodes is a dedicated set of wires so as the number of nodes increases the use of wires increases exponentially and the routing becomes more difficult. Point-to-point scheme suffers from low wire usage efficiency for low bandwidth channels and also pose a high usage of hardware. In terms of scalability, the point-to-point interconnect is not a good choice. As the complexity of the systems is increasing, the requirements like low area overhead and higher bandwidth are demanding more efficient interconnection architectures. [6] [1]

Speed performance of design depends upon the IP/Module internal delays in addition with the propagation delay through the interconnect. Latency and throughput of the design mainly depends upon the two communicating modules. In case of some shared medium, latency and throughput will also depend upon the arbitration technique. There is no extra arbitration logic required to connect the two modules

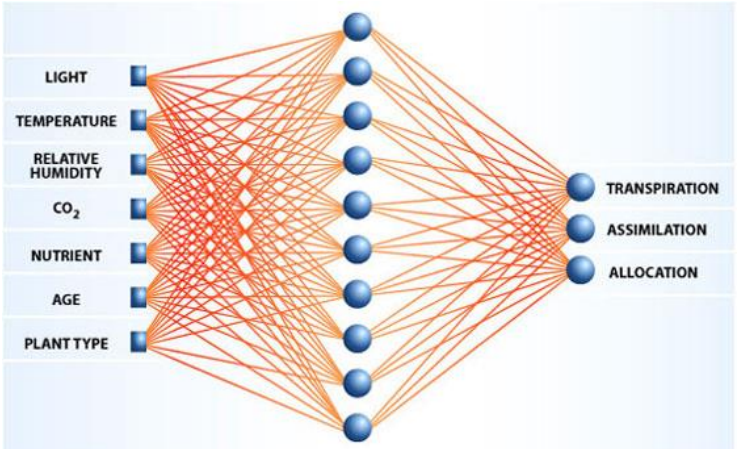


Figure 2-4 *Point to Point Interconnect*

2.2.2. Bus Interconnect

Bus is a single physical communication channel in which wires are grouped to form a single medium and shared amongst multiple entities. Each module communicates with other on the same physical interconnection. Typically one entity acquires the bus ownership (known as bus master) and places signal on the bus while the other entities response to the master. Bus can be synchronous in which transmission starts and ends on the edge of a clock while in asynchronous bus transmission starts and stops depending upon the acknowledgement signals. Figure *2-5* shows an example of a shared bus. An arbiter is deployed to control the access of the bus between the nodes. On the other hand, in the point-to-point interconnect each module communicates with the other module on a dedicated path. Bus architecture clearly reduces the number of wires and also reduces overall area needed for communication and control [1].

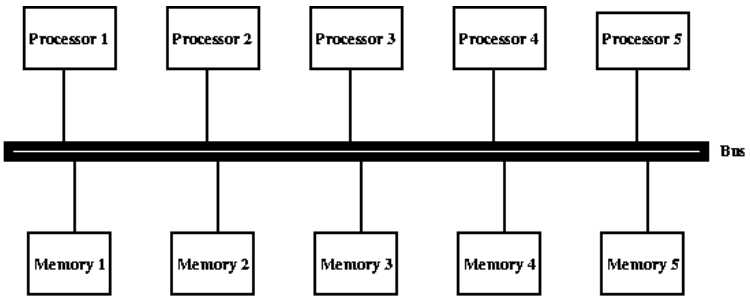


Figure 2-5 *Shared BUS*

Bus is widely accepted communication architecture amongst System-on-a-Chip (SoC) architectures and currently on-chip networks are made with buses. Bus based interconnection network provides design reuse and flexibility to the SoC designers. Designing teams focusing on future modules can easily design the newer modules around the current standard bus. As more processing elements are deployed on a single chip, interconnection network is also gaining importance. [1]

BUS ARCHITECTURE

Bus architecture has two major parts; bus topology and bus protocol.

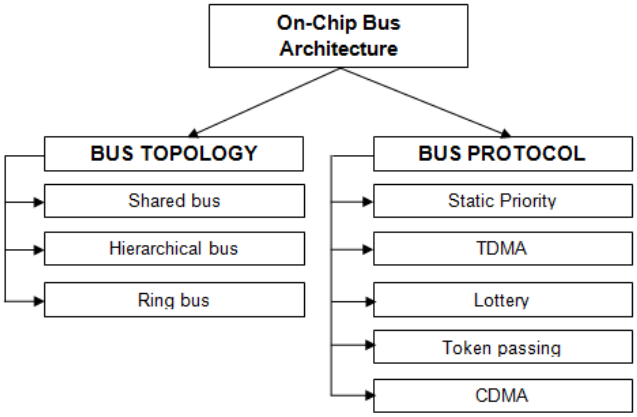


Figure 2-6 *Parts of Bus Architecture*

BUS TOPOLOGY

Bus based architecture has different types of bus topology where topology of the bus is the physical arrangement of the bus. It plays key role to determine cost, complexity, power and performance of the architecture. [2] The simplest scheme is the single shared bus, in which all components are connected to a single shared bus. A more efficient topology is the hierarchical bus, in which multiple shared buses are connected in parallel using bridges while allowing parallel data transfer. For further high performance systems full bus crossbar topology is used. In this topology each Master is connected to each slave with point-to-point interconnection providing superior parallel response. Finally Ring bus topology is used as high performance bus topology, in which components are connected in the form of a ring. [17]

o SHARED BUS

Shared bus is in which multiple entities share a common medium for communication. Figure *2-7* shows an example of a shared bus. Single/Multiple master and Single/Multiple slaves are connected to a shared bus and can communicate with each other. The arbiter in case of multiple masters examines the requests from multiple masters and gives access to a specific master on the basis of the protocol defined in arbiter. In case of a single master arbiter can be skipped. There are several advantages of a shared bus like the simplicity, extensibility, low area, easy to build and easy to manage. [1] [6]. As the number of masters on a shared bus increases its latency increases because bus is unavailable for transfers when it's busy in serving one master. There also disadvantages of a shared bus which includes high energy consumption, latency and low bandwidth when load is high. Such configuration is good for small sized SoC having few components and it doesn't scale well to handle large systems [18]. A system bus in computer is an example of a shared bus.

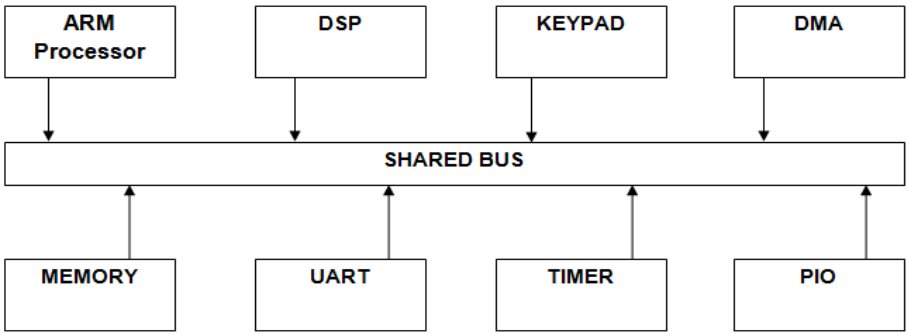


Figure 2-7 *Shared Bus*

Shared bus topology can be multiplexed or non-multiplexed. Multiplexed bus is when data and address share same set of signal lines. This reduces the number of interconnections but also increases the latency. In non-multiplexed data and address has separate signal lines which increases the number of interconnection but can be moved in as little as one clock cycle and latency can be decreased.

There are two types of bus interconnection; Tri-state based or multiplexer based. Tri state interconnection has tri state buffers. Tri state buffers give three logic levels as output which are "OFF", "ON" and "HIGH IMPEDANCE" and high impedance means logical disconnection in the circuit. This interconnection works in the way when one MASTER(Sender) wants to send the data it turns its buffer "ON" while the rest of the MASTERs turn their buffer to high impedance. Similarly the SLAVE (Receiver) which wants to read the data turns its buffer ON. This approach has two drawbacks. It is slower than direct interconnection because switching buffer ON and OFF has some minimum timing limit which should be met [10]. Second, three state buffers are not available on all the devices like Xilinx Spartan-6 XC6SLX45T. Figure *2-8* shows the two masters and two slaves connected via four three state buffers shown as arrows on wires. While the other is multiplexer based interconnection; its advantage is that it can be implemented in any device unlike tri-state based interconnection, its disadvantage is it requires large number of logic gates and large number of routed interconnects. Figure *2-9* shows the multiplexor based interconnection in which each module gets the access to the bus via control signal and multiplexor.

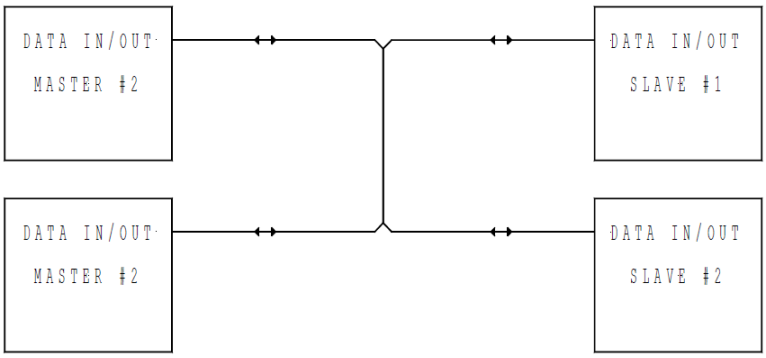


Figure 2-8 *Tri-State interconnection [10]*

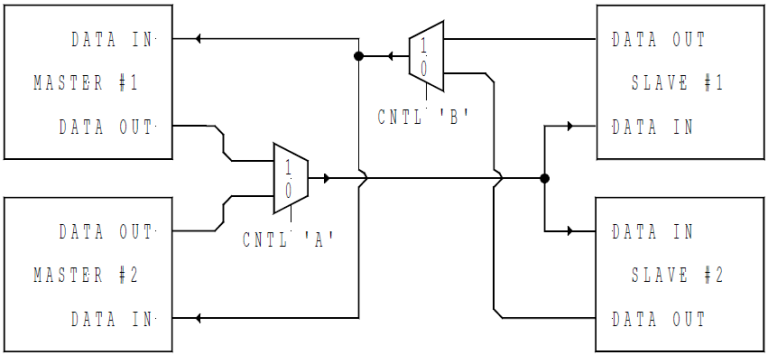


Figure 2-9 *Multiplexer Interconnection [10]*

- **HIERARCHICAL BUS**

This architecture is made of several shared buses connected to each other through bridges. Modules are placed at a level according to their requirement. High performance components are placed on high performance buses while the low ones on low performance bus. This system keeps the two levels separately in which the low performance modules don't load the high performance buses. Communication between the buses having different performance occur through bridges which also pose some overhead and latency. Multiple data transfers are possible in parallel. Buses can have different clock frequencies so bridge can get more complex for transactions amongst the buses [2]. *Figure 2-10* shows the example of a hierarchical bus where three buses are connected to each other via bridges. Common example of such an architecture is AMBA [8] and CoreConnect Bus [7].

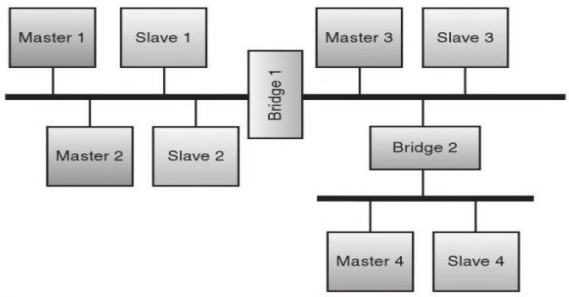


Figure 2-10 *Hierarchical Bus [2]*

- **CROSSBAR BUS**

A crossbar circuit takes N inputs and connect each of them to the M outputs. *Figure 2-11* shows the crossbar bus with master and slaves. At each wire intersection there is a pass transistor which is called crosspoint connector which short circuit two wires to make a connection. Each wire is a bus with unique master. Centralized arbiter controls the crosspoint connector and depending upon the input and output requests it makes connections. An example of crossbar bus topology SoC is Niagra Multiprocessor SoC [17]. This bus is used for high performance system which needs parallel data transfer

while the solution can be excessive for smaller systems [19]. It takes large area, more power and to achieve routing closure is impossible [14].

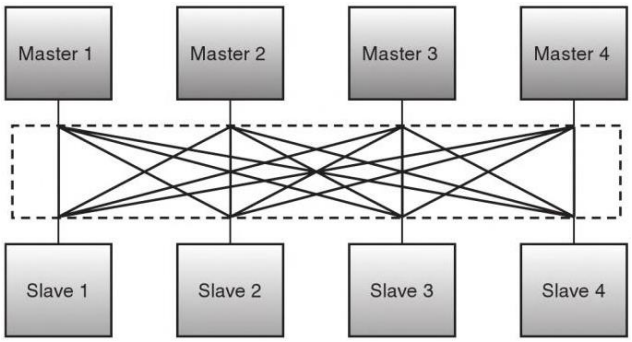


Figure 2-11 *CrossBar Bus [2]*

- **RING BUS**

In a Ring bus topology each node communicate using an interface made in a ring manner and the protocol used is token pass protocol. A token is passed to each entity and if the entity wants to access the bus it can keep the token till the time it is communicating and then it passes the token to the next entity. Data is transferred in clockwise or anti-clockwise direction. It is also a high performance ring topology. It takes less area while provides reasonable bandwidth [2].

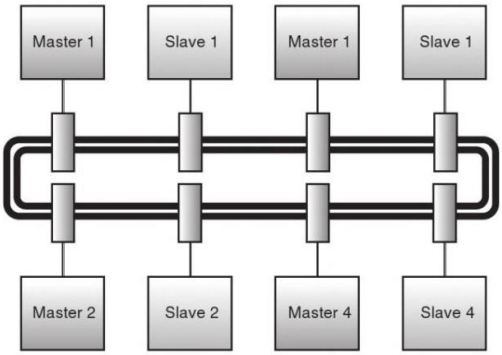


Figure 2-12 *Ring Bus [2]*

BUS PROTOCOL

Bus arbitration protocols are the set of rules, in which the transaction occur on the channel. Arbitration protocols allow the master to access the bus and transmit. Entities send their request to transmit to the central arbiter and arbiter depending upon the implemented protocol allows the access or reject. It includes arbitration mechanisms like Round robin access, Time division multiplexed access or Priority based access protocols. Buses use several communication protocols for channel sharing and resource allocation.

- **STATIC PRIORITY**

Typically shared bus architecture uses this protocol, in which a central arbiter gives access to the masters on the list waiting for access. Priority number is given to each connected master and the master having the highest priority gets the first access and so on.

Transactions can be pre-emptive or non-pre-emptive. AMBA and CoreConnect Bus uses this protocol [8] [7].

- **ROUND ROBIN PROTOCOL**

This protocol works as follows; each entity can access the shared media on its turn. It gives equal priority to all the entities. It grants access to the bus in rotating basis like a rotating switch as you can see in the Figure *2-13* . When an entity releases the bus, access is given to the next entity requesting for access and if some entity is not requesting it can skip. If currently served entity is Master#1 and the next entities requesting for access are MASTER#0 and MASTER#3. Access will be given to the MASTER#3. [10]

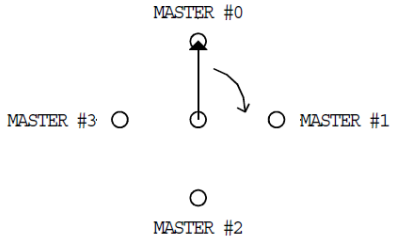


Figure 2-13 *Round Robin Arbiter [10]*

- **TIME DIVISION MULTIPLEXED ACCESS (TDMA)**

Time is divided in slots and each slot is assigned to a particular master. Master can only get access to a channel in its given time slot. Several algorithms are made to cope up with the problem of unused slots.

- **LOTTERY**

In this protocol a centralized system collects the requests from entity for the ownership of shared channel. An entity getting the ticket will get access to the bus. A static or dynamic number Xlottery ticketsX is assigned to them [1].

- **TOKEN PASSING**

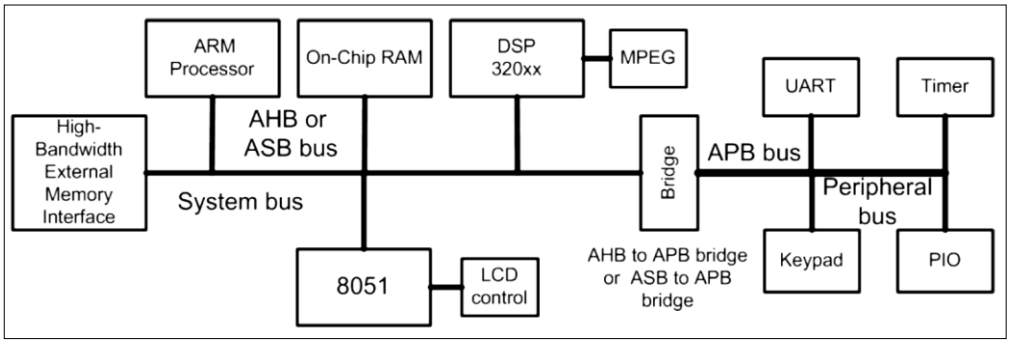
Token passing protocol is used in the ring topology. A special data word token is passed to all the nodes on ring and the node which has the token can access the shared channel. When the node is done with its transaction it passes the token to its next neighbour in the ring and the node which doesn't want to have access can simply pass the token to its neighbour.

2.2.3. SoC Buses

Advance Microcontroller Bus Architecture (AMBA)

AMBA (Advance Microcontroller Bus Architecture) is a bus standard developed by ARM with an intention to introduce an efficient on chip bus for communication. This bus is divided into two main parts; one is for system communication and other is for

Figure 2-14 *Example of AMBA Bus [1]*



peripheral communication and they are connected via a bridge. It has single edge clock protocol, split transactions, several BUS Master, pipelined operations and non-tri-state implementation. [8] [1] Buses defined with the AMBA system are

- **ASB (Advanced System Bus)**
  This is one of the system buses which are used for cost effective and simple solutions and it supports many things like pipelining, multi master and burst transfer, non-multiplexed address and data bus and centralized decoder and arbiter.

- **AHB (Advanced High-performance Bus)**
  This is also a system bus which is designed for high performance designs. It provided very high bandwidth and support processors, peripherals, RAM interfaces and APB Bridge. It supports multi master operation, burst transfer, split transactions, wide data bus, Synchronous no multiplexed bus, separate read data buses, and non tri-state multiplexed operations.

- **APB (Advanced Peripheral bus)**
  It is used to connect low power and low speed peripherals. It is a secondary bus and it doesn't have clock and data access is controlled by select and strobe. Bridge is the master and devices like Timer, parallel I/O, UART, keypad etc are slaves. It is a static bus with simple addressing, simple un-pipelined interface and latched address and control system.

Advanced Extensible Interface (AXI) BUS

AXI bus is introduced by AMBA; AXI3 in 2003 and AXI4 in 2010. AXI Interconnect IP Core is available in Xilinx ISE Design suite which can be used in the projects. It is suitable for low latency and high bandwidth designs. Unlike AMBA it provides high frequency operation without the use of bridges. It meets the interface requirements of large range of components and is suitable for memory controllers. It is flexible in the implementation of interconnections and is also backward compatible with AHB and APB interfaces of AMBA Bus. [20] [1] [21]

AXI has three types of interfaces

- AXI4- for high performance memory mapped
- AXI4-Lite- for simple, low-throughput memory-mapped communication
- AXI4-Stream- for high speed streaming data

AXI specification describes an interface between AXI Master and Slave. They are connected together using a structure called Interconnect block and in case of AXI4 and AXI4-Lite it is AXI Interconnect. It is used for memory mapped interfaces only while the AXI4-stream interconnect can be used for AXI-4 stream bus implementation.

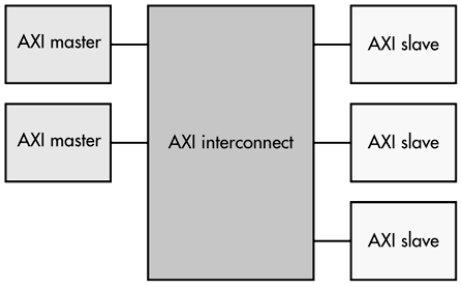


Figure 2-15 *AXI Bus*

Both AXI4 and AX4-Lite has different channels between Master and Slave

- Read Address Channel
- Write Address Channel
- Read Data Channel
- Write Data Channel
- Write Response Channel

AXI interconnect can be implemented in two modes;

Crossbar mode: Shared address and multiple data crossbar architecture with parallel lines for write and read data channel. This mode is optimized for performance.
Shared Access mode: Shared write and read data and single shared address pathway. This mode is optimized for area.

Some of the key features of AXI bus are

- Separate address/control and data phases
- Support for unaligned data transfers using byte strobes
- Burst based transactions
- Separate read and write data channels
- Able to issue multiple outstanding addresses
- Out of order transaction completion
- Can add register stages to get timing closure
- 32,64,128,256,512 or 1024 bits data width for AXI and 32 bit data width for AXI4-Lite

- Connects 1-16 masters to 1-16 slaves
- Optional register-slice pipelining and data path FIFO buffering

AVALON BUS

Avalon bus architecture is designed by Altera to connect its processor and peripherals in System-on-Programmable Chip (SoPC). It is designed for FPGA SoC design and used mainly by NIOS-II soft core processor by Altera. It has synchronous interface and it has pre-defined signals and timing to connect Master and Slave interface. It uses separate address, data and control lines. It supports multi-master and Master and salve interacts with each other in a technique called slave-side arbitration. Its other services are data-path multiplexing, address decoding, dynamic bus sizing, latent transfer capabilities and a streaming read and write capabilities. Altera SOPC builder, a system development tool, automatically generates a switch fabric logic to support the transfers by Avalon interface. [9]. Altera also started using AXI Bus but Avalon bus is specifically designed for the NIOS-II Embedded processor and it is still using it.

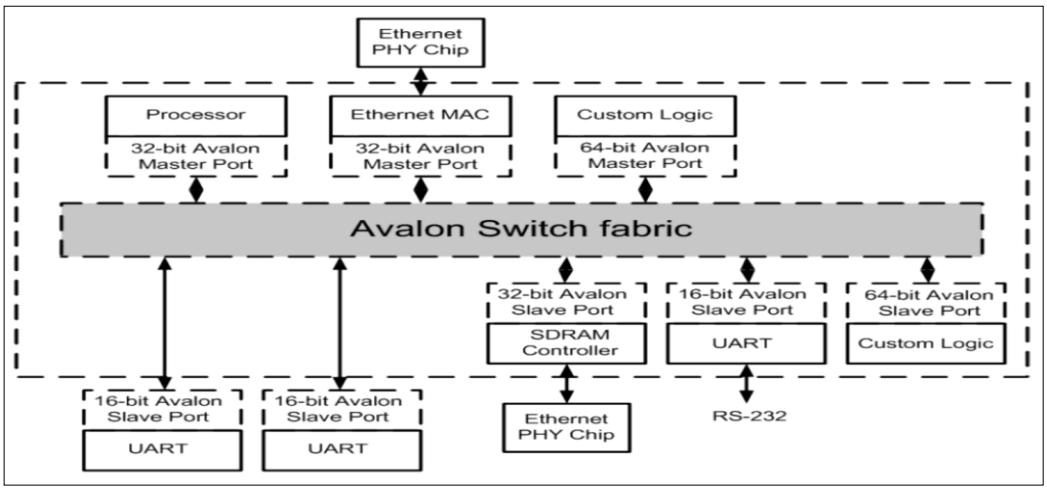


Figure 2-16 *Example of Avalon bus [1]*

CORECONNECT BUS

CoreConnect bus is developed by IBM. It is a hierarchical bus which is made of three buses each having its own functionality to make a complete System on Chip. [7]

- **PLB (Processor Local Bus)**
  It is multi master and synchronous bus designed to achieve high performance and lower delays. Its separate address and data bus support simultaneous read and write transfer. Masters are attached to the bus through separate addresses, read-data and write-data buses. Slaves are attached through shared and decoupled address, read data and write data buses. Arbitration unit supports 16 masters and any number of slaves.

- **OPB (On-chip Peripheral Bus)**

It is designed to connect low performance and low speed peripherals like UART, serial and parallel ports. It is fully synchronous, dynamic bus size, separate address and data bus, multiple masters, single cycle data transfer between master and slave. Its uses distributed multiplexer. PLB is connected to OPB through OPB Bridge and the bridge acts as slave to PLB and master to OCB.

- **DCR (Device Control Register Bus)**
  It is a single master bus with low speed data path. It is used for passing status and setting configuration information between the cores. It is also designed for testability purpose. It is a synchronous bus with ring topology. It has 10-bit address bus and 32 bit data bus. Its arbiter works on static priority function.

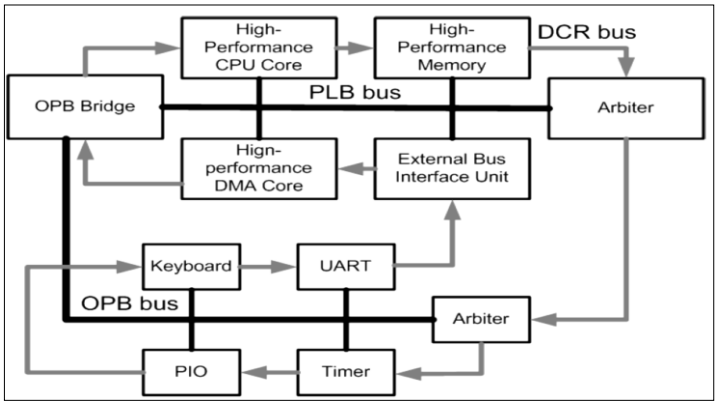


Figure 2-17 *CoreConnect Bus [1]*

WISHBONE BUS

Silicore corporation designed SoC interconnect architecture for portable IP Core independent of FPGA and SoC. It is put on public domain by OpenCores in August 2002. This particular information is extracted from [10]. It offers following features

- Simplicity, Reliability, Portability, Compact, non-hierarchical and Flexible
- Supports structured design methodologies
- Modular data bus width and operand sizes
- Variable interconnection methods which support point to point, shared bus, data flow and crossbar switch
- Single Clock data transfer
- Modular address widths
- Slave less redundant logic using partial decoding scheme
- User Defined tags
- Multi master capabilities
- User defined arbitration methodology
- Independent of hardware technology, delivery method, synthesis tools, router and layout technology

WishBone uses Master/Slave architecture. Master interface initiate the data transaction and slave responds to request. Master and slave communicate through an interface called INTERCON. This interconnection is variable and can be set according to the user needs like point to point, data flow, shared bus or crossbar switch. It is a large synchronous circuit and is designed to operate over an infinite range of frequencies and the only limit is the maximum frequency of the physical circuit.

Types of WishBone Interconnection

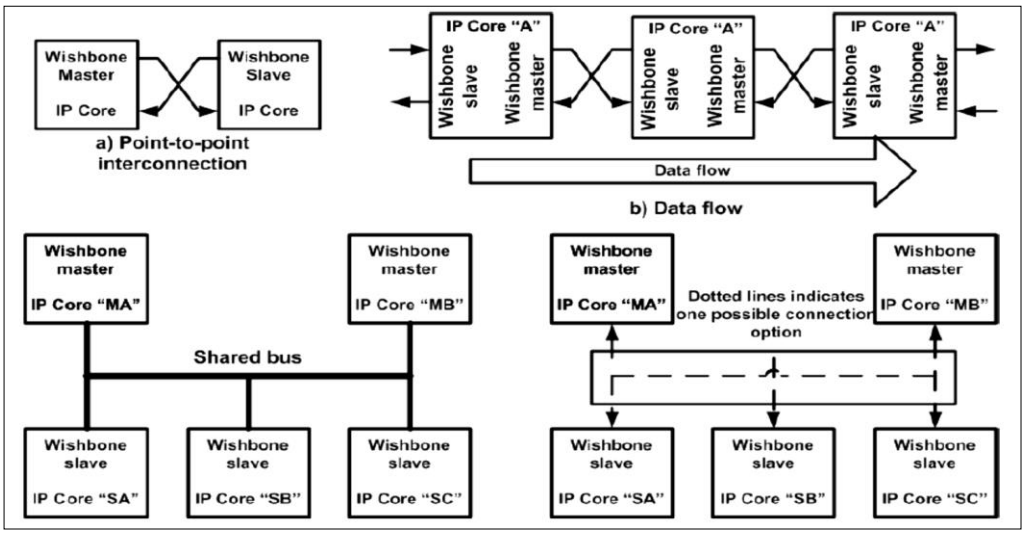


Figure 2-18 *WISHBONE Bus [10]*

Point to Point interconnection is the simplest way to connect two WishBone IP cores. It allows connecting a single Master interface to a single Slave. For example master can be a microprocessor core and slave can be serial I/O port.

Data flow interconnection is used when data flows in a sequential manner. Each single core has both Master and Slave interface. Data flows from one core to another and it exploits pipelining and speed up execution. Such architecture can be used e.g. signal processing.

Shared bus interconnection is used when connecting more than one master with one or more than one slave. Master initiates a bus cycle and targets slave, an arbiter gives an access to a particular master and decides when and which master will gain access. Shared bus can use priority or round robin arbiter. Shared interconnection are compact and uses less resources but latency can increase as the number of master's increase. Shared bus can be implemented with multiplexer or three state buses.

Crossbar switch interconnection is used when connecting two or more masters with two or more slaves. Master initiates a bus cycle and arbiter decided when a master can access a slave. And a dedicated link is established between master and slave. Overall data rate is higher than the shared bus but it requires more resources like the interconnection logic. Unlike shared interconnection more than one master can us the interconnection as long as two masters don't access the slave at the same time.

Summary of On-Chip Buses

This table summarizes and compares all the features of the important buses like WISHBONE, AMBA, AVALON, and CORECONNECT. [1] [5] [9] [8] [7] [10]

Table 1 *Summary of On-Chip buses*

Main feature	Sub Feature	WISHBONE	AMBA	AVALON	CORECONNECT
Status	Open	Yes	Yes	Yes (Partial)	Yes
	Registration	No	Yes	Yes	Yes
	License	No	No	Yes	No
Architecture	Hierarchical	No	Yes	No	Yes
	Pipelined	No	Yes	Yes	Yes
	Multiplexed	Yes	Yes	Yes	Yes
Topology	Point to Point	Yes	No	Yes	No
	Shared	Yes	Yes	Yes	Yes
	Data Flow/ Ring	Yes	No	No	Yes
	Crossbar Switch	Yes	Yes	No	No
Arbitration	Static Priority	Yes (Application Specific)	Yes (Application Specific except APB)	Yes (Slave Side)	Yes
	TDMA	Yes (Application Specific)	Yes (Application Specific except APB)	Yes (Slave Side)	No
	Round Robin	Yes (Application Specific)	Yes (Application Specific except APB)	Yes (Slave Side)	No
	Lottery	Yes (Application Specific)	Yes (Application Specific except APB)	Yes (Slave Side)	No
	Token passing	Yes (Application Specific)	Yes (Application Specific except APB)	Yes (Slave Side)	No
Bus Width	Address Bus width(bits)	1-64	1-32	1-32	8-256
Clocking	Synchronous	Yes	Yes	Yes	Yes
	Asynchronous	No	No	No	No
Operating Frequency		User defined	User defined	User defined	User defined

2.2.4. Network on Chip (NoC)

Network on chip's idea is taken from computer networks formed by routers which are responsible for receiving and forwarding data packets and requests hence making a network of communication at Local or Global level. NoC is implemented within a System

on Chip or FPGA using micro routers, links and other components which are responsible for communication amongst the IPs and other routers on the chip. A NOC router in mesh NoC has five connections, one for connecting to the local processing element attached with the router and the other 4 for communication with its neighbouring routers on East, West, South and North. Network Adapters control the communication like packetizing and de-packetizing between the router and the core connected to that router. NoC links are inexpensive and length is short as compared to the computer networks. Area usage, power consumption and latency are very crucial for the NoC but the computer networks are more flexible with such things. NoC provides many advantages as compared to other on-chip communications. NoCs are scalable, more bandwidth, less latency, multiple connections amongst the cores, design re-usability, low area and low power consumption [13].

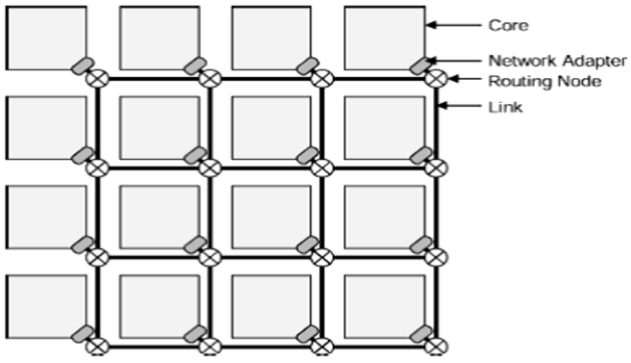


Figure 2-19 *Sample 4x4 Mesh NoC [13]*

NoC Router

Router is the main component and backbone of NoC. Routers are used in network to receive data from the sender and forward it to the receiver. It receives the incoming pattern calculates it destination and finds out the best path for its delivery. NoC router is built on OSI model where each layer has its own job. A router usually has 5 ports in which one is dedicated for the local port and rest 4 are East, West, North and South to communicate with other routers. Figure *2-20* shows the example NoC router. Router mainly consists of parts like FIFOs, Crossbar and Arbiter. FIFO stores the incoming packets from the respective ports. Each port has its own FIFO. Crossbar directs the input to the appropriate output port Arbiter- Arbitration scheme is implanted in arbiter like round robin or static priority. Arbiter decides when and which input packet will be directed to its output port.

NoC architecture has certain parameters to provide the desired functionality to a system.

NoC Topology

Distribution of nodes in the network and establishing links between them according to some scheme is topology. It also affects the area, power consumption, latency and frequency of the network. Popular topologies are Ring topology, Tree, Butterfly, Polygon, Mesh topology and torus. User can also define custom topology according to the requirements. 2D mesh is the common topology used in the FPGAs [22] [13].

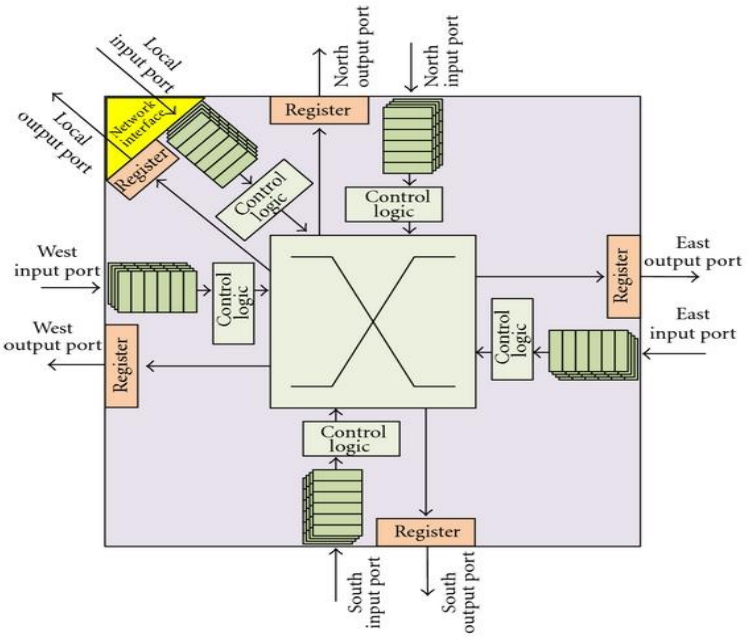


Figure 2-20 *NoC Router*

Switching Techniques

The mechanism which governs the passing of message from upstream to downstream is called switching. It directly affects the latency of the network. Circuit Switching and Packet switching are the common switching techniques. **Circuit switching** network reserves a dedicated physical path between the sender and receiver and it also needs setup time to establish or remove the path. Circuit switching advantage is that its bandwidth is predictable. On the other hand **Packet Switching** transmits packets without a dedicated path. Packet switching routers send data in packets called flits. Packet is delivered to the destination depending upon the routing information within the packet.

Flow Control

It lays down the policy the way network will provide resources for the message. Resources like buffers, channels, ports and control logic. Resource allocation depends upon the switching technique used. Packet switching requires buffers while circuit switching technique doesn't. Handshaking and credit based protocols are popular.

Routing Algorithm

Routing algorithm describes and defines the path between the source and destination taken by the packet. The objective is to make the routing scheme efficient so that there is less area overhead and latency and high performance can achieved. An efficient algorithm will save the network from issues like deadlock, live lock and starvation and helps to control the congestion. XY algorithm is the most famous amongst the others as it is simple and area overhead is low. It is implemented using distributed routing, in which each node forwards the packet to the next node depending upon the routing information included in packet header. Packet is sent to the X axis of the node first from where it reaches the Y

axis of destination node and then from Y axis it is forwarded to the node. It is one of the cheapest algorithms to achieve deadlock free network and prevents live lock.

Arbitration

NoC routers receive simultaneous messages on all of its nodes. It is the job of arbiter to decide and grant the output port to the incoming packet. There are various schemes of arbitration Round robin, first come first serve, priority based, and priority based round robin. First two ensures best effort data delivery while the last two are used for guaranteed traffic. Static arbitrations are limited to specific order and easy to implement but they can suffer from starvation problem while the dynamic arbitrations take decisions at run time depending upon the network condition, but are more difficult to implement and will take more area. These are efficient, flexible and ensures starvation free network.

NoC Evaluation Criterion

Cost metrics

Power consumption and area are the most focused criteria for NoC cost evaluation. Goal of the designer is to minimize them so that it can also be used in small applications where area and power is limited. When NoC is implemented in FPGA which has fixed logic units and routing paths more focus and techniques are required to lower these criterion. Buffers in the NoCs router are considered to be most area hungry items. Worm Hole switching technique is considered to be low cost because of their low buffering requirement. Dimensions and the amount of buffers in router play important role in the cost metrics of the NoC [22].

Performance Metrics

Data transaction time is important in evaluating performance metrics. Message delivery speed is determined by the operating frequency but throughput and latency are also important metrics for NoC. Data transferred in period of time is throughput and can also be related with bandwidth. Throughput in NoC is divided into intervals like overall application, packet throughput measured per system, IP core and router calculated as an average. Latency is the time taken for sending data from a source node to a destination node. It is calculated as average packet delay/flit traversing in the network. The lowest bound of latency is the zero latency or best case latency; it is the latency when no congestion is present in network. Internal delay of the router and other parameters like serialization play important part in overall latency. Such delays can be reduced by looking and adjusting the parameters like switching techniques, flow control and reducing routing decision time [13].

2.3. Summary of On-Chip Interconnects

Point to point architecture has direct connection between each module through a set of dedicated wires. It works for small systems but as the complexity increases they will need more dedicated wires for communication. The system won't be scalable, routing will be difficult and there will be area overhead and hence the complexity increases. Point to

point communication for system having few components can run effectively with such light weight architecture. The communication channel between two nodes is a dedicated set of wires so as the number of nodes increases the use of wires increases exponentially and the routing becomes more difficult. Point-to-point scheme suffers from low wire usage efficiency for low bandwidth channels and also pose a high usage of hardware. If the system needs expansion in the future then this scheme may not be a good choice. As the complexity of the systems is increasing and requirements like low area overhead and higher bandwidth are demanding more efficient interconnection architectures. [1] [2]

A Crossbar Bus can be used in multi core SoCs where more than one master can simultaneously access several slaves. There are multiple ways for data to be transferred between masters and slaves in a crossbar switch interconnection. Therefore two or more masters can communicate with slaves at the same time, as long as it isn't the same slaves. As compared to a shared bus, it leads to a higher data transfer rate. In this type of interconnection, there is always an arbiter to control the bus. Arbiter decides which master may communicate with which slave. There are number of shared bus architectures by different companies. Each of them has its own application, advantages and disadvantages. Shared bus is used when low area overhead and scalability are required. Shared bus is suitable for small systems, or more complex hierarchal multi buses, using sophisticated protocols and bridges, to serve larger systems. Bus is a shared interconnection where each module is connected to the bus and acquires the bus control based on bus protocol. The disadvantages are the bandwidth is limited and the scalability is restricted. As the number of (IPs) grows, the use of these techniques becomes a bottleneck because of scalability complications and efficiency. Buses have their own advantages and disadvantages but they are used if they are fitting in a particular application. Buses can be used when the complexity of the system is less and the requirement for maximum operating frequency is not very high. Many disadvantages are associated with bus interconnections. Noticeably, the bandwidth is limited, concurrent communications are not possible, and the scalability is restricted and causes speed degradation. [1] [2] [13]

NOC is implemented using micro routers, links and other components which make the communication possible amongst the modules and other routers on the chip. NOCs have certain advantages but they are used for complicated and high end systems for specific applications. The basic concept of NoC is to communicate across the chip in the same way that messages are transmitted over the Internet today. Communication is achieved by sending message packets between blocks using an on-chip packet-switched network. NoCs work well for systems with large number of IPs which require high throughput and high operating frequency. However for smaller systems Area overhead, complexity and power dissipation won’t be optimal as compared to the SoC buses. [6] [13] [23]

3. ANALYSIS OF INTERCONNECTIONS

This chapter gives an analysis about a SoC design, which is implemented with point to point interconnection and shared Bus. WISHBONE shared bus architecture is selected as a new way to implement the system. It is modified, implemented and analysed according to the design requirements in accordance with the WISHBONE rules.

3.1. Analysis

Analysis includes the discussion about the design implemented by point-to-point architecture and the problems caused by such interconnection. Shared bus architecture is implemented with WISHBONE standards using Master, Slave and interconnection scheme. Writing and reading of the master bus follows the WISHBONE standards. Pros and Cons of the new proposed design are discussed and also features of new design are listed.

3.1.1. Design with point-to-point interconnection

This design is asynchronous and has one Processor which is connected to around 400 to 800 entities (Registers) through point to point interconnection as we can see in Figure *3-1*. Processor is an external processor outside of the FPGA which is connected to the FPGA through an external bus. This design is one part of the large design of company's project.

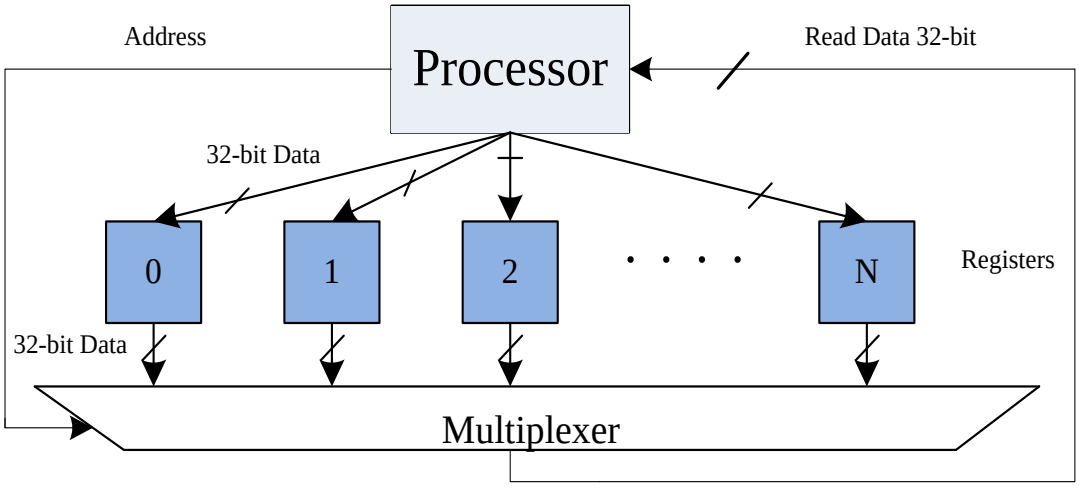


Figure 3-1 *Block Diagram of Point to Point Design*

The design resulted in large area as each connection between processor and registers is 32- bits wide. This design was implemented on Xilinx Virtex-6 FPGA. The processor is directly connected to the registers without any logic in between. The design also resulted in a large multiplexer when Registers Write back to the processor as you can see in *Figure 3-1*. Pipelining cannot be implemented as the design is asynchronous. Large

multiplexer created timing issues in a multi clock domain and the design failed to fulfil the timing constraint of 100 MHz with FPGA filled around 65%. As the external processor is operating at very high speed so the requirement of the design is high speed. According to the Non-Disclosure Agreement with the company, the exact details of the design cannot be discussed here.

3.1.2. Design with shared bus interconnection

There are several interconnection types available to solve the problems. Each of them has its own advantages and disadvantages. We will discuss them here and will find suitable interconnect which supports the architecture of our design of Single Master and Multiple Slave.

As already discussed, the crossbar interconnect architecture is designed for good parallelism where several Master cores are communicating with the slaves simultaneously while the required design only includes one Master core with several slaves. Crossbar interconnect architecture doesn't fit in this context. Crossbar bus can be used even with the single master when the master is required to access the different slaves simultaneously which is not our case.

NoC is the most recent development in the domain of SoC interconnect, which is designed to handle large number of IP cores, large bandwidth and scalability but suffers from high area overhead and complex design interface issues [24]. As this interconnect architecture is designed for multi master cores, it doesn't fit into the design requirements of the required design architecture, which is with single master core.

There are several versions of the bus architecture like shared bus, crossbar bus, ring bus and hierarchical bus. Crossbar bus, hierarchical bus and ring bus are used for parallelism and multi cores. Shared bus is rather simple to implement, takes less resources and is scalable up to a certain point [6] [1]. The only choice left behind to solve this particular issue will be possible by the shared bus interconnect because it supports the architectures single master into single slave, multiple master into multiple slave and finally Single Master x Multiple Slave architecture, which is the architecture of our problem.

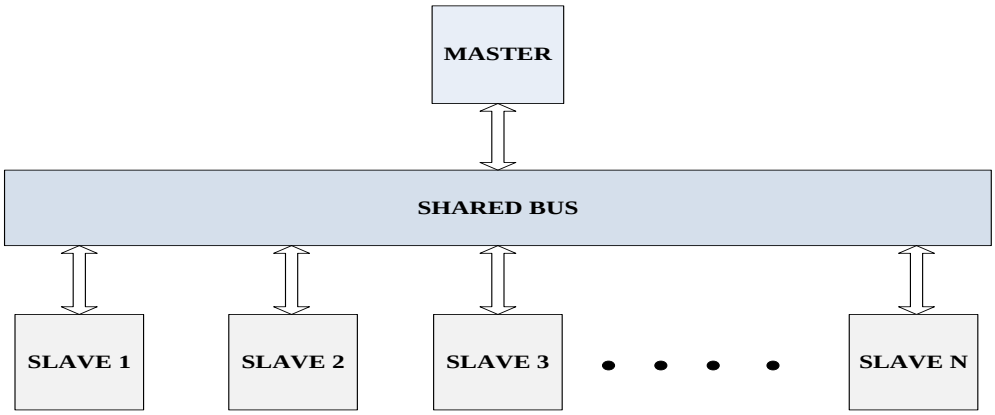
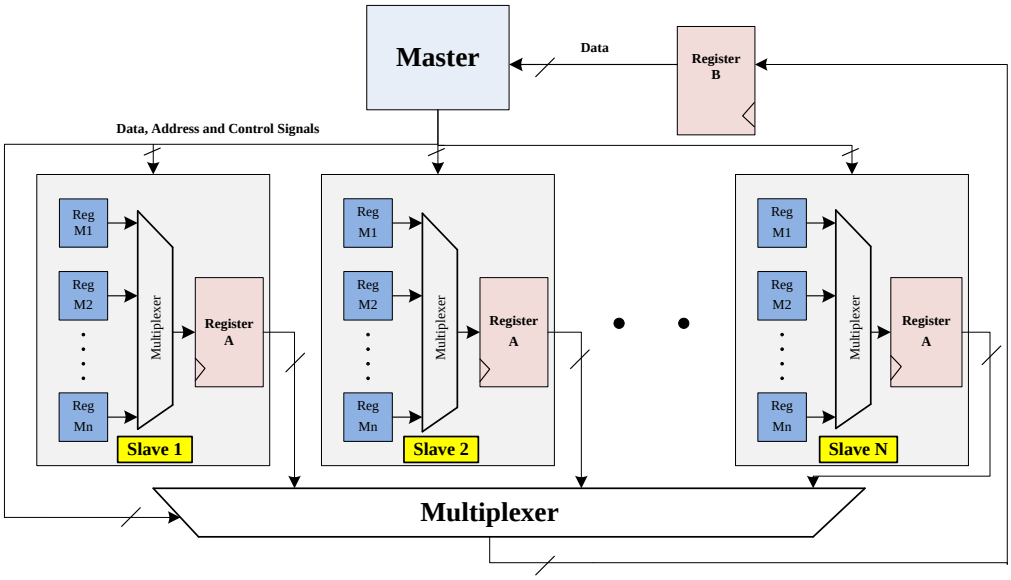


Figure 3-2 *Shared bus Block Diagram*

Shared bus block diagram can be seen in *Figure 3-2* which is the architecture of the proposed solution. It shows that Master/Processor is connected to the slaves through a shared bus. Design is further explained later with design in detail.

The detailed block diagram of the proposed design with shared bus can be seen in *Figure 3-3*. Master / External processor is connected to the multiple slaves. Unlike the design with point-to-point architecture, total number of registers are divided and put inside the slaves and also big multiplexer is divided. There will be n- number of registers and N- number of slaves. Writing the data takes one clock cycle while reading the data takes two clock cycles as read back is divided into two pipeline stages. Register A represents the first stage and the Register B is the second stage of pipeline. Interconnect includes control signals, address signal and data signal. Interconnect also includes other logic and address decoder which will be explained in later section. Each slave contains a fixed number of registers or memory spaces in case slave is realized as a memory. The number of slaves and registers within each slave is decided by the designer. Designer will take into account that the timing and resource consumption depends upon the total number of slaves and number of registers within slaves. It is shown in the Chapter Results. Data is written to each slave via shared data, address and control lines through shared bus while data is read using two stages of multiplexers.

Figure 3-3 *Proposed Design with shared bus*



Slaves also contain write back multiplexers which selects the data Register A based upon the address from the Master and the second Register B is a part of a shared bus which selects the slave based upon the address. Detailed description of this new block diagram is in section 3.1.3.

3.1.3. Benefits and Features of Shared bus design

Benefits of Shared bus design

- Interconnect sharing by the entities will reduce the overall utilization of interconnect area.
- Synchronous design will lead to pipelining and increase the overall efficiency of the system.

- Shared bus will make it easy to Add/Remove the entities hence making the system scalable and flexible.

Features of Shared bus design

- **Bus Architecture** for the new design will be a shared bus which will support the design of single master and multiple slaves.

- **Bus Topology** in proposed shared bus design will be non-multiplexer based topology in which data and address interconnections are separate and not shared. It uses more wires but average latency can be reduced compared to asynchronous point-to-point topology.

- **Bus Interconnection type** will be multiplexer based as tri-state interconnections are not portable and also tri-state buffers are not available in all the devices.

- **Synchronicity** is a feature of clocking in the bus. Single clock will be used for communication medium and connected entities. The new design will be synchronous and makes it possible to do pipelining.

- **Bus Protocol** for data transfer is handshaking. Read and Write cycle are explained in detail in the WISHBONE bus section below.

- **Address and Data Bus** WISHBONE data bus width can be 8, 16, 32 or 64 bits and the address bus width is 1 to 64 bits.

3.2. WISHBONE Bus

The proposed shared bus design will be realized by the Wishbone shared bus standard. WISHOBONE bus standard is chosen as we have seen in Chapter 2 Table 1 the comparison between the important interconnects. WISHBONE bus appears to be simple, flexible and portable because of certain factors. WISHBONE bus is open standard and it requires no registration or license. It is in the public domain and maintained by Open Cores and its IP cores are free to use in any product. It can be used in projects and designs without any royalty payment and financial issues.

CoreConnect and AMBA are hierarchical buses made for complex designs for multi masters. AMBA is hierarchical, Avalon is point to point and CoreConnect is a hybrid structure. WISHBONE bus is more flexible in terms of architecture and topology. It supports, point to point, dataflow, crossbar and shared bus architecture.

Latency in WISHBONE bus is user defined and application specific. In multi master interconnection latency depends upon the arbitration technique and number of masters. In our project there is a single master and many slaves so 1 clock cycle latency can be achieved but the results would be different for pipelining vs. non pipelined designs.

3.2.1. WISHBONE shared bus

WISHBONE shared bus interconnection is designed to connect two or masters with one or more slaves. As the general idea of the WISHBONE shared bus is already given in Chapter 2 but it will be discussed in more detail in this section. For further rules and standard specification, WISHBONE standard specification document can be consulted at [10]. General diagram for multi master and multi slave architecture can be seen in Appendix 1. It comprises of components like Master, Slave, Arbiter, Interconnect and Simple System Controller (SYSCON).

This architecture has memory mapped addressing. It's an architecture which allows the data to be stored and recalled in memory at unique addresses. A unique address range will be assigned to each slave for addressing. "Master" is an entity or IP core that initiates the data transaction with the "Slave" entity. Master provides address and control signals and slave responds to the particular master. "Slave" is an entity which is responsible for responding to the Master signals and slave is implemented as a memory module to make it memory mapped architecture. "Arbiter" is an entity that works like a traffic cop in a multi master architecture which grants access to one master at a time to the bus. Different arbiter protocols are discussed in Chapter 2 which can be implemented in the arbiter.

"Interconnect" consists of wires and logic gates that provides interface to the master and slave to communicate with each other. Other than the Master, Slave, System Controller and Arbiter, all the things like wires, logic gates, multiplexers and address comparator that we see in Appendix 1 figure combine to make interconnect. WISHBONE shared bus interconnection general block diagram can be seen in Figure *3-4*.

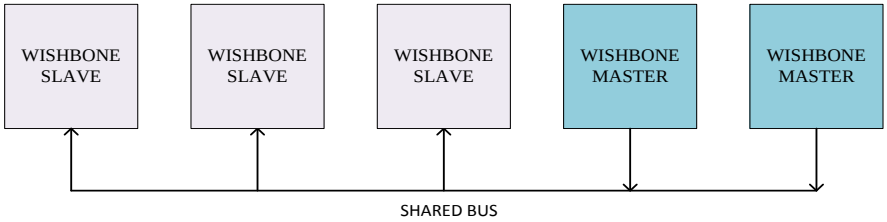


Figure 3-4 *Shared bus interconnection*

SYSCON called as system controller which generates clock [CLK] and reset [RST] signals which are compatible with WISHBONE. The clock output signal is directly connected to the input clock signal but the reset signal [RST] is a single clock pulse generator in accordance with the WISHBONE reset timing rules. [EXTTST] signal is a RESET signal input from the user. Detailed Rules for reset are listed in the WISHBONE specification document [10] in Chapter 3 section 3.1.1. State diagram and state machine explanation can be found in Appendix 2.

These components are explained in more detail in the example implementation of a multi master and multi slave shared bus.

3.2.2. WISHBONE Master and Slave Signals

WISHBONE Master and Slave shared bus comprises of many signals but the signals relevant to the shared bus and the ones, which fit in our design are listed here. There are some common signals between Master and Slave like Clock, Data In, Data Out and Reset.

Clock: Clock input signal coordinates with all the activities in the Master, Slave and interconnect. At the rising edge of clock all WISHBONE output signals are registered. All input signals should be stable before the rising edge of Clock.
Data In: Data In signal is binary data array used to accept the data which is being read from the Slave. Maximum supported port size is 64-bits.
Data Out: Data Out signal is used to send binary data to the slave. The data port is for data output and the maximum size can be 64-bits.
Reset: The reset signal is used to reset or restart the WISHBONE interface. This signal resets the whole WISHBONE interface but excluding the other cores in a system. It can be connected to them if required.

MASTER SIGNALS
WISHBONE shared bus master are listed here

- **ACK_I:** The Acknowledgement signal is asserted to indicate the successful response of the SLAVE and normal termination of the bus cycle.
- **ADR_O:** This binary address signal is used to send the address to the SLAVE. The size of the signal depends upon the size of data port.2^n=Size of the data port, where n will be the width of address port.
- **STB_O:** The strobe output signal [STB_O] shows a valid data transfer cycle. SLAVE responds with [ACK_I] signal in response.
- **WE_O:** The Write Enable Output signal [WE_O] shows whether the current signal is READ/WRITE. Signal is LOW on Read and signal is HIGH on Write.

SLAVE SIGNALS
WISHBONE slave signals are listed here

- **ACK_O:** The acknowledge output [ACK_O] is asserted for the normal termination of the signal in response to strobe signal of MASTER.
- **ADR_I ():** The address input signal [ADR_I ()] is used as an address for the SLAVE in a memory mapped structure.
- **STB_I:** The strobe input [STB_I], when asserted, indicates that the SLAVE is selected. A SLAVE shall respond to other WISHBONE signals only when this [STB_I] is asserted. The SLAVE asserts either the [ACK_O] signals in response to every assertion of the [STB_I] signal.
- **WE_I:** The write enable input [WE_I] indicates whether the current local bus cycle is a READ or WRITE cycle. The signal is negated during READ cycles, and is asserted during WRITE cycles.

3.2.3. Wishbone General Single Read Cycle

Wishbone single read cycle for master can be seen in *Figure 3-5*. The figure is taken from the datasheet of the Wishbone bus and unnecessary signals are removed.

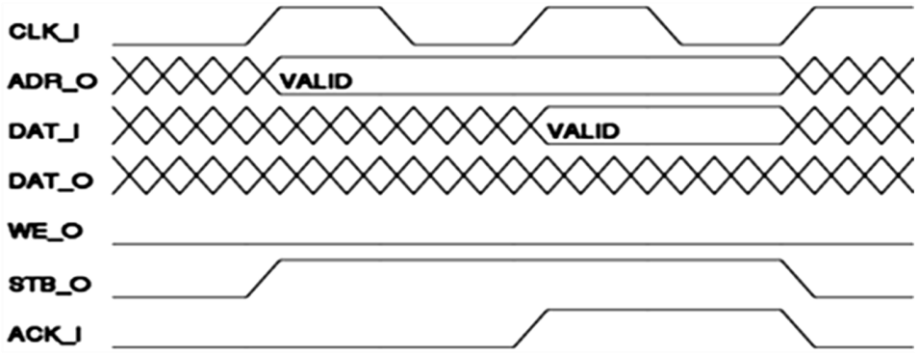


Figure 3-5 *Wishbone Single Read Cycle*

CLOCK EDGE 1
MASTER presents a valid address on [ADR_O ()].
MASTER negates [WE_O] to indicate a READ cycle.
MASTER asserts [STB_O] to indicate the start of the phase.

CLOCK EDGE 2
SLAVE decodes inputs, and responding SLAVE asserts [ACK_I].
SLAVE presents valid data on [DAT_I ()].
SLAVE asserts [ACK_I] in response to [STB_O] to indicate valid data.

CLOCK EDGE 3
MASTER receives data on [DAT_I ()].
MASTER negates [STB_O] to indicate the end of the cycle.
SLAVE negates [ACK_I] in response to negated [STB_O].

3.2.4. Wishbone General Write Cycle

Wishbone single write cycle for master can be seen in *Figure 3-6*. The figure is taken from the datasheet of the Wishbone bus and unnecessary signals are removed.

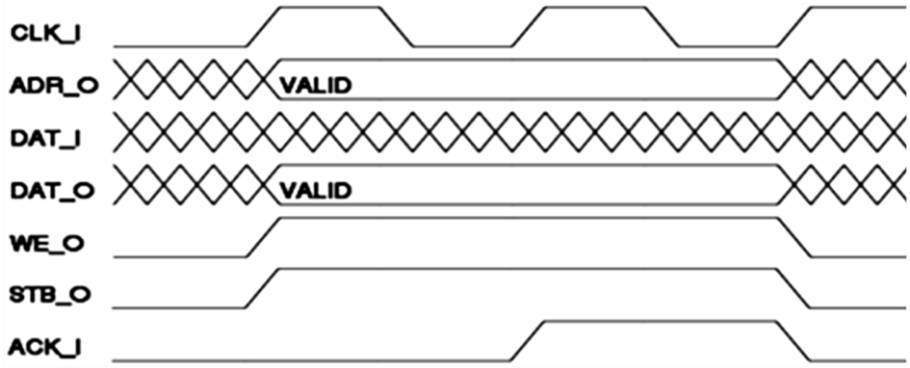


Figure 3-6 *Wishbone Single Write Cycle*

CLOCK EDGE 1
MASTER presents a valid address on [ADR_O ()].
MASTER presents valid data on [DAT_O ()].
MASTER asserts [WE_O] to indicate a WRITE cycle.
MASTER asserts [STB_O] to indicate the start of the phase.

CLOCK EDGE 2
SLAVE decodes inputs, and responding SLAVE asserts [ACK_I].
SLAVE prepares to latch data on [DAT_O ()].
SLAVE asserts [ACK_I] in response to [STB_O] to indicate latched data.

CLOCK EDGE 3
SLAVE receives data on [DAT_O ()].
MASTER negates [STB_O] to indicate the end of the cycle.
SLAVE negates [ACK_I] in response to negated [STB_O].

These timing diagrams show the general Read and Write cycles for the Wishbone shared bus. The timing diagram from the actual implementation of the design will be shown in the next chapter. There are some signals which will be skipped for our shared bus implementation because they don’t fit in the context of single master-multiple slave design.

4. IMPLEMENTATION OF SHARED BUS

This chapter has the complete implementation of the wishbone shared bus. Wishbone shared bus for multiple Master and multiple slave design is used as a base design to start, which is in Appendix 1. It is modified according to the requirements of the single master and multiple slave design. Short snippets of code are included and discussed. There are also timing diagrams of Read and Write cycle from ISim simulator.

4.1. Shared Bus Design Implementation

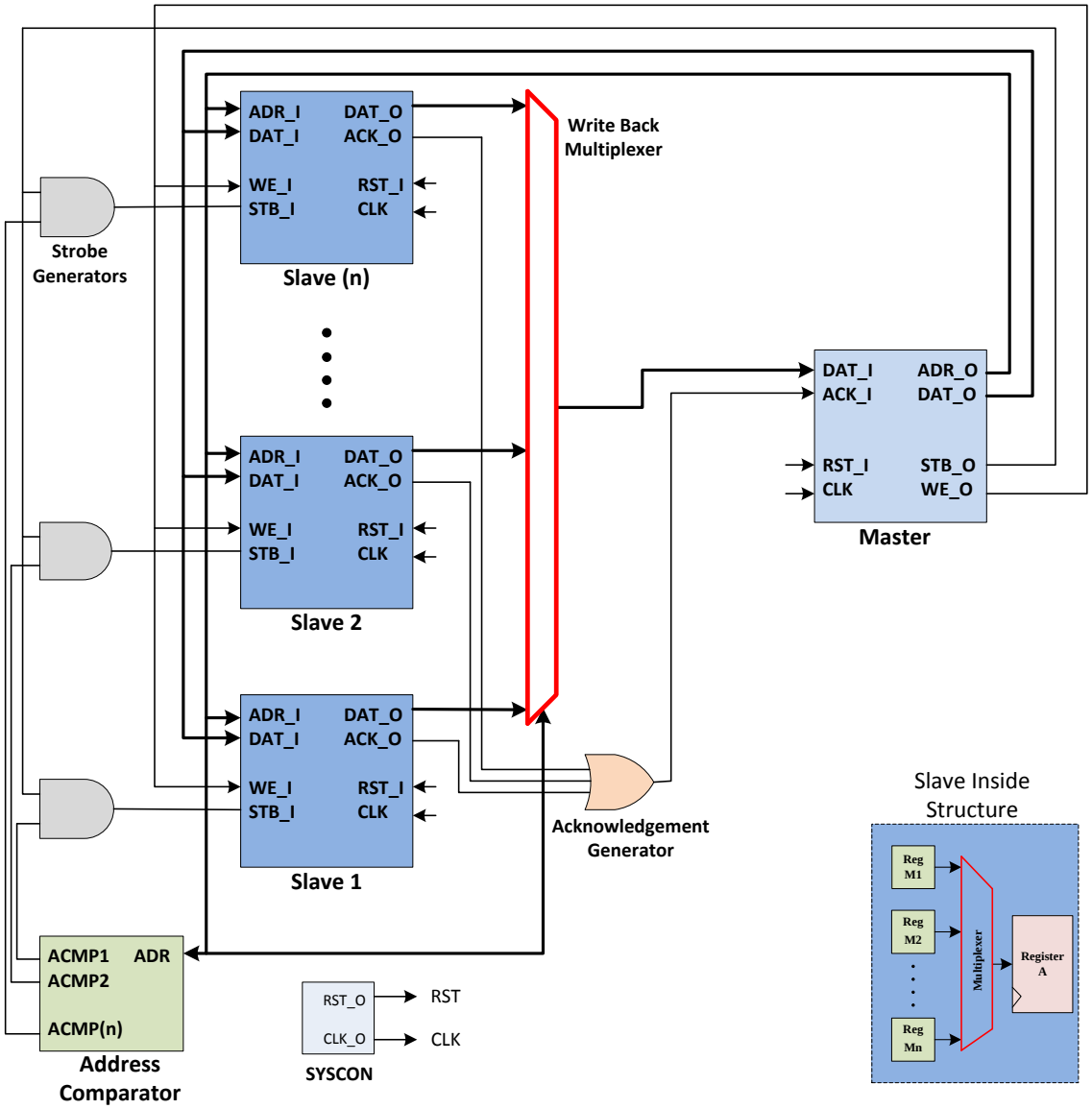


Figure 4-1 *Proposed Shared Bus Design*

The proposed shared bus design can be seen in Figure *4-1*. Bus and Interconnection topologies are already discussed in Chapter 2 Shared bus. With only single master, design will be simpler than multi master design and there is no need of arbiter. Data, address and control interconnections are shared unlike the point-to-point interconnections. The design consists mainly of the following components Master, Slaves, Strobe generator, Acknowledgement generator, Address Comparator, System Controller and Write Back Multiplexer. The figure shows a top level block diagram of the design with necessary details. There is also a separate block, which shows the inside structure of the Slave.

4.1.1. Master of the Shared Bus

Figure *4-1* shows the idea of the general block diagram of the proposed design. There is a single master in this design and signals shown here are the WISHBONE standard signals. For the testing purpose WISHBONE Master Wrapper is made and can be connected to any Master and in this case it's DMA. The Master (DMA) will communicate with the WISHBONE Master Wrapper and the wrapper will communicate outside with slaves using standard WISHBONE handshake protocols. Master Wrapper and its interface with Master under test can be seen in Figure *4-2*. WISHBONE standard input signals are Acknowledgement (ACK), Clock (CLK), Data (DAT) and Reset (RST) while the output signals are Address (ADR), Strobe (STB), Write/ Read Enable (WE) Signal. Details of these signals are already described in the explanation of the WISHBONE Master Signals. Address and Data bus width are made generic in VHDL design which is passed on to the Master from the Top level entity. Master has five signals Address out, Data out, Write / Read, Enable and Data In and except Enable signal all the other signals are directly connected to the corresponding WISHBONE signals.

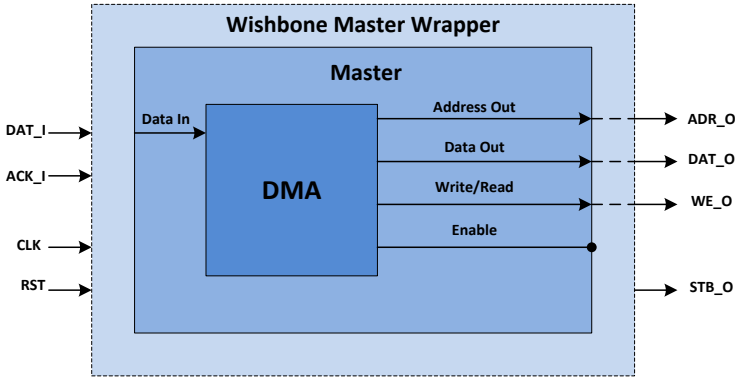


Figure 4-2 *WISHBONE Master Wrapper*

Enable signal is used to enable the Master wrapper to start the Write or Read cycle. Master is implemented as synchronous Direct Memory Access (DMA) unit to Read / Write data in the memory. For testing purpose Write/Read, Address, Data and Enable signals are provided from test bench either from the input file or VHDL test bench. As the Enable signal goes HIGH the DMA depending upon the Read/ Write signal starts to Write or Read from the Slave.

4.1.2. Slave of the Shared Bus

WISHBONE Slave wrapper is designed which can be connected with any slave. Standard WISHBONE signals are shown in the Figure *4-1*. WISHBONE Slave interface and Slave connection can be seen in Figure *4-3*. Slave is implemented as synchronous/asynchronous Read RAM which can be distributed or block.

Distributed RAM is made by using Look up Tables (LUTs) which are available in throughout of the FPGA. Maximum of 401Kb of Distributed RAM can be implemented in Xilinx Spartan-6 XC6SLX45T Device. SLICEM within the Configurable Logic Block (CLB) of the FPGA can be configured to be implemented as Distributed RAM. Such RAM is synchronous Write and asynchronous Read but can be also configured as synchronous Read. DRAM is fast, local and better for small memory requirements. Block RAM is a dedicated two port memory in FPGA. This device discussed above has 116 block RAMs of size 18Kb. Block RAM is synchronous Write and Synchronous Read. It also has the feature of pipelining with the output register and cab be used for large memory space requirements.

RAM can be also implemented as Xilinx IP Core RAM and connected to the Slave wrapper. Slave is implemented as a generic model as it takes the address and data size from the top level and also the size of the memory is passed from the top level which makes it generic and flexible model for testing. Xilinx tool can infer the type of RAM whether block/distributed from the modelling style or this slave is designed as we can force the tool to make it block or distributed using the VHDL code as

```
attribute ram_style: string;
attribute ram_style of REG : signal is "<Block/Distributed>";
```

Memory Read & Write operation is synchronous with clock. As this memory poses no wait state so Acknowledgement signal is tied to the Strobe signal. As we will see in the later sections the type of RAM and ratio between number of slaves and memory size with each slave affects the resource usage and timing parameters.

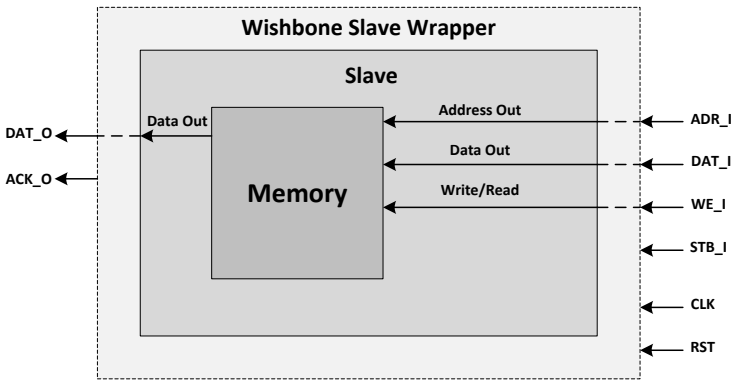


Figure 4-3 *WISHBONE Slave Wrapper*

4.1.3. Interconnect of Shared Bus

Interconnect consists of wires connecting entities, Address comparator, Read back multiplexer, Acknowledgement generator and strobe signal generator to enable the relevant slave as we can see in Figure *4-1*.

Address comparator takes the most significant bits which are reserved for Slave addressing from the Master Address. It analyses the address and assert the relevant ACMP (N) output pin which is connected to the corresponding Slave. Address comparator and this WISHBONE bus protocol implementation uses partial address decoding. It uses one-hot encoding scheme for the output where only relevant bit for particular slave is HIGH and rest are LOW.

Partial Address Decoding Registers are now divided into different blocks called Slaves. Slaves can also be implemented as memory rather than individual registers. Address bus is logically divided into two groups. Most significant bits are reserved to address each individual slave. Each slave is assigned a pre-determined static address. Address bus goes to the address comparator which depending upon the address decoding table, decodes the address and assert the HIGH signal for relevant slave. For example if we have four slaves and each slave has eight registers the address decoding table will be like as seen in Figure *4-4*. Address bus will be 5 bits in which two most significant bits are reserved to address the slave while the least significant 3 bits are used by each slave to address the registers or memory location within slave.

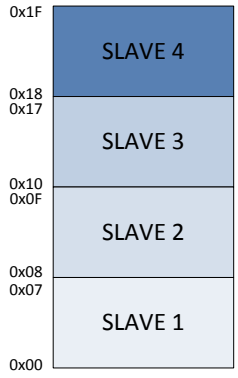


Figure 4-4 *Slaves Address Space*

Strobe Generators are AND gates which acts as Enable for the slave. They take Strobe, and ACMP(N) signal from the address comparator. Slave is active if all signals are in logic HIGH state.

Acknowledgement Generator; ACK_O signals from the slaves are connected to the OR gate and output is connected to the Master ACK_I signal. Data out from the slaves is multiplexed through a write back multiplexer whose select pin is the most significant address bits from the master. Acknowledge (ACK) signals from the slaves are connected to the OR gate and its output is connected to the input of the master Acknowledgement input pin.

Interconnect entity in VHDL Top level is designed in such a way that it makes the overall design generic. Width of the data and address bus can be provided and will be passed to the other entities from the Top Level block called as interconnect. Address bus

width is divided into two sections. MSBs are reserved to address individual slaves and LSBs are reserved to address unique partitions within slave. For example, if we have 4 Slaves each having 8 partitions/registers, then 2 MSBs are reserved for 4 slaves and 3 LSBs are reserved for partitions and the total length of address bus width will be 5 bits.

```
-- TOTAL ADDRESS BITS
1 ADR_BITS_LENGTH              : integer: =5;
-- TOTAL DATA BITS
2 DATA_BITS_LENGTH             : integer: =32;
-- Bits reserved to Address Slaves
3 SLAVE_ADR_BITS_LENGTH        : integer: =2;
-- Bits reserved to Address partitions within slaves
4 SLAVE_PARTITIONS_LSB_LENGTH : integer: =3;
-- Optional to select different no of slaves otherwise 2^no.bit for
slave
5 NO_OF_SLAVES                 : integer: =4
-- Optional to select different no of partitions with each slave
otherwise 2^no of partitions within each slave
6 NO_OF_REGISTERS_SLAVE        : integer: =8
```

The code snippet of the top level entity to implement and test a generic model is shown above. The numbers can be changed to implement the desired configuration.

1. Size of the address bits is the $\log_2$ (NO_OF_SLAVES x NO_OF_REGISTERS_SLAVE)
2. Data bits size is the size of the data bus
3. It is $\log_2$ (**NO_OF_SLAVES**) address bits reserved to address the slaves.
4. It is $\log_2$ (**NO_OF_REGISTERS_SLAVE**) address bits reserved to address the slaves.
5. This parameter gives the flexibility to set different number of slaves otherwise by default 2^ **NO_OF_SLAVES** number should be entered.
6. This parameter gives the flexibility to set different number of partitions within each slave otherwise by default 2^ **NO_OF_REGISTERS_SLAVE** with each slave number should be entered.

Top level entity takes the number for total number of slaves and generates them in for loop using generate statement. As we can see in code below component “Memory” is port mapped using for loop. The tool automatically generates the required number of slaves depending upon the input number “NO_OF_SLAVES”. This saves us from writing redundant code for port mapping each slave.

```
GEN_SLAVE:
      For I in 0 to NO_OF_SLAVES -1 generate
            SLAVE: Memory
            GENERIC MAP (ADR_BITS_LENGTH =>SLAVE_PARTITIONS_LSB_LENGTH,
                        NO_OF_REGISTERS =>NO_OF_REGISTERS_SLAVE,
                        DATA_BITS_LENGTH=>DATA_BITS_LENGTH)
            Port map (
                        ACK_O    => ACK_SLAVE (I),
                        ADR_I    => ADR (SLAVE_PARTITIONS_LSB_LENGTH-1
downto 0),
                        CLK_I    => CLK,
```

```
                    DAT_I   => DWR,
                    DAT_O   => DAT_OUT (I),
                    STB_I   => STB_SLAVE (I),
                    WE_I    => WE);
End generate GEN SLAVE;
```

4.2. Test Bench

A simple Test bench is written in VHDL to provide the stimulus to the Master. For the simulation purpose stimulus to the Master are address, data, enable and Write Enable. Clock period is 10 ns.

Write Data

A code snippet from test bench for writing the data is

```
Enable <='1';
WE_O<='1';
Temp_Address<="b00000001";
Temp_Data<="xAAAAAAAA"
```

Read Data

A code snippet from test bench for reading the data is

```
Enable <='1';
WE_O<='0';
Temp_Address<="b00000001";
```

4.2.1. Write Cycle

During the single write cycle the Master puts the data on [dat_o] and address on [adr_o] signal and asserts the Strobe out signal [stb_o]. Master asserts the Write Enable signal [we_o] to inform the slave that it is write operation. As slave in this implementation is RAM, which operates on zero wait state it replies by asserting acknowledge signal. In this case [ack_i] signal from slave is tied to the Strobe signal so it's the responsibility of the Master to negate the strobe signal when it's done with writing the data. This design takes only one clock cycle to write the data in the RAM. Latency for writing the data is one clock cycle. Timing diagram for write cycle can be seen in *Figure 4-5*.

The adr_o = XXX_XXXXX means the most significant 3 bits are for the 8 slaves and least significant 5 bits are for the memory within the slaves. In this timing diagram the first data "0xaaaaaaaa" for address "0b00000001" is written in the 1st slave and its 2nd memory location and the second data "0xbbbbbbbb" for address "0b00100001" is written in 2nd slave and its 2nd memory location

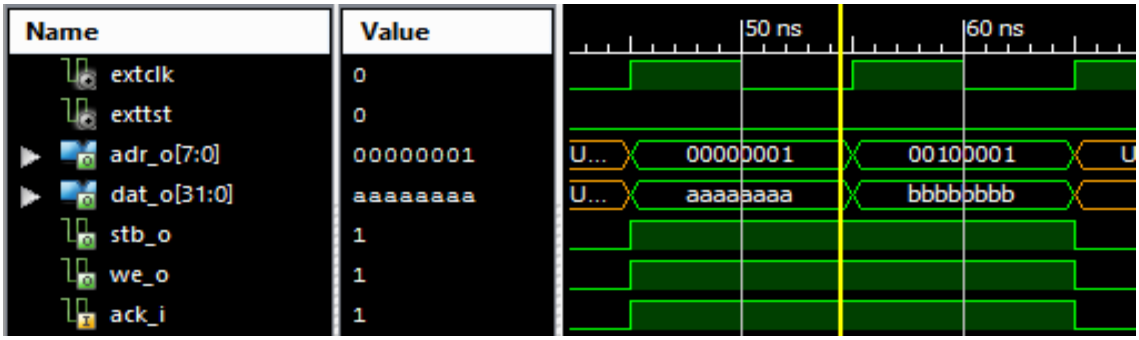


Figure 4-5 *Write Cycle*

4.2.2. Read Cycle

Read cycle is initiated by the Master. During the single read cycle Master puts the address on [adr_o] signal at Clock Edge 0 and similarly like Write cycle asserts the [stb_o] signal. [we_o] signal is negated to show the read operation. Slave asserts the [ack_i] signal and in response Master pulls down Strobe. This design takes two cycles to read the data because reading the data from the slaves is pipelined to make the clock fast. RAM in slave takes one clock cycle for reading the data and one extra cycle is added by Read Back Mux register. Data being read is same as the data written in the previous section from same address.

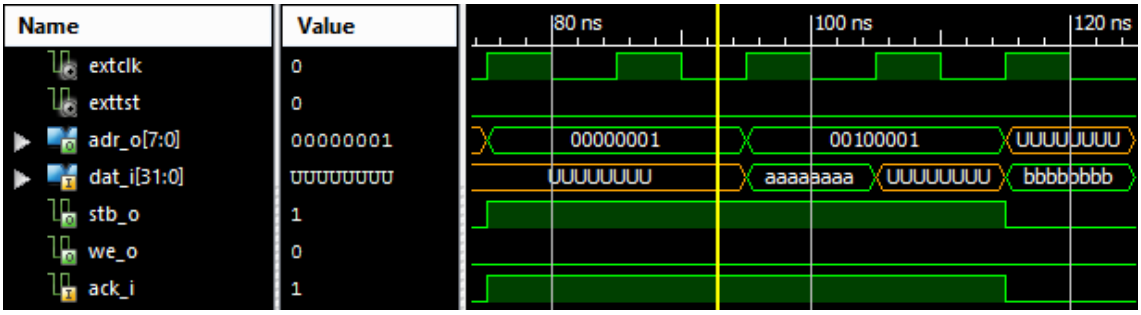


Figure 4-6 *Read Cycle*

4.3. FPGA filled with Dummy Logic

FPGA is filled with dummy logic so that when timings of the design are tested, the design is close to the practical case. This logic fills the Xilinx Spartan-6 XC6SLX45T Device up to 60-65 %.

PORT

```
DUMMY_IN :  in std_logic_vector(49 downto 0)
DUMMY_OUT: out std_logic_vector(49 downto 0)
```

ARCHITECTURE

```
CONSTANT NO_OF_DUMMY_STAGES: Integer: = 700;
```

```
Type t_dummy_array is array (NO_OF_DUMMY_STAGES-1 downto 0) of
std_logic_vector(DUMMY_IN'range);
Signal dummy_array: t_dummy_array := (others => (others =>
'0'));

PROCESS (EXTCLK)
Begin
     If (rising_edge(EXTCLK)) then
          DUMMY_OUT <= dummy_array(NO_OF_DUMMY_STAGES-1);
          dummy_array(0) <= DUMMY_IN;
          For i in 1 to NO_OF_DUMMY_STAGES-1 loop

               If i<2 then
                    dummy_array(i) <= dummy_array(i-1);
               End if;

               If i>=2 then
                    dummy_array(i) <= dummy_array(i-1) xor
                    dummy_array(i-2);
               End if;

          End loop;
     End if;
END PROCESS;
```

4.4. Clock Constraint

Clock Constraint is added for the clock in the constraints file with TIMESPEC. Design is tested with 10 ns, 5 ns and 4 ns.

NET "EXTCLK" TNM_NET = EXTCLK;
TIMESPEC TS_EXT_CLK = PERIOD "EXTCLK" 4 ns HIGH 50%;

This WISHBONE shared bus design for single master and multiple slaves is tested by changing the number of slaves and partitions within the slaves. The results are discussed in the next chapter. This design saved the interconnect utilization as compared to the point-to-point design also achieved higher timing constraints.

5. EXPERIMENTAL RESULTS

We will discuss the resource consumption and timing analysis of the design implemented by the point to point system and wishbone shared bus system.

5.1. Point to Point Design Results

The design was implemented using point to point interconnection on Xilinx Spartan-6 FPGA. The design had one Master which was connected to around 400 to 800 registers.

5.1.1. Resource utilization

Resource utilization in this particular design was more based on the interconnect utilization. There is no logic utilization for the interconnection as we have for the bus and NoC design. In this case, resource utilization is calculated in terms of the interconnect utilization or simply the utilization of routing resources like wires and switches. It is difficult to calculate the wire utilization but we can approximate with some numbers. If data bus is 32-bit wide then 32 dedicated wires are going to each register. Total wires can be approximately 32 times the number of registers.

5.1.2. Timing Analysis

This design failed to achieve the timing constraint of 100MHz in a multi clock design with FPGA filled around 60-65%. The exact details of the design cannot be discussed as thesis is completed with a company and NDA is signed.

5.2. WISHBONE Shared Bus Design Results

This design is implemented in Xilinx Spartan-6 FPGA XC6SLX45T Speed grade -3. The results for the proposed design consist of one Master and multiple slaves. The design was tested for each slave having 8-64 registers and the overall design was tested for 8-32 slaves. Slaves are implemented in several different ways like distributed or block RAM. The design was also tested with empty FPGA containing only the wishbone shared bus design and the other case in which extra dummy logic is added to the design and the FPGA is filled 65%. Results of the proposed design include the resource utilization and timing analysis of the different design models.

5.2.1. Resource Utilization

In this section, I have discussed the total design resource utilization and also the resource utilization of individual components like master, slave and interconnect. Logic utilization for interconnect is important here, which is very low. Resource utilization for other entities like master and slave depends upon the design and implementation.

Total Resource Utilization

Table 2 shows the resource utilization of the total design in terms of Slice Reg and LUTs. It includes the Master, Slaves, Interconnect and System Controller. We can observe that if we double the number of slaves the total resource utilization doubles but if we keep the slave number constant and only increase the number of registers the total resource utilizations increases slowly.

Table 2 *Total Resource Utilization*

No. of Slaves	No. of Registers 32 bit	Slave with DRAM Slice Regs	Slave with DRAM LUTs
8	8	374	535
8	64	382	644
16	8	636	952
16	64	670	1287
32	8	1242	2336
32	64	1355	2810

Figure 5-1 shows the average percentage of resource utilization of each component. Detailed table for each component is in Appendix 4. We can see that slave utilizes the major area while the interconnect and master uses the least.

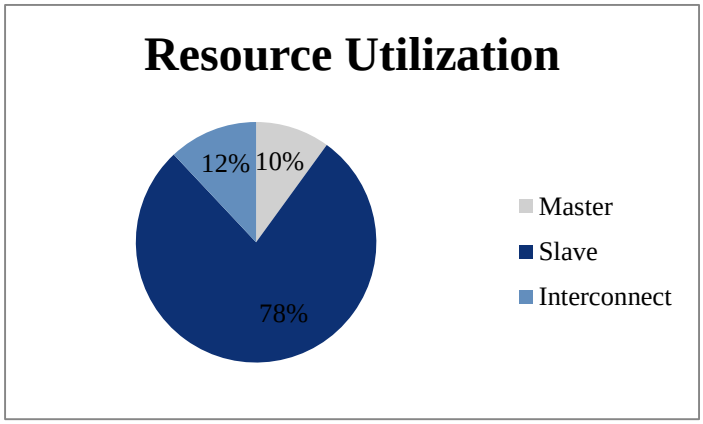


Figure 5-1 *Percentage of Resource utilization by each component*

Resource Utilization by each Component

Interconnect

The most important results are of interconnect utilization because other components are implementation dependant but the resource consumption for interconnect will remain the same. Interconnect resource utilization consists of the write back multiplexer, address comparator and some logic gates. Logic utilization for interconnect is negligible and wire utilization as compared to the design with point-to-point connection is very low. 32-bits for data, 1-bit for strobe, 1-bit for Write Enable and in the maximum case of 32-slaves and 64-registers the address is 11- bits. This makes in total 45-bits which means 45 wires

from Master to slaves. And those wires are shared by each slave. The important point to note here is such large design would be impossible with point-to-point interconnection. *Table 3* shows the resource consumption of interconnect for each design.

Table 3 *Resource Utilization by Interconnect*

Slaves	Registers	LUTs
8	64	64
16	64	128
32	64	328

Master

Master uses a very small number of resources and irrespective of the number of slaves it uses only 50-70 LUTs. The reason for such low resource consumption is that Master here only takes address and data from the test bench and forwards it to the slave. It acts as Dynamic Ram Access entity.

Slave

Table 4 shows the resource utilization for each design. Slave when implemented as synchronous read distributed memory. Detailed resource consumption can be found in Appendix 4.

Table 4 *Resource utilization by Slaves*

Slaves	Registers	Slices	Slice Regs	LUTs	LUT RAM
8	8	128	256	423	148
8	64	152	256	531	256
16	8	282	512	851	278
16	64	347	512	1083	512
32	8	606	1024	1950	565
32	64	746	1024	2417	1024

5.2.2. Timing Analysis

Timing analysis of the design is done by implementing slave as distributed RAM (DRAM) and Block RAM (BRAM) to analyse the impact of slave side on the timing of the design. Also as discussed earlier, the design was implemented on empty FPGA and the FPGA which was 64% filled with extra dummy stuff. Timing parameters are observed after the Place and Route.

Table 5 shows the timing analysis of the design in which slave is implemented with synchronous read DRAM and also BRAM on an empty FPGA. The design failed the timing constraint of 250 MHz when the number of slaves reaches 32 and the registers are 64. The critical path which failed the timing constraint is the Write Enable signal from Master to the slaves. The design with 32-slaves and 64- registers passed the target timing constraint of 200 MHz.

Table 5 *Timing Analysis of design with empty FPGA*

No of Slaves	No of Registers 32-bit	Slave with DRAM 250MHz	Slave with BRAM 250MHz
16	8	Passed	Passed
16	64	Passed	Passed
32	8	Passed	Passed
32	64	Failed	Failed

Table 6 shows the timing analysis of the design when target FPGA is filled with dummy stuff and FPGA is filled up to 64% and then following designs are implemented on the FPGA. This also shows the design containing 32-slaves with each having 64-Registers which makes in total 2048 registers passed the timing constraint of 200MHz even if the FPGA is 64% filled.

Table 6 *Timing analysis of design with 64% filled FPGA*

No of Slaves	No of Registers 32 bit	Slave with DRAM 200MHz	Slave with BRAM 200MHz	Slave with DRAM 250MHz
8	64	Passed	Passed	Passed
16	64	Passed	Passed	Failed
32	64	Passed	Failed	-

Timing Analysis Results

The design with point to point interconnection failed with the timing constraint of 200 MHz when the design only had 400 registers. As we can see in *Table 6*, wishbone shared bus design with 32-slaves and 64-registers which makes in total of 2048 registers passed the timing constraint of 200 MHz.

5.2.3. Critical Path

I will discuss the critical path of the design which failed to pass the timing constraint of 250 MHz. Critical path in most cases is the control signals going from the Master to the slaves. Critical path for the design with 32-Slaves and 64-Registers is Write Enable signal going from Master to all the Slaves. This signal has big fan out as compared to the other signals and connected to all the slaves directly without any logic in between. Otherwise the goal of the design is to fulfil the timing constraint of 200 MHz which it did successfully.

5.2.4. Latency and Throughput

Latency

Latency for writing is one clock cycle and for reading from RAM is one clock cycle. If write back path from slave to master with multiplexer in between is registered / pipelined than latency for reading is two clock cycles.

Throughput

Writing

Throughput for writing 32-bit data, latency is one clock cycle and clock is 200 MHz

Throughput= (Frequency/Latency)*Data size per cycle
= ((200x10^6 Cycles/sec) ÷1) ×32- bits/ Cycle
= 762 MB/s
If clock is changed to 250 MHz then
Throughput = 953 MB/s

Reading

Also throughput for reading 32-bit data, one clock cycle latency and 200 MHz clock
Throughput = 762 MB/s

And for 250 MHz clock
Throughput = 953 MB/s
If latency is two clock cycles then throughput will reduce to 381 MB/s.

Combined for Reading and Writing

Combined throughput for complete writing and reading process is when you write in one cycle and read in next cycle. Latency is 2 clock cycles for complete process and clock is 200 MHz.

Throughput= (Frequency/Latency)*Data size per cycle
= ((200x10^6 Cycles/sec) ÷2) ×32- bits/ Cycle
= 381 MB/s

This ends the discussion of the performance analysis and the timing constraints of the design.

6. CONCLUSION

This chapter summarizes the master thesis work and it also includes the future direction and room for improvement in this work

6.1. Conclusion

In this thesis, on chip communication architectures are analysed, which included point to point interconnection, shared bus and Network on Chip. Important on chip shared buses like Wishbone, CoreConnect, Avalon, AXI and AMBA are discussed and compared with each other. Company's previous design was implemented using point-to-point interconnection and causing timing problems. It is analysed in this thesis and alternate approaches like shared bus and Network on Chip's feasibility are discussed to solve the timing problems and interconnect resource utilization. Shared bus is identified as potential interconnect to solve the problem. There are different shared bus models available like single shared bus, ring bus and hierarchical bus. The single shared bus model is selected because design consisted of single master and multiple slaves. There were different shared bus standard available for single master and multiple slave design. Out of the different shared bus standards, Wishbone bus is selected to solve the particular problem because it is simple, flexible and open source.

Wishbone shared bus based design is implemented, which can support the design of one Master connected to Multiple Slaves. The design with point-to-point interconnection failed to meet the timing constraint of 200MHz with 400 registers. New proposed Wishbone shared bus design successfully met the timing constraint of 200MHz with 5 times the number of registers than the previous design. Interconnect utilization by the wishbone shared bus is significantly lesser than the point-to-point design, which used dedicated connection to each entity while the new design used the shared bus. A shared bus uses more logic resources than point to point interconnection, but it improves the timing, reduce interconnect utilization and is scalable.

6.2. Future Work

Shared bus design is optimized to achieve a timing constraint of 200 MHz but it can be further optimized in future to pass timing constraint of higher frequencies. The control signals from Master to slaves are the critical path and bottleneck which can be pipelined. Built-in tool features for optimizing can also be used.

REFERENCES

[1] M. Mitic and M. Stojcv, "An Overview of On-Chip Buses," FACTA UNIVERSITATIS, Niš, 2006.

[2] S. Pasricha und N. Dutt, On-Chip Communication Architectures: System on Chip Interconnect, Morgan Kaufmann, 2010.

[3] R.UMA, V. Vijayan, M. Mohanapriya and S. Paul, "Area, Delay and Power Comparison of Adder Topologies," *International Journal of VLSI design &*, vol. 3, no. 1, p. 161, February 2012.

[4] H. W.H. and P. T.M., "A Design Methodology for Efficient Application-specific on-chip," *IEEE Trans. Parallel Distributed System*, vol. 17, no. 2, pp. 174-179, February 2006.

[5] M. Sharma and D. Kumar, "WISHBONE BUS ARCHITECTURE – A SURVEY AND COMPARISON," *International Journal of VLSI design & Communication Systems (VLSICS)*, vol. 3, no. 2, pp. 107-124, April 2012.

[6] T. S. Mak, P. Sedcole, P. Y. K. Cheung and W. Luk, "ON-FPGA COMMUNICATION ARCHITECTURES AND DESIGN FACTORS," Department ofComputing, Electrical and Electronic Engineering, London, 2006.

[7] IBM, "IBM Microelectronics," [Online]. Available: http://www.ibm.com/chips/products/coreconnect. [Accessed 29 05 2014].

[8] ARM, "Arm. amba specifications v2.0. ARM," [Online]. Available: http://www.arm.com.

[9] ALTERA, "www.altera.com," [Online]. Available: www.altera.com/literature/manual/mnl_avalon_spec.pdf. [Accessed 24 06 2014].

[10] OpenCores, "WISHBONE OpenCores," 2014. [Online]. Available: http://cdn.opencores.org/downloads/wbspec_b4.pdf. [Accessed 14 May 2014].

[11] Open Cores, "Open Cores," [Online]. Available: http://opencores.org/. [Accessed 14 August 2014].

[12] W. Wong, "Electronic Design," Penton Electronics Group, 17 July 2012. [Online]. Available: http://electronicdesign.com/fpgas/understanding-fpga-processor-interconnects. [Accessed 01 June 2014].

[13] A. Imbewa, "Design Space Exploration of FPGA-Based NoC Routers," University of Windsor, Windsor, 2012.

[14] Z. Marrakchi, H. Mrabet, U. Farooq and H. Mehrez, "FPGA Interconnect Topologies Exploration," *International Journal of Reconfigurable Computing*, vol. 2009, no. 259837, p. 13, 2009.

[15] J. Rose und D. Hill, „Architectural and physical design challenges for one-million gate," in s *Proceedings of ACM/SIGDA International Symposium on Field Programmable*, 1997.

[16] M. I. MASUD, „FPGA ROUTING STRUCTURES: A NOVEL SWITCH BLOCK AND DEPOPULATED INTERCONNECT MATRIX ARCHITECTURES," The University of British Columbia, Columbia, 1998.

[17] S.Phillips, „VictoriaFalls: Scaling highly-threaded processor cores," in s *HotChips*, 2007.

[18] J. Shandle, „EETIMES," 24 09 2002. [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1275901. [Zugriff am 01 06 2014].

[19] S.Brini, D.Benjelloun und F.Castanier, „A flexible virtual platform for computational and communication architecture exploration of DMT VDSL modems," in s *DATE*, 2003.

[20] M. Harnisch und Doulos, „DOULOS," 2010. [Online]. Available: http://www.doulos.com/knowhow/arm/Migrating_from_AHB_to_AXI/. [Zugriff am 2014].

[21] XILINX, „AXI Reference," Xilinx, 2012.

[22] Diolan, „Diolan," 2013. [Online]. Available: https://www.diolan.com/dln_doc/spi-bus-interface.html. [Zugriff am 2014].

[23] M. K. Namork, „Development of a test system for Network on Chip," Norwegian university of Science and Technology, 2011.

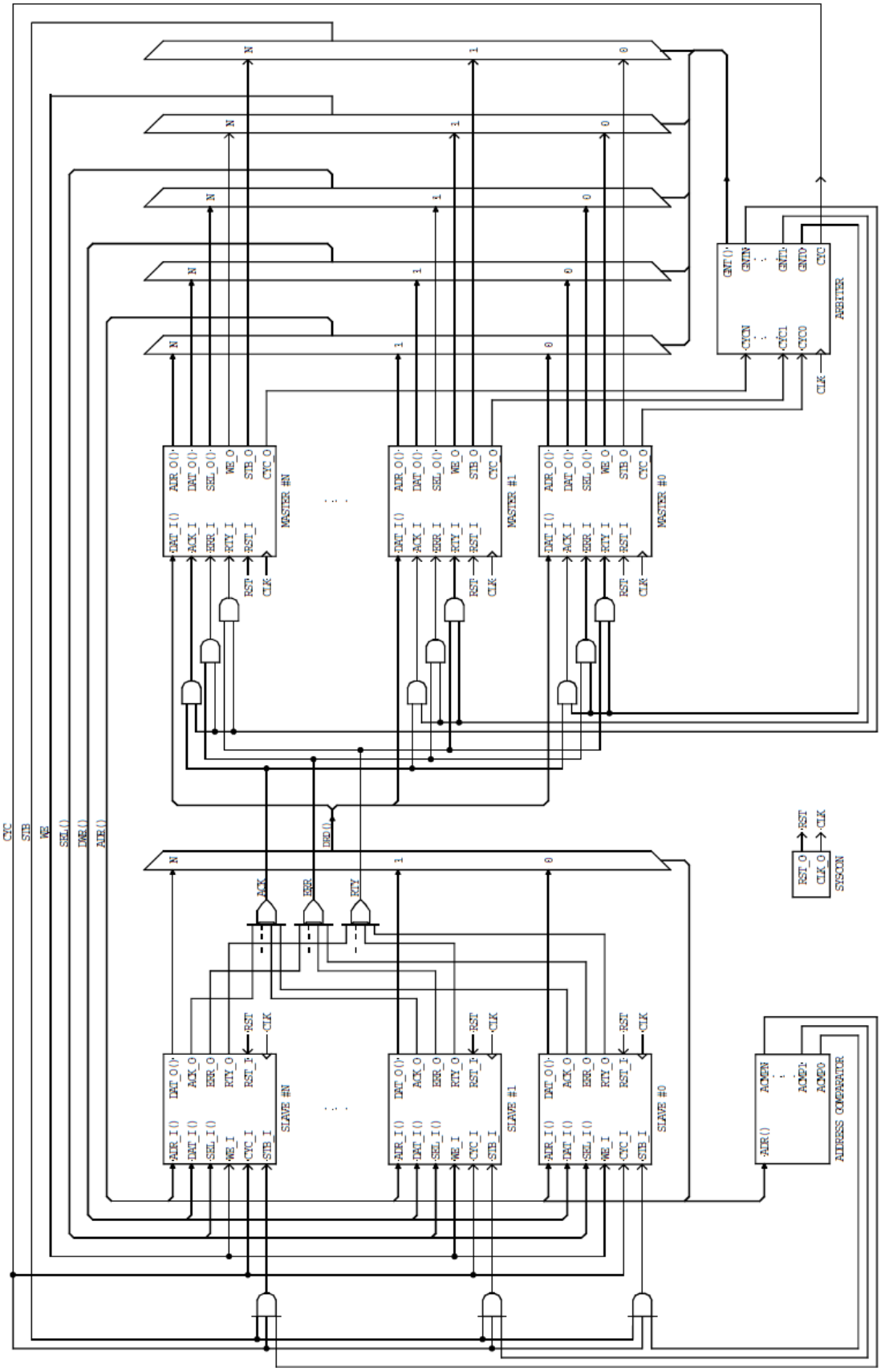
[24] Arteris, "A comparison of Network-on-chip and Busses," Arteris, Guyancourt Cedex, 2005.

[25] J. Alam, „Academia," 2005. [Online]. Available: https://www.academia.edu/4102154/DESIGN_AND_IMPLEMENTATION_OF_EFFICIENT_BUS_ARCHITECTURE_FOR_SYSTEM-ON-CHIP. [Zugriff am 2014].

[26] Silicore Corporation, "OpenCores," October 2001. [Online]. Available: http://opencores.org/websvn,filedetails?repname=uart_block&path=%2Fuart_block%2Ftrunk%2Fdocs%2FWishbone+Public+Domain+Library%2FWBVHDLIB.zip. [Accessed June 2014].

[27] M. Khellah, S. Brown and Z. Vranesic, "Minimizing Interconnection Delays in Array-based FPGAs," IEEE 1994 Custom Integrated Circuits Conference, San Diego, 1994.

APPENDIX 1
WishBone MxN Shared Bus



APPENDIX 2

Area utilization by each component

SLAVES	REGISTERS	INTERCONNECT		MASTER			SLAVE 1				TOTAL SLAVES			
		Slices	LUTs	Slices	Slice Reg	LUTs	Slices	Slice Reg	LUTs	LUT RAM	Slices	Slice Reg	LUTS	LUT RAM
8	8	32	64	56	118	48	17	32	54	19	128	256	423	148
8	16	32	64	62	132	45	17	32	54	19	128	256	423	148
8	32	32	64	58	118	49	16	32	53	18	123	256	417	142
8	64	32	64	62	126	49	19	32	67	32	152	256	531	256
16	8	32	128	57	124	39	26	32	62	17	282	512	851	278
16	16	32	128	59	137	48	26	32	62	17	284	512	851	278
16	32	32	128	59	146	56	26	32	62	17	284	512	851	278
16	64	32	128	65	158	49	30	32	76	32	347	512	1083	512
32	8	181	328	75	218	58	32	32	91	17	606	1024	1950	565
32	16	177	328	87	246	52	31	32	91	17	603	1024	1950	565
32	32	174	328	94	294	73	31	32	91	17	604	1024	1960	575
32	64	172	328	106	331	65	35	32	106	32	746	1024	2417	1024

Timing Report for the whole design

No of Slaves	No of Registers (32 bit)	Slave with DRAM 10ns	Slave with BRAM 10ns	Slave with DRAM 5ns	Slave with BRAM 5ns	Slave with DRAM 4ns	Slave with BRAM 4ns
8	8	6.393	5.864	4.728	4.937	3.952	3.870
8	16	6.604	6.684	4.879	4.897	3.944	3.925
8	32	5.482	7.041	4.760	4.759	3.927	3.938
8	64	5.639	6.537	4.926	4.957	3.964	3.927
16	8	6.136	5.377	4.915	4.939	3.958	3.963
16	16	6.219	7.421	4.942	4.949	3.969	3.939
16	32	7.511	6.727	4.856	4.917	3.891	3.969
16	64	7.288	6.813	4.957	4.918	3.955	3.956
32	8	8.974	6.164	4.969	4.922	3.961	3.966
32	16	6.961	6.665	4.965	4.889	3.975	4.612
32	32	8.884	9.499	4.953	4.962	3.980	4.644
32	64	8.857	8.397	4.961	4.968	5.318	4.691