University of Cincinnati

Date: 10/2/2013

I, Kishan Sarpangala , hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Engineering.

It is entitled:

Arbitration Techniques for SoC Bus Interconnect with Optimized Verification Methodology

Student's name: **Kishan Sarpangala**

This work and its defense approved by:

Committee chair: Carla Purdy, Ph.D.

Committee member: Raj Bhatnagar, Ph.D.

Committee member: George Purdy, Ph.D.

6958

Arbitration Techniques for SoC Bus Interconnect with Optimized Verification Methodology

A thesis submitted to the

Division of Research and Advanced Studies

at University of Cincinnati

for partial fulfillment of the

requirements for

MASTER OF SCIENCE

in the School of Electronic and Computing Systems

of the College of Engineering and Applied Sciences

Oct 2, 2013

By

Kishan Sarpangala
B.Tech, Electronics and Communication Engineering,
Visvesvaraya Technological University, 2010

Thesis Advisor and Committee Chair: Dr. Carla Purdy

Abstract

The die size of the VLSI chip has been drastically decreasing over the years while the density has been increasing exponentially with more functionality squeezed into a smaller space. The need for manufacturers to have an edge in the market and to keep up with Moore's law are the driving factors which motivated the industry to move towards SoC (System on Chip). There is more on-chip integration of different IP (intellectual property) cores, in a smaller space, with more functionality being included. This leads to problems in designing and verifying the entire design. In particular, verifying the SoC bus interconnects is difficult and presents major challenges especially with respect to time to market. One aspect of this verification involves power usage. As the complexity of the SoC increases in the communication architecture, the power consumed becomes a major concern. In terms of power use the most important component in any communication architecture is the bus interconnect and the arbiter is the major component in the bus interconnect.

In this research, an effective arbiter for the bus communication which supports six priority policies will be designed. It's named Blue-Jay. Arbitration policies are applied using one of the three approaches to data multiplexing- transfer, transaction, and desired transfer length. Hence, there are total of eighteen possible arbitration schemes. To verify the functioning of our design we have developed an optimized verification methodology, based on formal methods and System Verilog. Our results show higher reliability than the commonly used AMBA arbitration scheme.

Acknowledgements

Firstly, my sincere Thanks to my advisor, Dr. Carla Purdy for her consistent support and guidance throughout my thesis which enabled me to complete my thesis easily without much hurdle. I am sure we have exchanged minimum 70 emails and phone conversations over the past few months, this speeded up my research. Thanks to her, the embedded class became interesting enough that I didn't dare miss even one of her classes. I would also like to thank Dr. George B Purdy and Dr. Raj Bhatnagar, they have been kind enough to agree to be part of the committee as well as spend their valuable time in proofreading my work; I thoroughly enjoyed their classes as well.

Thank to my parents for being there for me during difficult times and always supporting, motivating me throughout my life. They are my back bone. My parents have done lot of sacrifices through the years and I am genuinely in debt for these kindness and actions. Also sincere thanks to my one and only sister for her constant support mentally and emotionally, without which I wouldn't have been able to graduate from this University. She has been a constant support from my school days, a person who I can rely on, who would share her ears for my troubled heart even during late hours, never failing to quip and make valuable suggestions.

My lab mates especially Shaival Jhaveri made my graduate days so much fun. Thanks to Shaun K peter, Rissen Joseph and Akhil Kalathungal; as well as my other friends who have made my masters pleasurable.

Table of Contents

List of Illustrations

List of Tables

List of Abbreviations

1) SoC – System On Chip
2) DSP – Digital Signal Processor
3) AMBA – Advanced Microcontroller Bus Architecture
4) TDMA – Time Division Multiple Access
5) RR – Round Robin
6) GDS II – Graphic Data System II
7) UVM – Universal Verification Methodology
8) ICC – Integrated Chip Compiler
9) SVA – System Verilog Assertions
10) SP – Static Priority
11) AHB – Advanced High-performance Bus
12) SAIF – Switching Activity Interchange Format
13) TSMC – Taiwan Semiconductor Manufacturing Company
14) UVM – Universal Verification Methodology
15) UART – Universal Asynchronous Receiver/Transmitter
16) DMA – Direct Memory Access
17) INC – Increment
18) CLR – Clear

Chapter 1

INTRODUCTION

System on Chip (SoC) is a combination of processors, memory units, peripherals and bus interface. These units are tailored to meet the needs of a particular specific application domain.

The complexity of designing a system on chip is rapidly increasing. It's a common trend to see a micro architecture which is specific application based being crammed into a small die space. The main reason for the semiconductor industry to adopt the system on chip is that there is a steep reduction in overall cost in system design. SoC guarantees there is increased performance and also the power consumption by the overall system is reduced. Different semiconductor industries lease out their intellectual property either in soft intellectual property or in hard intellectual property format to other companies. These intellectual properties are then incorporated together as a system on chip and sold as one high performance unit. Chip based design has been given precedence over the traditional board based design. In Figure 1.1 the system on chip evolution is depicted. In Figure 1.2 board based design to chip based design is depicted, the common trend in semiconductor industry for the past few years.

We can see that Moore's law is being extended here. Consumers are demanding cheaper products which have more in built functionality. Most importantly they are asking for a better battery life. The work of verifying the design and the complexity of testing increase drastically. Time to market is the most important factor for chip manufacturers; the engineers are trying constantly to decrease this time frame. It takes more time and also more cost to fix a bug if it is found when the design is mature. Hence it's very important that during the early stages of a design, we find as many bugs as possible. [1] This can be achieved by adopting better verification methodology, where there is a mix of traditional testbench approach combined with formal verification.

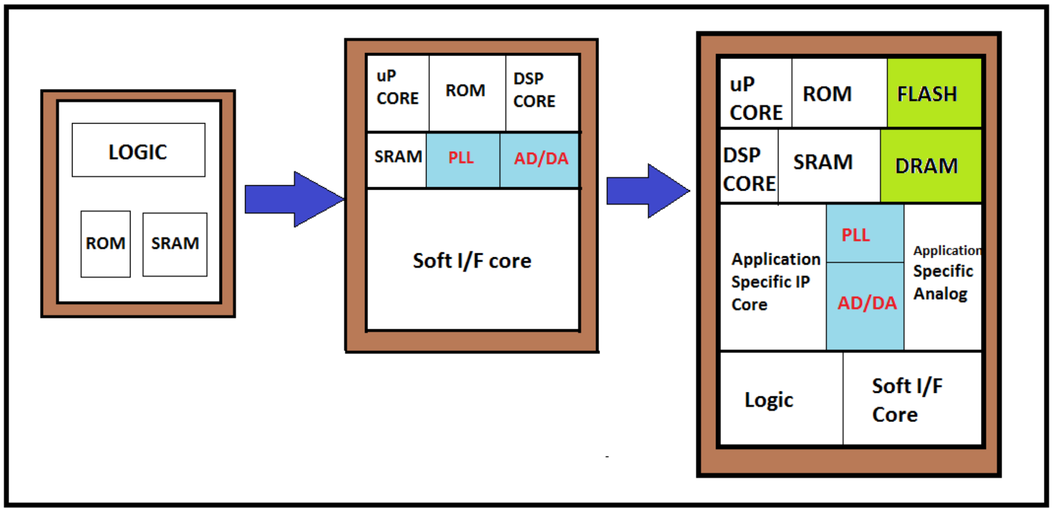


Figure 1.1: Evolution of SoC

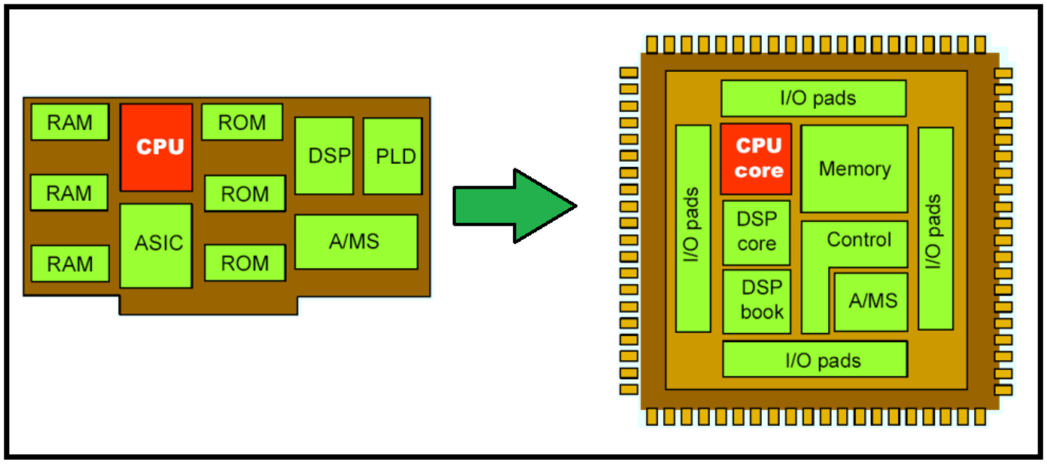


Figure 1.2: Board based design to chip based design

1.1 System on chip design

For this research work a single core is taken which is obtained from open-core website [1]. This single core is replicated to form a quad core. The cores are connected through a bus matrix interconnect design. An effective arbiter for the bus communication will be designed which supports six priority policies. The arbiter is named Blue-Jay. These policies are applied using one of the three approaches to data multiplexing- transfer, transaction, and desired transfer length. Thus eighteen possible arbitration schemes are present in total. We provide a better design for the bus interconnect arbiter with a slave side arbitration scheme. The performance is compared with master side arbitration schemes. We also compare the number of transistors synthesized and the power consumption of different arbiters.

To verify the functioning of our design we have developed an optimized verification methodology, based on formal methods and System Verilog. Our results show higher reliability than the commonly used AMBA arbitration scheme.

1.2 Bus matrix interconnect design

We will design a bus matrix interconnect design which uses extensively the slave side arbitration methodology. This kind of arbitration scheme is quite different from the traditional master side arbitration approach. The changes are described in chapter two. In this project a flexible arbiter with multiple layers of bus matrix which supports slave side arbitration is designed. The arbiter supplies the following protocols:

1. Pre-emptive Static Priority

2. Fixed Priority Arbitration with Hold Control
3. Round-Robin
4. Dynamic Priority
5. TDMA
6. TDMA/RR

The main aim is to ensure that the master will give maximum performance. This is possible with the addition of data multiplexing, transfer transaction and desired transfer length from the master. These three strategies are used together with any of the supported protocols. The performance will be compared with master side arbitration. The benchmarks which will be the foundation for comparison will take into account the number of requests that can be handled, the different data transfer modes that can be supported and also the type of decoding scheme which will be incorporated.

Architecturally it's always a better design to have a system on chip bus interconnect with multiple active agents. It's also better to design active agents which require higher bandwidth. This will increase the overall capacity to transmit information which is possible only with the use of an efficient protocol. This protocol will support correct non-interfering message transmission and increase efficiency.

The design is simulated using the Synopsys tools Modelsim and Verdi. Synthesis was done by Xilinx and Synopsys Design Compiler. The layout is done through Synopsys ICC (Integrated Circuit Compiler). Primetime was used extensively to make sure the timing closure is met. The formal verification tool Magellan was used. Power Compiler was used to calculate power consumption.

Verdi helped extensively to resolve X propagation issues within the design and also to connect the dangles between different components/components. Descriptions of all these Synopsys tools can be found at [19].

1.3 Testbench and formal verification methodology

A test bench methodology is depicted in Figure 1.3. This is the most common methodology adopted by the industry. Engineers need to exhaustively verify the logical correctness of SoC designs before the GDSII is sent to foundry where manufacturing happens. Despite this, however, bugs do easily manage to creep into the system on chip bus interconnect designs. Simulation alone can't verify the system on chip bus interconnects. To increase coverage, asserting test cases and efficient test case generation is needed. This drives the need to generate as many test cases as possible. They can be specific or random in nature. A simple test bench methodology is depicted in Figure 1.3. A more complex and efficient design is needed for corner cases in actual designs.

We can find bugs through proper test-bench design. But we need to generate test cases which feed into other blocks, that's more effective [2]. The work load is reduced as we start to integrate design blocks which stimulate each other. Multiple block simulations will help us to uncover more bugs, but the downside is that the test will run slower. The main idea is to reduce the simulation dependence, so that at the top level of design under test, the entire design will be tested. Simulation performance will be reduced significantly [2]. Various blocks must be tested, the test cases should strive to test all blocks while performing activities concurrently. Basically it means that we should keep the input-output ports in active mode, the caches should be used, the main processors should crunch data, and the memory should be refilled. This guarantees that timing bugs are bound to occur. All this action of data alignment will corrupt data [2]. Therefore we need to use formal verification.

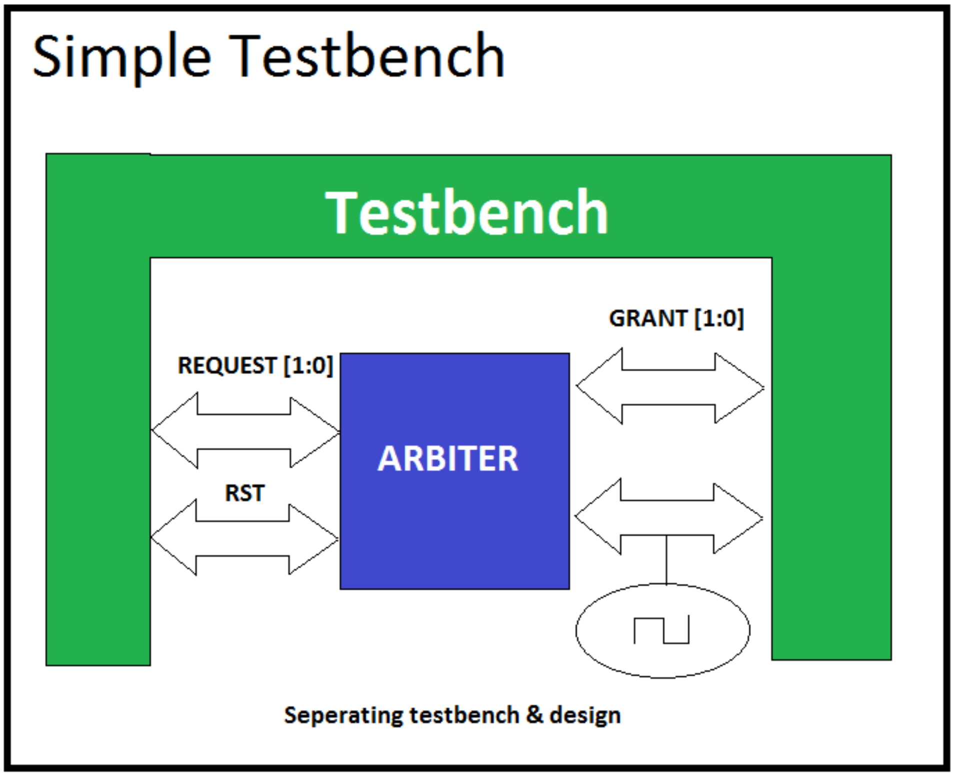


Figure 1.3: Simple test bench [2]

Formal verification works on the principle of bug hunting by using guide posts. This is the advanced methodology used in the industry. These guideposts are user specified and will change with the design changes. This property is expressed as cover property and it’s enabled through assertions usually written in System Verilog. The overall coverage flow for verification is depicted in Figure 1.4.

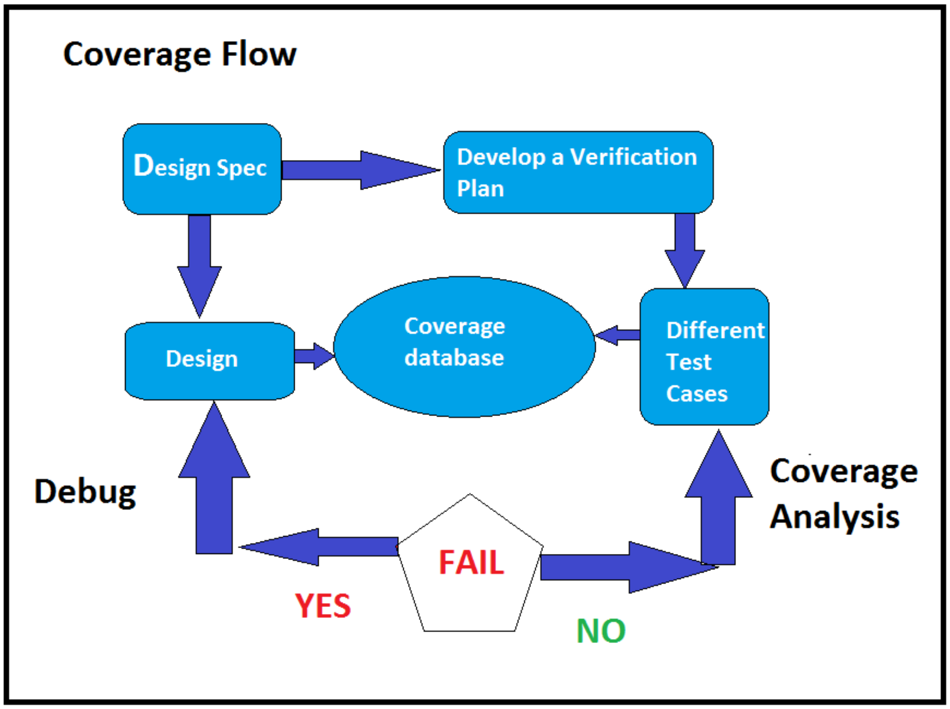


Figure 1.4: Coverage flow [4]

Every property debugging the system must be associated with a guide post. [4] This approach is depicted in Figures 1.5, 1.6 and 1.7.

SVA (System Verilog Assertions) cover property defines how the guidepost needs to be used by the user. It is always expected that through these SVA properties we can define the system. Every guidepost is associated with some property. [4] In this thesis work the Magellan tool from Synopsys has been used to define the guideposts. Coverage modules are being used by the Magellan tool to implement guideposts. A point to be noted is that we cannot use property based modules to implement/design guideposts. We can use SVA covers which will help to specify and design the guidepost so that the coverage goals can be achieved easily. Magellan assumes that the guidepost coverage goals you define are easier to reach than the properties they are intended to verify. [13]

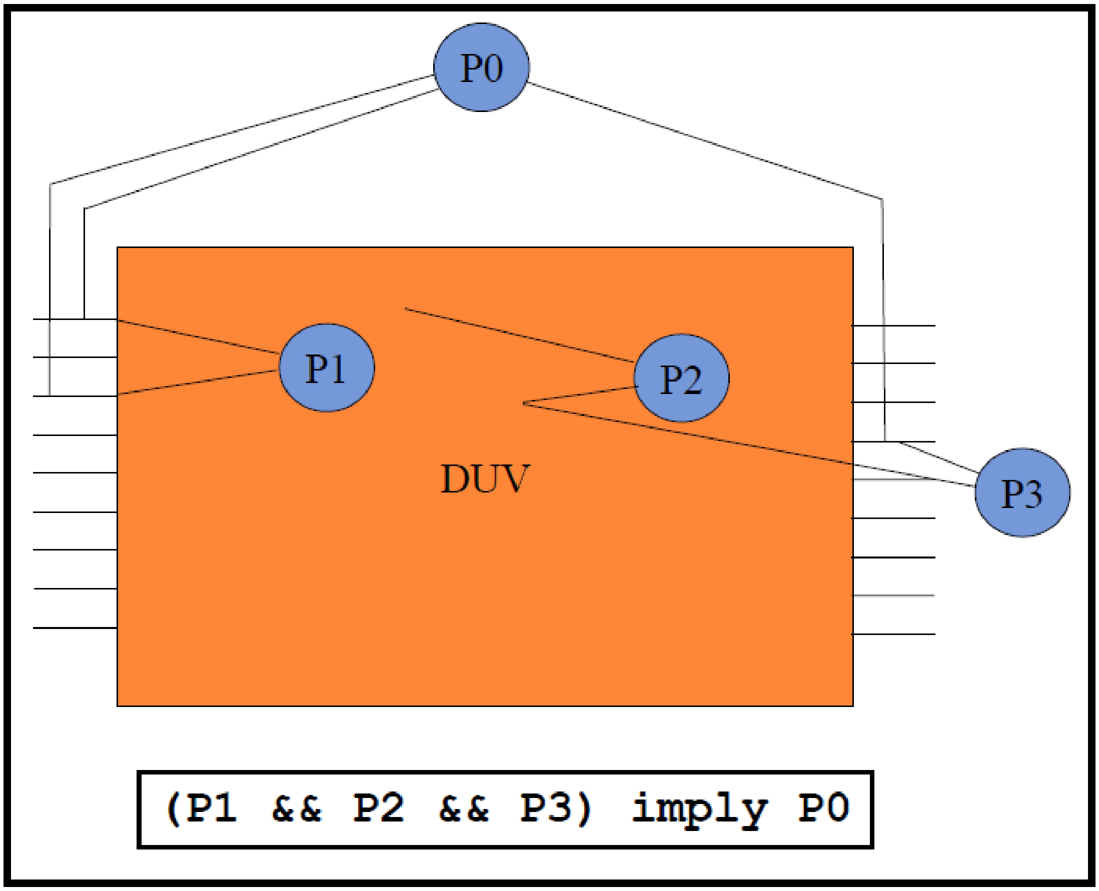


Figure 1.5: Reduce the complexity through specification partitioning [4]

A potentially useful guidepost is one that is close to the property in the state space of the design. [13].

1.4 Verification approach

Verification of bus interconnects in a system on chip is a major challenge. Formal verification helps to pace up the time to market the product by extensively resolving and debugging corner cases. To verify the following features like virtual address space, power management features, also the correct routing of transactions and bus protocol translations while still trying to complete the project milestones is not an easy task but a difficult one. [3]

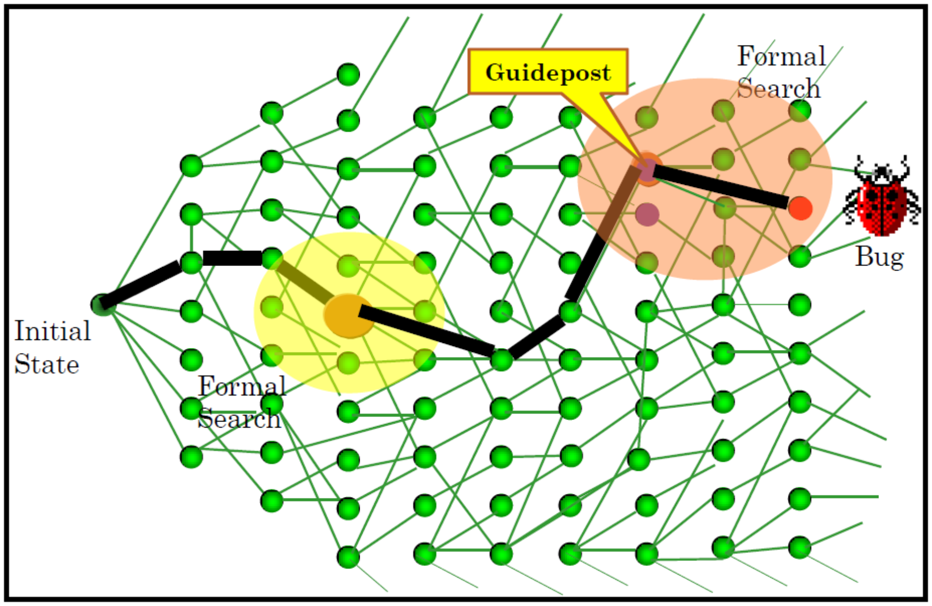


Figure 1.6: Bug hunting with guidepost [17]

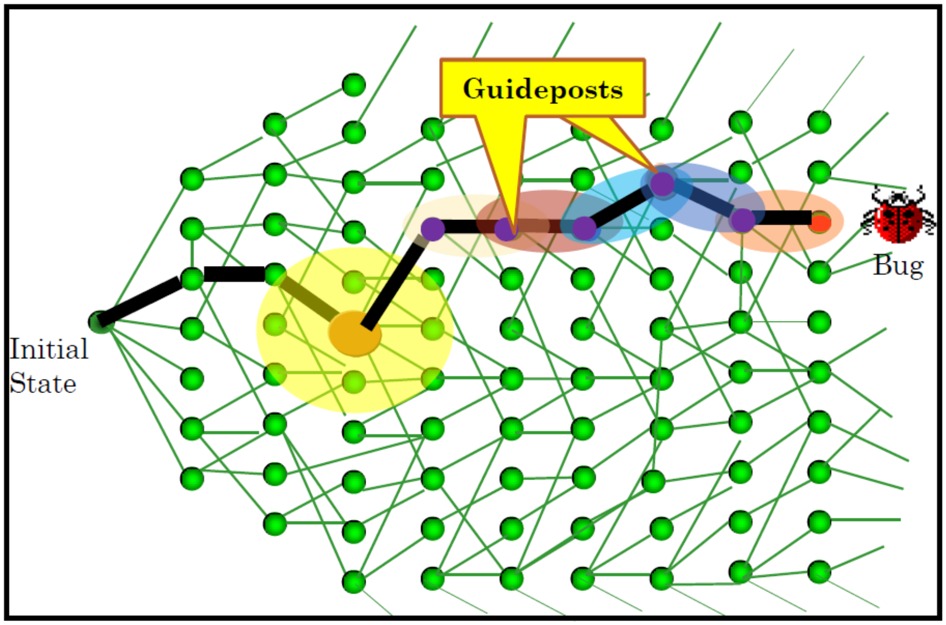


Figure 1.7: Sequence of guideposts expands reach of formal search [17]

1.5 Thesis organization

To the best of my knowledge, this is the second work with respect to modeling for power estimation in arbitration techniques for SoC bus interconnect. The only other work, by Srinivasan et al. is for single core [15]. This thesis not only estimates and models the power consumption for quad core but also verifies the design using formal tools while making sure there is no X propagation. It expands on the work in [15].

In Chapter 2, we briefly discuss, bus based communication architecture's fundamental concepts and different data transfer modes and survey the available papers on these topics.

In Chapter 3, we discuss and analyze the general implementation flow adopted for this research work.

In Chapter 4, different simulation results verify the intended design. Synthesis and power Results are analyzed and discussed. The design is compared with other arbiters.

In Chapter 5, conclusions and future work are discussed.

Chapter 2

BACKGROUND

2.1 Bus based communication architecture's fundamental concepts

In any system on chip (SoC), it's through the bus communication architecture different components are dictated. Through this complex bus architecture the data can be sent efficiently to all the components of a SoC. Thus bus based communication is the preferred interconnection mechanism used in complex SoC of today. This architecture becomes the channel. Also this is a shared medium. Parallel bus or a single bus can be designed based on functionality for better efficiency. It's a norm to use parallel bus in today's designs and this is extensively leveraged in today's system on chip architectures. While the rest of the components ignore this particular data which is send through the shared bus. Data send in the shared bus acts more like a broadcast center. Only a particular component only is connected through this bus is supposed to consume the data sent in the shared bus. The data are sent on the bus by an individual component and arrive at the destination component. An acknowledgement is sent back to the transmitting component to indicate the arrival. [10] The communication transaction, which supports temporal (duration and sequence of messages exchanged) and spatial (eg, message size) characteristics, is called a bus protocol. The shared bus communicates with various components through this protocol. The protocol enables smooth receiving and transferring between different components in the bus architecture. The protocol is able to handle multiple requests of sending (or receiving) at the same time. [10] One or more shared buses along with logic components that implement the details of a particular bus protocol form the basis of a bus-based communication architecture. [10]

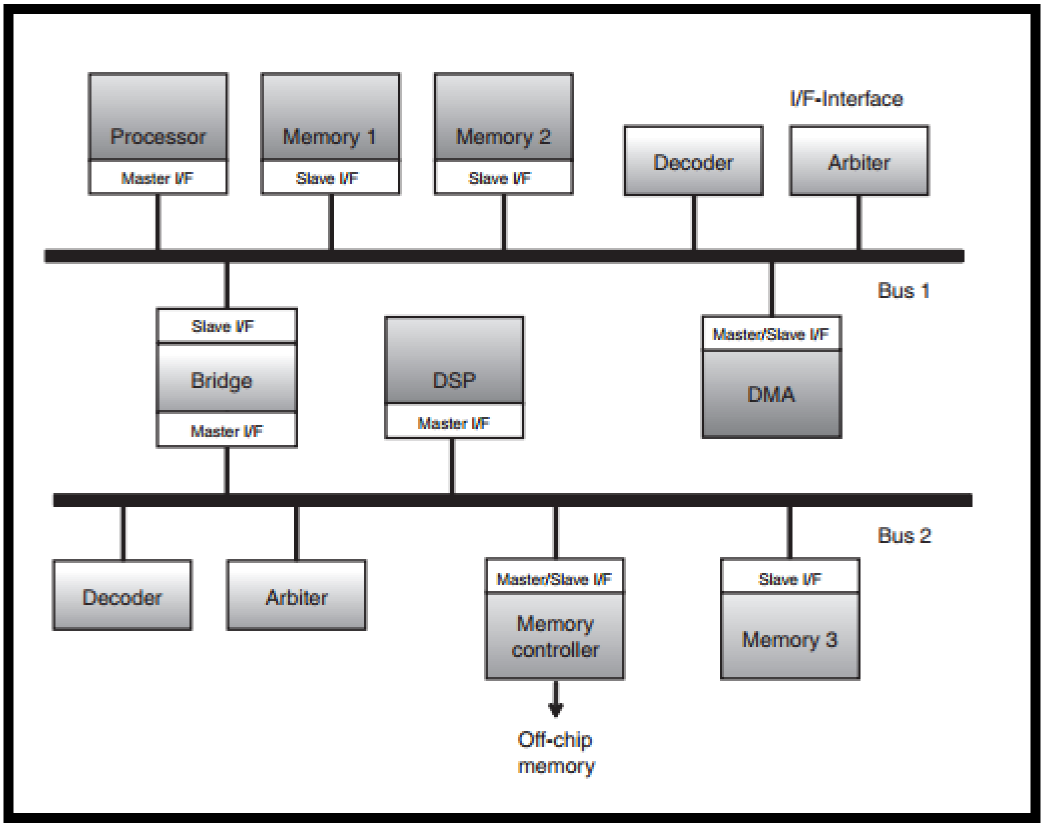


Figure 2.1: SoC with bus-based communication architecture [10]

There are different components which are connected to a bus based communication system. The components which initiate and control read/write data transfers are referred to as masters. Digital signal processing units and the core processors house the bus masters. Master port is a set of signals which connects the master with the shared bus. Slaves are connected to the shared bus and respond to requests by the master. They cannot initiate data transfers on their own. Now to improve the communication we will have buffers along with complex voltage and frequency converters whose sole job is to improve the performance time. These complex components form part of the interface, which can be very simple, such as a set of wires, or more complex depending on the complexity of the bus. This interface's sole purpose is to be work with masters and slaves and connect them with the bus [10]

There are some cases where we have to design hybrid components. Master and slave properties can be combined in a hybrid component. DMA (Direct Memory Access) is an example of a hybrid component. The DMA is represented in Figure 2.1. DMA (direct memory access) allows the processor to write and also read from a register unit. DMA must be configured to write or read mode based on requirement. The requirement can be memory read or memory write. As a master port, DMA will initiate and control data transfers between memory blocks (which would otherwise have been managed by the processor). As a result the processor is freed up to perform other activities, which typically improves system performance. [10] Similarly, the MC or the memory controller has both master and slave ports. Through DSP, the MC can be set or the MC's port configuration can be changed to either slave or master mode. Once the MC is configured, it can initiate and control data transfers using its master port. [10]

We also need bridges and most importantly arbiters for this bus communication to be successful. This allows two or more buses running in parallel, connected through bridges.

The bridges can be run by the same protocol or by different protocols. Here the implementation will include different protocols. Both the buses operate in different frequencies. This can create lot of issues. Let's consider a scenario for Figure 2.1. How does Bus number 1 communicate with

Bus number 2? Control data transfer need to be initiated initially, possibly with the help of The Direct Memory Access and Processor component on Bus number 1. Through these components data is sent to the Bus number 2 via the Bus number same port. During this transfer, the master port on Bus number 2, sends the necessary data to the destination. A single bridge will work as the DSP unit and MC will not initiate or control the data transfers between the components on Bus number 1. This is shown in Figure 2.1. Another bridge which has the slave port on Bus number 2 and a master port on Bus number 1 would have been required if the mentioned components had to transfer data to Bus number 1. [10]

While performing the data transfer the decoder will decode the destination address from the master and make sure the appropriate slave receives the data necessary. The decoder can be a separate logical component or a component integrated with the interface. If multiple masters request access to the shared bus, the arbiter; will need to determine who should be granted access to the shared bus. Usually there will be some sort of priority scheme supported. Which particular data transfer is critical for the system is determined by this scheme.

The thesis is based on different arbitration schemes available in SoC which are discussed next.

2.2 Arbitration

Arbitration Often different masters who are connected to the same shared bus make requests at the same time. The shared bus can support or initiate only one particular data transfer time.

Which master gets the authority to make that particular data transfer and which masters have to wait in a queue is decided by the arbitration mechanism. To start that particular data transmission the arbiter signals a particular master to go ahead only then will the master.

A distributed centralized arbiter can be implemented just like a decoder. Figure 2.2 is a distributed configuration of an arbiter. [10] Several arbitration schemes are used in the bus based communication. An arbitration scheme must ensure critical data transfer with no starvation and ensure a fair amount of freedom to access. Establishing fixed values to the masters is a solution, this provides a static priority (SP) scheme. SP schemes can be preemptive or non preemptive. The different arbitration schemes include

1) Pre-emptive SP
2) Fixed priority arbitration with hold control
3) Round-robin (RR) arbitration
4) (TDMA) time division multiple access

Pre-emptive SP implementation gives importance to critical data transfers. In this scheme a current ongoing data transfer is terminated immediately if a higher priority master makes a request. SP scheme is simple to design. [10] However, there are some issues with this scheme. This type of design needs to be implemented carefully to make sure there are no starvation among other lower priority masters. Overall effectiveness of the arbitration scheme will be spoiled if there is frequent access of the higher priority masters to the bus. [10] Pre-emptive SP can also be called fixed arbitration. Each of these masters is capable of initiating bus request signals. For example let Master 0 of AHB bus system be the default master, this bus system has 5 masters in total. [9] In addition there is a possibility that under certain circumstances the arbiter may allocate ownership of the bus to a dummy master. Fixed priority scheme is depicted in Table 2.1, determines the priority between the alternate masters. [9] HMASTER signal gives the priority for the master, one being the highest priority. HSPLIT signal decides the point at which data is split.

If no master makes a request from the shared bus, it's usually granted to Master 0. Master 0 is meant to be used by CPUs only, as it's a very greedy master and demands the shared bus all the time. Suppose a bus cycle is terminated due to a retry response, the arbiter can be re-allocated to a more significant master who will now have access to the shared bus which it was requesting earlier. [9] If there are no more significant masters making requests to the shared bus, the arbiter will continue to allocate the shared bus to the master that required retrying the transfer of data. If a bus cycle is terminated with split response, there is a high probability that the arbiter will

Priority **Description**	**HSPLIT**	**HMASTER**	**Description**
1	4	0100	AHB Master 4
2	3	0011	AHB Master 3
3	2	0010	AHB Master 2
4	1	0001	AHB Master 1
5	0	0000	AHB Master 0
6	X	1111	Dummy Master

Table 2.1: Fixed priority scheme [9]

reallocate the shared bus to any alternative master which had been currently requesting the shared bus. This is a common situation if the master is of lower priority than the master which had initiated the split transfer. Then the corresponding lock signal is asserted and made high if a master makes a locked transfer. Until all of the transfers which have been initiated are complete the arbiter will not reallocate the shared bus entity. [9]

Fixed priority arbitration with hold control: Except when a fixed length burst transfer involves a master this arbitration is very similar to fixed arbitration. The arbiter will not change the currently selected master until the penultimate transfer of the burst since only then will the master de-assert its grant request output. The grant outputs will change normally when the master ends the burst unless the burst is terminated early due to a split/retry transfer. [9]

If we need no starvation within the bus system than we should go ahead with the Round-robin (RR) arbitration scheme. In the RR method the key to the bus is granted in a circular or round-robin fashion. This guarantees that each master will be given access to the bus eventually. [10] Master and slave will communicate and transfer data as long as there is no more information to be sent, this will continue up till the maximum allowed time Ownership of the bus passes to the next master in line. [10]

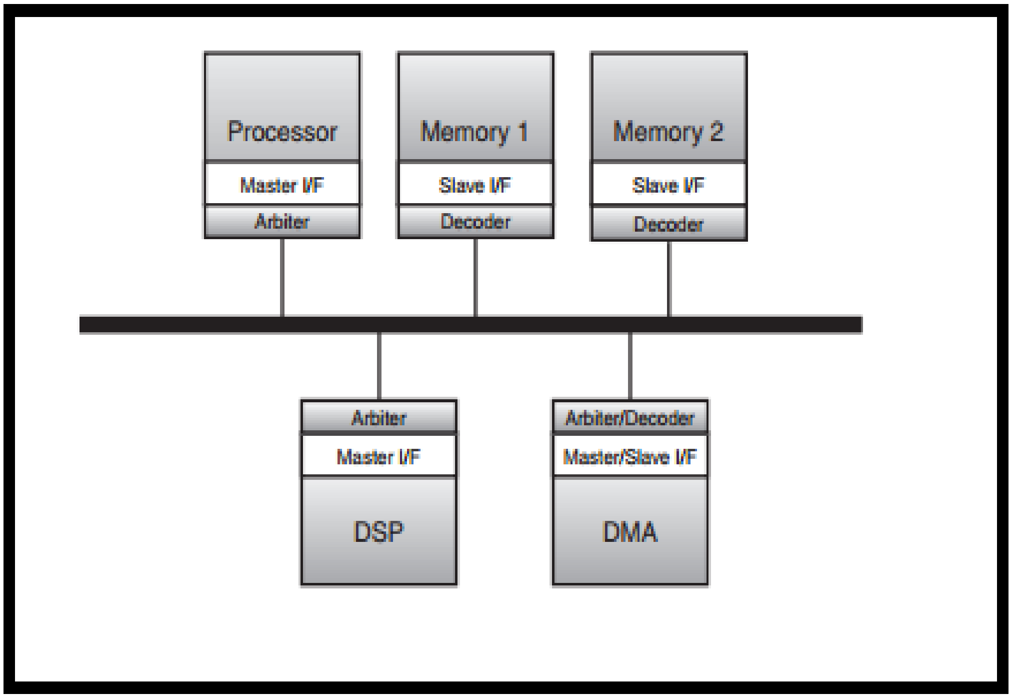


Figure 2.2: Distributed arbiter [10]

1RR scheme is very easy and simple to design and implement, it also ensures that a fine and equitable BW (Bandwidth) is distributed through the bus, but it suffers from a drawback compared to schemes like SP. When there is critical data transfer occurring it might have to be suspended or in wait mode for a long duration before it can proceed further. [10]

The (TDMA) time division multiple access arbitration is another type of scheme which is also very popular. While making sure that the lower priority masters are not starved this methodology makes sure that a fixed, higher bus BW (bandwidth) is given to the masters which have higher data transfer needs. Fixed time slots or time frames which are varying are given to every master. This basically depends on the BW (bandwidth) requirements of the master. Very important that we assign the number of time slots to each master. It's important that the critical data transfers are not affected and there is very little wait time to get access. Time frame should be long and sustainable enough to ensure a single data transfer while also making sure that the other critical data transfers are not affected. Also there should be very little wait time for access. This situation can also be looked in a different perspective. There is a chance for wastage if the master possesses the current time-slot and does not issue a request for the time slot. The time-alignment during communication is very important in this methodology. It's completely based on the probability of dynamic variations of the request patterns. Usually this scheme is implemented as single level but more complex level schemes can be developed if necessary [10].

To create a two level arbitration methodology like the Sonics SMART Interconnect we need to combine two or more arbitration schemes into one. TDMA/RR arbitration scheme design is implemented here a two-level. Different masters are assigned time slots at the same time based on TDMA scheme. [10]Let's imagine a scenario where the master does have enough data to send during its particular time slot. Then in the second level, the Round Robin scheme makes sure to choose a different master to grant bus access to. The cost of implementing a complex scheme is that it occupies more space. But this scheme is better off than TDMA methodology and ensures better and more efficient use the shared bus.

Dynamic priority (DP) methodology is a more complex scheme but a highly efficient and useful one. The priorities of masters can be changed at runtime (i.e., while the system is executing). [10] Here the priorities are dynamically adapted to the changing traffic profiles of an application. Additional logic is used to analyze data traffic at runtime. [10]

In our design we have implemented the dynamic priority and also the TDMA/RR scheme. This ensures better performance. The masters which need to send larger amounts of data are always given the utmost importance. There are issues in such schemes as well. Keeping track of different priorities as well as data traffic profiles at various localized points during execution will lead to high implementation costs since this requires a large number of registers.

There is an easier way to design a dynamic priority scheme by making it into a programmable priority (PP) scheme. Set the priority for masters on the bus dynamically with the arbiter's programmable registers. [10] At every transfer on the bus arbiters are invoked. Hence in a bus-based design we should always consider the bus to be a critical path. If an arbitration is complex in nature which when implemented will always consume more than one cycle to make a decision. This will usually affect the overall performance. This calls for careful thought while designing a complex multi-cycle arbitration methodology for some arbitration applications. At times a simple well thought out arbitration scheme which when synthesized occupies less space and needs fewer transistors may be preferable.

Dynamic priority is a scheme whose main strength is varying the master during execution of a specific application. This gives the masters higher injection rates based on higher priority. This requires additional logic which can analyze traffic at runtime. This kind of arbitration also adapts to changes in the data traffic profiles which are common. There is a high implementation cost where there is need for several registers to track all.

If the overall performance needs to be improved than a possible solution is to design a complex multi-cycle arbiter which will be based on pipeline methodology. Ensuring an overall improvement in performance. This requires profiling the design based on applications. Selecting and exploring an appropriate arbiter is based on the application and design flow which are the motivation factors. This calls for the need to profile the application in early stages of design flow. [10]

2.3 Data transfer modes

There are different modes by which the required data can be sent on the bus. The modes are specialized and they follow specific standards. In our design the following modes have been implemented:

1. Single non-pipelined transfer
2. Pipelined transfer
3. Burst transfer
4. Split transfer
5. Out of order transfer

2.3.1 Single non-pipelined transfer

To send data on the bus this is the easiest and simplest way. In due time, master will be granted access after it makes request for access to the shared bus. Address of the data is sent out in the next cycle and then data is written which occurs in the subsequent cycle. In the subsequent cycles, the master waits for the slave to send the read data. [10] Figure 2.3 shows that the master sequential example performing two single read data transfers.

The master makes a request to access the shared bus through the arbiter by asserting BUSREQ control signal. This is done in the first cycle, after which the arbiter grants the permission. Arbiter gives the permission to access the master only during the second cycle when the signal GRANT is asserted. [10] The slave receives the address (A1) from the master and then read data only when it is granted access to the shared bus. Master is than at the beginning of the third cycle. Requested data (D_A1) in the fourth cycle will be send back by the slave. Initially the slave will sample the read request at the beginning of the fourth cycle. [10] At the initial stages of the fifth cycle, to read more data from the slave, the master will need to make a request again to the shared bus through the arbiter. At the initial stages of the fifth cycle the read data is sampled off by the bus through the master. At the beginning of the ninth clock cycle the master will sample the data from the slave. This will follow a set of sequence of events quite similar to the first data transfer. [10] Typically slaves take multiple cycles to return data. In some cases slaves even have to write data. One master is connected to the bus, arbitration is not needed in such cases. Since there is no possibility of simultaneous bus transfers. [10] Bus request and grant are absent in such a cases, they are the first two cycles. The data transfer will now take only two cycles.

Conversely, when there are multiple masters connected to the bus it is possible that in such a case arbitration will be required. The arbiter takes multiple cycles to decide which master to grant bus access to. Possible through complex arbitration scheme such as the DP-based one. The arbiter takes multiple clock cycles when the bus clock frequency is very high, the arbiter takes time to respond as it has to choose from a set of arbitration schemes.[10]

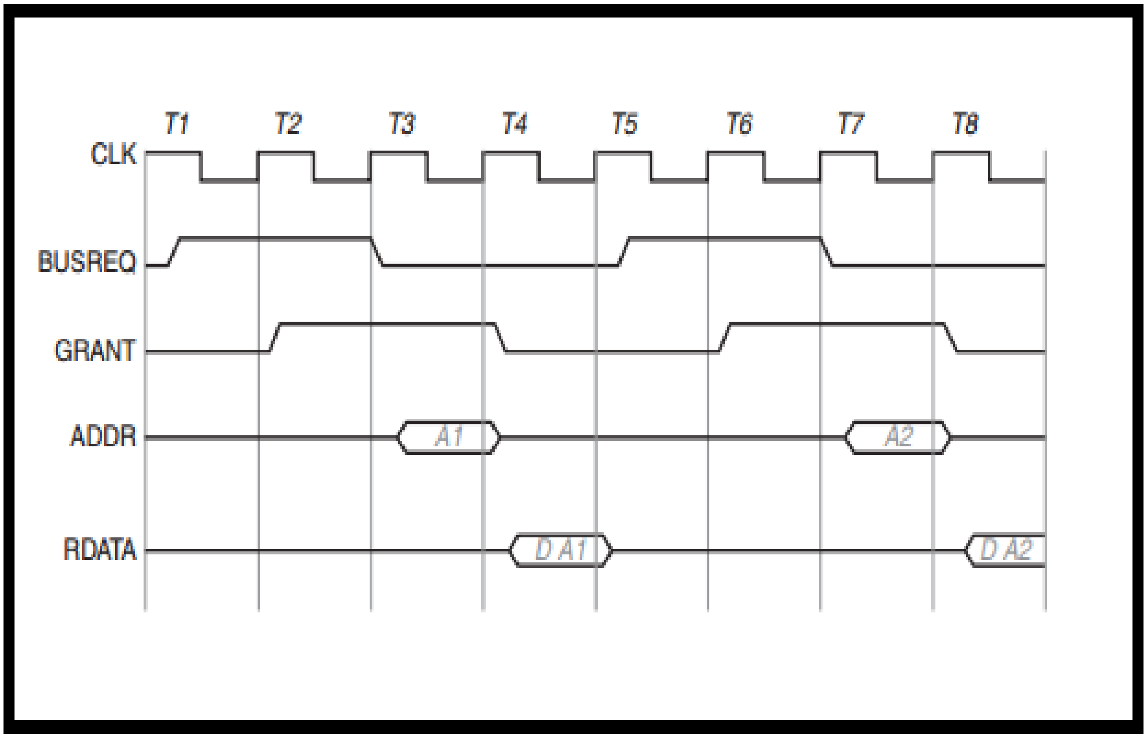


Figure 2.3: Single non-pipelined data transfer [10]

As many as four cycles will be needed to complete a single read data transfer for Single non-pipelined data transfer mode as shown in Figure 2.3. Under the assumption that a single cycle is needed for arbitration. There is a need for multiplexed address and data buses in bus-based communication architectures, especially for single non-pipelined transfer. We now look, how to reduce this number of cycles in the transfer modes. [10]

2.3.2 Pipelined transfer

For multiple data transfers to improve the overall bus transfer than pipelined methodology is adopted. There is a driving need to overlap the address and data phases at times. Overall bus throughput is increased by adopting this mode. There are two write data transfers for every two write data transfer requests which might be initiated by different masters this is highlighted in Figure 2.4. This is a perfect example of data transfer which is pipelined. [10] This figure shows that two masters, both make request to access the shared bus at the first cycle. Masters M1, M2 after making their requests for shared bus will need to wait; suppose M1 master is now given access at the same cycle. This decision will be taken by the arbiter. In the second cycle, the data to write (D_A1) in send. This is performed by the M1 (Master) which transmits the address of its destination to the slave (A1). [10] Access to the shared bus is given to the second master M2, in the next cycle this is performed by the arbiter, as it’s smart enough. Even before the write transfer is finished this permission is granted. The master M2 receives the address from the slave (A2) in the third cycle. The data phase of master M1 and the address phase of master M2 overlap. [10] In the fourth cycle, data will be written on master M3 which is transmitted from master M2. This will complete the actual transfer.

. To reduce the time for a data transfer such an overlapped transfer is implemented. This improvises the bus utilization. Pipelined (or overlapped) arbitration to be implemented a more complex arbiter will be needed. Only through complex arbitration will a pipelined transfers be implemented. [10] Separate address and data buses is a must for pipelined transfer implementations where there is no multiplexing of address and data signals. [10]

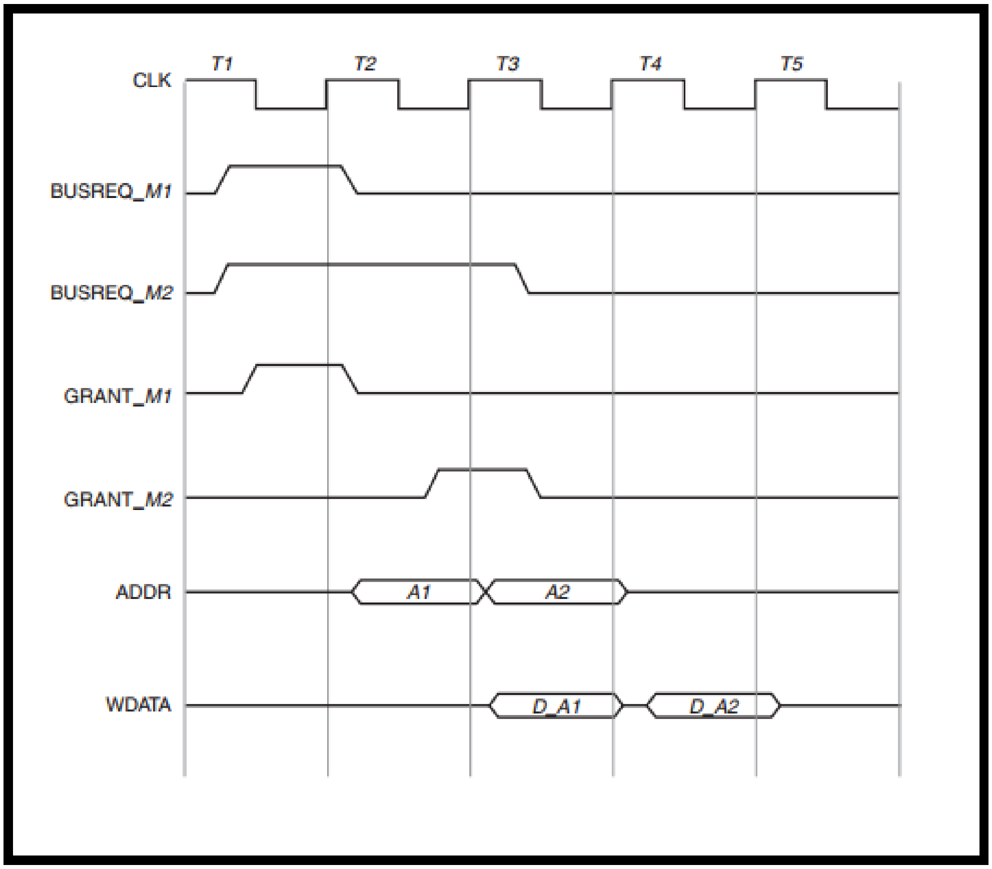


Figure 2.4: Pipelined transfer [10]

2.3.3 Burst transfer

The same master requires arbitration for every individual data transfer depicted in Figure. 2.5. Bus performance is improved by requesting arbitration only once for multiple data transfers, this is the burst transfer mode depicted in Figure 2.6 [10]. Figure 2.6 shows an example of burst data transfer by a master which is non-pipelined. The scenario depicted has a master needing to write four data items to a slave on the bus. Four data items is sent by the master after it is granted permission to access the shared bus by the arbiter at the beginning of the second cycle.

The request will be made at the beginning of the first cycle by the master.

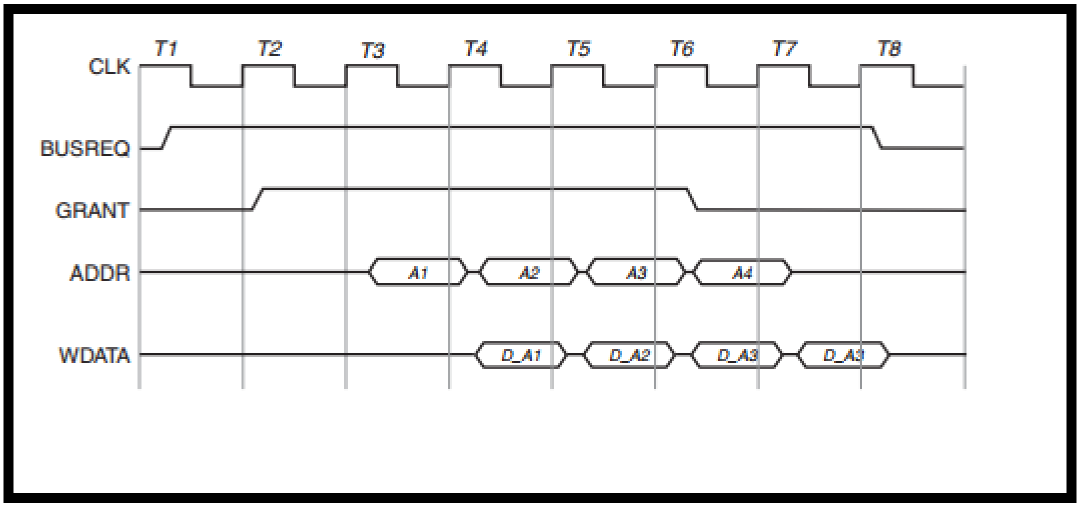


Figure 2.5: Burst pipelined transfer mode [10]

Typically, the master inform the arbiter of the length of the burst (four in this case) through the control signals (not shown in the figure). In the third cycle, address of the first data (A1) item is send to the master. In the fourth cycle the data is written to the slave (D_A1). The master is given permission for a burst of four data items. Re-arbitration at this point is not required since the arbiter has already granted bus access to the master. At the beginning of the fifth cycle, the master simply proceeds to send the address of the next data item (A2) in the burst. Till all four data items have been sent to the slave the data transfer continues. Compared to the single transfer mode shown in Figure 2.3, data items sent by the master is avoided in a burst transfer, which significantly reduces data transfer time. This is shown in Figure 2.6, pipelining if allowed within the burst transfer can reduce the overhead of arbitration and performance can be improved even further. This is depicted in Figure 2.6 and Figure 2.5 which highlights the same case; as four data items is received by the slave when the master sends it, but data transfers within the burst are overlapped that is the data phases and address of the data items. To considerably improve bus utilization and performance this pipelined burst mode is implemented. Which reduces the data transfer time compared to the non-pipelined burst mode in Figure. 2.6.

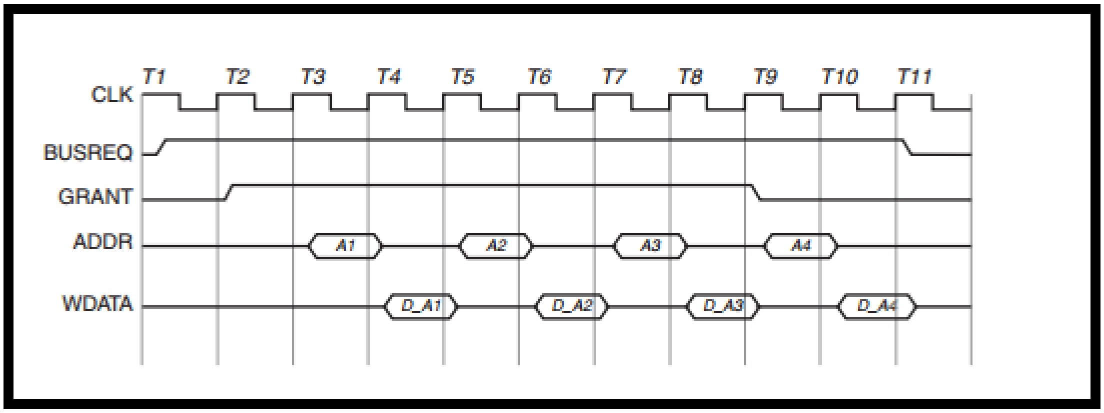


Figure 2.6: Master writing four data items in burst transfer mode: Non-pipelined burst transfer mode [10]

2.3.4 Split transfer

Slave can take multiple cycles to return the requested data or write the data during a data transfer on a bus which is very much possible. No other master can gain access to the shared bus till the transfer is completed. In such a case shared bus is typically held by the master. Till the slave completes the transfer, the shared bus remains idle for multiple cycles. [10] Under-utilization of the shared bus and reduced performance occurs during such a scenario. Improved bus utilization is possible by "splitting" the data transfer, a split transfer [4] is a special type of transfer mode. Which allows slave to be utilized for data transfers by other masters and by not using the idle cycles that would otherwise be used for waiting. Typically split transfer mode works as follows. [10] Once the arbiter receives a SPLIT signal from a slave, it initiates the transfer by masking the request from the master. There by preventing the master from getting any further access to the shared bus. Arbiter than initiate’s data transfers on the shared bus by using arbitration scheme to grant bus access to one of the other masters potentially waiting. Slave will send signals to the arbiter to “un-split” the master which will be initiated later, when the slave is ready to complete the transfer [10]. In due time the master gets access to the bus again while the arbiter un-masks the request from the master in the meantime. The slave can perform rest of the transfer. Therefore an effective mechanism for improving the communication performance in bus-based systems is through split transfer mode. This allows the idle cycles in a data transfer to be utilized. [10] Of course in the presence of split capable slaves and arbiters only then this mode will be operated.

2.3.5 Out-of-order transfer

The slave performs the SPLIT mode and to allow multiple transfers from different masters, best option is to use the SPLT transfer mode. Even from the same master it’s is possible to use the SPLIT transfer mode which is described above. The SPLIT is done by the slave and be in progress simultaneously on a single shared bus. This is Out-of-order (OO) data transfer’s basic idea [1]. Masters can initiate data transfers without waiting for earlier data transfers to complete this is to improve system performance because multiple data transfers can be processed in parallel. [10] To complete execution of data items in any order. Now each data transfer has an ID associated with it, so we can execute them in any order we like. It is possible to complete the transfer of the second data item even before the first data item by using the concept of ID. Execution of two data transfers, in a sequential manner made possible with the ID.

Master issues ID to data items, based on that the order of execution is decided; basically master decides this. Different masters or from the same master the data items will have different IDs and have no ordering restrictions, hence they can complete in any order. [10] Faster memory blocks to slower memory blocks can transfer data items in out of order fashion. This ability to complete data transfers out of order means without waiting for earlier transfer the data items can be sent. Overall system performance is improved significantly as a result of better bus utilization. Predictably, such an advanced and complex data transfer scheme there is an overhead involved in this type of implementation. Every data transfer in the system additionally needs extra signals to transmit IDs. [10] Secondly, master can reorder received data interfaces and for that need; the master is extended to handle data transfer IDs. Thirdly, to ensure that the proper data transfer ordering is maintained, slaves require additional logic at their interface to decode and process IDs. [10] Reordering depth is a parameter based on which the read (or write) data will be able to specify the maximum number of read (or write) data transfers which are pending in the slave. This parameter can be reordered if necessary. Requires more logic and increase system cost if larger reordering depths is needed, though it can significantly improve performance. Value for this parameter should be decided carefully by the designer [10]. Reordering depth beyond which the performance won't improve for a given application; that is the maximum value for that application [8]. Performance profiling will ensure that threshold corresponding to the maximum level of data traffic parallelism fir that application is reached. To ensure that data transfer ordering is maintained additional bus logic must also be added (to the arbiter module, or separately), for transfers originating from multiple masters.

Chapter 3

GENERAL IMPLEMENTATION FLOW

The generalized implementation flow diagram of the project is represented in Figure 3.1. We were able to cover the previous work on arbiters which helped us in determining the current requirements for the design. Later the specification for the architecture was defined and identified. Then the RTL modeling was carried out in Verilog. Through RTL coding the entire hardware architecture behavior was designed. Once the RTL modeling was done, simulation was carried out and then verification was carried out using System Verilog. Functional verification was done in consecutive steps and intended architecture had to pass all the test cases.

The developed RTL model was then translated to the format needed for the Magellan tool. Once the functional verification was clear, the RTL model was taken through the synthesis process. General flow has 9 stages which are mentioned in the Figure 3.1 while the three main operations of synthesis process are

1. Optimization of net-list
2. Translation
3. Mapping

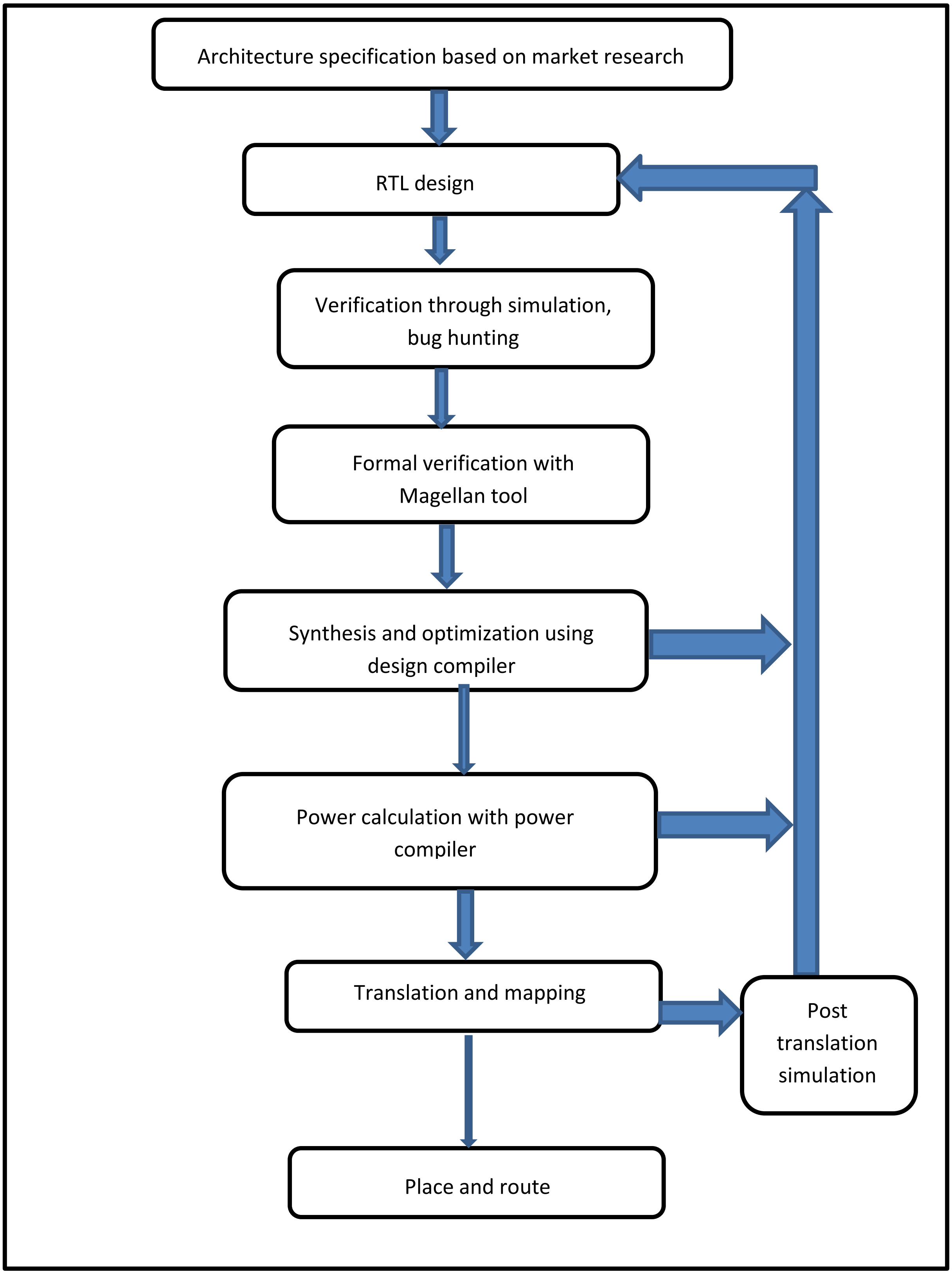

Architecture specification based on market research
RTL design
Verification through simulation, bug hunting
Formal verification with Magellan tool
Synthesis and optimization using design compiler
Power calculation with power compiler
Translation and mapping
Post translation simulation
Place and route

Figure 3.1: General flow

3.1 Verification methodology

3.1.1 Basic test bench functionality

The purpose of a test-bench is to determine the correctness of the design under test (DUT). The following steps make this possible:

- ✓ Generate stimulus
- ✓ DUT Stimulus is applied on DUT
- ✓ Response is captured
- ✓ Correctness is checked
- ✓ The overall verification goals are checked and progress is measured.

Figure 3.2 depicts the above mentioned steps.

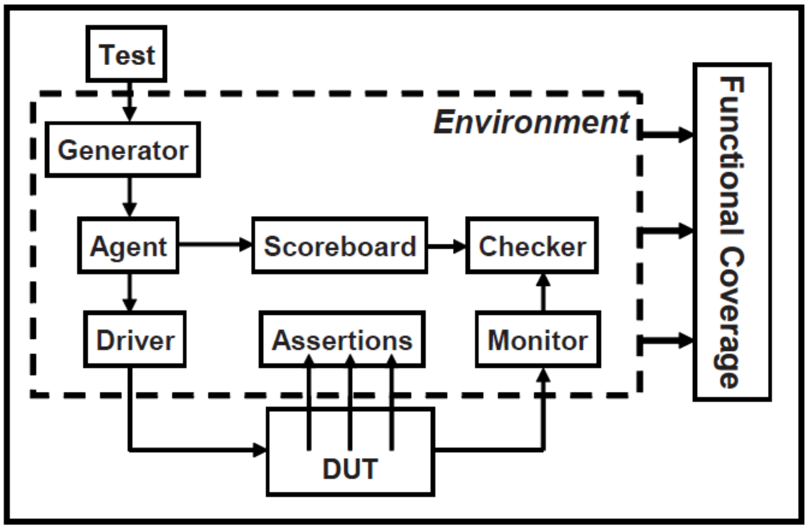


Figure 3.2: Test-bench functionality [2]

3.1.2 Formal Verification

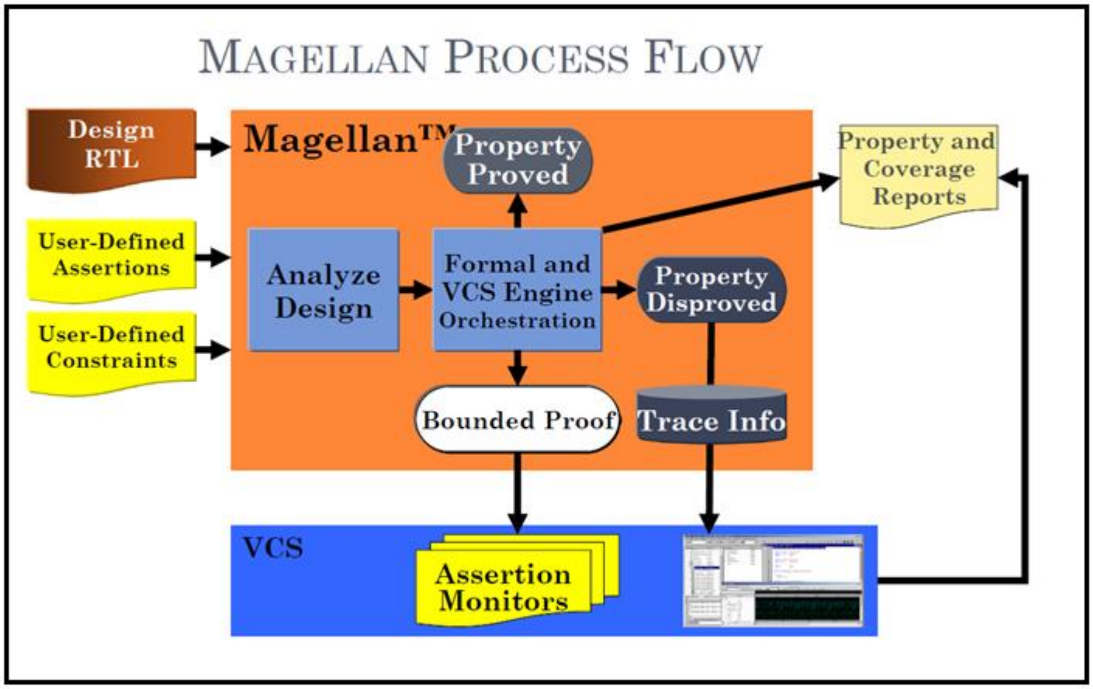
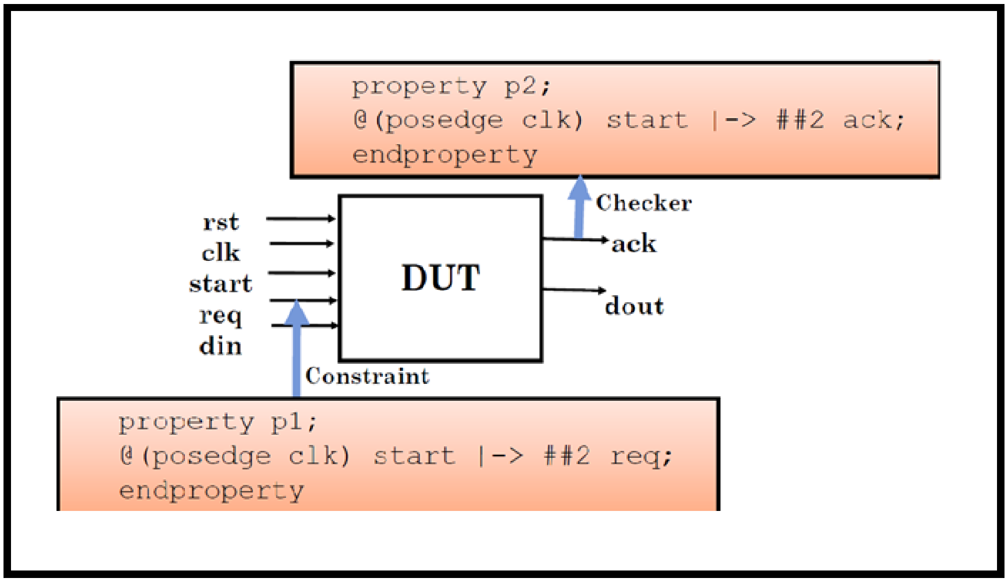


Figure 3.3: Magellan process flow [4]

To increase the cover properties of a design which will enable us to verify a particular design extensively we use the Magellan tool. Through this tool significant events within the arbitration system can be covered. For example it's possible to check complex sequences for bus protocols like burst read, burst and even multiple write and multiple read. Using properties of property of cover groups, the required information can be collected and used to analyze the delays such as packet latency.

Possible scenario is mentioned below where Figure 3.4 shows an example of assertions. Figure 3.5 diagram depicts the results, while Figure 3.6 and 3.7 give us the overall views of the flow process.



```
property p2;
@(posedge clk) start |-> ##2 ack;
endproperty
```

```
property p1;
@(posedge clk) start |-> ##2 req;
endproperty
```

Figure 3.4: Assertion for a DUT [4]

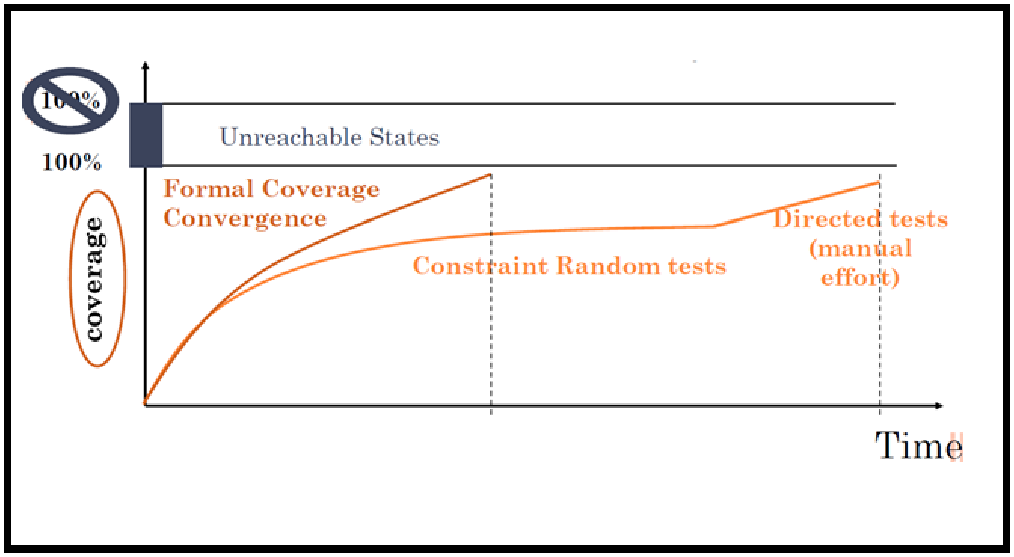


Figure 3.5: Formal coverage [4]

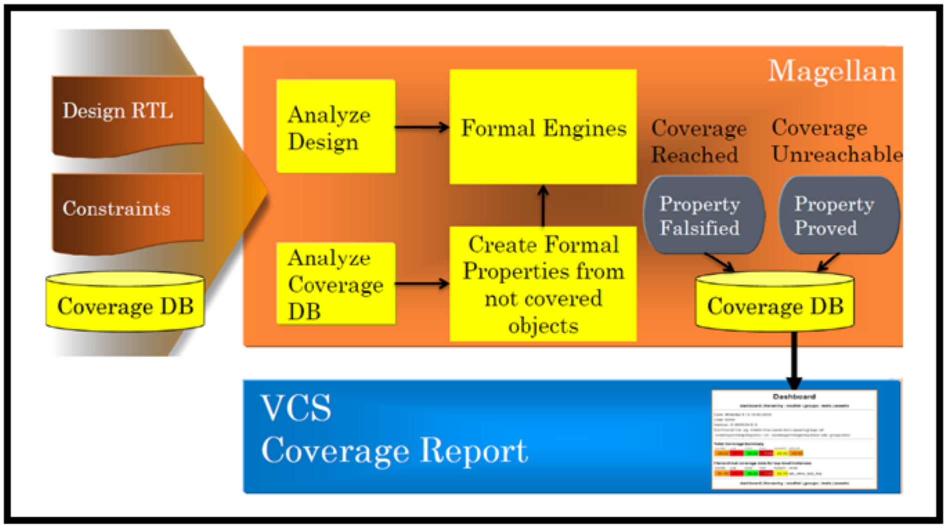


Figure 3.6: Converge convergence flow using the formal tool [4]

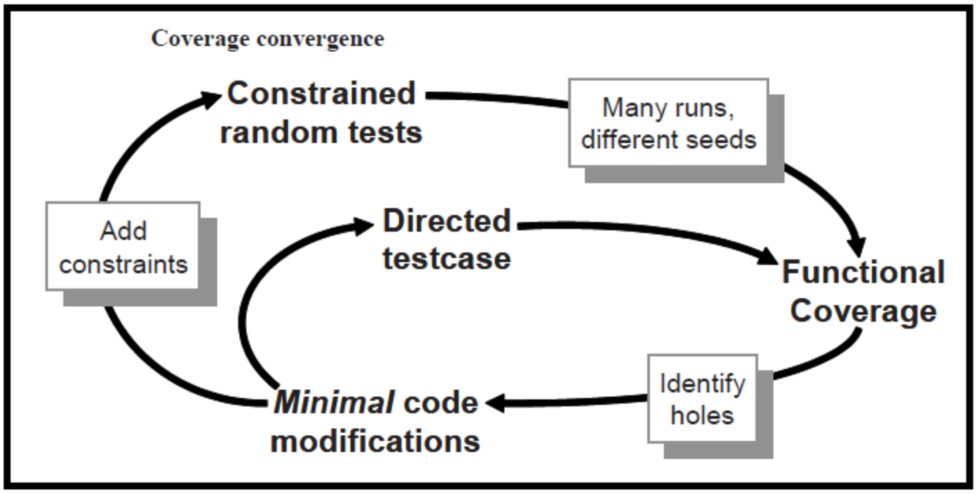


Figure 3.7: Coverage convergence [4]

The behavior of the system is verified by increasing the assert properties, which verify the functionality. Unreachable coverage objects can be reached through this formal verification methodology and the number of direct tests can be reduced.

3.2 Low Power techniques

Different low power optimizations are discussed below

1) Multi Vt cells
2) Power-driven clock gating
3) Power gating
4) Cloning clock gates
5) Cell sizing
6) Switching OFF inactive blocks

Multi Vt cells

Multi threshold library cells have different threshold voltages for MOS devices. The standard cell libraries provide more than one flavor of cells, each with different power speed characteristics which are typically determined by the Vt of the transistors used in the cells. [6]

Power driven clock gating

Knowing the activity of a clock gater can help determine the tradeoff between power savings and cost of putting in the clock gaters. The activity of a clock gater can be obtained from simulations and provided in an activity file, such as SAIF. If a clock gate is active most of the time, it may better not to have the clock gater. Optimizing the selection based on clock gater activity is known as power driven clock gating. [6]

Power gating

In this technique, the supply voltage to a block that is not active or not in use can be turned off. A header cell can be used to disconnect the VDD or a footer cell can be used to disconnect VSS. Both ways (using the header or footer cell) have the same effect – the power to the block is switched off using a control signal. [6]

The basic strategy of power gating is to provide two power modes: a low power mode and an active mode. The goal is to switch between these modes at the appropriate time and in the appropriate manner to maximize power savings while minimizing the impact to performance. [7] With power gating, we have to retain some critical register contents (FSM states) and save and restore the state quickly and efficiently to get the block fully functional after power up. [8] Selective power gating is shown in Figure 3.8.

Cloning clock gates

Clock gates can be cloned sometimes to help meet timing. The cloning can be for the gater enable pin or for the flip flop being driven. This results in a likely increase in clock power but helps in meeting the timing requirement.

Cell sizing

Downsizing the cells on the non-critical paths can help reduce the dynamic power consumption. A lower strength cell has lower dynamic power consumption. This is normally part of the area recovery and the power recovery steps during the timing and power driven implementation. In this approach the place and route tool would reduce the cell strength at non critical timing paths to save area and reduce power after the initial implementation is completed.

Lower strength cells normally provide a lower input capacitance load on the previous stages than the higher strength cells. Higher strength cells are useful for driving long traces or for driving higher heavier loads. Low strength cells are typically for low power.

During physical design of an ASIC, a program can target power improvement by reducing the sizes of the cells in non-critical paths. The lower strength cells most likely are not footprint compatible. However, the lower strength cells have usually lower area than the higher strength cells and thus fit within existing area.

Switching OFF inactive blocks

In this method, inactive blocks are shut down to save power. By turning the blocks off, the active power goes down to zero. The control for turn off is internal to the chip and shutdown is accomplished using power switches.

Power gating involves gating off the power supply so that the power to the inactive blocks can be turned off.

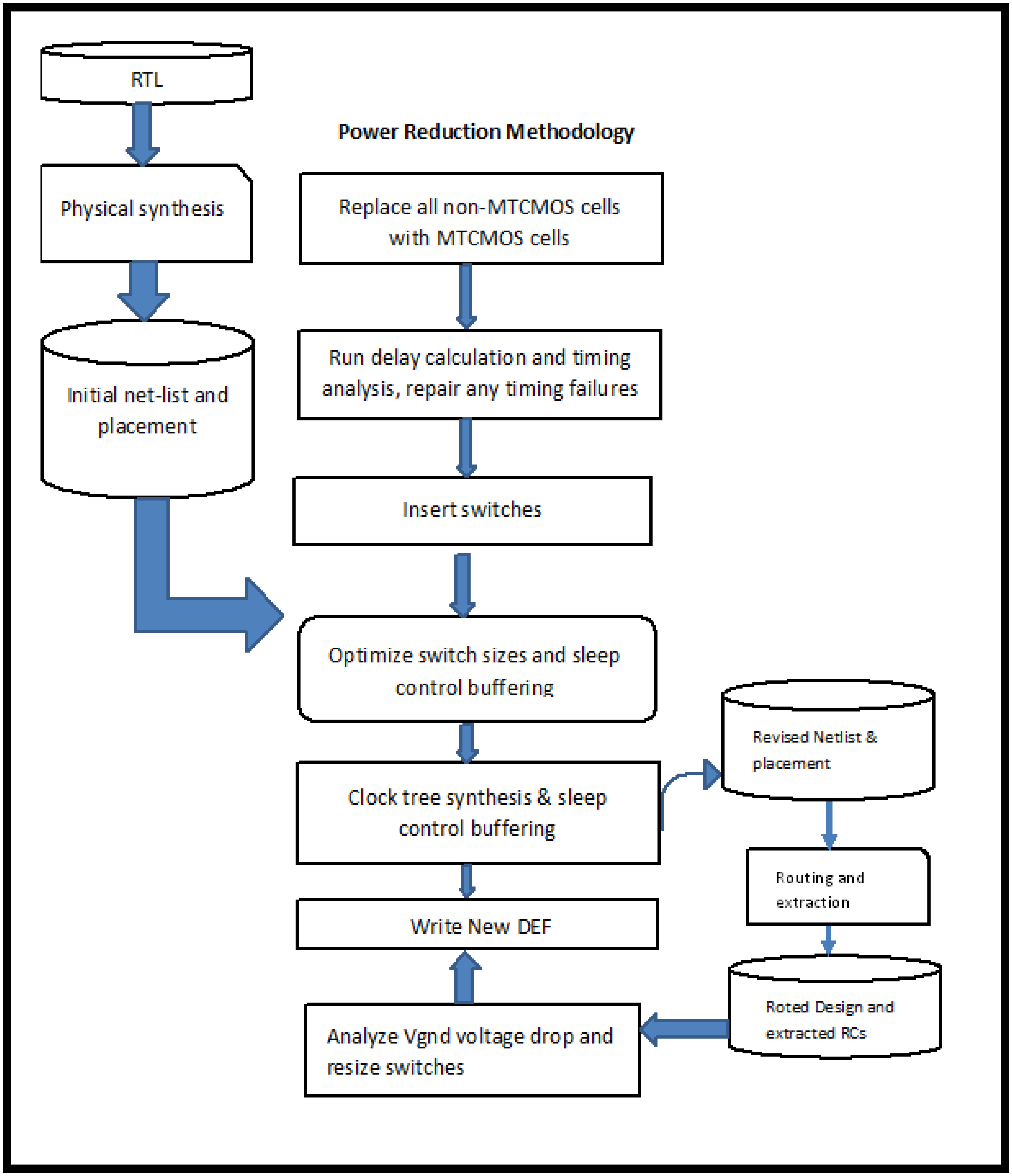

RTL
Physical synthesis
Initial net-list and placement
Power Reduction Methodology
Replace all non-MTCMOS cells with MTCMOS cells
Run delay calculation and timing analysis, repair any timing failures
Insert switches
Optimize switch sizes and sleep control buffering
Clock tree synthesis & sleep control buffering
Write New DEF
Analyze Vgnd voltage drop and resize switches
Revised Netlist & placement
Routing and extraction
Roted Design and extracted RCs

Figure 3.8: Selective power gating design flow

Chapter 4
RESULTS AND DISCUSSION

The on-chip bus was implemented as discussed in the previous chapter. The arbiter design is given in Figure 4.1. In this chapter the simulation and synthesis results are discussed. The entire design is simulated through Modelsim and Verdi tool is used to connect the dangling signals. The functionality of the design was also checked using Magellan and System Verilog. A test bench was developed to monitor the bus request signal and corresponding bus grant signal

Through simulation different modes - Pipelined Burst, Single Non- Pipelined Transfer, TDMA/RR., Dynamic Priority and Round Robin - were verified.

Once the functional verification was done, then using Design Compiler the entire design was synthesized using 90nm technology library from TSMC and thus the netlist was generated.

Power results were obtained using Synopsys Power Compiler by monitoring the switching activity of each net in the bus architecture.

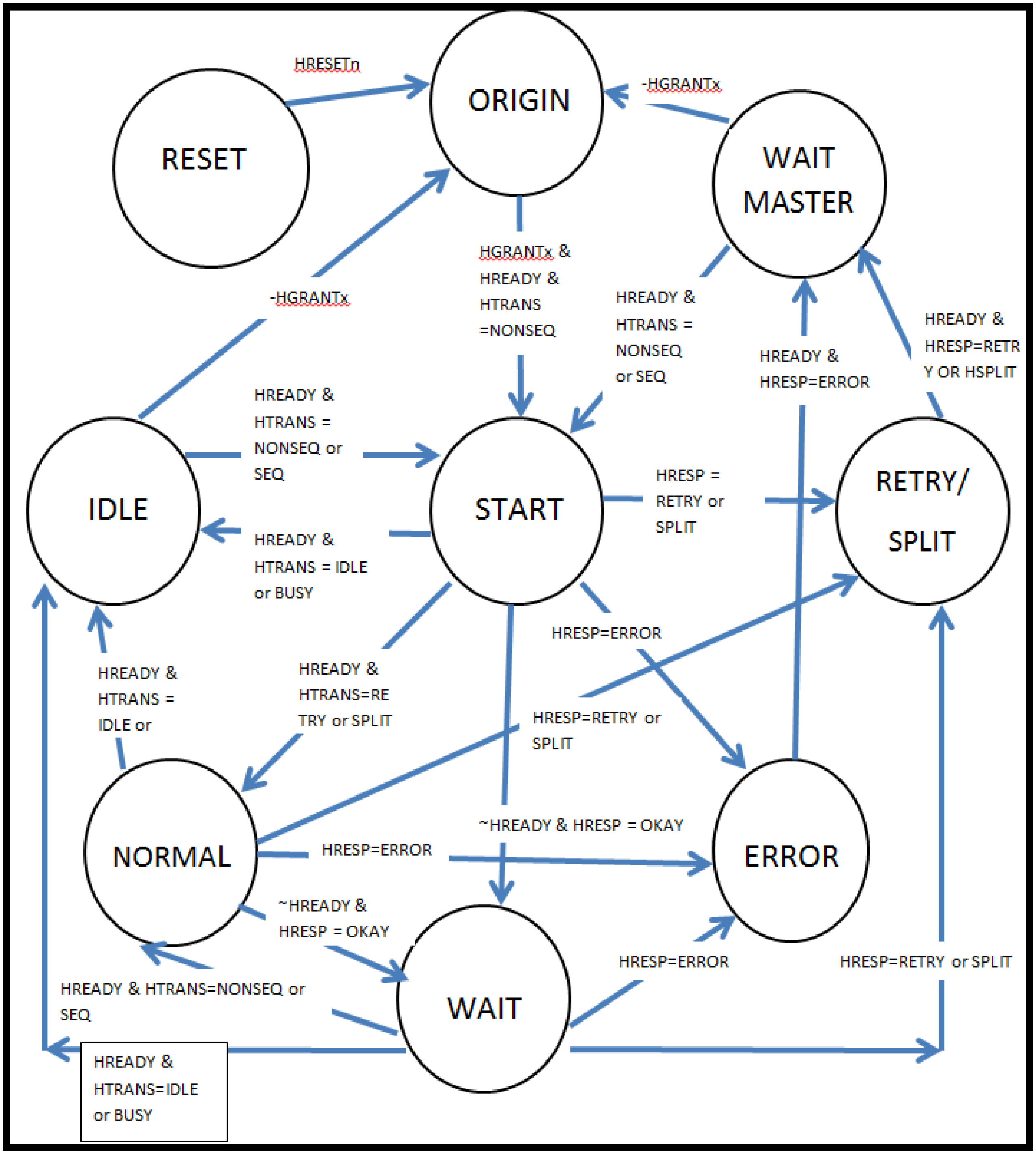
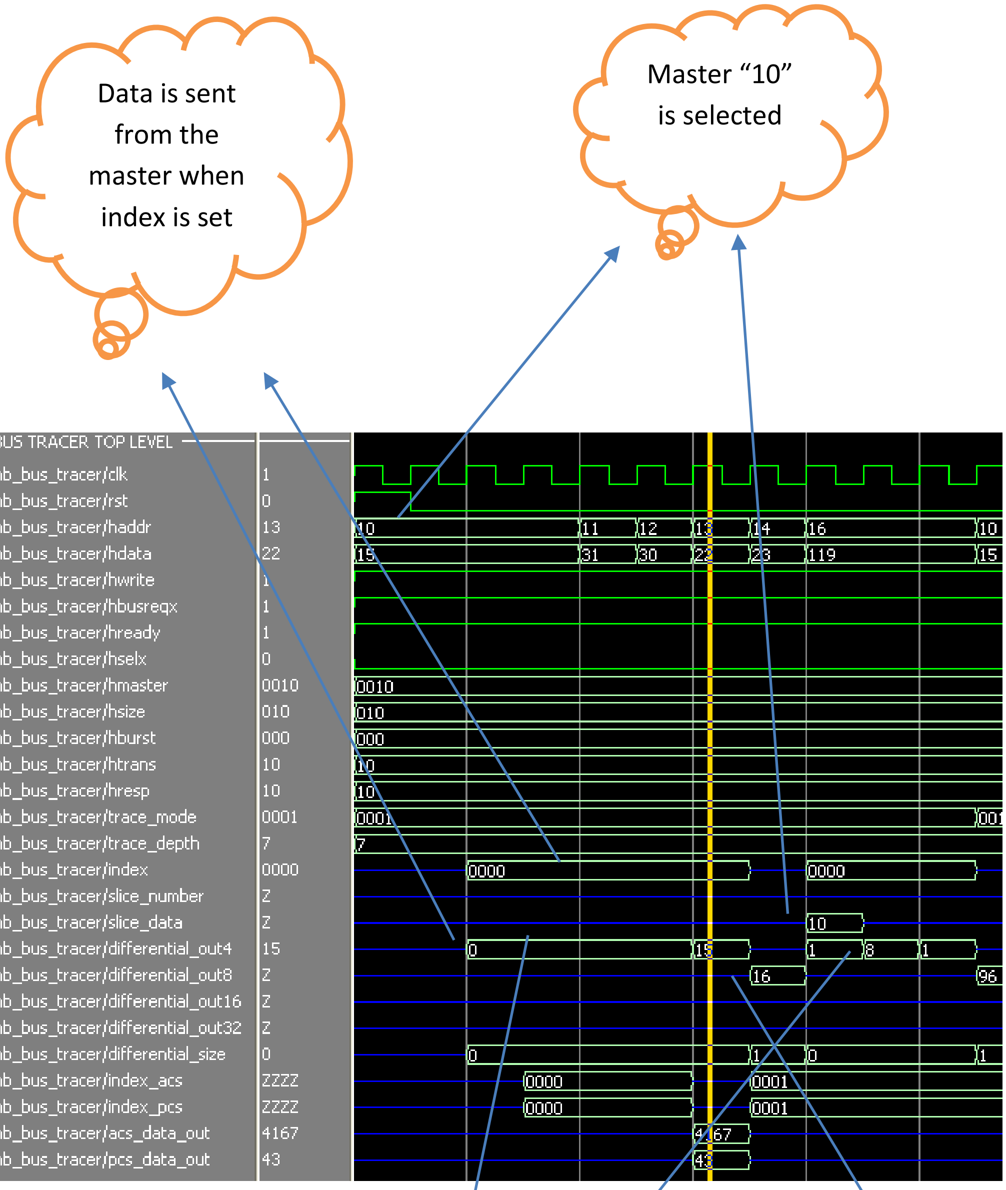

HRESETn
ORIGIN
-HGRANTx
RESET
WAIT MASTER
HGRANTx & HREADY & HTRANS =NONSEQ
-HGRANTx
HREADY & HTRANS = NONSEQ or SEQ
HREADY & HRESP=ERROR
HREADY & HRESP=RETRY OR HSPLIT
HREADY & HTRANS = NONSEQ or SEQ
IDLE
START
HRESP = RETRY or SPLIT
RETRY/ SPLIT
HREADY & HTRANS = IDLE or BUSY
HRESP=ERROR
HREADY & HTRANS = IDLE or
HREADY & HTRANS=RETRY or SPLIT
HRESP=RETRY or SPLIT
~HREADY & HRESP = OKAY
NORMAL
HRESP=ERROR
ERROR
~HREADY & HRESP = OKAY
WAIT
HREADY & HTRANS=NONSEQ or SEQ
HRESP=ERROR
HRESP=RETRY or SPLIT
HREADY & HTRANS=IDLE or BUSY

Figure 4.1 State diagram of an arbiter

4.1 Simulation results

4.1.1 Mode pipelined burst/single non- pipelined transfer mode



ON CHIP AHB BUS TRACER TOP LEVEL

/on_chip_ahb_bus_tracer/clk	1
/on_chip_ahb_bus_tracer/rst	0
/on_chip_ahb_bus_tracer/haddr	13
/on_chip_ahb_bus_tracer/hdata	22
/on_chip_ahb_bus_tracer/hwrite	1
/on_chip_ahb_bus_tracer/hbusreqx	1
/on_chip_ahb_bus_tracer/hready	1
/on_chip_ahb_bus_tracer/hselx	0
/on_chip_ahb_bus_tracer/hmaster	0010
/on_chip_ahb_bus_tracer/hsize	010
/on_chip_ahb_bus_tracer/hburst	000
/on_chip_ahb_bus_tracer/htrans	10
/on_chip_ahb_bus_tracer/hresp	10
/on_chip_ahb_bus_tracer/trace_mode	0001
/on_chip_ahb_bus_tracer/trace_depth	7
/on_chip_ahb_bus_tracer/index	0000
/on_chip_ahb_bus_tracer/slice_number	Z
/on_chip_ahb_bus_tracer/slice_data	Z
/on_chip_ahb_bus_tracer/differential_out4	15
/on_chip_ahb_bus_tracer/differential_out8	Z
/on_chip_ahb_bus_tracer/differential_out16	Z
/on_chip_ahb_bus_tracer/differential_out32	Z
/on_chip_ahb_bus_tracer/differential_size	0
/on_chip_ahb_bus_tracer/index_acs	ZZZZ
/on_chip_ahb_bus_tracer/index_pcs	ZZZZ
/on_chip_ahb_bus_tracer/acs_data_out	4167
/on_chip_ahb_bus_tracer/pcs_data_out	43

Figure 4.2 Simulation results of arbiter in pipelined burst transfer

Pipelined Burst Transfer

Single Non-pipelined Transfer

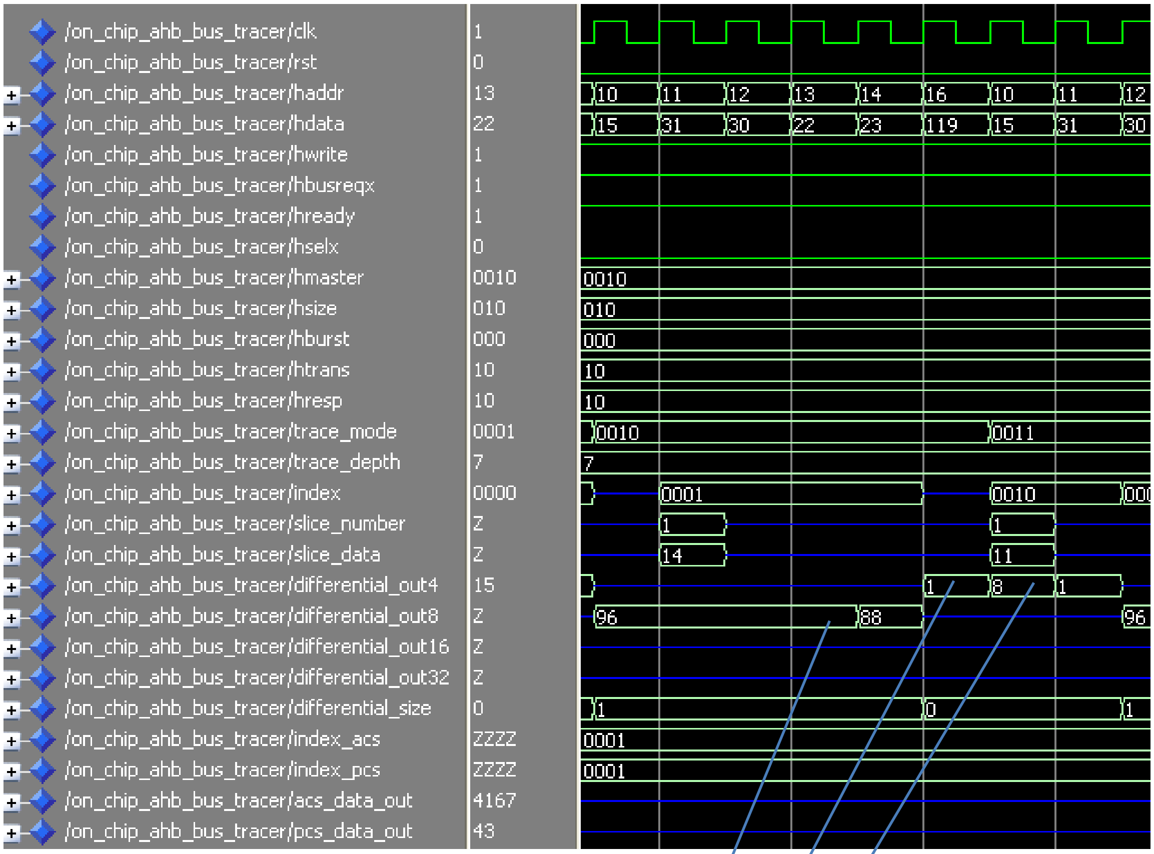
Simulation results for on-chip Bus data transfer in Mode PB (Mode Pipelined, Burst Transfer) as shown in Figure 4.2.Input signals for the data transfer from master to slave include program address, Data value and Control signals(ACS,PCS), hmaster. Hmaster decides the master priority.

In MODE PB, the hmaster “0010” and the type of data transfer gets selected initially as “10” for burst transfer mode. Differentail_out8 and Differential_out4 are selected for data transfer modes. Single non- pipelined transfer mode is set initially than pipelined burst transfer occurs. So now we are able to observe the most detailed bus activities. Looking at the detailed signals it’s easy to diagnose the cause of the error, hence this mode is very useful.

As shown in Figure 4.2, HADDR, HDATA, ACS, PCS are applied. Index, Slice_No, Slice_data selects the slave output data for program address. The different data transfer data modes for the address/data value signal are defined by Differential_size. The two control signals are access control signal and protocol control signal.

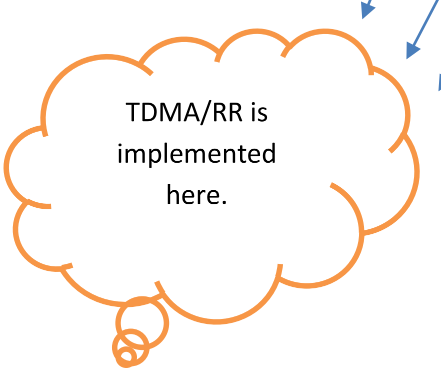
4.1.2 Mode TDMA/RR

Simulation results of on-chip bus transfer in Mode TDMA/RR (Mode Time Division Multiplexing and Round Robin) are shown in Figure 4.3. Mode RR and Mode TDMA combination is used and here the data is abstracted at both signal and timing level. Control signals, data value and program address are the on-chip bus signals. There are two control signals, Access control signal (ACS) and protocol control signals (PCS).



/on_chip_ahb_bus_tracer/clk	1
/on_chip_ahb_bus_tracer/rst	0
/on_chip_ahb_bus_tracer/haddr	13
/on_chip_ahb_bus_tracer/hdata	22
/on_chip_ahb_bus_tracer/hwrite	1
/on_chip_ahb_bus_tracer/hbusreqx	1
/on_chip_ahb_bus_tracer/hready	1
/on_chip_ahb_bus_tracer/hselx	0
/on_chip_ahb_bus_tracer/hmaster	0010
/on_chip_ahb_bus_tracer/hsize	010
/on_chip_ahb_bus_tracer/hburst	000
/on_chip_ahb_bus_tracer/htrans	10
/on_chip_ahb_bus_tracer/hresp	10
/on_chip_ahb_bus_tracer/trace_mode	0001
/on_chip_ahb_bus_tracer/trace_depth	7
/on_chip_ahb_bus_tracer/index	0000
/on_chip_ahb_bus_tracer/slice_number	Z
/on_chip_ahb_bus_tracer/slice_data	Z
/on_chip_ahb_bus_tracer/differential_out4	15
/on_chip_ahb_bus_tracer/differential_out8	Z
/on_chip_ahb_bus_tracer/differential_out16	Z
/on_chip_ahb_bus_tracer/differential_out32	Z
/on_chip_ahb_bus_tracer/differential_size	0
/on_chip_ahb_bus_tracer/index_acs	ZZZZ
/on_chip_ahb_bus_tracer/index_pcs	ZZZZ
/on_chip_ahb_bus_tracer/acs_data_out	4167
/on_chip_ahb_bus_tracer/pcs_data_out	43

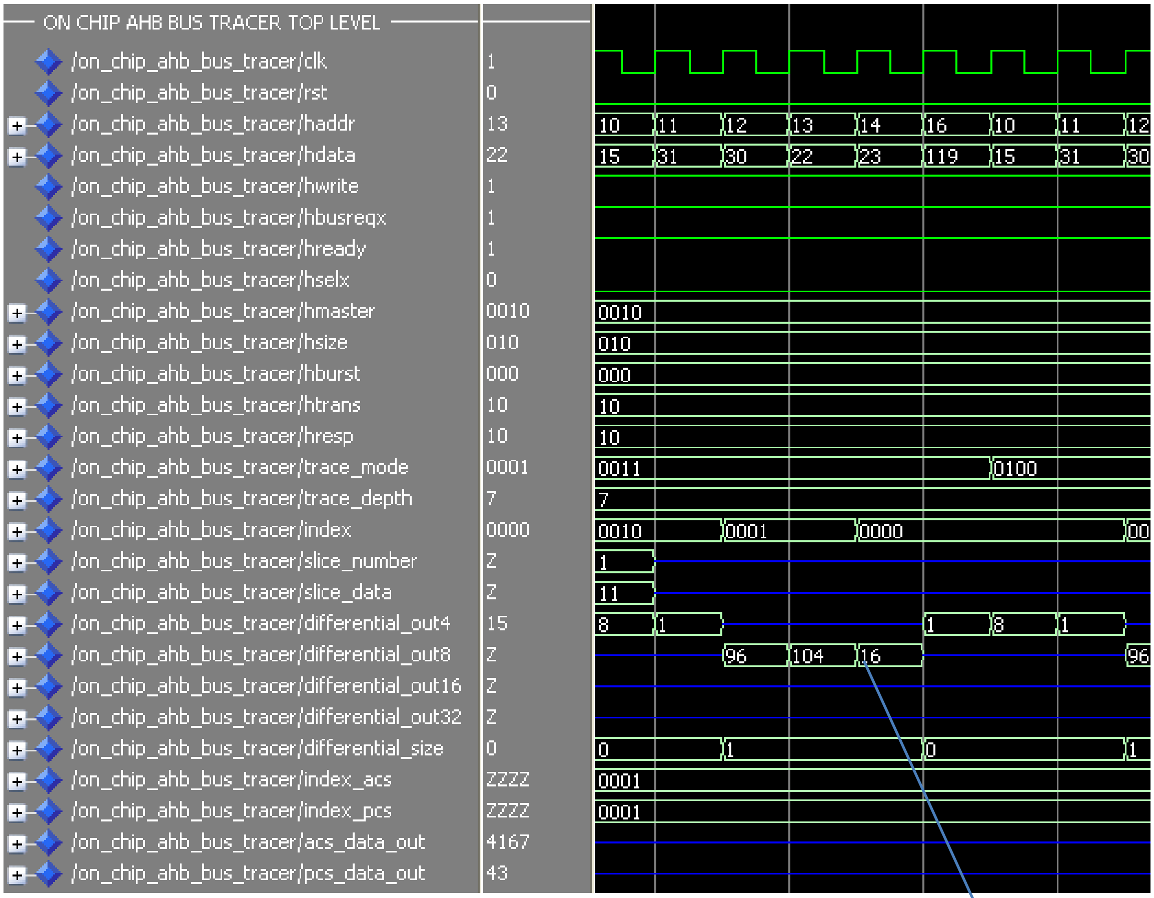
Figure 4.3 Simulation results of mode PT



As shown in Figure 4.3, HADDR, HDATA, ACS, PCS are applied. Index, Slice_No, Slice_data select the slave. Differential_out4, Differential_out8, Differential_out16, Differential_out32, Differential_size decide the different data transfer mode.

4.1.3 Mode dynamic priority

Simulation results of the arbiter for on-chip bus with Mode DP (Mode Dynamic Priority, Burst Transfer) are shown in Figure 4.4.Input signals for the on-chip arbiter include program address, Address value and Control signals. Access Control Signal - ACS, protocol control signal –PCS form the control signals. As shown in Figure 4.4, HADDR, HDATA, ACS, PCS are applied. Index, Slice_No, Slice_data compressed output data for program address.Differential_out4, Differential_out8, Differential_out16, Differential_out32, Differential_size decide the different data transfer mode.



ON CHIP AHB BUS TRACER TOP LEVEL

/on_chip_ahb_bus_tracer/clk	1
/on_chip_ahb_bus_tracer/rst	0
/on_chip_ahb_bus_tracer/haddr	13
/on_chip_ahb_bus_tracer/hdata	22
/on_chip_ahb_bus_tracer/hwrite	1
/on_chip_ahb_bus_tracer/hbusreqx	1
/on_chip_ahb_bus_tracer/hready	1
/on_chip_ahb_bus_tracer/hselx	0
/on_chip_ahb_bus_tracer/hmaster	0010
/on_chip_ahb_bus_tracer/hsize	010
/on_chip_ahb_bus_tracer/hburst	000
/on_chip_ahb_bus_tracer/htrans	10
/on_chip_ahb_bus_tracer/hresp	10
/on_chip_ahb_bus_tracer/trace_mode	0001
/on_chip_ahb_bus_tracer/trace_depth	7
/on_chip_ahb_bus_tracer/index	0000
/on_chip_ahb_bus_tracer/slice_number	Z
/on_chip_ahb_bus_tracer/slice_data	Z
/on_chip_ahb_bus_tracer/differential_out4	15
/on_chip_ahb_bus_tracer/differential_out8	Z
/on_chip_ahb_bus_tracer/differential_out16	Z
/on_chip_ahb_bus_tracer/differential_out32	Z
/on_chip_ahb_bus_tracer/differential_size	0
/on_chip_ahb_bus_tracer/index_acs	ZZZZ
/on_chip_ahb_bus_tracer/index_pcs	ZZZZ
/on_chip_ahb_bus_tracer/acs_data_out	4167
/on_chip_ahb_bus_tracer/pcs_data_out	43

Figure 4.4 Simulation results of mode DP

Dynamic priority

4.1.4 Mode RR

Simulation results for the on-chip bus arbiter with Mode RR (Mode Round Robin, Cycle level) are shown in Figure 4.5.Input signal for on-chip bus are program address, Address /Data value and Control signals. There are two control signals they are Access control signal (ACS), Protocol control Signal (PCS).

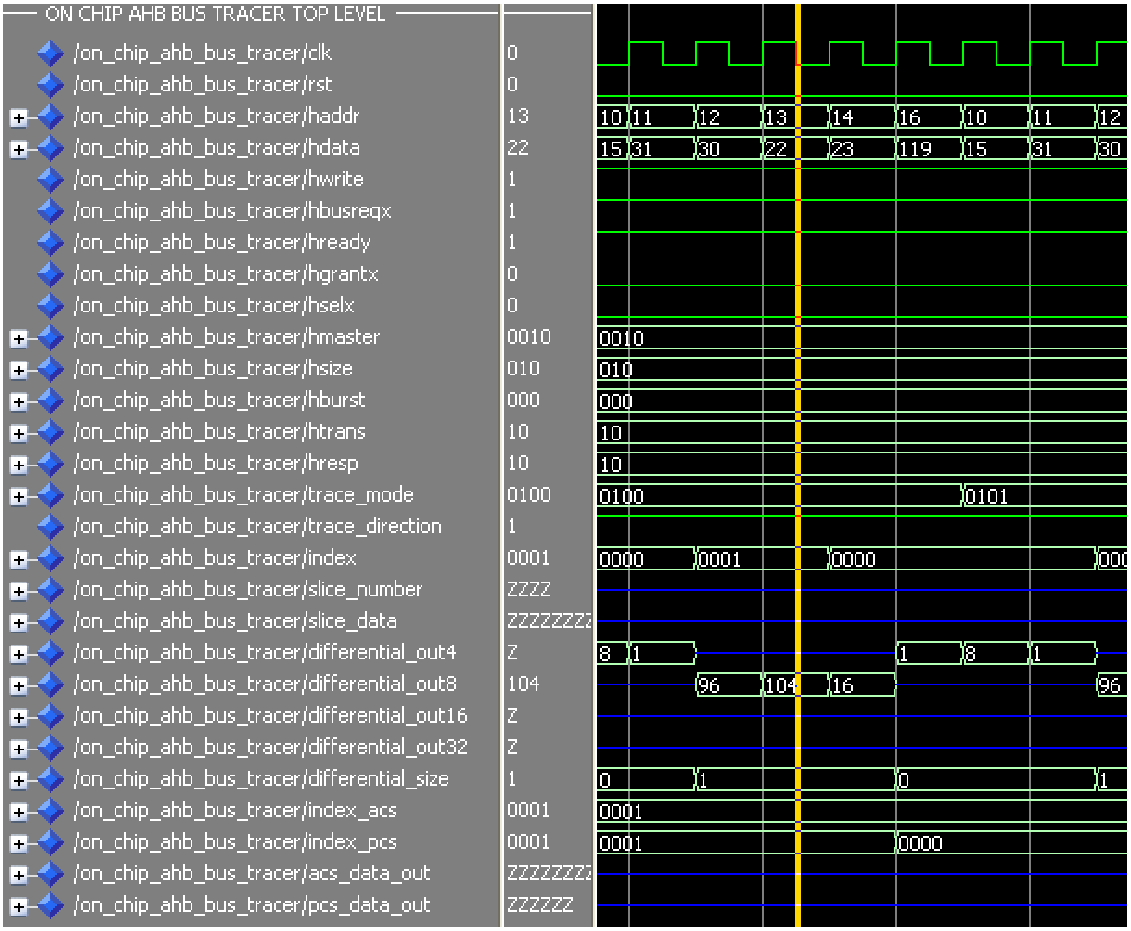

ON CHIP AHB BUS TRACER TOP LEVEL
/on_chip_ahb_bus_tracer/clk 0
/on_chip_ahb_bus_tracer/rst 0
/on_chip_ahb_bus_tracer/haddr 13
/on_chip_ahb_bus_tracer/hdata 22
/on_chip_ahb_bus_tracer/hwrite 1
/on_chip_ahb_bus_tracer/hbusreqx 1
/on_chip_ahb_bus_tracer/hready 1
/on_chip_ahb_bus_tracer/hgrantx 0
/on_chip_ahb_bus_tracer/hselx 0
/on_chip_ahb_bus_tracer/hmaster 0010
/on_chip_ahb_bus_tracer/hsize 010
/on_chip_ahb_bus_tracer/hburst 000
/on_chip_ahb_bus_tracer/htrans 10
/on_chip_ahb_bus_tracer/hresp 10
/on_chip_ahb_bus_tracer/trace_mode 0100
/on_chip_ahb_bus_tracer/trace_direction 1
/on_chip_ahb_bus_tracer/index 0001
/on_chip_ahb_bus_tracer/slice_number ZZZZ
/on_chip_ahb_bus_tracer/slice_data ZZZZZZZZ
/on_chip_ahb_bus_tracer/differential_out4 Z
/on_chip_ahb_bus_tracer/differential_out8 104
/on_chip_ahb_bus_tracer/differential_out16 Z
/on_chip_ahb_bus_tracer/differential_out32 Z
/on_chip_ahb_bus_tracer/differential_size 1
/on_chip_ahb_bus_tracer/index_acs 0001
/on_chip_ahb_bus_tracer/index_pcs 0001
/on_chip_ahb_bus_tracer/acs_data_out ZZZZZZZZ
/on_chip_ahb_bus_tracer/pcs_data_out ZZZZZZ
10 11 12 13 14 16 10 11 12
15 31 30 22 23 119 15 31 30
0010
010
000
10
10
0100 0101
0000 0001 0000 000
8 1 1 8 1
96 104 16 96
0 1 0 1
0001
0001 0000

Figure 4.5 Simulation results of Mode RR

4.2 Verification results

4.2.1 Magellan tool readings

Rule Name	Cycles	Result
Master can write	1	Exhaustive Pass (100%)
Slave can read	1	Exhaustive Pass (100%)
Master 1 transfer data to the slave 4	6	Exhaustive Pass (100%)
Dynamic Priority	12	Exhaustive Pass (100%)
Single Burst	3	Exhaustive Pass (100%)
Pipelined Burst	7	89% Success
Round Robin	10	Exhaustive Pass (100%)
RR/TDMA	15	92% Success

Table 4.1: Magellan tool readings

Through user specified properties and also extracted structural properties, Magellan tool can be highly effective. With the help of VCS tool which is inbuilt with Magellan we can count the number of cycles. The Magellan tool provides us the readings of success or failure with percentages. This enables us to reach corner cases and also deep data logic within a complex design. The above table gives us the required readings for our design.

4.2.2 Universal Verification Methodology (UVM) test-bench

Through effective verification planning we were able to spend less time on debugging, just 10% of the entire verification time. Usually more than 30% of the time is spent on this task. More time was dedicated for test execution.

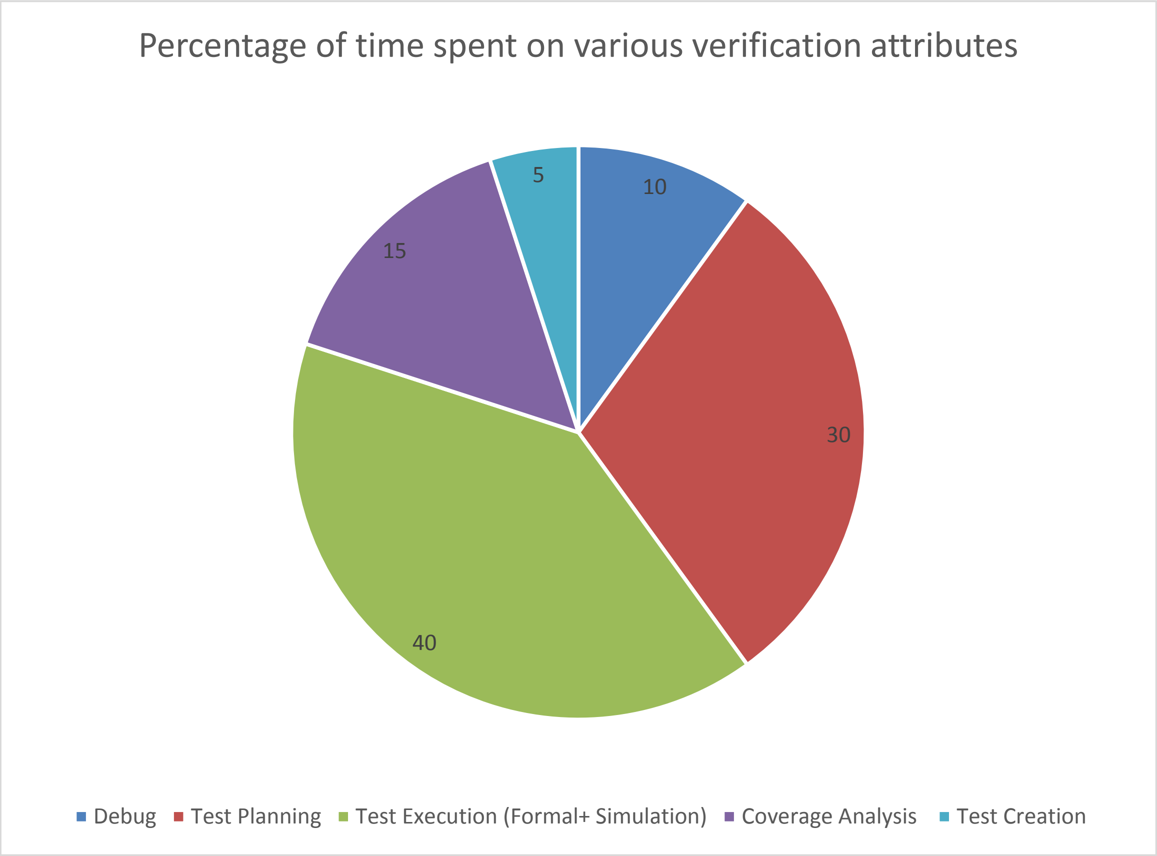


Figure 4.6: Percentage of time spent on various verification attributes

4.3 Synthesis results

The developed Blue-Jay arbiter for the On-Chip Bus is simulated and verified for its functionality. RTL model is taken through synthesis using the Design Compiler; this is done as soon as the functional verification is done. In the synthesis process, the specific technology 90nm library from TSMC is used. The optimized RTL model will be converted to the gate level net-list and mapped to the structural level. Also multiple optimization steps were performed at each stage. Also the "Don't Care" conditions were recognized or inserted as part of the language synthesis process. Structural optimization includes the Boolean minimization.

The design of On-Chip Bus arbiter with 16 different types of arbitration schemes is synthesized and its results are analyzed as follows.

The RTL (Register Transfer Logic) can be viewed. In Figures 4.7, 4.12 and 4.13. It's usually depicted as a black box model after synthesis of the design. It shows the outputs and the inputs of the particular design. We can see gates, flip-flops and MUX by double-clicking on the diagram, in Figures 4.8, 4.10, 4.11, 4.16, 4.17, 4.18 and 4.19.

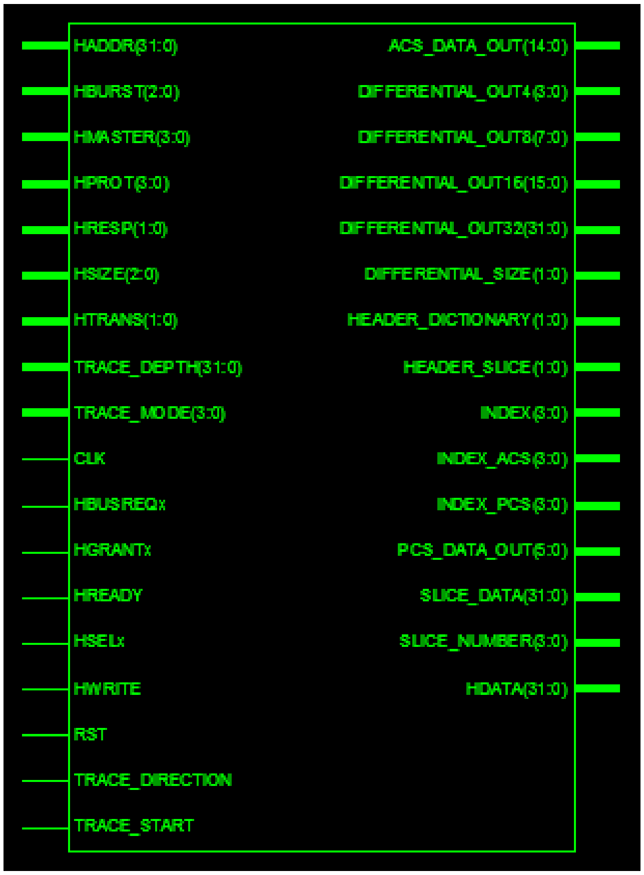


Figure 4.7: RTL schematic view of arbiter

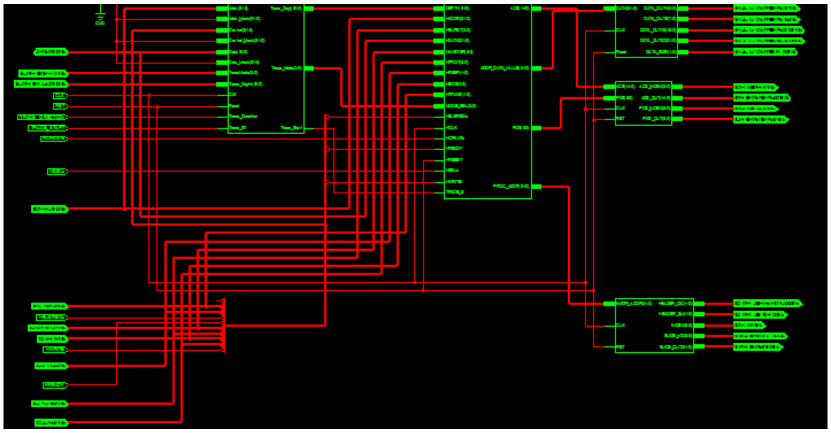


Figure 4.8: Internal view of RTL schematic

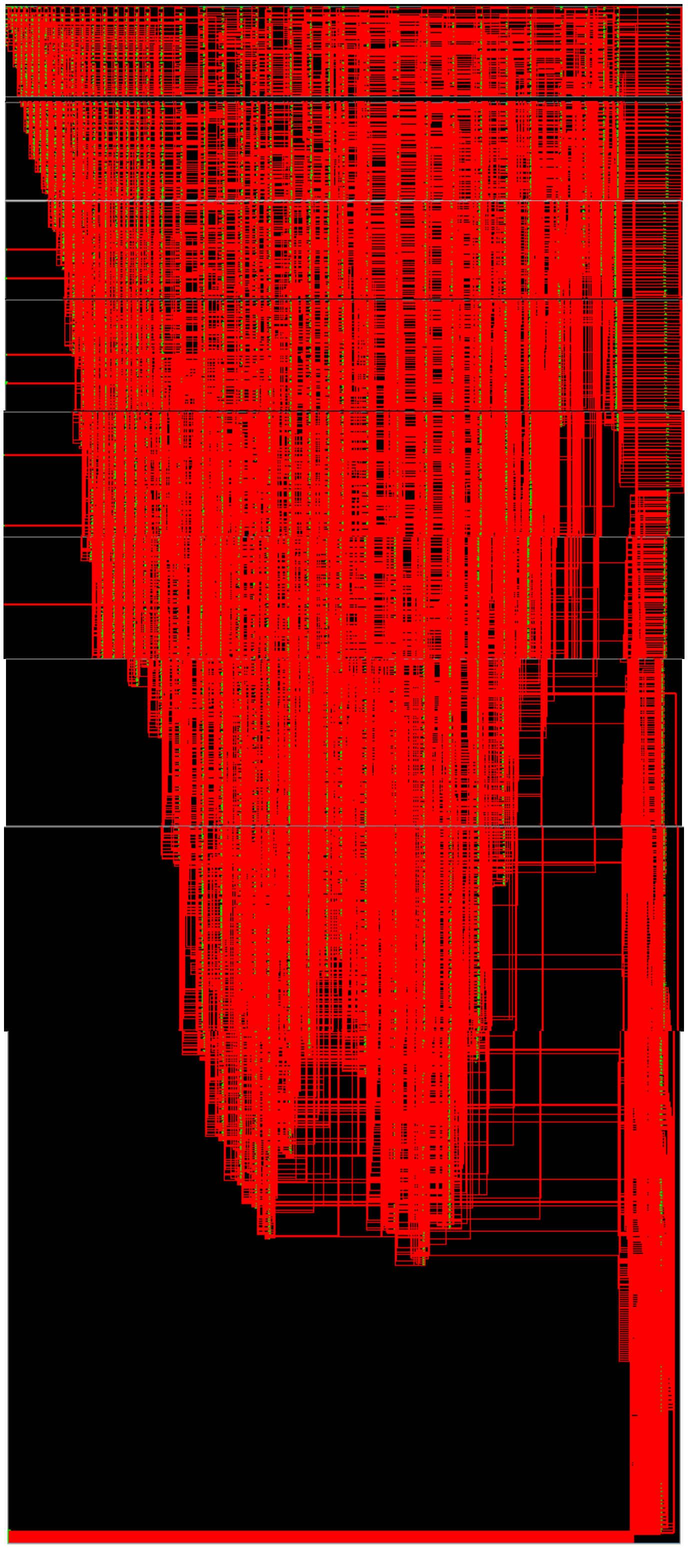


Figure 4.9: Technology schematic view of arbiter (Top Level)

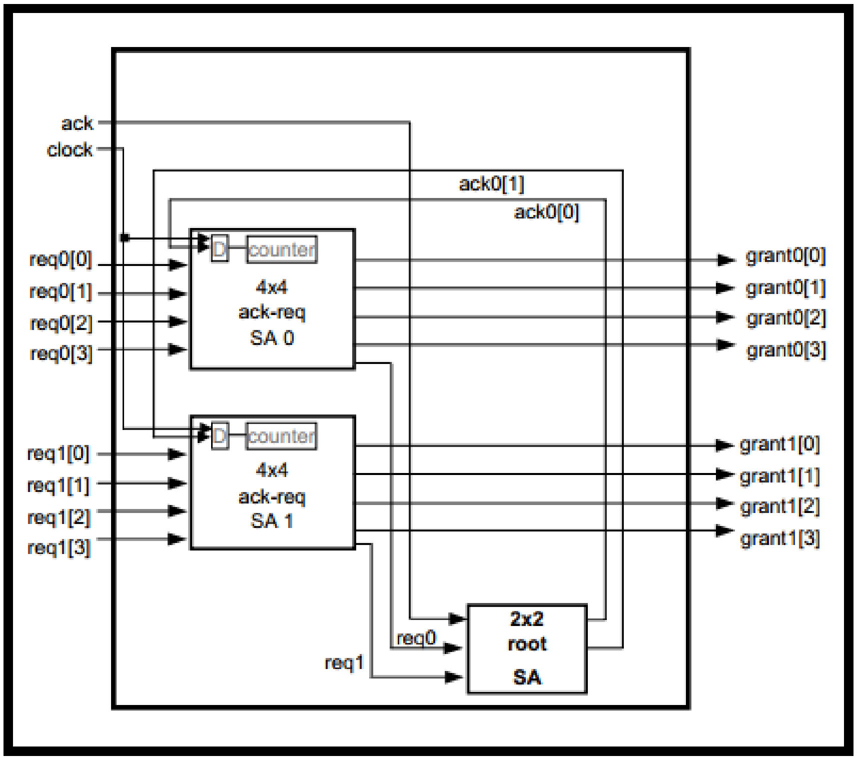


Figure 4.10: Schematic view of priority based grant 8X8

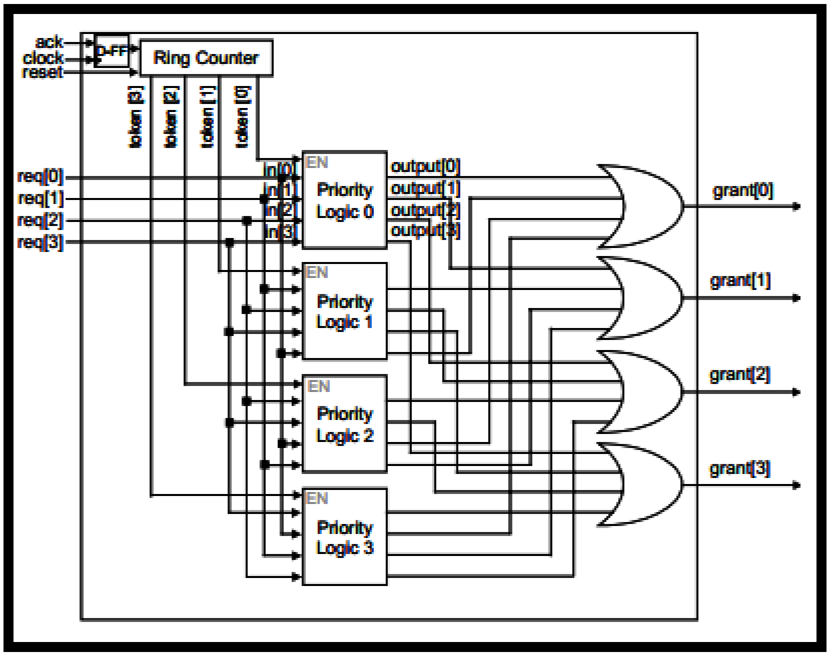


Figure 4.11: Schematic view of priority based grant 4X4

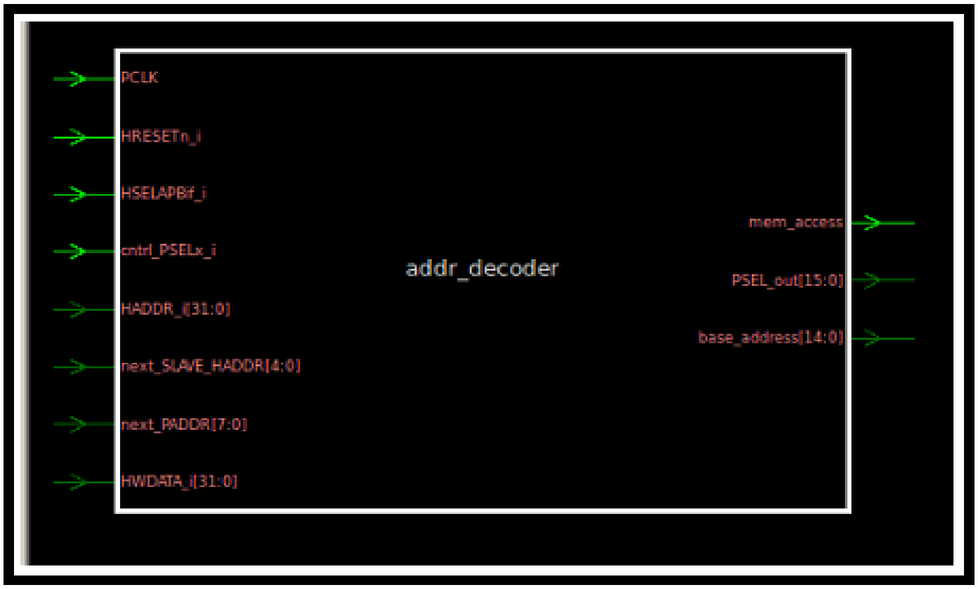


Figure 4.12: Block diagram of decoder

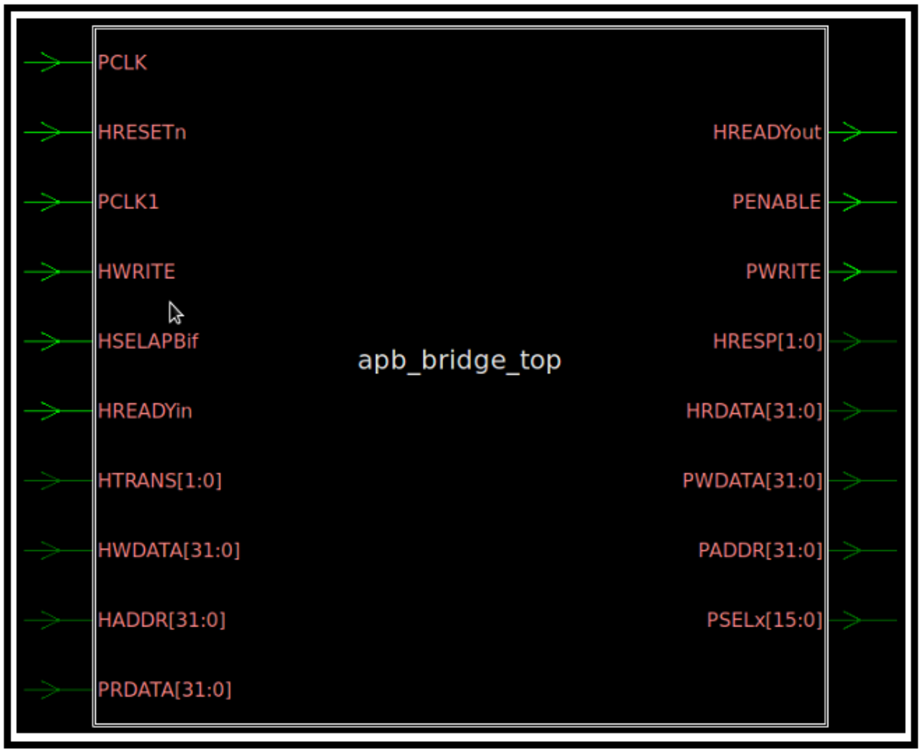


Figure 4.13: Block diagram of the bridge

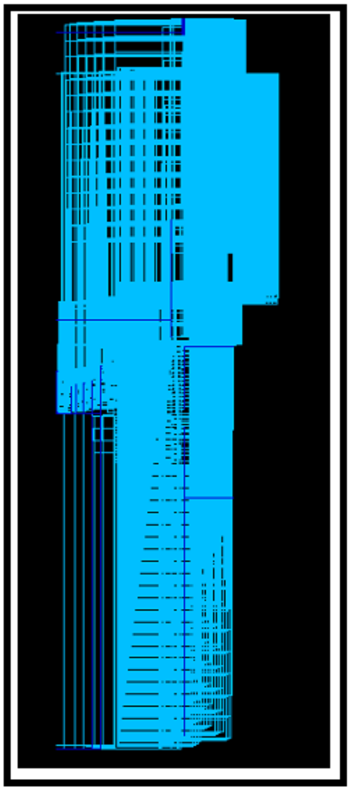


Figure 4.14: Schematic diagram of the bridge

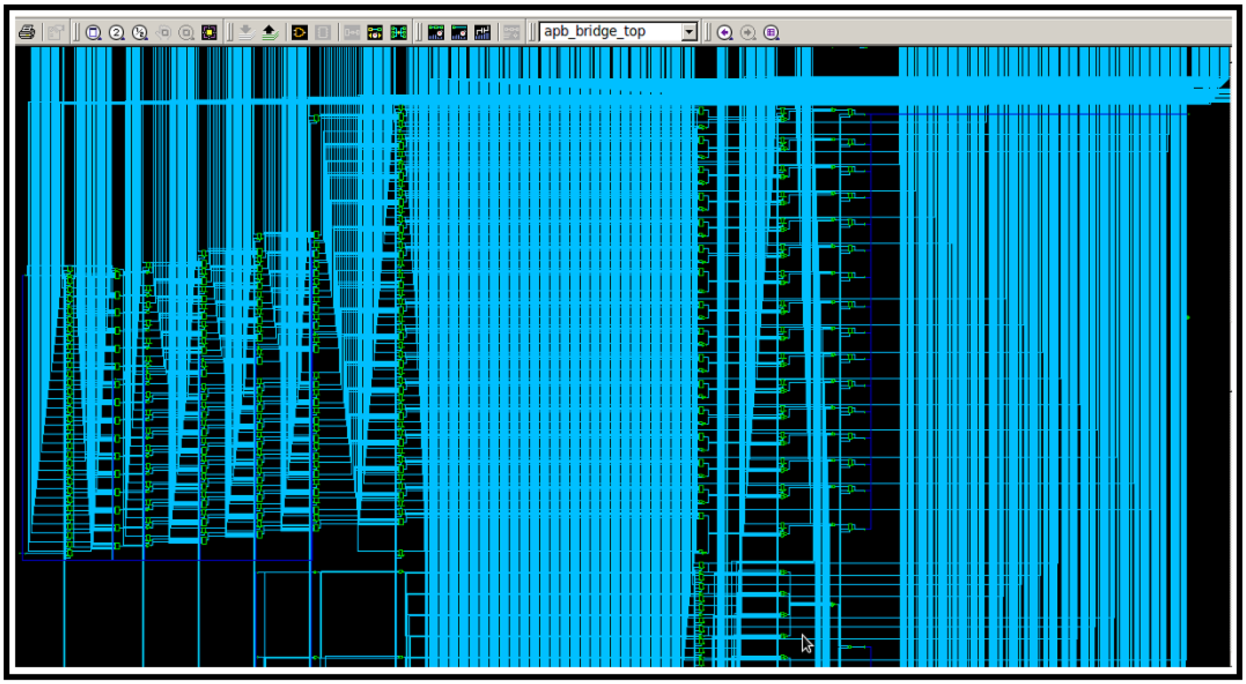


Figure 4.15: Schematic diagram of bridge (zoomed view)

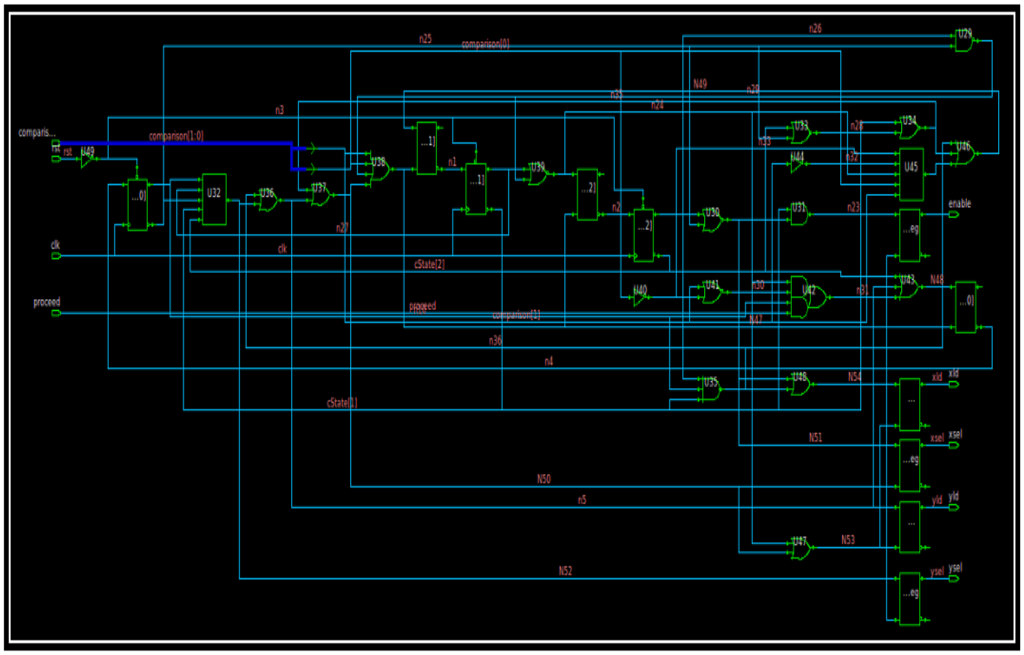


Figure 4.16: Schematic diagram of DMA

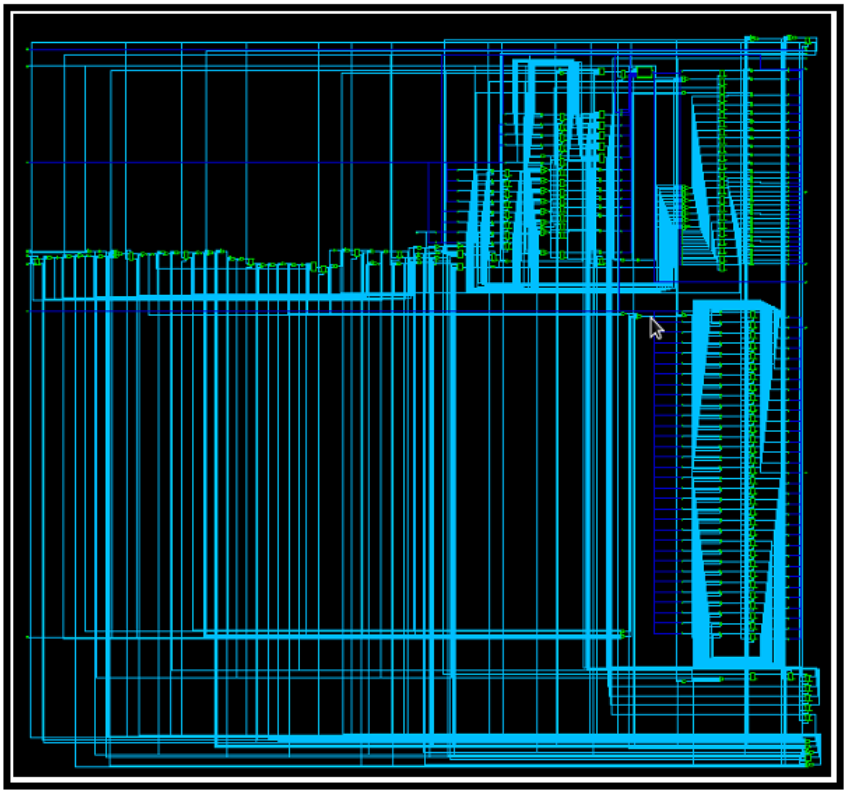


Figure 4.17: Schematic diagram of DMA (broad view)

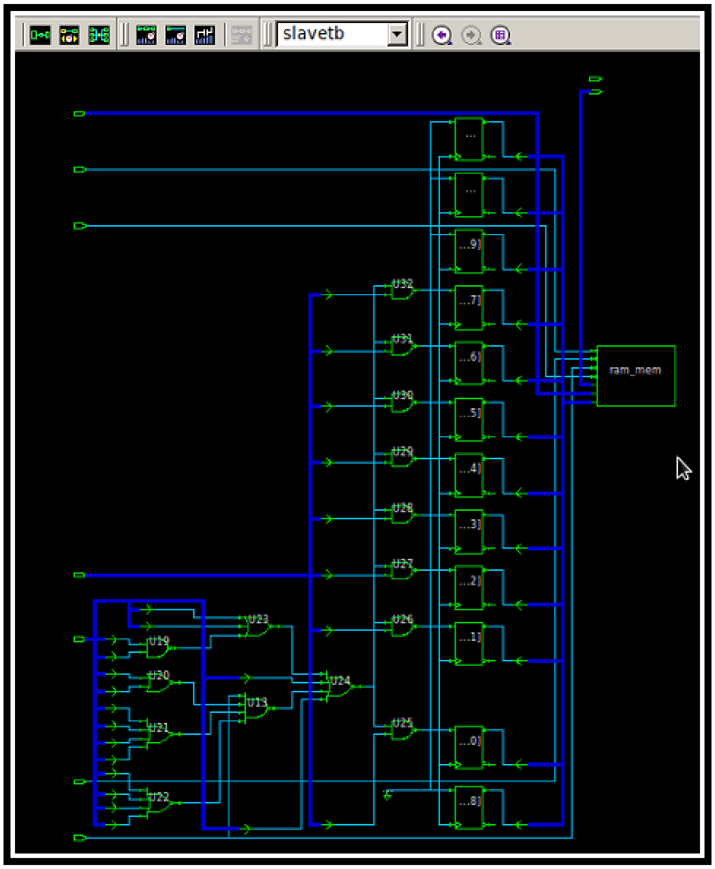


Figure 4.18: Schematic diagram of decoder

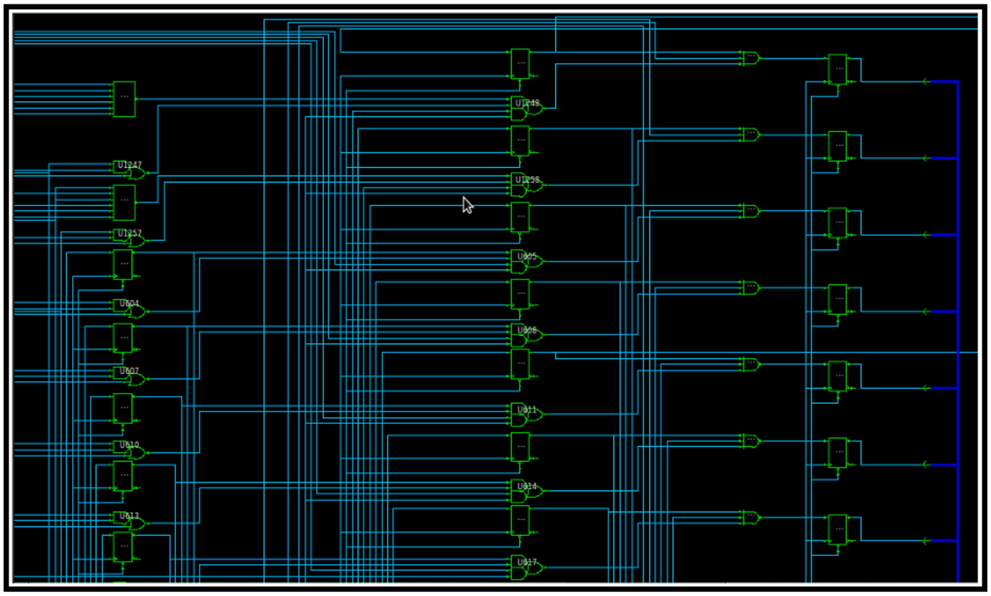


Figure 4.19: Schematic diagram of bridge

Table 4.2: Results for the arbiter synthesis

	Total Wire Length(m)	Average wire length(um)	Total cell count	Cell area (mm2)	Timing Slack (ps)	Number of IO
With clusters	0.988	86	9777	0.987	2001	14
Without clusters	1.087	91	11901	1.09	500	15
improvement	10%	5.8%	21.7%	10.44%	-300%	

Table 4.3: Timing summary of arbiter

	Total Wire Length(m)	Average wire length(um)	Total cell count	Cell area (mm2)	Timing Slack (ps)	Number of IO
With clusters	0.45	40	3021	0.57	200	09
Without clusters	0.77	45	3500	0.99	700	13
improvement	71.1%	12.5%	15.8%	73.6%	200%	

Table 4.4: Results of decoder synthesis

Minimum period	1.25ns (Maximum Frequency:800.5MHz)
Minimum input arrival time before clock	5.864ns
Maximum output required time after clock	2.934ns
Maximum combinational path delay	3.306ns

Table 4.5: Timing summary of decoder

Minimum period	2.02ns (Maximum Frequency: 495 MHz)
Minimum input arrival time before clock	3.44ns
Maximum output required time after clock	3.934ns
Maximum combinational path delay	7.6ns

In the timing summary depicted in Tables 4.3 and 4.5, details regarding time period and frequency are approximate since placement has not been completed. After place and routing is over, we get the exact timing summary. Hence the maximum operating frequency of the synthesized arbiter design is given as 800.5 MHz and the minimum period as 1.25ns. Also the maximum operating frequency of the synthesized decoder design is given as 495 MHz and the minimum period as 2.02ns.

Table 4.2 shows that the total wire can be reduced with cluster formation. There is overall improvement in the arbiter design with the cluster approach. The wire length has been decreased by 10%, the overall area has been decreased by 10.44% and, last but not least, there is significant improvement in total cell count.

Table 4.4 depicts overall improvement in decoder synthesis with the cluster approach. There is overall improvement in the decoder design with the cluster approach. The wire length has been decreased by 71.1%, the overall area has been decreased by 73.6% and last but not the least, there is significant improvement in total cell count.

Table 4.5 depicts the Timing summary of the synthesized decoder where the total combination delay is just 7.6ns. Also the maximum output required time after clock is just 3.934ns.

4.4 Power utilization results

Various instructions have been executed and the power consumption for each instruction is highlighted in Table 4.6. Multiplication takes the most power.

The arbiter Blue-Jay consumes more power while executing a transaction between the processor core and the bridge. This transaction needs to be avoided at all costs and the importance should be given to transfer data from stored memory. Various transactions with their power consumption are shown in Table 4.7.

Instruction	Power Utilized (µw)
XOR	4.96
ADD	4.81
MUL	5.2
MOV	3.99
SHR	5.1
For LOOP	4.73
INC	4.8
CLR	4.3

Table 4.6: Instruction power consumption

Transaction (Data Path)	Total Power (µw)
Processor core to bridge	125.63
Bridge to UART	88.63
Bridge to decoder	45.51
DMA to bridge	115.89

Table 4.7: Transaction power consumption

4.5 Comparison between Blue-Jay and other existing arbiters

Here we compare the features of our arbiter with related bus arbiters: ARM's ABMA (Advanced Microcontroller Bus Architecture [16], CoreConnect [17] and Wishbone [18] and analyze the area, delay and power characteristics. AMBA, CoreConnect and Wishbone are compared in the related work [15].

In this section, we briefly compare the structural area and power utilized by the different arbiters. Blue-Jay, AMBA and CoreConnect have hierarchical structures, where two or more buses are connected through a bridge. There are many inherent advantages for this type of architecture. It has the ability to transfer data independently as long as the components do not refer to resources residing on other buses.

Blue-Jay can support a larger load per data bus line, there is less delay per data transfer, though there is larger energy consumption, and lower bandwidth as compared to AMBA arbiter. In Blue-Jay, slave devices remain in the idle state until they have been accessed. In contrast Wishbone architecture uses only a single bus for high-speed components and also to support its low speed peripheral components. [15] In Blue-Jay basic operations are performed in one of the following two ways: a) the arbiter locks the bus for exclusive use by the master requesting for the bus access; and b) the arbiter does not grant access to a locked slave. AMBA and CoreConnect use a mechanism of locking the slave and the bus for a short time. The access time for this type of centralized arbitration is proportional to the number of masters connected to the bus. Both AMBA and Wishbone buses do not define the arbitration mechanism and support the design of an arbiter with respect to specific applications requirements. CoreConnect defines the arbiter with different priority schemes using a soft-core. The considered SoC buses support various data transfer types. All the buses support handshaking data transfer types. Blue-Jay, AMBA and CoreConnect also support pipelined transfers and split transfers. [15]

From the area plot, Figure 4.21, it can be deduced that maximum area is utilized by the Wishbone bus system, as each master is connected to the slave interface and decoder, while each slave is connected to master interface and the arbiter. This is a complex interconnection system resulting in an area approximately 5 times more than the Blue-Jay bus system. The Blue-Jay arbiter consumes less power than any of the other three arbiters. Figure 4.20 depicts this.

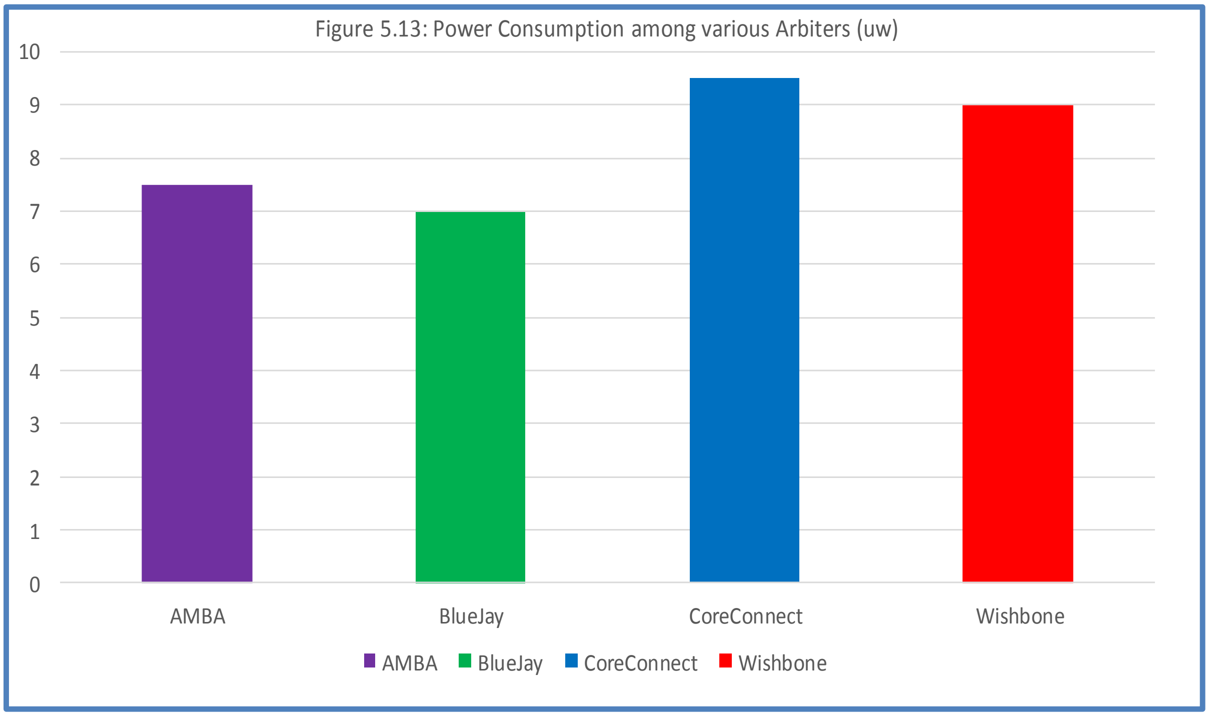


Figure 4.20: Power utilized by different bus arbiters

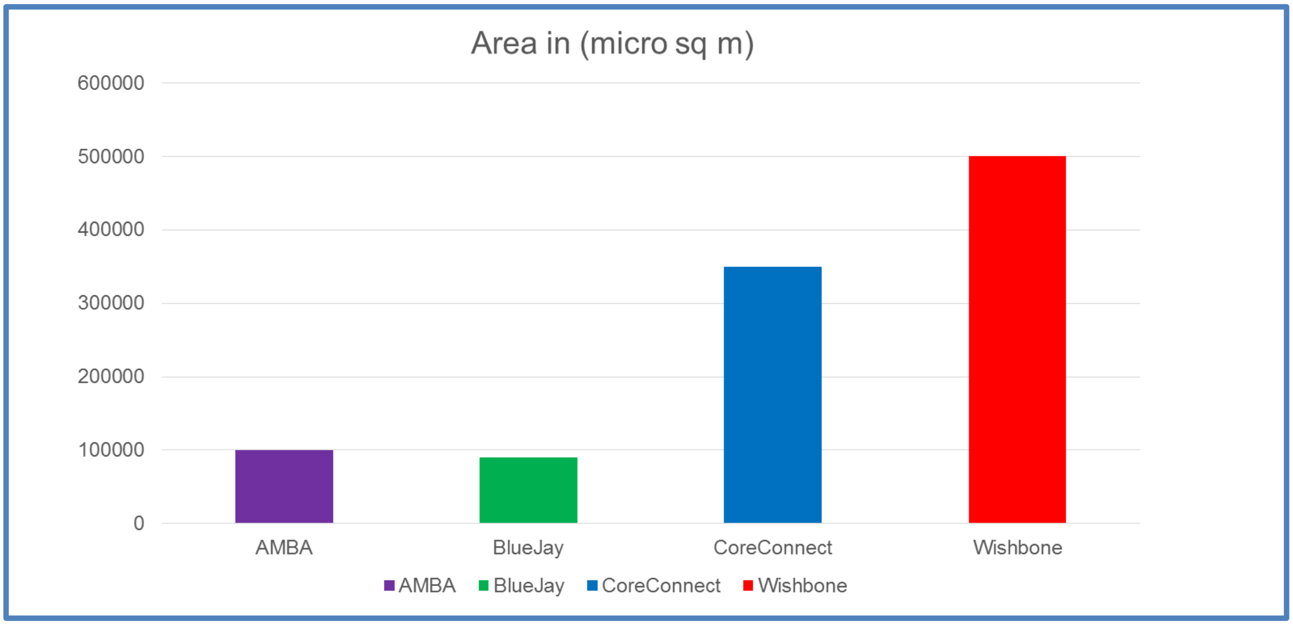


Figure 4.21: Area utilized by different bus arbiters

CoreConnect with crossbar switch uses 28% less area than Wishbone. Blue-Jay with its simple structure and user defined arbitration scheme is very efficient in terms of area which is depicted in the Figure 4.21. In terms of delay, CoreConnect has peak value, as the number of slaves attached to a PLB macro directly affects the maximum attainable PLB clock rate. This is because large systems tend to have increased bus wire load and a longer delay in arbitrating among multiple masters and slaves.

From Figure 4.22 and Figure 4.23 we conclude that the Blue-Jay has superior design with respect to arbiter and bridge design. The Blue-Jay arbiter consumes just 23% of the total power in the bus interconnect design.

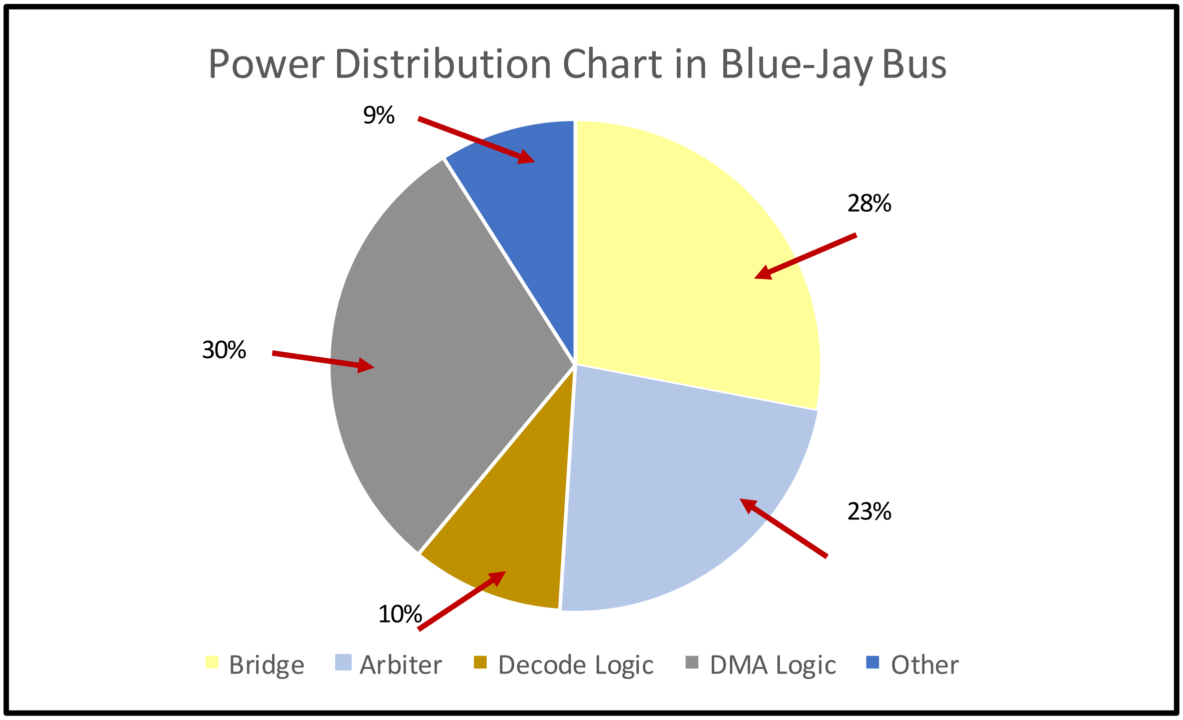


Figure 4.22: Power distribution chart in the bus interconnect Blue-Jay

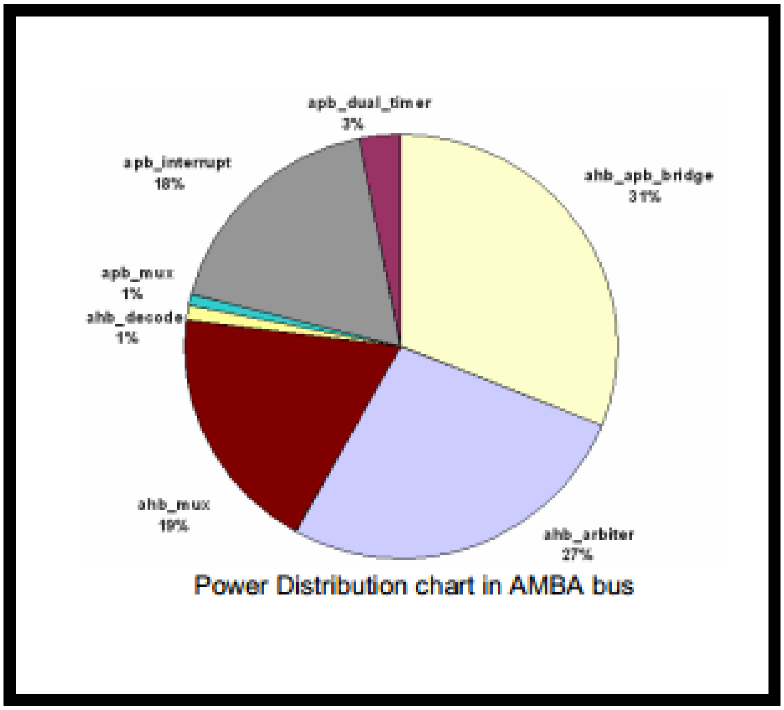


Figure 4.23: Power distribution chart in AMBA bus [15]

Chapter 5
CONCLUSION AND FUTURE WORK

5.1 Conclusion

We have shown the importance of power and area analysis of the arbitration technique in on-chip bus based SoC design. We have also compared the structural and functional differences between three on-chip bus systems. We have evaluated the effect of the arbitration technique on the power consumption of an on-chip bus system. We have also discussed some of the issues facing SoC designers in determining which bus architecture to use in order to provide flexible and efficient bus communication for complex microprocessor SoC designs.

Our newly designed arbiter for the on-chip bus was successfully implemented. The coding was done in Verilog. The RTL simulations were performed using Modelsim from Mentor Graphics and Verdi from Synopsys. The synthesis was done using Design Compiler. The arbiter for the on-chip bus works at a frequency of 800.5 MHz. The Blue-Jay arbiter works properly for all the modes Mode FC, Mode FT, Mode BC, Mode BT, and Mode MT. Blue-Jay arbiter behavior was verified for all test cases.

The bus arbiter costs only about 2144 slice registers and uses 2144 flip-flops, which is relatively small in a typical SoC. In packing module the largest component is the (First In First Out) FIFO buffer. Compression module being the second largest component. Pre-periodical triggering module and Post-periodical triggering module implementation cost is 1032 gates only. Event register is the major component of the event generation module, 1500 gates is used roughly. We implemented two event register in this paper. Extra registers can be added if necessary. The gate count is now reduced because of iterative method. ping-pong architecture is optimized in this research work this is accomplished by sharing the data path instead of duplicating the entire hardware components. The bus arbiter is capable of running at 800 MHz, circuit speed. The design can be easily synthesized using design compiler synthesis technology which is sufficient for most of the current SoC standard. The bus interconnect design can be easily partitioned into pipeline stages and the streamlined compression processing flow can also be implemented for faster clock speed if necessary.

5.2 Future work

In future,

- This work can be improved by implementing it with Advanced Risc Machines (ARM) processors.
- This design can also be used in all System-On-A-Chip (SoC) applications where debugging and performance analysis is difficult.
- The cores can be optimized for better overall power consumption.
- The Bus arbiter design can be modified, it can easily be portioned into more pipeline stages with streamlined compression processing flow to ensure a faster clock speed.

REFERENCES

[1] The Myths of EDA – the 70% Rule, http://www.chipdesignmag.com/martins/2008/11/27/the-myths-of-eda-the-70-rule/ (Accessed: 30 June 2013).

[2] Chris Spear and Greg Tumbush, *System Verilog for Verification: A Guide to Learning the Test Bench Language Features*, Marlboro, MA: Springer June 2006.

[3] François Cerisier and Mike Bartley, SoC Interconnect Verification Challenge, http://www.design-reuse.com/articles/31993/soc-interconnect-verification-challenge.html (Accessed: 1 June 2013).

[4] Mandar Munishwar, Formal Vrification, UCSC extension class notes, http://www.ucsc-extension.edu/ (Accessed: 2 Aug 2013).

[5] http://www.chipdesignmag.com/martins/2008/11/27/the-myths-of-eda-the-70-rule/ (Accessed: 2 Aug 2013).

[6] Rakesh Chadha and J. Bhasker, *ASIC Low Power Primer*, Marlboro, MA: Springer, June 2006, page 50-120.

[7] Charwak Apte, "Power Gating Implementation in SoCs", http://nanocad.ee.ucla.edu/pub/Main/SnippetTutorial/PG.pdf (Accessed: 1 July 2013).

[8] Ismo Hänninen, Class slides, http://www.tkt.cs.tut.fi/kurssit/9626/S08/Chapter_5.pdf (Accessed: 27 July 2013).

[9] Prakash Srinivasan, A. Olugbon and A.T Erdogan, "Power Analysis of Arbitration Techniques for AMBA AHB based Reconfigurable System on Chip", *24th Norchip Conference, 2006.*

[10] Sudeep Pasricha and Nikil Dutt, *On Chip Communication Architecture: System on Chip Interconnect*, the Morgan Kaufmann Series in Computer Architecture, May 2008.

[11] P. Srinivasan, A. Ahmadinia, A.T Erdogan and T. Arslan, “Integrated Heterogenous Modelling for Power Estimation of Single Processor based Reconfigurable SoC Platform”, *2007 IEEE International Symposium on Circuits and Systems.*

[12] ESL and SoC and Embedded world, http://funningboy.blogspot.com/2010_06_01_archive.html (Accessed: 10th June 2013).

[13] *Magellan User Guide Version C-2009.09*, Synopsys Corporation September 2009.

[14] Eung S. Shin, Vincent J. Mooney III and George F. Riley, *Round-robin Arbiter Design and Generation*, *System Synthesis,* IEEE 2002, Pages 243-248.

[15] P. Srinivasan, A. Ahmadinia, A.T. Erdogan and T. Arslan, “Power Evaluation of the Arbitration Policy for Different On –Chip bus based SoC Platform”, *IEEE International SOC Conference, 2007.*

[16] AMBA Specification Rev 2.0. ARM Ltd., 1999. http://www.arm.com. (Accessed: 11th June 2013).

[17] *Processor Local Bus Arbiter Core User’s Manual*, Version 2.0, IBM Corporation, 2001. http://www.ibm.com/chips/products/coreconnec (Accessed: 13th June 2013).

[18] Wishbone System-on-Chip (SoC) Interconnection Architecture for Portable IP Cores Revision B.1, Silicore Corporation, 2001. (Accessed: 17th June 2013).

[19] http://www.synopsys.com/Tools/Pages/default.aspx (Accessed: 4 September 2013).