Verification Of IP at SoC Level

Major Project Report

Submitted in partial fulfillment of the requirements
for the degree of

Master of Technology
in
Electronics & Communication Engineering
(VLSI Design)

By

Mayank Tiwari
(14MECV10)

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481

May 2016

Verification Of IP at SoC Level

Major Project Report

Submitted in partial fulfillment of the requirements
for the degree of

Master of Technology
in
Electronics & Communication Engineering
(VLSI Design)

By

Mayank Tiwari
(14MECV10)

Under the guidance of

External Project Guide:

Mr. Praveen Kumar
SoC Verification Engineer - APG,
STMicroelectronics Pvt. Ltd.,
Greater NOIDA

Internal Project Guide:

Dr. Usha Mehta
Professor (VLSI Design),
Institute of Technology,
Nirma University, Ahmedabad.

Electronics & Communication Engineering Branch
Electrical Engineering Department
Institute of Technology
Nirma University
Ahmedabad-382 481

May 2016

Certificate

This is to certify that the Major Project entitled **"Verification Of IP at SoC level"** submitted by **Mayank Tiwari (14MECV10)**, towards the partial fulfillment of the requirements for the degree of Master of Technology in VLSI Design, Nirma University, Ahmedabad is the record of work carried out by him under our supervision and guidance. In our opinion, the submitted work has reached a level required for being accepted for examination.The results embodied in this major project, to the best of our knowledge,haven't been submitted to any other university or institution for award of any degree or diploma.

Date: Place: Ahmedabad

Internal Guide

Dr. Usha Mehta
(Professor,EC)

Program Co-ordinator

Dr. N. M. Devashrayee
(PG-Coordinator(VLSI Design),EC)

Director

Prof. P. N. Tekwani
(Head of EE Dept.)
(Director, IT-NU)

Declaration

This is to certify that

1. The thesis comprises my original work towards the degree of Master of Technology in VLSI Design at Nirma University and has not been submitted elsewhere for a degree.

2. Due acknowledgment has been made in the text to all other material used.

- Mayank Tiwari
14MECV10

Acknowledgements

First and foremost, sincere gratitude to my manager Mr. Rajesh Jeswani. Also I want to thank STMicroelectronics Pvt. Ltd., Greater NOIDA for assigning me such project and guide me through.

I would like to express my gratitude and sincere thanks to my mentors Mr. Narender Kumar and Mr. Praveen Kumar at STMicroelectronics Pvt. Ltd. , Greater NOIDA for their valuable guidance throughout this period. They have given me valuable advices and support for my project work which I am very lucky to benefit from.

I would like to express my gratitude and sincere thanks to our Director Dr. P. N. Tekwani, Head of Electrical Engineering Department for allowing me to undertake this thesis work and for his guidelines during the review process.

I would like to thank my Program Coordinator, Dr. N. M. Devashrayee, Professor, EC (VLSI Design), Institute of Technology, Nirma University, Ahmedabad for giving valuable support and motivation throughout the academic period.

I would also thank to my Project Guide, Dr. Usha Mehta, Professor & PG Coordinator, VLSI Design, Institute of Technology, Nirma University, Ahmedabad for being a source of inspiration, giving valuable support and timely guidance for the project work.

I take immense pleasure to thank my team members Mr. Nikul Kumar Savaliya and Mr. Ravin Shah and other colleagues , special thanks for helping me on this path and for making project at STMicroelectronics more enjoyable.

I wish to thank my classmates for their delightful company which kept me in good humor throughout the journey.

Last, but not the least, no words are enough to acknowledge constant support and sacrifices of my family members because of whom I am able to complete the degree program successfully.

- Mayank Tiwari
14MECV10

Abstract

In today's regularly advancing technological world its very important for the manufacturer to achieve a very great quality in the minimum time, in short a good time to market is high on demand. The verification of design enacted makes it a very vital part to ensure the quality of product and to ensure a bug-free design when provided to the customer. Here in this thesis the main area of focus is to complete a verification flow from RTL[Register Transfer Level] to GLS[Gate Level Simulation] with timing constraints. The synthesized gate level netlist is generated, which also has the influence of physical constraints and timing constraints that are generated from RTL. Verification Plan is designed as per the functional equivalency understood from the design. In SoC level most of the IP's come under the category of the Black Box verification, which makes it very vital for the verification engineer to design the flow strictly as per specifications provided by the IP owner.Once the plan is formed the test-cases are developed as per the plan and the IP design constraints, the connectivity with the SoC environment, interconnects and access the core.

In this project our aim is to cover a complete verification flow covering all the IPs functionality and other peripherals. Every team member is assigned with the verification ownership of different IPs that are included in the SoC. I was assigned the ownership of some critical IPs such as Graphic Engine and Smart Card Interface[SCI]. I had developed a complete test-bench flow stretching from verification plan to RTL verification and finally completing the verification flow at Gate Level

Contents

List of Figures

Abbreviations

AXI Advanced eXtensible Interface

DMA Direct Memory Access

AMBA Advanced Microcontroller Bus Architecture

ID Identification

IP Intellectual Property

SoC System on Chip

GUI Graphic User Interfaces

RTL Register Transfer Logic

DPF Digital Picture Frames

MID Mobile Internet Device

HDL Hardware Description Language

AHB Advanced High-performance Bus

ICG Integrated Clock Gating

Chapter 1

Introduction

A question often comes to our mind about SoC, what is it and why it has taken the market by storm in such a less time. Let us first make ourselves clear that a System On a Chip (SoC) is not an application or innovation in fact its an art that came into existence and resemblance with the sudden increasing interest in VLSI. Typical SoCs consists of many IPs, however a general SoC usually have contents like microprocessor or microprocessor sub-system, microprocessor bus, peripheral bus like AHB, bridging between buses for different frequencies and other IP's.

What is so specific about Automotive SoC ?
The SoC designed for the extreme purpose like that of for Automotive/Industrial requirements demand very high an error free performances from their designs since these products have to prove themselves worthy to be working at extreme temperatures, humidity and other similar constraints. They include a series of different IP like that of CAN, ETHERNET, FLEXRAY and internal clock oscillator that ensures full functionality without including external crystal oscillator. Most important part for these is the way of their packaging.

A process used to demonstrate the functional correctness of a design to making sure you are verifying that you are indeed implementing what you want to ensure that the result of some transformation is as expected. Verification is a process used to demonstrate the functional correctness of a design.Figure 1.2 shows the verification process.

1.1 SoC Verification

A common industrial experts does point out that many of the bugs generated in the SoC are a result of an incomplete integration of IP cores. Now in order to verify it we are required to develop testcases that does cover out the functional, properties of the IP that may lead to development of critical bugs later on in RTL level. However this is not as simple as it sounds, it requires a lot of strategies and scenario coverage that has to be included during

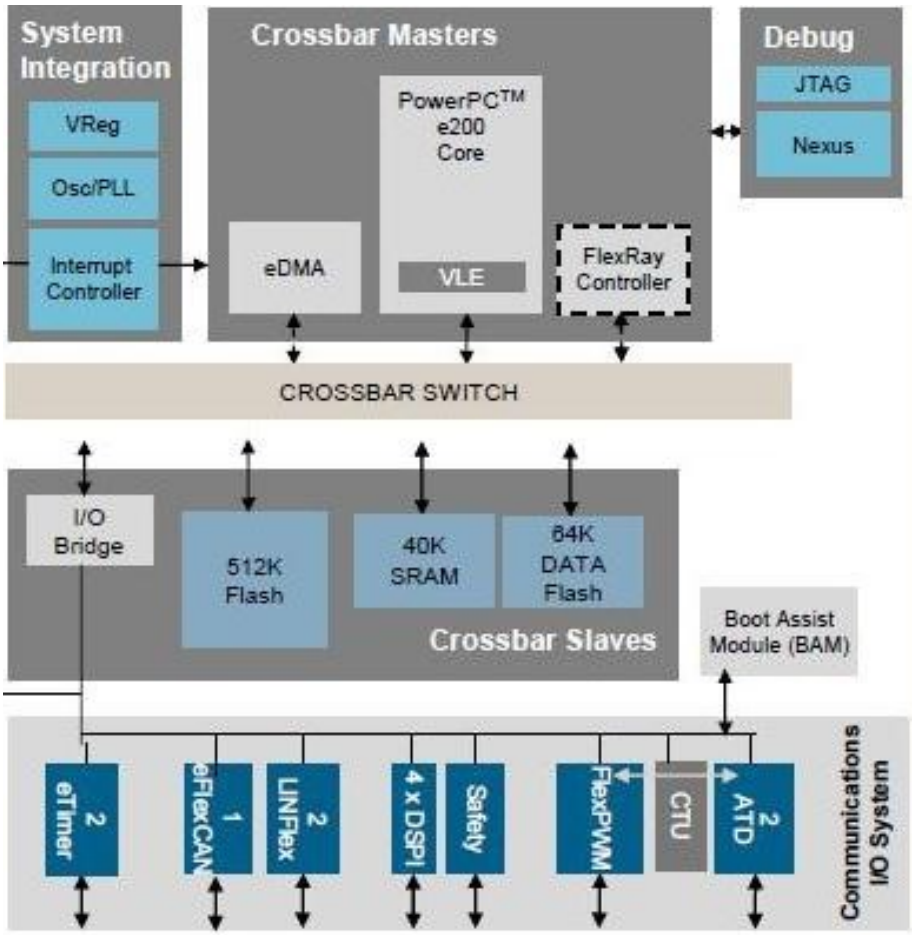

Figure 1.1: SoC Architecture

verification process. Challenges that are faced during this whole process could be defined as:

- **Integration:** As stated earlier in the above paragraphs the most important aspect for a verification engineer is to check the integration of IP with the SoC in case of the SoC verification. One major assumption made during IP verification at SoC level is that IP is properly and perfectly verified by the third party vendor. Thus special focus is just kept on verifying its functionality related to SoC.

- **Complexity:** Another huge challenge faced during SoC verification is related to the complex nature of the SoC now a days. The requirements of smaller, compact and low power SoC architectures have no doubt given a great satisfaction to the users but on the other hand they have created a lot of havoc for the Verification engineers. It actually rises a regular demand of defining such test scenarios that can meet the challenging complicated scenarios and corner cases that may give birth to any design bug unexpectedly

The challenges that are mentioned above does demand a strong and rigorous need for verification of each of SoC components separately, which requires very compatible and reliable

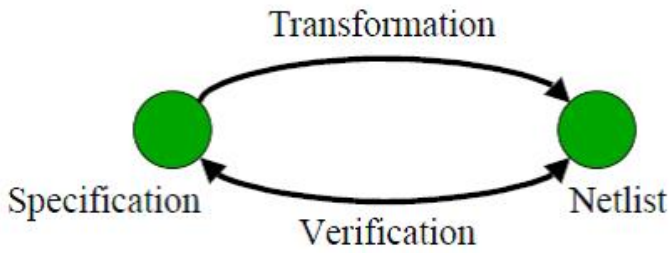

Figure 1.2: Verification Process

methodologies and tools that can handle a full system verification. This requirement for extensive verification points out towards the need of a high level of automation; in order to make this verification task practical.

1.2 Trends in Traditional SoC Verification

Companies and design groups around the world have many different approaches to verification and specifically to SoC verification. Let's take a look at some of the current trends and their limitations.

Test plans: Consists of a detailed test plans that covers all the functional behaviour of the IP with respect to the designated SoC environment. To check this behaviour we have to develop a series of testcase accordance to its design environment. However due to complex nature of the SoC it becomes extremely difficult to write a test case detailing each and every functional aspect and all the corner scenarios via directed tests.

Chapter 2

Literature Review

This section covers the basic and advanced features of the AMBA[Advanced Micro-controller Bus Architecture] BUS protocols like AXI, AHB and APB. Understanding of these protocols is very necessary in order to reason the proper flow of data and instruction in both directions that is from IP to Core or from core to IP. Its not necessary to understand this protocol only for the purpose of knowing the data flow, but it plays a very vital role in the debugging in case a if any probable design bug is encountered.

2.1 AMBA AXI Protocol

The AMBA AXI protocol is targeted at high-performance, high-frequency system designs and includes a number of features that make it suitable for a high-speed sub-micron interconnect.

The important aspects of the AXI protocol are:

- Availability of different buses for address/control and data phase
- Support for unaligned data transfers by the use of byte strobes
- Availability of burst-based transactions which requires to be issued only with starting address
- Assignment of the different Read/Write data channels that can enable low-cost *Direct Memory Access* (DMA)
- Powerful feature that allows it to issue multiple outstanding addresses
- Ability to complete transaction out of order
- Can perform easy addition of register stages to provide timing closure.

As well as the data transfer protocol, the AXI protocol includes optional extensions that cover signaling for low-power operation.

2.1.1 Architecture

The AXI protocol is burst-based. Every transaction has address and control information on the address channel that describes the nature of the data to be transferred. The data is transferred between master and slave using a write data channel to the slave or a read data channel to the master. In write transactions, in which all the data flows from the master to the slave, the AXI protocol has an additional write response channel to allow the slave to signal to the master the completion of the write transaction(1).

The AXI protocol enables:

- address information to be issued ahead of the actual data transfer
- support for multiple outstanding transactions
- support for out-of-order completion of transactions

Figure 2.1 shows how a read transaction uses the read address and read data channels.

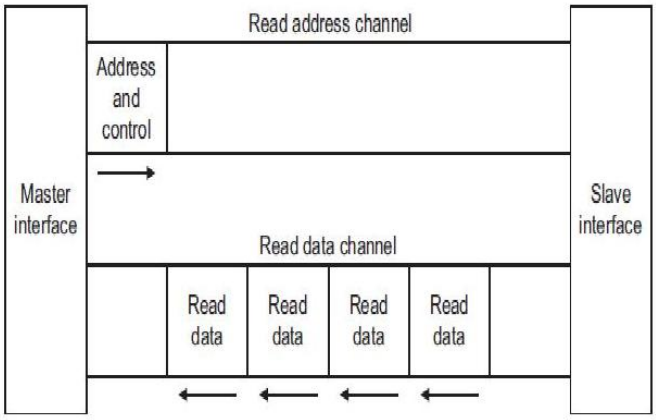

Figure 2.1: Channel architecture of reads

Figure 2.2 shows how a write transaction uses the write address, write data and write response channels.

Channels consists of a set of information signals and uses a two-way VALID and READY handshake mechanism. The information source uses the VALID signal to show when valid data or control information is available on the channel. The destination uses the READY signal to show when it can accept the data. Both the read data channel and the write data channel also include a LAST signal to indicate when the transfer of the final data item within a transaction takes place(1).

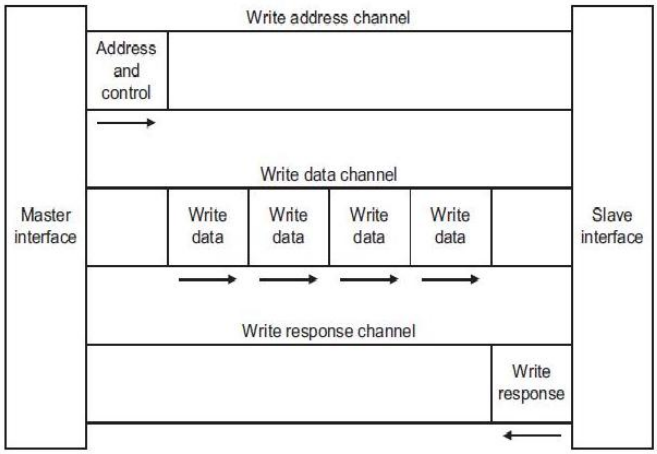

Figure 2.2: Channel architecture of writes

Interface and Interconnect

A typical system consists of a number of master and slave devices connected together through some form of interconnect.

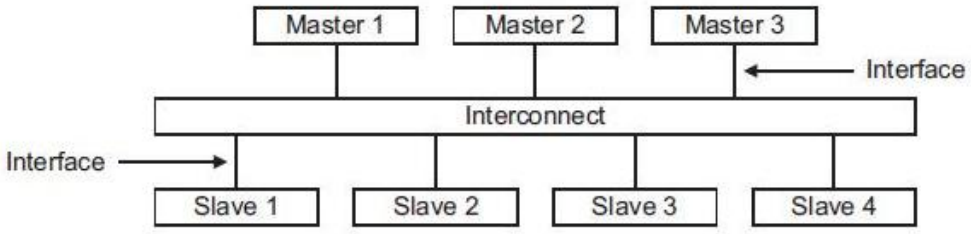

Figure 2.3: Interface and Interconnect

The AXI protocol provides a single interface definition for describing interfaces:

- between a master and the interconnect
- between a slave and the interconnect
- between a master and a slave.

Most systems use one of three interconnect approaches:

- shared address and data buses
- shared address buses and multiple data buses
- multilayer, with multiple address and data buses

Register Slices

Each AXI channel transfers information in only one direction. It is also possible to use register slices at almost any point within a given interconnect. It can be advantageous to use a direct, fast connection between a processor and high-performance memory, but to use simple register slices to isolate a longer path to less performance-critical peripherals(1).

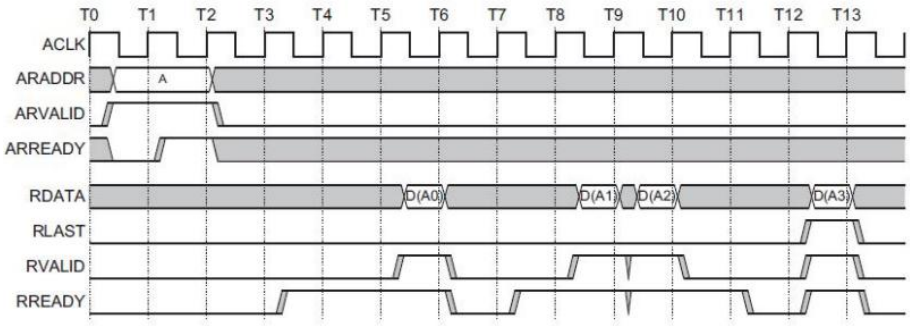

Figure 2.4: Read burst

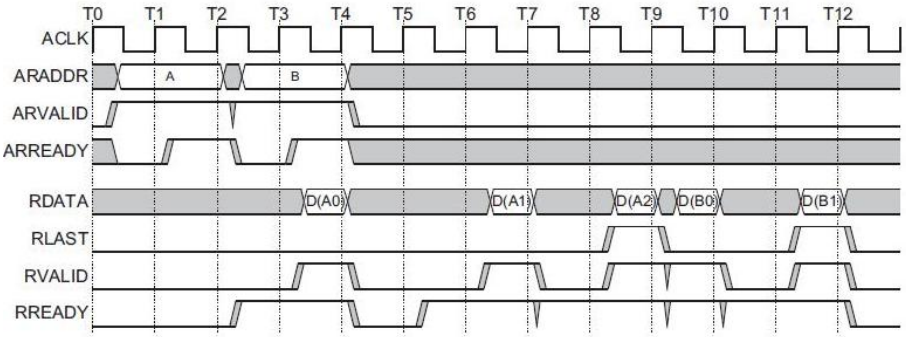

Figure 2.5: Overlapping read bursts

Write Burst Example

The master then sends each item of write data over the write data channel. When the master sends the last data item, the WLAST signal goes HIGH. When the slave has accepted all the data items, it drives a write response back to the master to indicate that the write transaction is complete(1).

Transaction Ordering

The AXI protocol enables out-of-order transaction completion. It gives an ID tag to every transaction across the interface. The protocol requires that transactions with the same ID tag are completed in order, but transactions with different ID tags can be completed out of order(1).

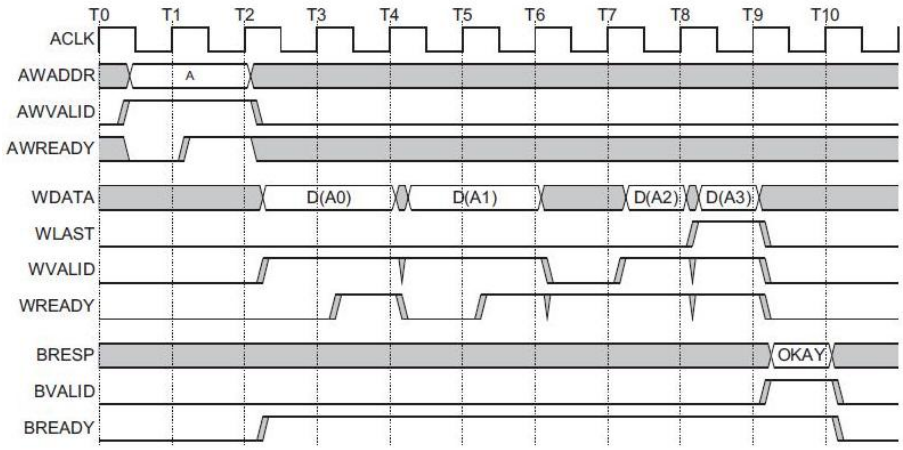

Figure 2.6: Write burst

Out-of-order transactions can improve system performance in two ways:

- The interconnect can enable transactions with fast-responding slaves to complete in advance of earlier transactions with slower slaves.
- Complex slaves can return read data out of order. For example, a data item for a later access might be available from an internal buffer before the data for an earlier access is available.

If a master requires that transactions are completed in the same order that they are issued, then they must all have the same ID tag. In a multimaster system, the interconnect is responsible for appending additional information to the ID tag to ensure that ID tags from all masters are unique.

2.1.2 Handshake Process For AXI

All five channels use the same VALID/READY handshake to transfer data and control information. This two-way flow control mechanism enables both the master and slave to control the rate at which the data and control information moves. The source generates the VALID signal to indicate when the data or control information is available. The destination generates the READY signal to indicate that it accepts the data or control information. Transfer occurs only when both the VALID and READY signals are HIGH(1).

In Figure 2.7, the source presents the data or control information and drives the VALID signal HIGH. The data or control information from the source remains stable until the destination drives the READY signal HIGH, indicating that it accepts the data or control information. The arrow shows when the transfer occurs(1).
In Figure 2.8 the destination drives READY HIGH before the data or control information is valid. This indicates that the destination can accept the data or control information in a

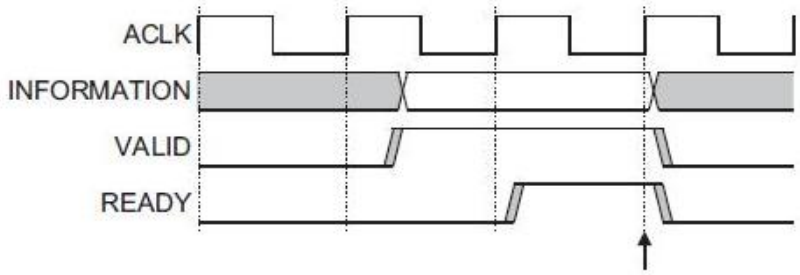

Figure 2.7: VALID before READY handshake

single cycle as soon as it becomes valid(1).

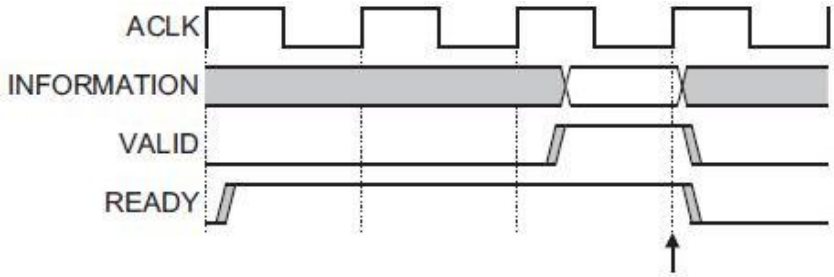

Figure 2.8: READY before VALID handshake

In Figure 2.9, both the source and destination happen to indicate in the same cycle that they can transfer the data or control information. In this case the transfer occurs immediately. The arrow shows when the transfer occurs(1).

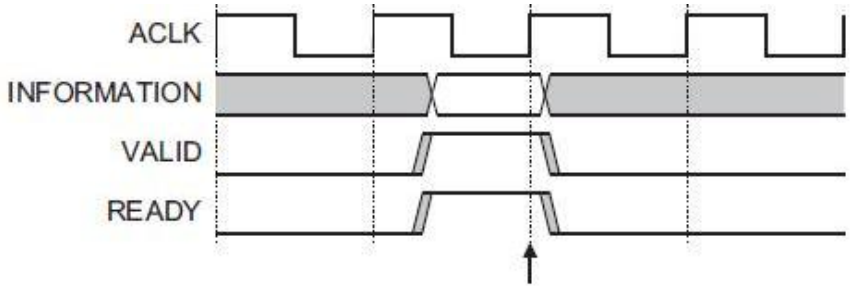

Figure 2.9: VALID with READY handshake

2.2 AHB Bus Protocols

HCLK It's the clock input to all elements in an AHB system and is assumed to come from some external clock generator. All of the AHB logic is triggered at the rising edge. The AHB specification does not give any specific clock frequency for AHB, but implementations above about 200MHz are pretty uncommon. In case if multiple AMBA clock domains are

required in the system, then it is required in order to build multiple AHB buses with asynchronous AHB2APB bridges to connect them together. As a matter of fact it is allowed and in fact common for devices to use other clocks thus it is only their AHB interface that must use HCLK.

HRESETn An active low signal which is an input to all elements in AHB. It is typically asserted low for a few cycles.

HADDR[31:0] 32 bit output bus from the master, it indicates the address to be used for a transfer. It acts as an input to the decoder and to all of the other slaves. The decoder uses the most significant bytes [MSBs] of the address to decode select line for the slave to be used after this the slave further uses the remaining bits to decode the address which is to be accessed within the slave (eg a particular register or memory location). The address is then aligned according to the transfer size (HSIZE), its applicable even for IDLE transfers.

HSIZE[2:0] It's the master output that indicates the size of the read or write transfer to be performed. The protocol supports up to 1024 bit transfers, but masters and slaves typically use 8, 16 and 32 bit sizes (and 64/128 in higher performance systems with larger data-bus widths).

HTRANS[1:0] Its an output from the master and input to slave which symbolises the type of transfer intended to be done. The allowable types are Non-Sequential (NONSEQ), Sequential (SEQ), Idle (IDLE) and Busy(BUSY). Idle indicates that no transfer will be performed. The system respond to idle transfers with "HREADY=1 and HRESP=OKAY". Busy cycles occur according to the part of a burst and it indicates that the master is currently unable to perform the next transfer in the burst on the current cycle. Non-sequential indicates that the current transfer is not related to previous transfers (i.e. a single transfer, or the first of a burst). Sequential indicates the second or subsequent transfer of a burst. For sequential transfers, control signals may not change value compared to the previous transfer and there are rules about how the address relates to the previous address (depending upon the type of burst).

HWRITE An output from the master which indicates the direction for data transfer i.e. for a write from master to slave and for a read from slave to master.

HBURST[2:0] It's a master generated bus which tells the system about the type of burst it is performing i.e. a single transfer or a fixed, incrementing (INCR) or wrapping (WRAP) burst. Also gives information to the slave related to the length of the burst (1, 4, 8, 16 or undefined length).

HPROT[3:0] Allows the master to instruct the slave about the characteristics of the transfer including whether it is privileged or non-privileged, opcode or data and also about the cacheable and write buffer properties of the access. Not all slaves need to make use of this bus.

HWDATA[n-1:0] Master output data-bus which contains the data that master wants to write to an address in a slave. The data-bus width is not fixed by the protocol but is typically 32 or 64 bits.

HRDATA[n-1:0] Slave output data-bus (input to the master) which contains data that is to be read by master from the selected slave.

HSELx A set of system-specific decoder outputs which is generated combinatorially from the decoder's HADDR input (ie no register should be used). It produces a group of select signals, predefined for each slave.

HRESP[1:0] It's an output from slave (input to the master) which indicates about the state of the transfer The slave gives the OKAY response when a successful transfer is done. ERROR response is generated when a problem was encountered with the transfer. RETRY response is generated indicating that the master should attempt to access again SPLIT allows the slave to perform a slow transfer offline, without locking up the bus and requires the master to retry the access when it is next granted the bus when the split access completes.

HREADY It's an output generated from each slave which shows that the transfer is completed. Holding HREADY low is effectively placing wait state during ongoing transfer. Note that slaves will also have an HREADY input so that they can see when previous transfers from other slaves have completed. The protocol does not enforce any quality-of-service or deadlock requirements, so if a slave holds HREADY permanently low, it can deadlock the AHB system. HBUSREQx Output from a master and input to the arbiter signifies that the master request to granted control of the bus Is accepted.

HGRANTx Output from the arbiter, input to the master, indicating that the master has been given the bus.

HLOCK/HMASTLOCK In a full AHB system consisting of multiple masters, a master will produce an output HLOCK to show that the following transfer is locked and the arbiter will produce a corresponding HMASTLOCK signal, with same timing as HMASTER.

In an AHB-lite system, the master will often output HMASTLOCK directly (or will need a small amount of additional logic to do so). A series of locked transfers are atomic, which means that no other transfer from another master may occur during the locked sequence in

short it restricts the arbiter to change any grant while this happens. This is typically used for the control of shared resources in multi-master systems.

2.3 Transfers

1. **Basic Transfers:-** In AHB-Litebasic the transfer contains two phases:

   (a) **Address:**This usually remains for one **HCLK** cycle and have an option of extension by the use of the last bus transfer in accordance to the need.

   (b) **Data:**This could require many **HCLK** cycles in order to finish the required data transfer, it thus uses the **HREADY** signal by which it controls the total number of clock cycles required for the transfer to complete.

   HWRITE This typically have command over the direction of data flow i.e. it tells whether the data transfer to or from the master. Thus, when: **HWRITE**, When this signal goes HIGH this means that the master is about to do a write transfer and the master thus transmits the data over the write data bus, **HWDATA[31:0]** **HWRITE**, When this signal goes LOW this means that the master is about to do a read transfer from the slave and the slave transmits the data on the the read data bus, **HRDATA[31:0]**.

   Example of simple transfer can be stated as below:-

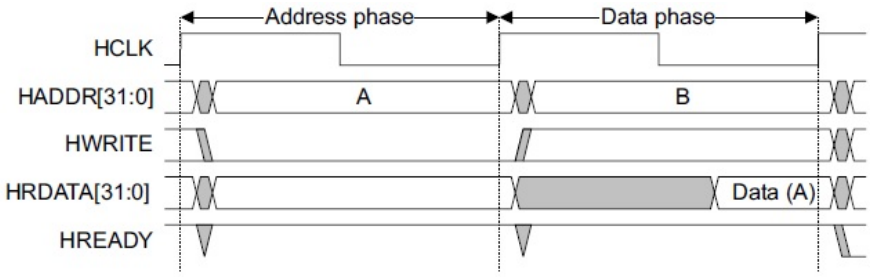

Figure 2.10: Basic WRITE Transfer

2. **Burst Operation:-**AHB supports the Bursts transfers of 4, 8, and 16-beats. It also supports a wide range of other bursts like undefined length bursts, and single transfers. Both incrementing as well as wrapping bursts are supported by this:

   Incrementing bursts, this means that the master is accessing the sequential locations always in the regular incremental form, thus here the address assigned to each transfer in the current burst will always represent the incremented address version of the previous address.

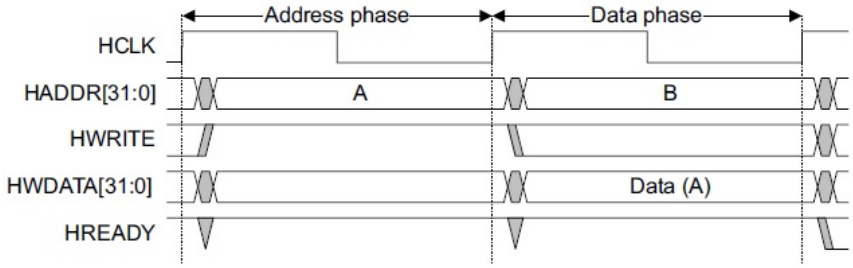

Figure 2.11: Basic READ Transfer

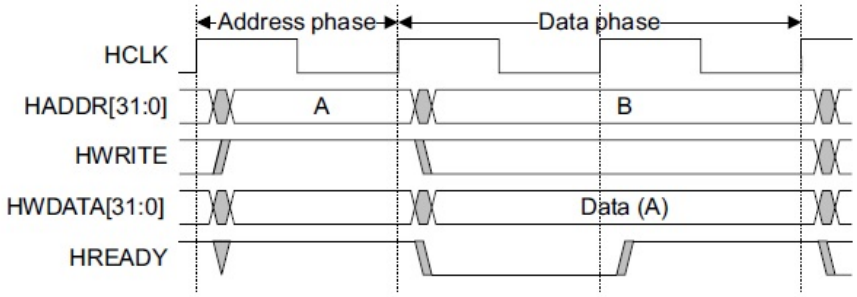

Figure 2.12: Basic Transfer with wait state

Wrapping bursts, this means that the master is accessing the address locations that wraps around whenever they are intended to leap-forward an desiugnated address boundary. The address boundary here is defined as per the product of the number of beats in a burst stated as per in **HBURST** and the size of the transfer stated as per in **HSIZE**.

Although its allowed for the master to access as much memory as it has defined but its recommended that Master shall not attempt to start an incrementing burst targeting to take memory more than 1KB, since at this point it will start writing in the different location of some other slave.
Single transfers are performed by Master using either: **SINGLE** burst **UNDIFINED** length burst that is burst of length one.

Condition for Burst Termination

(a) **Burst termination after a BUSY transfer**
Once the burst starts, the master could use BUSY transfers if it decodes that more time is required before continuing with the next transfer in the burst(1).
In case of an undefined length burst, i.e. INCR, the master may insert BUSY transfers in order to decide if more data transfers are required or not. In such cases,it is acceptable for the master to perform a NONSEQ or IDLE transfer that then terminates the undefined length burst immediately(1).
However, the protocol does not allow the master to end a burst with a BUSY

HBURST[2:0]	Type	Description
b000	SINGLE	Single burst
b001	INCR	Incrementing burst of undefined length
b010	WRAP4	4-beat wrapping burst
b011	INCR4	4-beat incrementing burst
b100	WRAP8	8-beat wrapping burst
b101	INCR8	8-beat incrementing burst
b110	WRAP16	16-beat wrapping burst
b111	INCR16	16-beat incrementing burst

Figure 2.13: Burst Transfer Table

transfer in case of a fixed length bursts of type like(1):
incrementing INCR4, INCR8, and INCR16(1).
wrapping WRAP4, WRAP8, and WRAP16(1).
These fixed length burst types must terminate with a SEQ transfer. The master is not permitted to perform a BUSY transfer immediately after a SINGLE burst. SINGLE bursts must be followed by an IDLE transfer or a NONSEQ transfer(1).

(b) **Early burst termination**
Bursts can be terminated by either:

- Slave error response(1)
- Multi-layer interconnect termination(1)

Slave error response
In case if slave provides an ERROR response then the master can cancel the remaining transfers in the burst. Although it's not a necessary requirement so it's acceptable for the master to continue the remaining transfers in the burst.If the master does not complete that burst then there is no requirement for it to rebuild a burst when it again accesses that slave next time. For example, if a master only completes two beats of an eight-beat burst then it is not necessary for the master to complete remaining five transfers when the next time it accesses that slave(1).

Multi-layer interconnect termination
Even though masters are not permitted to terminate a burst request early, slaves must be designed to work correctly if the burst is not completed. However this

property is not required to be discussed here since our SoC works in the single processor based environment(1).

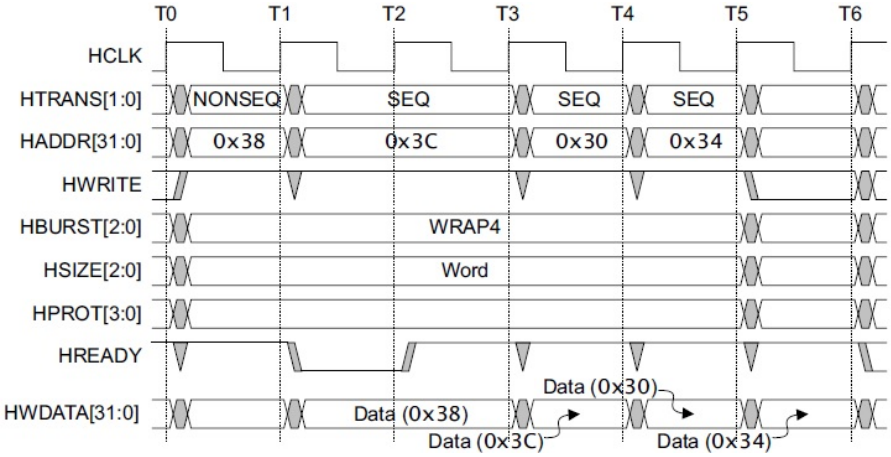

Figure 2.14: Burst example1

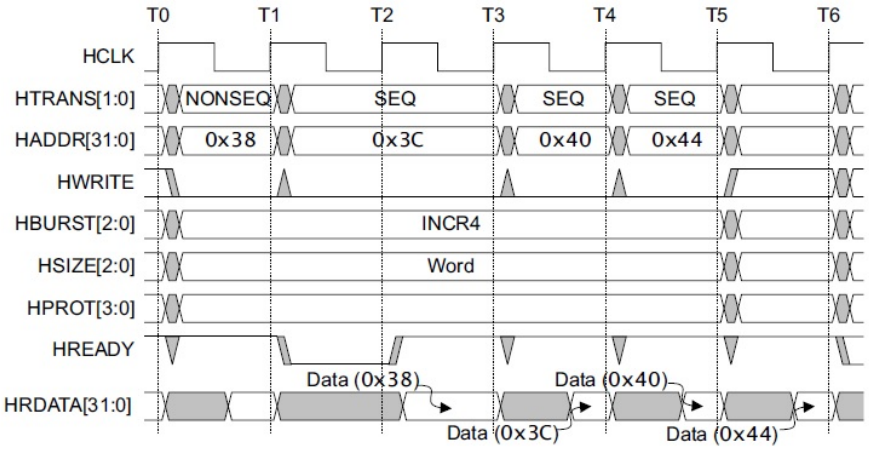

Figure 2.15: Burst example 2

Chapter 3

Graphic Engine - IP working and Verification Flow

Graphics processing IP defines a high-performance GPU core that delivers hardware acceleration for graphics displays on these devices. Addressable screen sizes range from the smallest cell phones to 720p displays.

IP has been designed for easy integration into the SoC, providing high performance, high quality graphics, low power consumption, and the smallest silicon footprint for its class. The core is delivered as synthesizable RTL. It is technology independent and can be synthesized using any library. Dynamic power consumption is minimized by extensive use of multi-level hierarchical clock gating. accelerates numerous 3D graphics applications, including graphical user interfaces (GUI) and menu displays, Flash animation, and gaming, and it is a perfect fit for popular consumer devices like cell phones and smartphones, digital picture frames (DPF), digital signage, portable and in-dash GPS navigation systems, mobile internet devices (MID) and netbooks, handheld gaming consoles, set-top boxes and HDTV.

3.1 Design Description

1. **Host Interface**:- Allows the GCCORE to communicate with external memory and the CPU through AXI or AHB bus. In this block data crosses clock domain boundaries.

2. **Memory Controller**:- Internal memory management unit that is the block-to-host memory request interface.

3. **Graphics Pipeline Front End**:- Inserts high level primitives and commands into the graphics pipeline.

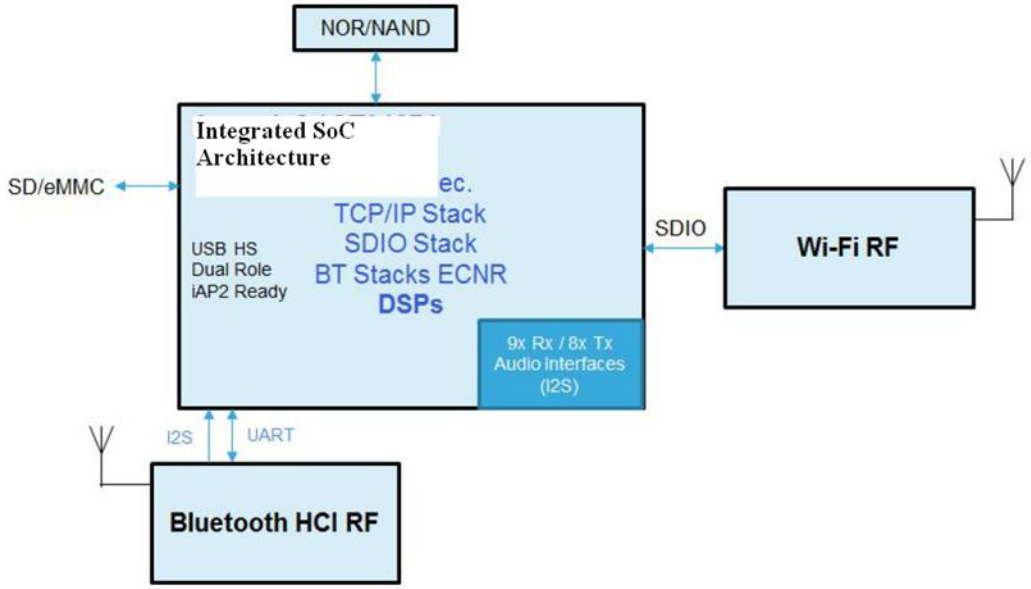

Figure 3.1: Basic connectivity Block Diagram

4. **Ultra-threaded Unified Shader**:- SIMD processor that performs as both vertex shader and fragment shader. When used as a vertex shader it performs geometry transformations and lighting computations. When used as a fragment shader it applies texture data and computes color values for each pixel.

5. **3D Rendering Engine**:- Converts triangles and lines into pixels. Computes slopes of color attributes and texture coordinates. Performs clipping.

6. **Texture Engine**:- Retrieves texture information from memory upon request by the fragment shader. Performs interpolation and filtering, and transfers the computed value to the fragment shader or the vertex shader.

7. **Pixel Engine/Resolve**:- Pixel engine does alpha blending and visible surface determination. Resolve does tiling and de-tiling.

3.2 Buffers Used

1. **Data Buffers**

   (a) **Clear Buffer**: This is a source buffer with clear value

   (b) **Command Buffer**: Contents of the Command Buffer the program uses. This file contains the contents of the COMMAND BUFFER the test program uses. The state loads for the VTX_BUFFER_ADDR, IDX_BUFFER_ADDR, TEX_BUFFER_ADDR,

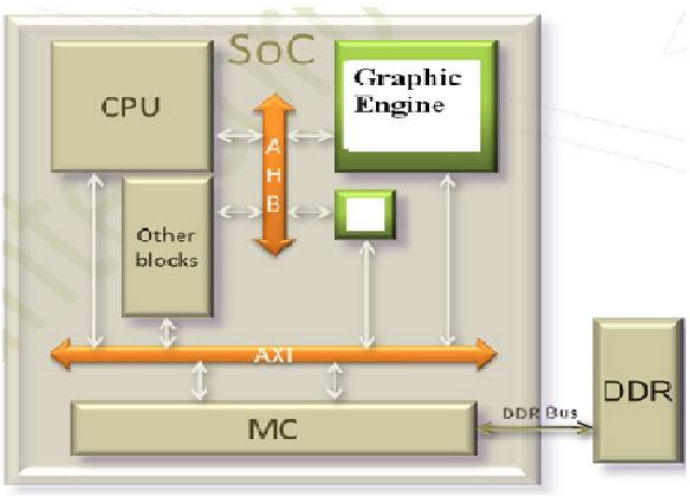

Figure 3.2: Block Diagram Of IP In SoC

DEPTH_BUFFER_ADDR, FRAME_BUFFER_ADDR are provided as symbolic names, these should be replaced by the corresponding memory addresses.

(c) **Composition Buffer**: Memory used to store composition data when test has composition.

(d) **Index Buffer(s)**: If the test program uses the DrawIndex command, this buffer specifies the order in which it will draw the vertices.

(e) **Result Frame Buffer**: This file is the contents of the frame buffer after the test is run. Compare the contents of your frame buffer to see if the test program worked properly.

(f) **Texture Buffer**: If the test program uses texture mapping, this buffer stores the texture data.

(g) **TLB Buffer**: Memory used as translation lookaside buffer.

(h) **Vertex Buffer**: Stores the attributes of the vertices that the test program will draw.

2. **Execution Buffers**

(a) **Depth Buffer**: Used to store the depth (Z) data. This buffer is both written and read during execution.

(b) **Frame Buffer**: Target memory to which the test program writes the output pixel values. This buffer is both written and read during execution.

(c) **Temp Buffer, Medi Buffer**: Memory where the test program writes its temp values.

3.3 Verification Plan

1. To develop verification plan for any IP we have to see if the IP is properly connected at the SoC level or not.

2. To check this we ran two basic tests on IP

   (a) **Initial value test**: In initial value test we just call all the registers and read its initial reset values, we now match its initial values in a prediofined file formate and in case it the value dosent matches in the pmap then an error is generated.

   (b) **Register access**: This test is ran after we have completed the initial value test, in this test we check the readable and writable bits decides upon the given specification of the IP and then check it with the expected values given in the specification. This test ensures the proper Read/Write operations of IP when accessed in SoC.

3. **Interrupt access test**:- Once when we have checked the connectivity of on the SoC level we have to test the IP's driving capability through the respective core . To check this we design the interrupt test in which we check whether the respective core is driving the IP properly or not by rising HIGH checking the status of interrupt and taking LOW and then again checking the interrupt status.
   The interrupt is been served to it by the interrupt bus which have dedicated pins for each associated IP.
   Buffers used:- Command Buffer, VTX Buffer, IDX buffer and Result Buffer

4. **Pixel Test**:- This test is used to check the total memory coverage of the IP. In this test the constraints written will be called properly through the buffers the values will thus be written and read back from the whole set to identify the total coverage .
   Buffers used:- Command Buffer, Vertex Buffer and Result Buffer

5. **Texture Test**:- Now to check the texture in the IP we have designed and used various buffers. These buffers include different values by which it will draw the different geometric styles i.e. rectangle, triangle ,etc. The final results will be compared with the previously stored values and error will be raised in case of any mismatch .
   Buffers used:- command buffers, texture buffer, result buffer and vertex buffer.

Chapter 4

Smart Card Interface [SCI] – IP functionality and verification flow

Smart Card Interface (SCI) is an Advanced Microcontroller Bus Architecture (AMBA) compliant System-on-a-Chip (SoC) peripheral that is developed, tested and licensed by ARM.

It is defined as a slave in AMBA slave module that get connection with the AMBA Advanced Peripheral Bus (APB), and interfaces to an external Smart Card reader. It has the ability to control data transfer to and from the smart card in an automated fashion. In order to reduce the interaction in between the IP and Core or with the bus protocol for high performances the Ip has defined some powerful terms by declaring FIFOs like Transmit(Tx) and Receive(Rx) FIFOs .

4.1 Features of the SCI

Following features are provided by the SCI:

- Compatibility to the AMBA Specifications thus contributes in the easy integration onto System-on-a-Chip (SoC).
- Has definition in both asynchronous T0 and T1 transmission protocols for data transfer.
- Consists of two clock rate conversion factor [F] = 372 or 512, with a good range of bit rate adjustment factors [D] = 1, 2, 4, 8, and 16 are also supported .
- It have eight characters deep buffered TX and RX paths.
- Can generate direct interrupts for TX and RX FIFO.
- Provides option for independent masking of all interrupts, thus making it very much approachable in terms functionality.

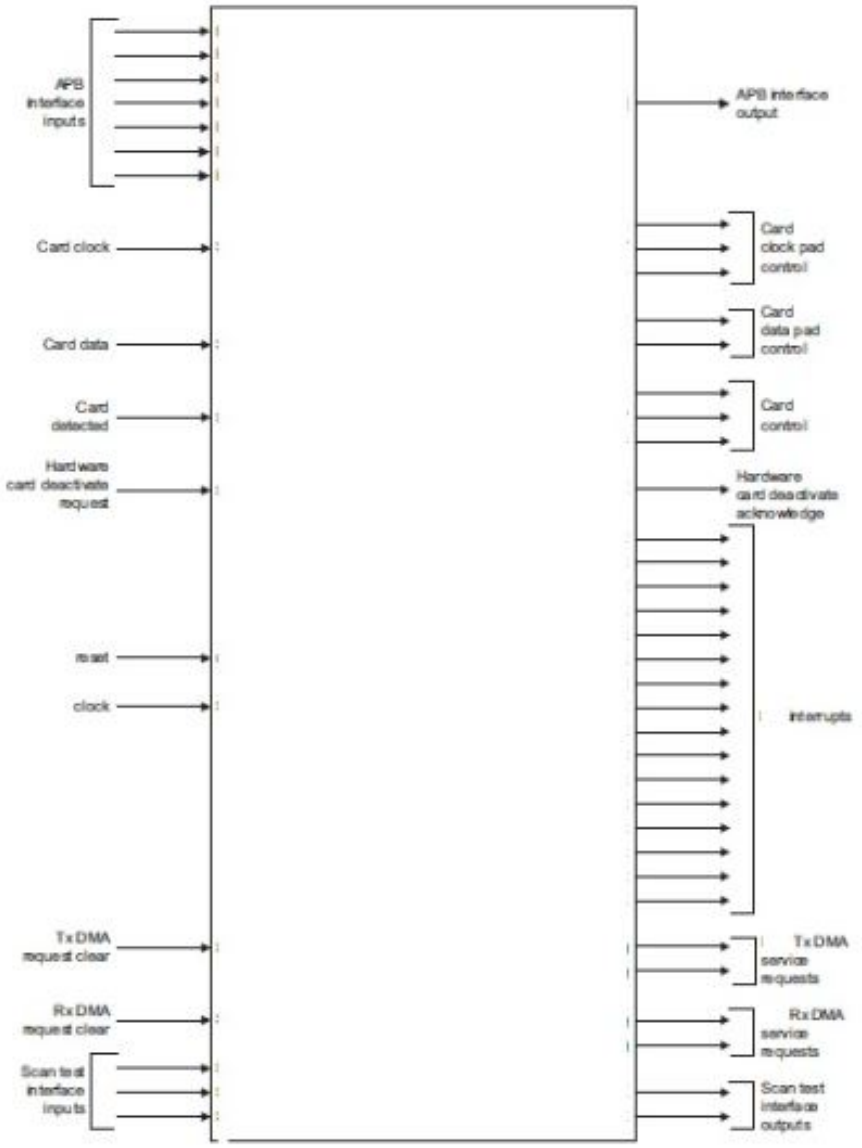

Figure 4.1: Basic Block Diagram of SCI

- It supports Direct Memory Access (DMA).

- A wide range of Interrupts working as a flag for different scenarios, defined within Interrupt status register.

- Hardware initiated card deactivation sequence on detection of card removal.

- Consists of Card deactivation and activation sequence that could be initiated through software or hardware both .

4.2 Functional Overview

The host CPU reads and writes data and control information through the APB interface. The transmit(tx) and receive(rx) paths are buffered with internal FIFO memories allowing up to 8 bytes that are to be stored independently in both transmit as well as in receive modes.

We can also transfer Data via DMA interface in SCI. The SCI includes a programmable baud rate generator and, in conjunction through a secondary value counter it provides a programmable elementary time units (etus). The SCI has been designed to enable close monitoring of all stages of a card session by using mask enabled interrupts. The interrupt architecture allows a choice of:

- A polled approach in which we examine the interrupt status register on assertion of a single common interrupt.(we mostly use this method)
- The interrupt sources can also be applied directly to the interrupt controller for immediate identification by reading a status registers corresponding bits.

The interrupts generated by the FIFOs can be asserted and de-asserted, their level is however defined via the threshold trigger levels. Hardware checks the errors generated by the parity by itself on the receive data. received data is been interpreted by the help of the user defined application software.

Card deactivation can be initiated automatically through hardware through card removal, but a process of deactivation via software by writing to the respective control register is also available. A second deactivation request can be generated through writing bit into the available control register.

4.3 Functional blocks of SCI

4.3.1 AMBA APB interface

The Read/Write decodes are generated by the APB interface which make the accesses for the status control registers and transmit/receive FIFO memories. APB defines performances for the low power extension and thus connects it with the higher bandwidths defined in the AMBA. In short it can be said as a local second bus to AMBA.

4.3.2 Register block

All of the data operations are performed on this block, it have the primary responsibility to store the data for Read/Write operations related to core and SCI.

4.3.3 Transmit and receive logic

This block implements the main functionality of the SCI with the control information being fed to the SCI by the SCI control block. It contains all the counters that are used for timing and various other stages of transactions that are required for a card session. Also, it drives all the card control signals with the exception of data. In this implementation some counters are multi-purpose also, that is, they indicates the transactions required for a particular transaction stage, thus helping in reducing the overall logic required for proper functional implementation.

The transmit and receive block controls the sequencing of the important aspects such as power, clock, reset and data line that are very much vital during during activation and deactivation processes.

4.3.4 SCI control logic

This block consists of the definitions related to the control signals that are generated by the SCI. This mainly consist of the parameters like timeouts and few other logic also.

4.3.5 Transmit FIFO

It is a circular buffer with a width of 8 and a depth of 8. Its main functionality includes the Read/Write operations to be done in the transmit FIFO which results an access to buffer locations that will be pointed by a Read and Write pointers respectively. The transmit FIFO transfers the data to the data register which is used in the buffer data transfer and this data is then send to the card.

4.3.6 Receive FIFO

It is a circular buffer with a width of 9 and a depth of 8. Its main functionality includes the Read/Write operations to be done in the receive FIFO which results an access to buffer locations that will be pointed by a Read and Write pointers respectively. Here one extra bit is to denote the parity of the data transferred rest of the data is stored in the least significant eight bits. The receive FIFO receives the data from the data register which is used in the buffer data receiver and this data is then send by the card to the pads.

4.3.7 Interrupt generation logic

SCI generates the individual maskable interrupts, however it also generates a combined interrupt as an output which is an OR function with all interrupts as the inputs and a

single output line going high whenever any interrupt line is high. Thus we can use the single combined interrupt with a system interrupt controller.

4.3.8 DMA interface

An interface is provided by the SCI in order to connect to the DMA which will be required to access the memory locations whenever some data is to be transferred.

4.3.9 Synchronizing registers and logic

It have both asynchronous and synchronous clock operations for many of the clocks, PCLK and SCICLK. However with customer requirements of usage we have made PCLK and SCICLK to work on the same frequency.

4.3.10 Test registers and logic

These are registers and logic for functional block verification, and integration testing. When in the normal condition these test registers will not be in use. The integration testing verifies that the SCI has been wired into a system correctly or not. It enables each input and output so that both may be written and read.

4.4 Verification Plan

It was a challenging task to define the verification plan for SCI since SCI takes the input as well as it gives back the output data to the GPIO[General Purpose Input Output] directly. Now since we are at very initial stage of designing , thus we cant decide the proper flow of data from the outer world at this stage.

To rectify this problem we designed a Driver [test-bench] through which we will transfer the data and can also drive the data on SCI assuming that the data is been written or read by SCI on the pads. For this Driver we have instantiated the design methodology of the SCI as it is by defining all those constraints that are required by us for testing. SCI is been connected to a number of peripherals like core, pads, DMA, etc. to check all this functional connectivity we have to develop following testcases.

- To develop verification plan for any IP we have to see if the IP is properly connected at the SoC level or not. To check this we ran two basic tests on IP.
  - **Initial value test:-** In initial value test we just call all the registers and read its initial reset values, we now match its initial values in a predefined file format and in case it the value doesn't matches in the pmap(a file format that contains all registers fields) then an error is generated.

  - **Register access:-** This test is ran after we have completed the initial value test, in this test we check the readable and writable bits decides upon the given specification of the IP and then check it with the expected values given in the specification. This test ensures the proper Read/Write operations of IP when accessed in SoC.

- **Interrupt access test:-** Once when we have checked the connectivity of on the SoC level we have to test the IP's driving capability through the respective core . To check this we design the interrupt test in which we check whether the respective core is driving the IP properly or not by rising HIGH checking the status of interrupt and taking LOW and then again checking the interrupt status. The interrupt is been served to it by the interrupt bus which have dedicated pins for each associated IP.

  IP will ask enable the interrupt line here, now the interrupt generated here is handled by the core. Once it has been handled we are assured of proper functional connectivity of IP with the core as well as interrupt service bus.

  Registers used:- Control register, Interrupt Status register, Interrupt enable line

- **Activation Deactivation Sequence:-** Usually as per the design constraints this IP must consists a proper activation and deactivation sequence since we have a option of detaching the whole IP as of the hardware from the SoC. However according to revised specifications by the integrator hardware activation process is not viable in this case.

  Thus we have to check the proper functionality of the whole deactivation and activation process through enabling the software activation bit first, this bit will make the power source enable. Once this power source is enabled one card signal which identifies the hardware presence of the SCI will go high. After the DETECT signal goes high a interrupt flag indicating the card in bootable stage will generate the high flag, this is immediately followed by the RESET and CLOCK signals going high.

  Similarly in a deactivation sequence first the CLOCK and RESET signals are disabled. Disabling of these signals indicates the starting of the deactivation process. After this DETECT and CARDOUT signals are generated.

  Registers used:- Control register1, Interrupt Status register, DETECT signal register, Clock frequency and Reset.

- **Loopback test:-** It's an internal loopback test where we check the compatibility of IP with the SoC environment . In this test we programme the given "Test Registers", these registers are very much responsible for the Primary Inputs(PI) and Primary Outputs(PO) of the SCI.Now using the Test output register we write some specific bits in some specific sequence and now on the other hand we poll for the specific bits of the input Test registers which are actually responsible for those output register bits.

In short in this way we actually test the exactness of IP in respect of the integration with the SoC by looping the output back in the input.

Registers used:- Test control registers, Test output registers, Test input registers

- **Data transfer test from IP to Memory:-** In SCI the data can be transferred using the DMA transfer channel . In SCI we have to first set the proper BAUD rate, Activation time, Deactivation time and watermark of FIFO. Once these constraints are set we are ready to go forward to set same constraints in the testbench architecture of the SCI . After compilation of this we set the DMA access as per defined by the integrator to us, in this we set the master and slave mode for the DMA. We know that DMA have two interfaces in here and it may be possible that a single IP may not have access to both of them as it happens in the case of the SCI, now for such a scenario we have to set the mode of interface to be used by the DMA for proper transfer.

  Once DMA channel is set we start to save the copy of data in the memory location and the Rx mode of SCI is enabled which transfers the data to the TB side of the design to be be accepted by TB through its predefined Rx FIFO.Here the notable thing is that we have verified a very important aspect of the IP which defines the transfer of data to the pads using DMA interface.

  Registers used:- Data Register, Tx and Rx FIFO, TIDE, Baud rate, Activation time, deactivation time

Chapter 5

Simulation Results and Flow Observations

The result of one of the testcase is shown here. Result here shows the comparison in terms of timing between its initial state and latest state. In the given simulated waveform the signals define the whole transfer of the data to their respective addresses . Also different transfers are compared here.

Figure 5.1 shows the transfer signals activation from the core done for the data/instructions to be written or read with respect to the IP. Here we can clearly see from the waveform how the basic initial signals reacts when the simulation is just started.

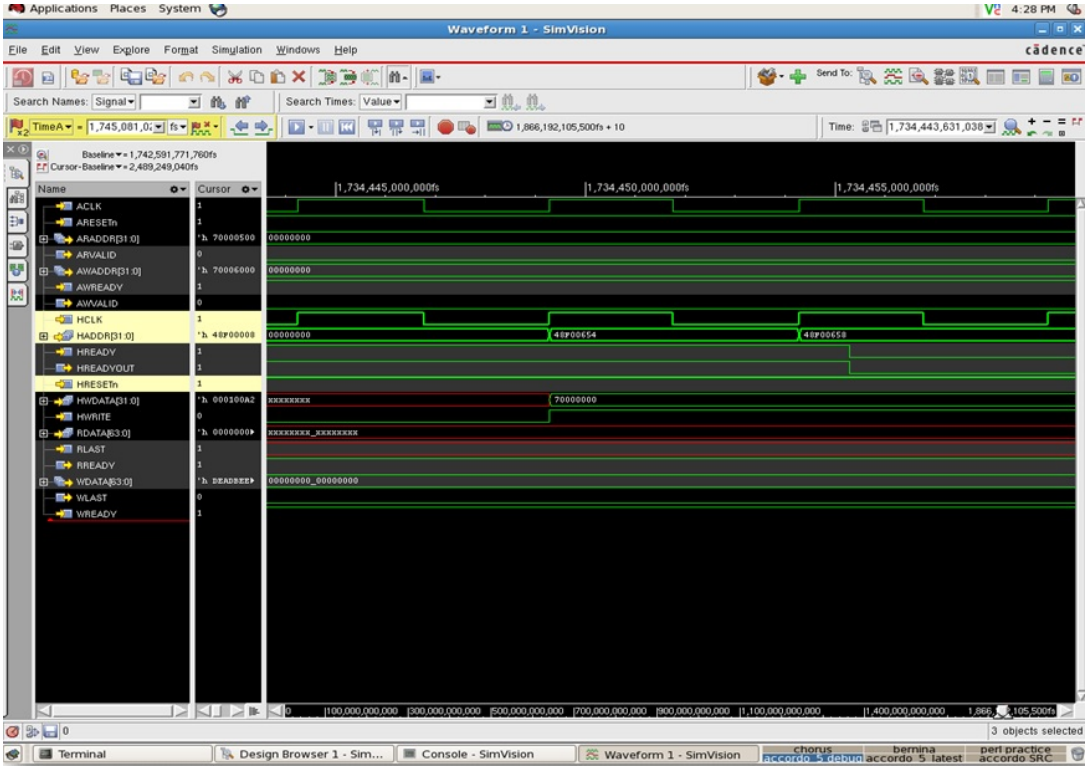

Figure 5.1: Signals From Core On Waveform

Here the **HWRITE signal is high which indicates that the write transfer is been performed on the slave. The address as per defined in the HADDR i.e. 0x48F00654 and 0x48F00658 indicates the starting address of the buffers from which the instructions is to be fetched.**

Figure 5.2 shows the transfer signals activation from the IP side done for the data/instructions to be written or read with respect to the IP.

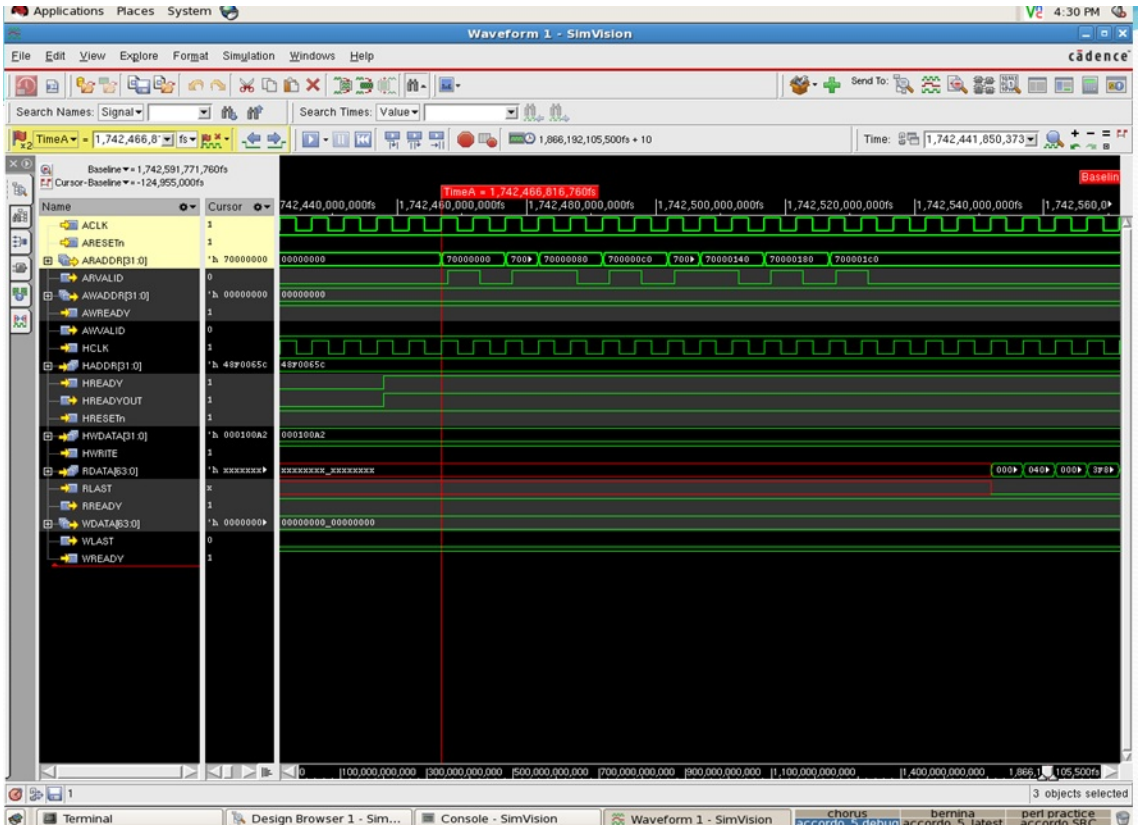

Figure 5.2: Signals From IP On Waveform

Here the HADDR shows the starting address of the slave IP's register trough which the transfers have started, rest all the sequences are perfectly same as per the core.

Figure 5.3 shows the transfer addresses that are to be accessed by the core for writing data/addresses in the IP register defined as per the sequence given in buffer from the IP side done for the data/instructions to be written or read with respect to the IP.

Figure 5.4 shows the read process that is initiated on the IP, here first the the core waits for two basic signals to start the process: first is the HREADY which should be HIGH, second is the status of the HWRITE signal which should be low at the time of read process. Once the read process starts the address from

which core is reading will be shown in the output.

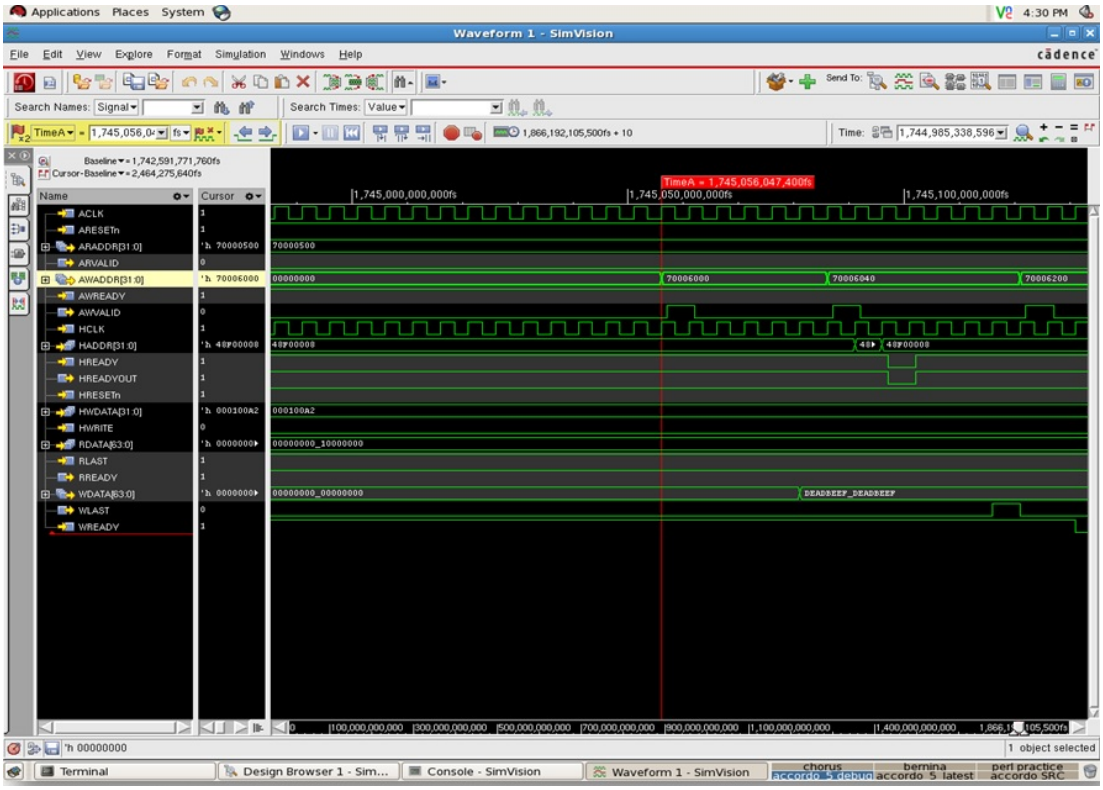

Figure 5.3: HADDR sequences of IP accessing through Core

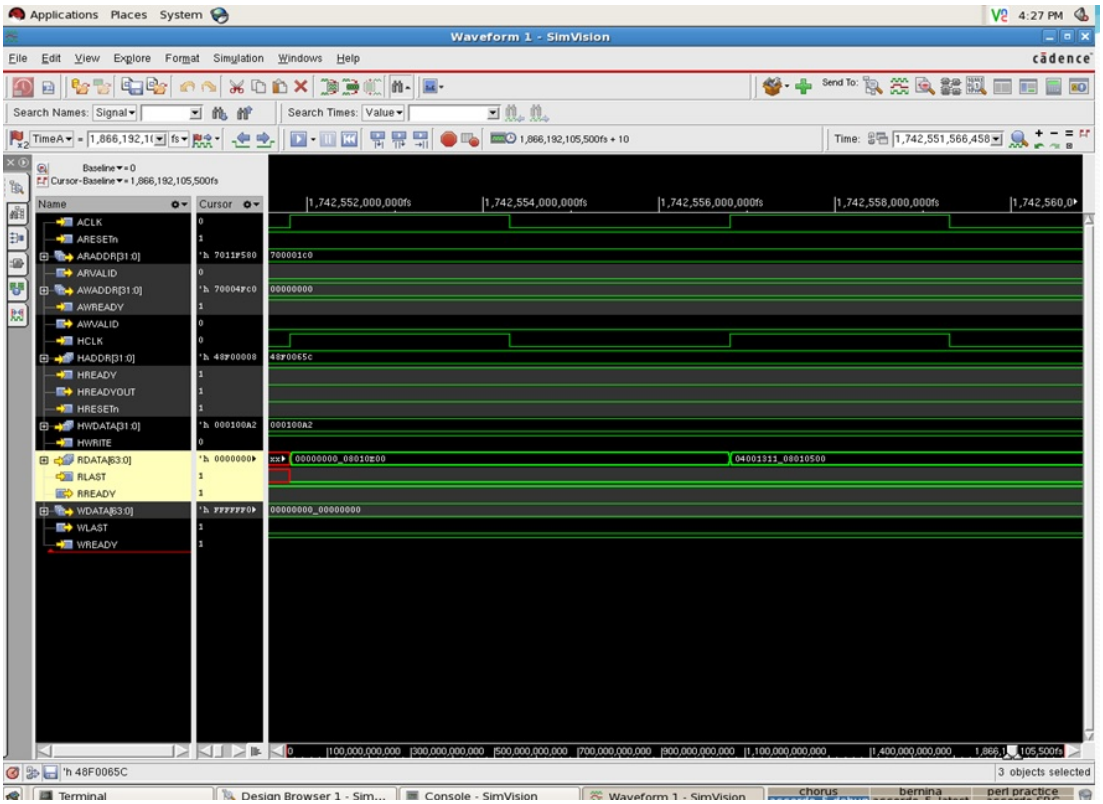

Figure 5.4: READ sequences of IP

Figure 5.5 shows the write process once this process is completed then the output is been iteratively compared with the predefined data saved by us at some memory location. Final status of the simulation is shown once all the comparisons are been done.

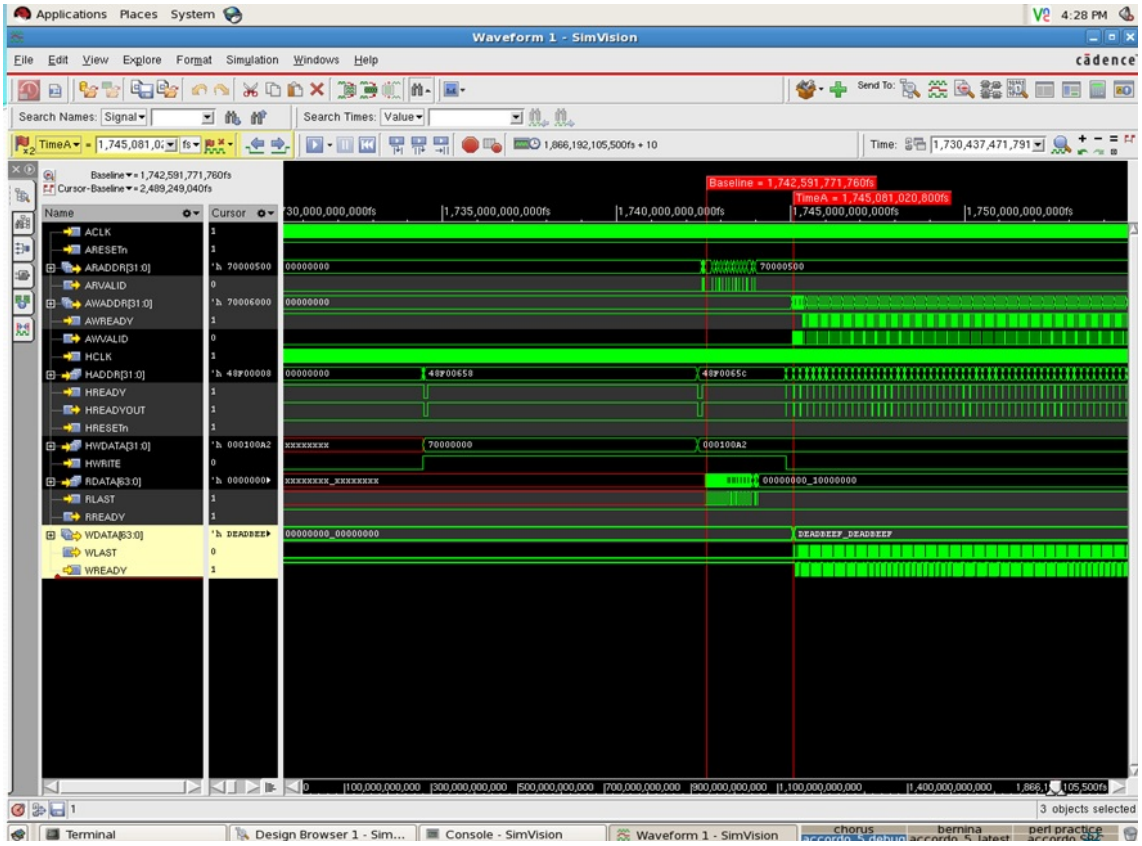

Figure 5.5: Write sequences of IP and comparison

```
1731225 ns CORTEX-M3: #### Wait for Ideal State ####
1766916 ns CORTEX-M3: R32: A=48F00004 R=FFFFFFFF
1770406 ns CORTEX-M3: #### Comparing output data with Result Buffer ####
1778889 ns CORTEX-M3: RE32: A=70006040 R=DEADBEEF E=DEADBEEF
1787410 ns CORTEX-M3: RE32: A=70006000 R=DEADBEEF E=DEADBEEF
1795941 ns CORTEX-M3: RE32: A=70006080 R=DEADBEEF E=DEADBEEF
1804467 ns CORTEX-M3: RE32: A=70001040 R=FFFFFF00 E=FFFFFF00
1810384 ns CORTEX-M3: R32: A=70006240 R=DEADBEEF
1812378 ns CORTEX-M3: #### Pass/Fail Status ####
1817132 ns CORTEX-M3: *** GRAPHIC_CTRL Test Final Status : PASSED (1)
1825519 ns CORTEX-M3: MailBox Read from C at address = 0x00000000  Read Data = 0x00000001
1827504 ns CORTEX-M3: No error/s from .sv side
1827960 ns CORTEX-M3:
1831315 ns CORTEX-M3: END_SIM:    PPPPP      AA        SSSS    SSSS
1834670 ns CORTEX-M3: END_SIM:    PP  PP    AAAA      SS  SS  SS  SS
1838024 ns CORTEX-M3: END_SIM:    PPPPP    AA  AA      SS      SS
1841379 ns CORTEX-M3: END_SIM:    PP      AAAAAAAA       SS      SS
1844734 ns CORTEX-M3: END_SIM:    PP     AA      AA   SS  SS  SS  SS
1848089 ns CORTEX-M3: END_SIM:    PP    AA        AA   SSSS    SSSS
1851467 ns CORTEX-M3: END_SIM: Simulation completed Successfully;
1851612 ns CORTEX-M3:
            TUBE: ** PROGRAM EXIT **

Simulation stopped via $stop(1) at time 1851611602423 FS + 10
ncsim> if ($env(GATE_SIM) == "1") {
> }
ncsim>
ncsim>
ncsim>
ncsim>
ncsim>
```

Figure 5.6: Write sequences of IP and comparison

Chapter 6

Conclusion

In Industry every IP requires a different verification environment in order to define its total functionality through various test-cases. In case of Graphic Engine the major challenge faced was the excess size of the buffers due to which the simulations were not even able to start. Similarly in case of SCI an whole new environment that consists of the definition of drivers was successfully created and was included thus ensuring its functional connectivity with the pads.

The GLS of both the IPs was a very challenging task due to its time consuming simulations and a very large memory consuming waveforms. However this was also completed completed successfully and many of the design bugs related to the multi-cycle path and false paths that are declared wrongfully while analysis in STA[Static Timing Analysis] that were unable to be identified through STA were also resolved.

References

[1] **AMBA Bus protocol Specifications** `http://www.arm.com/products/system-ip/amba-specifications.php`**, as on :- 21:34 16 December 2015**

[2] ***"Specification and working flow of Graphic Engine "*** **:STMicroelectronics Confidential Library as on:- 6 May 2016**

[3] ***"Functional specification and integration guide of Digital Smart Card Interface[SCI] "*** **:STMicroelectronics Confidential Library as on:- 6 May 2016**

[4] **E Balagurusamy *"Programming in ANSI C"* - Fourth Edition , 2008 , Tata McGraw-Hill Publication House**

[5] **Sameer Palnitkar *"Verilog HDL: A guide to digital design and synthesis"*, Second Edition , Pearson Publications**