MAC INTERFACES WITH ETHERNET PHY

M.Tech. Thesis

By

SAI KIRAN KANCHERLA

DISCIPLINE OF ELECTRICAL ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY INDORE

JUNE 2020

MAC INTERFACES WITH ETHERNET PHY

A THESIS

Submitted in partial fulfilment of the
requirements for the award of the degree
of
Master of Technology
in
Electrical Engineering
with specialization in
VLSI Design and Nanoelectronics
By

SAIKIRAN KANCHERLA

DISCIPLINE OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE

JUNE 2020

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **MAC INTERFACES WITH ETHERNET PHY** in the partial fulfilment of the requirements for the award of the degree of **MASTER OF TECHNOLOGY** and submitted in the **DISCIPLINE OF ELECTRICAL ENGINEERING Indian Institute of Technology Indore**, is an authentic record of my own work carried out during the time period from JUNE 2019 to JUNE 2020 under the supervision of Dr. SAPTARSHI GHOSH Assistant Professor at IIT INDORE .

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

18/06/2020

Signature of the student with date
K. SAI KIRAN (1802102016)

This is to certify that the above statement made by the candidate is correct to the best of my/our knowledge.

Saptarshi Ghosh 22/06/2020

Signature of the Supervisor of

M.Tech. thesis (with date)

Dr. SAPTARSHI GHOSH

Mr. K SAI KIRAN has successfully given his M.Tech. Oral Examination held on 22th June 2020

Saptarshi Ghosh

Signature(s) of Supervisor(s) of M.Tech. thesis
Date: 22/06/2020

Convener, DPGC
Date: 22/6/2020

V. Sridhar

Signature of PSPC Member #1
Date: 22/6/2020

Somnath Dey 22/06/2020

Signature of PSPC Member #1
Date:

ACKNOWLEDGEMENTS

First and foremost , I would like to express my gratitude to Dr. SAPTARSHI GHOSH Assistant Professor at IIT INDORE whose efforts and guidance helped me in completing my M. Tech. dissertation work successfully. I am extremely thankful to my supervisor PANCHAKSHARI SAHUKAR Senior Lead Engineer at NXP semiconductor for his motivation to help me to get deep knowledge of the research area and supporting me throughout the life cycle of my M. Tech. dissertation work.

I am also thankful to Dr. Vipul Singh, Head of the Electronics Engineering Department, for his fruitful guidance through the early years of chaos and confusions. I wish to thank the faculty members and supporting staff of Electronics Engineering Department for their full support and heartiest co-operation.

This thesis would not have been possible without the support of my friends. My deepest regards to my Parents for their blessings, affection and continuous support.

SAIKIRAN
1802102016

Abstract

With the unexpected rate of growth in technology, demand for a high speed and high performance devices has increased. IP digital Designing is now at a avery challenging state where requirement of complex design is growing exponentially. Designer can now design chips with lakhs of transistors on a single chip. Verilog hardware discription languages provide an ease to develop such complex designs.

AMBA bus architecture is one such complex design which can be developed using verilog. The APB which is a part of AMBA bus design is implemented in this project using verilog and verified using testbench.

Often, when the chip is manufactured there will be a lot of silicon bugs. These bugs can arise for a number of reasons such as manufacturing defects or improper design etc. These bugs can be anywhere on the chip. To pin point the bugs present in a specific area, additional logic is added to the design. This additional logic will not impact the main functions of the chip. One such logic is the ability for the chip to test itelf. This is called Built in self test (BIST). This comes under “design for Testibility”. A module for this issue is developed in this project. On an Ethernet PHY chip, Data is transferred over data path between the PHY and MAC layers of OSI Model. In the validation stage of the chip these data paths are checked for silicon bugs using BIST engine module discussed in depth.

An insight on different media independent interfaces is also given. These interfaces are divided based on the clockspeed, port requirement etc and best features of each interface is highlighted.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

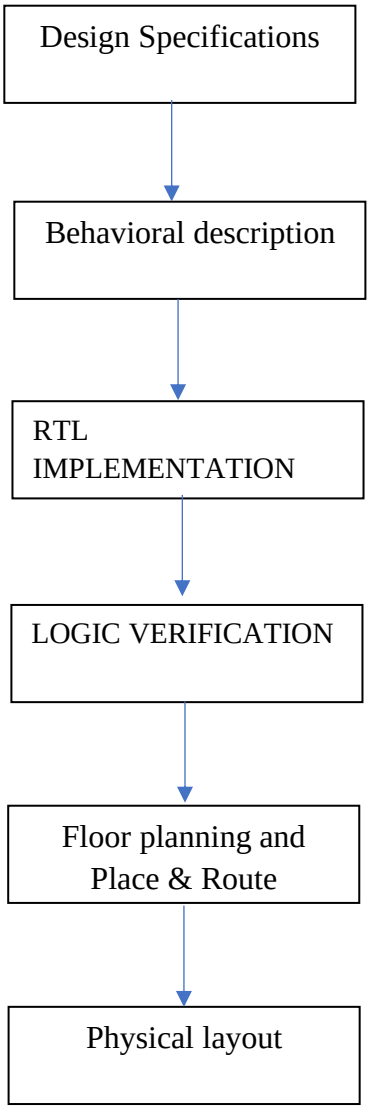
AMBA	Advanced Microcontroller Bus Architecture
APB	Advanced Peripheral Bus
AHB	Advanced High Performance Bus
ASB	Advanced System Bus
BIST	Built In Self-Test
CRC	Cyclic Redundancy Check
FCS	Frame Check Sequence
DUT	Design under test
FSM	Finite State Machine
GMII	Gigabit Media Independent Interface
MAC	Media Access Control
MII	Media Independent Interface
PERL	PERL scripting language
PHY	Physical Transceiver
RGMII	Reduced Gigabit Media Independent Interface
SGMII	Serial Gigabit Media Independent Interface
SoC	System-on-Chip
SFD	Start of frame Delimiter

Chapter 1

Introduction

Digital designing has evolved from designing a few hundred transistors to million transistors on a chip. A digital designer creates the hardware specifications, makes a description and then implements the logic. Hardware description Languages HDLs are used to make the tool understand the design and bring out the connections for the logic. So, dealing with millions of transistors is taken care of by these computer tools making it very easy for the designers to design any complex logic without limiting to the hardness in implementation.

Design flow :



The above design flow shows the steps through which a chip goes through before tape out. This is a very simplified version. After making the specification for hardware, RTL is developed. The RTL is then subjected to rigorous testing for functionality and bugs. Then an optimal location is chosen on the chip to give the best results.

Goal :

The main focus of the project is to develop RTL for AMBA APB protocol and use this module for the implementation of BIST engine. In-depth functional verification will also be done.

Overview:

Chapter 2 and 3 gives an insight about the AMBA APB slave and its applications. Protocol specifications are explained and these milestones are reached which is explained using the simulation waveform. Chapter 3 explains how APB slave protocol can be used with other designs.

Chapter 4 gives an over view on how data transfer take places from one network point to other. It describes how different layers of OSI models are used for safely transferring the data. Standard formats, Error correcting techniques are also explained.

Chapter 5 deals with the implementation of BIST engine, a DFT application, which helps in finding silicon bugs in post validation phase of the chip.

Verilog HDL is used for the implementation of the designs and each module functionality is verified using test benches.

Chapter 2

Implementation Of AMBA APB Protocol

2.1 Amba and its bus architecture:

Advanced Microcontroller Bus Architecture (AMBA) is a communication system used on 'System on chip'(SOC). It is an open standard interconnection for the purpose of connecting different blocks in a System-on-Chip .It is used for designing high performance embedded systems. It is used as "on chip" bus.

AMBA bus architecture consists of three parts,

- Advanced High Performance Bus(AHB).
- Advanced System Bus (ASB).
- Advanced Peripheral Bus (APB).

The AMBA specification are:[refer 9]

• This is development of embedded microcontroller products with one or more CPUs or signal processors.

• This is highly reusable peripheral appropriate for full-custom, standard cell and gate array technologies.

• It provides a road-map for advanced cached CPU cores and the development of peripheral libraries to minimize the silicon infrastructure required to support efficient on-chip.

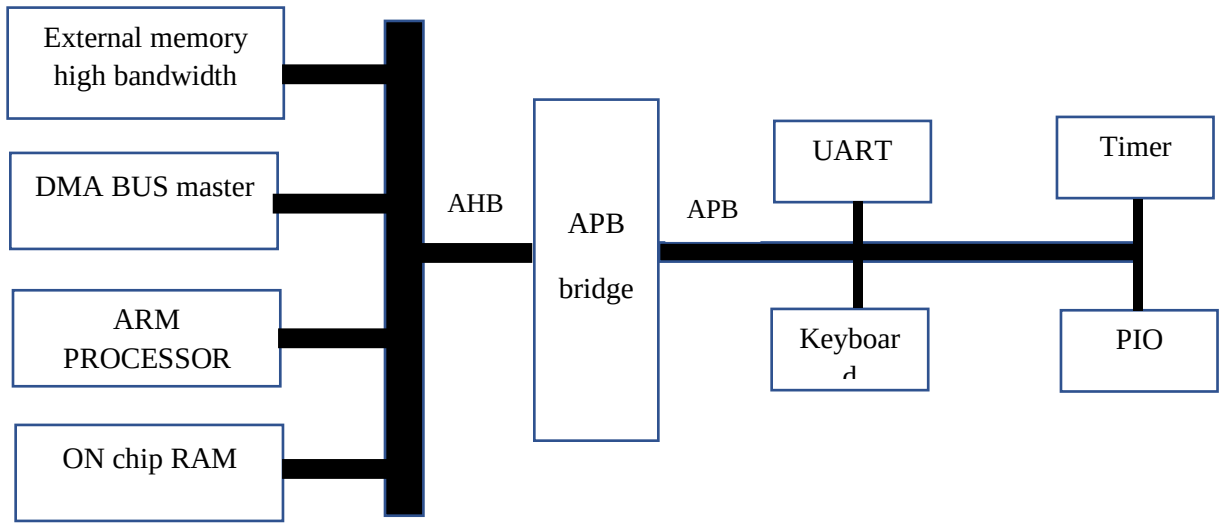


Fig 2.1 AMBA bus architecture

- Advanced High-performance Bus (AHB): The AMBA AHB is used to connect the high bandwidth devices. It is used for applications where operating frequency is high. The AHB acts as the high-

performance system backbone bus. Devices such as DMA BUS MASTER, ON CHIP RAM, external memory etc are connected on AHB. It enables us to connect the different devices in an efficient way. AHB is also simplifies synthesis and automated test methodologies to achieve high efficient design flow. It also implements

- Back to back read-write transactions.
- Transferring the bus control. .
- Single clock edge operation.
- Large data transfers over bus.

- Advanced System Bus (ASB): The AMBA ASB is for system modules. It is used as an alternative bus suitable for high speed and high bandwidth applications.it is used where AHB applications are not possible. ASB also supports the efficient connection different processors.
- Advanced Peripheral Bus (APB): The AMBA APB is a for low-power peripherals.

2.2 APB Protocol :

APB is the part of AMBA APB PROTOCOL. It is used where devices with low speed and low bandwidth are need to connect to the high speed devices such as ARM processor. AMBA APB is power consumption is very low due to its simple working steps. The interface complexity to support peripheral functions is greatly reduced. This member of AMBA protocol family is designed to support low-speed peripherals such as UARTs, keypads.

APB can be used in conjunction with either version of the system bus. It does not support pipelined transfers. It can only support single transfer at time. All the transitions happen the positive edge or at the negative edge of the clock. This makes it very easy to integrate APB device with any other designs. It is mainly used to access the control registers of the peripheral device. Reading the data from the peripheral device and write the data to the peripheral device is an important feature of the design. This working procedure is done through Finite State Machine.

Different versions of APB : [6]

The APB Specification Rev E, 1998, is now not used and advanced versions are as follows:

• AMBA 2 APB Specification

This specification defines the interface signals, the basic read and write transfers, and the two APB components the APB bridge and the APB slave.

• AMBA 3 APB Protocol Specification v1.0

The following interface signals are added:

- PREADY A ready signal to indicate completion of an APB transfer.

- PSLVERR An error signal to indicate the failure of a transfer.

This version of the specification is referred to as APB3.

• AMBA APB Protocol Specification v2.0:

The following interface signals are added:

- PPROT A protection signal to support both non-secure and secure transactions on APB.
- PSTRB A write strobe signal to enable sparse data transfer on the write data bus.

This version of the specification is referred to as APB4.

In this thesis, AMBA APB 2 is implemented.

2.3 APB Master & APB Slave connections:

Basic Block Diagram Of APB:

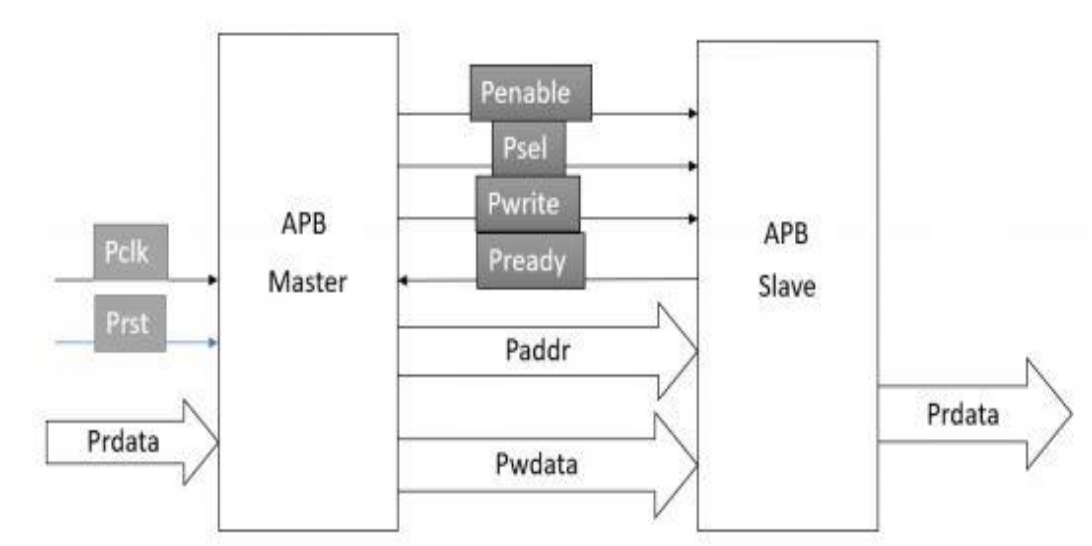


Fig 2.2 block diagram of APB

In the above figure, APB master and APB slave are shown. APB master acts as the bridge between the slave and AHB side. Different signals are shown in the above diagram which are explained in detailed. Pclk, PRESET, PSEL, PADDR, PWRITE, PWDATA are inputs to the APB slave. PRDATA and PREADY are the outputs.

2.4 Input and Output Signals :

These input and Output signals are standard signals defined in AMBA APB.

- **PCLOCK :** This represents the clock governing the transfers. The rising edge of PCLK times all transfers on the APB.

- **PRESET**: System bus equivalent Reset. The APB reset signal is active LOW.
- **PADDR**: This is the APB address bus. It can be up to 32 bits wide and is driven by the peripheral bus bridge unit.
- **PSEL**: The APB bridge unit generates this signal to each peripheral bus slave. It indicates that the slave device is selected and that a data transfer is required. There is a PSEL signal for each slave.
- **PENABLE** : This signal indicates the second and subsequent cycles of an APB transfer.
- **PWRITE**: It's a directional signal. It indicates an APB write access when HIGH and an APB read access when LOW.
- **PRDATA**: The selected slave drives this bus during read cycles when PWRITE is LOW. This bus can be up to 32-bits wide.
- **PREADY**: The slave uses this signal to extend an APB transfer.

2.5 APB finite state machine :

The following state diagram explains the operations of an APB.

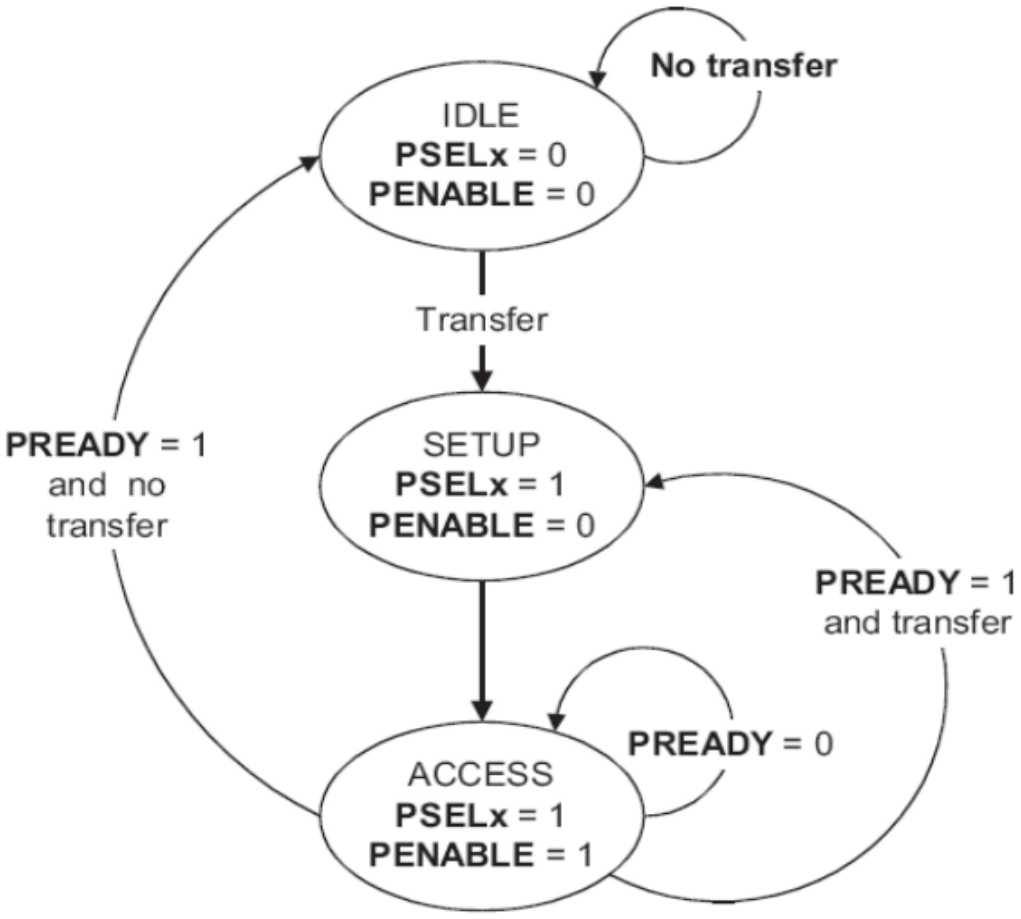


Fig 2.3 FSM diagram of APB [1]

1. IDLE: This is the default state of the APB. In this state PSEL and PENABLE are low. When transfer is required, PSEL is made high and the state jumps to SETUP phase. If there is no transfer, it will remain in IDLE.
2. SETUP: When a transfer is required the bus moves into the SETUP state, where the appropriate select signal, PSELx, is asserted. SETUP STATE is only for one cycle. At the next positive clock cycle, the FSM jumps to ACCESS .

3. ACCESS: The PENABLE signal is made high in the ACCESS state. The address, write, select, and write data signals must remain stable during the transition from the SETUP to ACCESS state. Exit from the ACCESS state is controlled by the PREADY signal from the slave:

(a) If PREADY is held LOW by the slave then the FSM state remains in the ACCESS state.

(b) If PREADY is driven HIGH by the slave then the ACCESS state is exited and the bus returns to the IDLE state provided there are no more transfer. If a transfer is made, then bus moves directly to the SETUP state.

2.6 WRITE and READ operations:

Write Cycle:

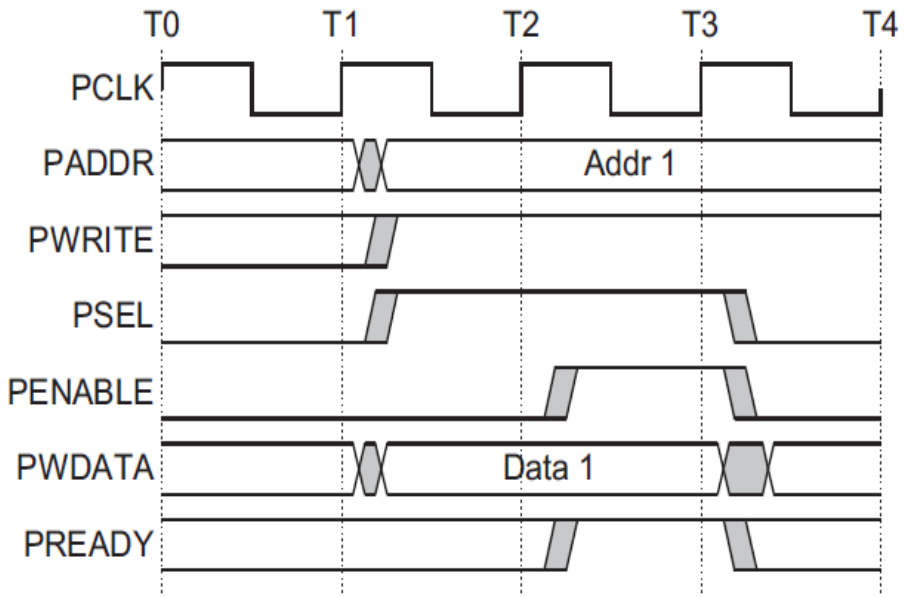


Fig 2.4 Write Cycle of APB SLAVE[1]

The write transfer starts with the address, write data, write signal and select signal all changing after the rising edge of the clock T1. The first clock cycle of the transfer is called the Setup phase.

After the following clock edge the enable signal is asserted at T2, PENABLE, and this indicates that the Access phase is taking place. The address, data and control signals all remain valid throughout the Access

phase. The transfer completes at the end of this cycle. The enable signal, PENABLE, is deserted at the end of the transfer at T3 . The select signal, PSEL also goes LOW unless the transfer is to be followed immediately by another transfer to the same peripheral.

PREADY signal from the slave can extend the transfer. During an Access phase, when PENABLE is HIGH, the transfer can be extended by driving PREADY LOW. The following remain unchanged for the additional cycles: PADDR , PWRITE , PSEL , PENABLE, PWDATA.

Read Cycle:

During read operation the PENABLE, PSEL, PADDR PWRITE, signals are asserted at the clock edge SETUP cycle. At the clock edge ACCESS cycle, the PENABLE, PREADY are asserted and PRDATA is also read during this phase.

The slave must provide the data before the end of the read transfer. PREADY signal can extend the transfer. The transfer is extended if PREADY is driven LOW during an Access phase. The protocol ensures that the following remain unchanged for the additional cycles: PADDR , PWRITE , PSEL , PENABLE.

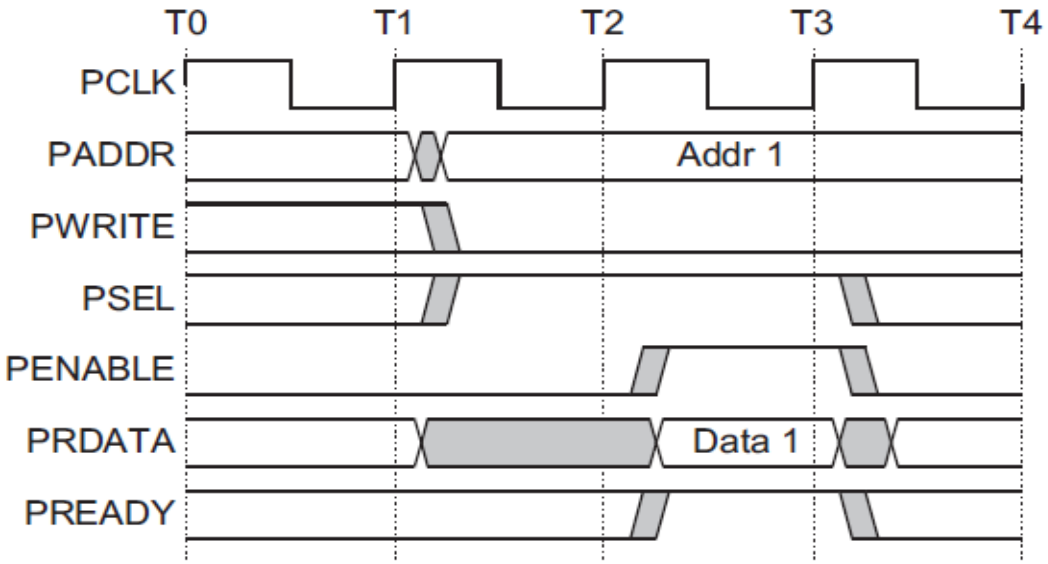


Fig 2.5 Read Cycle of APB SLAVE[1]

Test Plan :

After being well versed with the protocol , RTL code was written. Simultaneously a test plan was prepared. As we know that this consists of : 8 bit address bus, 32 bit data bus, memory of 255. Accordingly following tests were coded:-

1. Basic set ,reset ,clock check

2. The test to Write and Read to every valid location according to protocol. A special test of 00,AA,55 and FF. These values are written and read in every register.

This test checks that whether every bit of all registers are being occupied legally.

3. Every legal address is written into. These values are stored for reference. Then whole memory is read . It should match with all the values in reference.

4. Test to alternatively write and reading on register location. In this , first write on a particular address is followed by immediate read on that address is carried out. All values should match with reference.

5. Test for writing at various locations and Randomly Reading from those stored locations.

6. Default read check. This test should read the default value of the register if nothing is written to it.

7. Error for Invalid addresses. This is negative test. Invalid values should display appropriate response on application

2.7 Regression PERL Script :

One important thing to keep in mind during verification planning is to know its limitations. For industrial size programs, it is not realistic to expect the written code to naturally cover high level invariants or temporal properties. Its not that the code can't be written but the complexity and time to resources ratio required is not sustainable. Regression is a way to circumvent this problem.

Regression testing means you can run a test suite on your design at any time in an easy manner which require little additional setup and little analysis. The idea is that you can retest everything whenever something changes or periodically as a matter of course. Regression verification can be seen as an additional layer to improve the verification process. As regression test suites tend to grow with each found defect, test automation is frequently involved. Each test should report an unambiguous indication of pass or fail. This is also referred to as the Self-Checking Testbench. Viewing

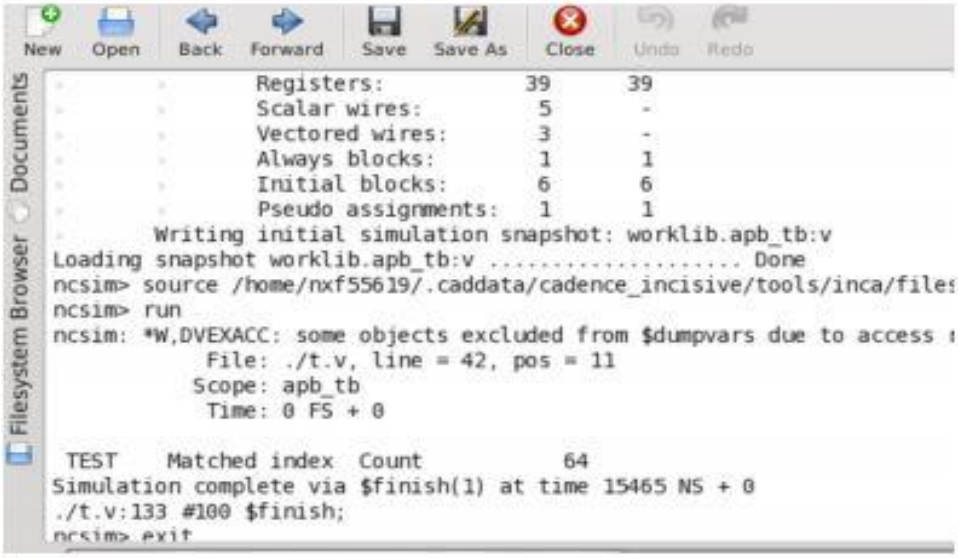


Fig 2.6 Log file

waveforms is great for diagnosing problems, but not the way to repetitively assess whether your test and your design is working.

So for APB various testcases were written to check every functionality of the DUT. With increased number of testcases and bug fixes, running every testcase manually every time is cumbersome. The next step was to automate the testbench for regression testing. This implies that with an automated testbench, the operator doesn't need to run the testbench manually for each Testcase.

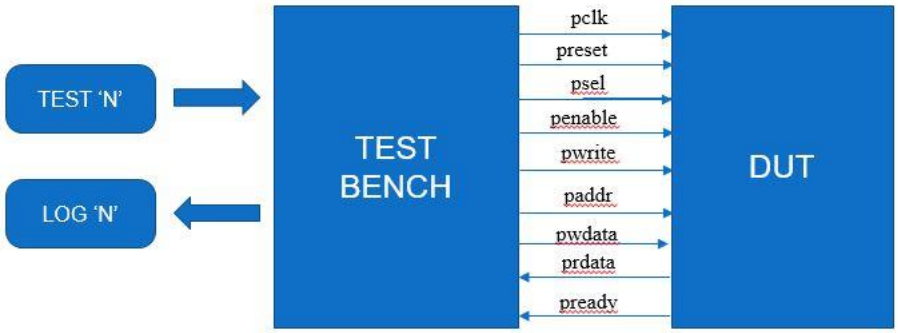


Fig 2.7 Block diagram showing perl regression Action

Fig 2.8 Result file

PERL takes the different test case scenarios and keeps each test case in the test bench one after the other and runs the DUT. The resultant log file for each test case is stored separately in a log file. Now for any number of Testcases, by running the script alone the operator can see the results at one location. Result file can be viewed to check which test case is PASSED and which testcase has FAILED.

2.8 Simulation Waveforms :

Test 1:

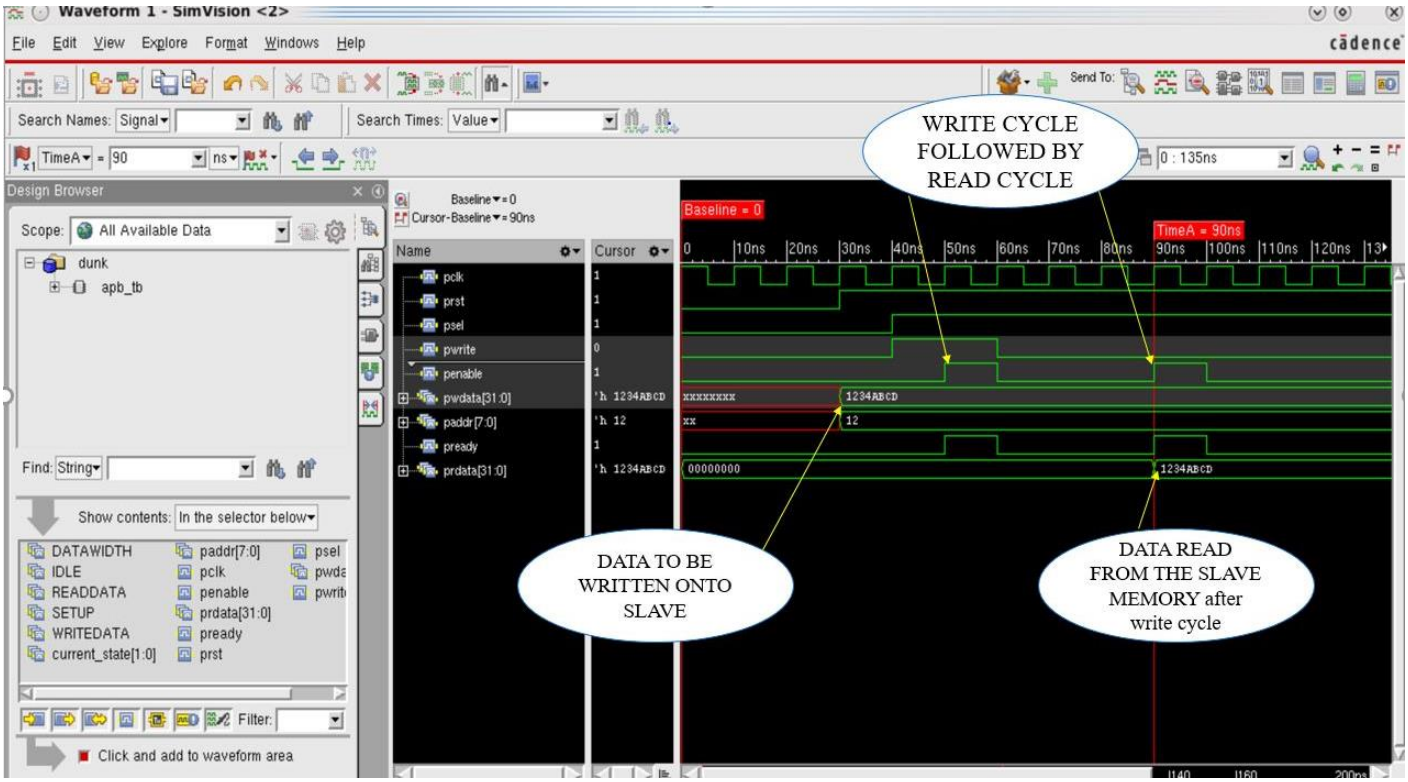


Fig 2.9 Simulation waveform for write and read cycles

The above waveform show a write cycle followed by a Read cycle.

- After the reset signal, Psel is made high.
- Then Pwrite is also made high to start a write cycle first.
- Then Penable signal is asserted and the data transfer takes place.
- Pready can be seen HIGH indicating the completion of the transfer.
- Also notice that the Psel line is not made low. This indicated that another transfer is going to take place at the same peripheral.
- Now, Pwrite is made low indicating a read cycle. At the same address a write cycle is performed followed by a read cycle. So the data transferred in the write cycle will be read back in the read cycle.

Test 2 :

In this test 2, data is being randomly written at different address locations and data is read back at a randomly chosen address location. Multiple such test are conducted

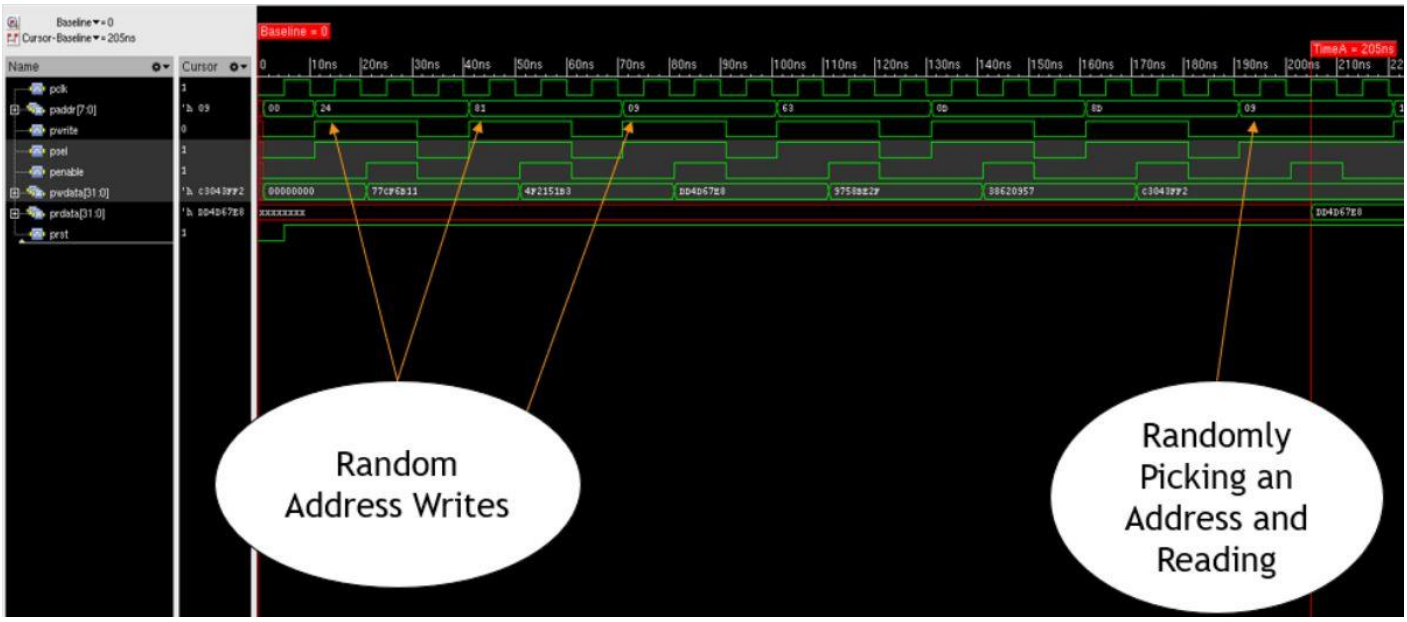


Fig 2.10 Simulation waveform showing random write and read cycles

Conclusion :

Now the AMBA APB Protocol timing diagram are achieved, we can use this APB slave logic in the Ethernet PHY to access the Ethernet control register by an ARM processor which is working on AMBA. This is explained clearly in the chapter 4.

Chapter 3

Interface Design for APB bridge and PHY layer registers

3.1 Problem statement :

In the previous chapter, APB slave protocol is explained. Also, how the control registers are accessed using the APB signals is also shown. But not all chips in the market use APB slave protocol. So question arises how to connect such circuits and access their registers.

Solution : The best solution is to build an interface which can help to understand the 3rd party chipset signals and creates a bridge between APB bridge and the 3rd part PHY. To do so we have to understand and map the signals correctly. This chapter explains about the interface which can help to access the registers of such device.

Specification of the 3rd party chipset :

CR Parallel interface : The CR Parallel interface is a synchronous, 16-bit data/address parallel port provided for on-chip access to control registers inside the PHY. The CR Parallel interface is enabled when input cr_para_sel is asserted to 1. While access to these registers is not required for normal PHY operation, this interface is included for users that want to access some of the PHY's diagnostic features during normal operation or to override some of the PHY's control signals.

The CR Parallel interface consists of following PHY interface signals: cr_para_addr, cr_para_clk, cr_para_rd_data, cr_para_rd_en, cr_para_wr_data, cr_para_wr_en, and cr_para_ack; and is enabled when input cr_para_sel is set to 1.

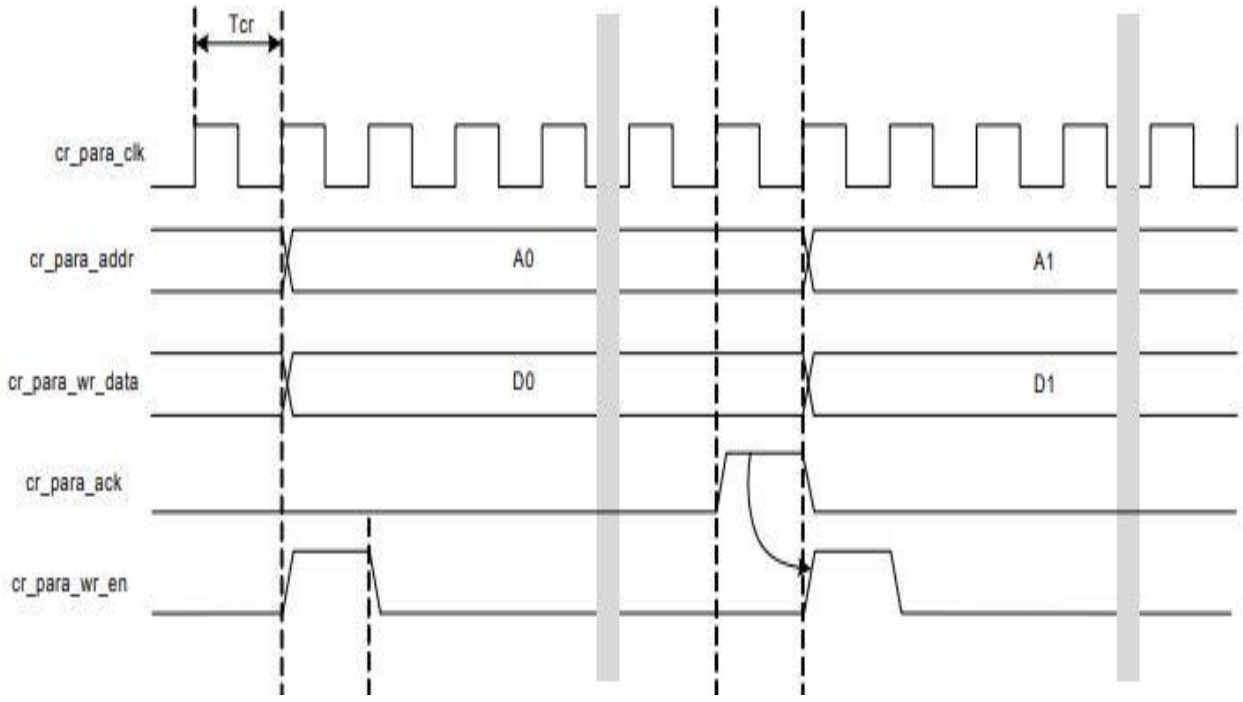


Fig 3.1 CR write cycle timing diagram

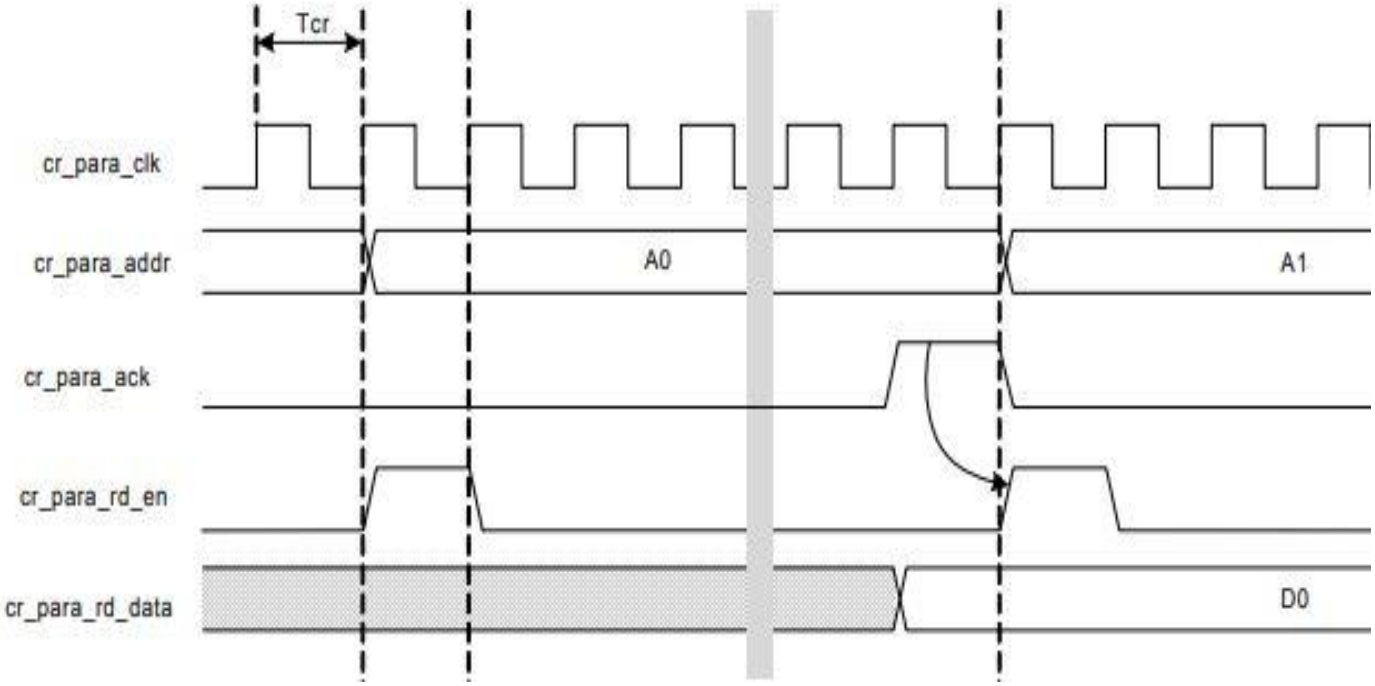
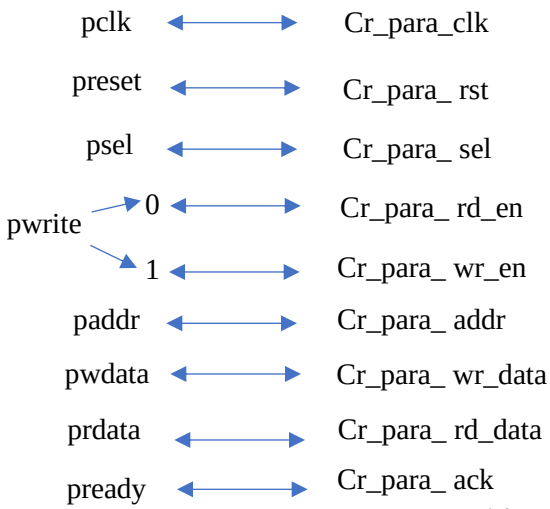


Fig 3.2 CR read cycle timing diagram

3.2 Interfacing APB bridge with PHY : By observing the above timing diagram, we can understanding the working of the respective signals.

- Cr_para_sel need to be high for the transfer to take place.
- Cr_para_rd_en should be high for a read cycle .
- cr_para_wr-en should be high for a write cycle.
- After each transfer cr_para_ack is made high by the device.
- Cr_para_wr_data is the input data to be written on to the device registers
- Cr_para_rd_data is the data read from the registers

Realation between APB slave signals and 3rd party device signals :Now we need to establish a relation between the signals so that we can access the registers .



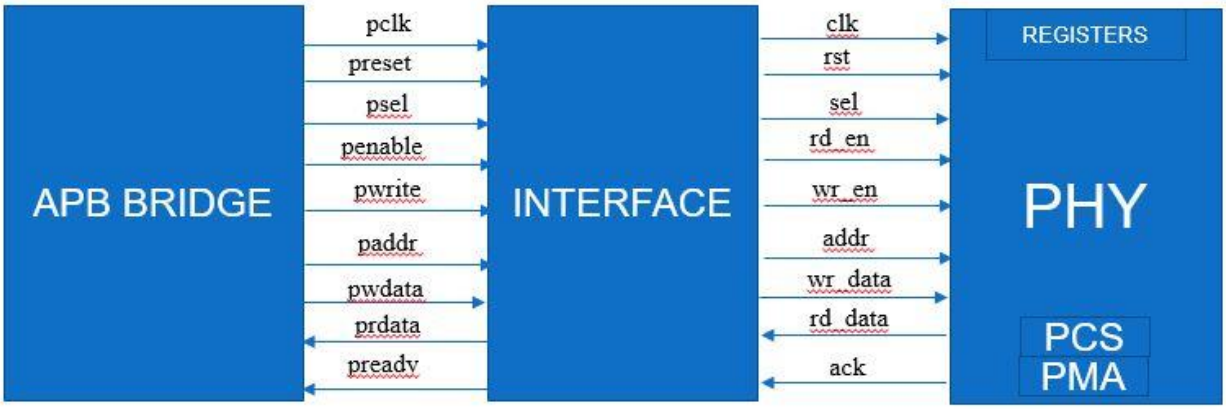


Fig 3.3 block diagram of interface between APB bridge and PHY

3.3 VERILOG and Testbench :

A interface is now built using the Verilog RTL code. The input side of the interface is APB slave and on the output side is the 3rd party chip. The Interface will take in the inputs from the APB bridge and generate the output signals which can be understood by the PHY.

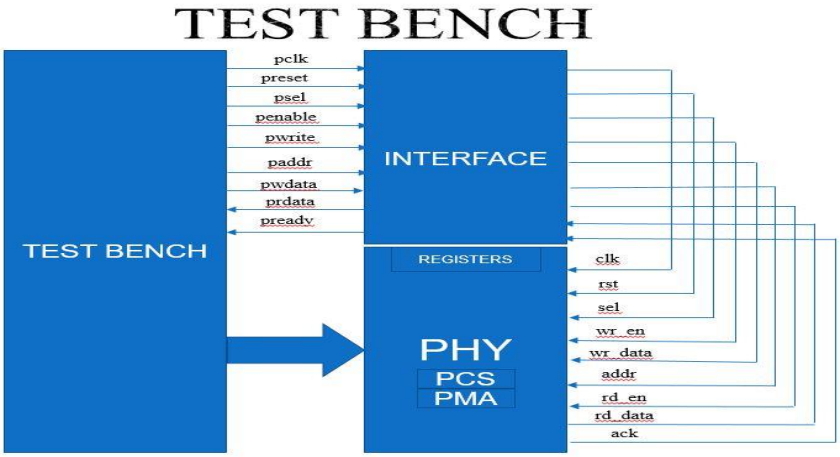
Pclk signal is given to the cr_para_clk inpujt of the PHY. Preset is connected to the cr_para_rst. Similarly other signals are generated.

Read and write cycle enable signals :

- When pwrite is 0, it denote a read cycle. So the interface will make cr_para_rd_en high when pwrite is 0.
- When pwrite is 1, it denotes a write cycle. So the interface will now make cr_para_wr_en high.

The outputs of the PHY are cr_para_ack and cr_para_rd_data. These two signals are connected to the Pready and prdata signals of the APB slave.

A test is developed as shown below :



The test bench is developed which can perform the actions of the APB bridge. The test bench will generate the signals according to the APB slave protocol. This generated signals are sent to the interface which will intern generate the signals for the PHY. Different test case scenarios are also made and tested.

3.4 Simulation Waveforms :

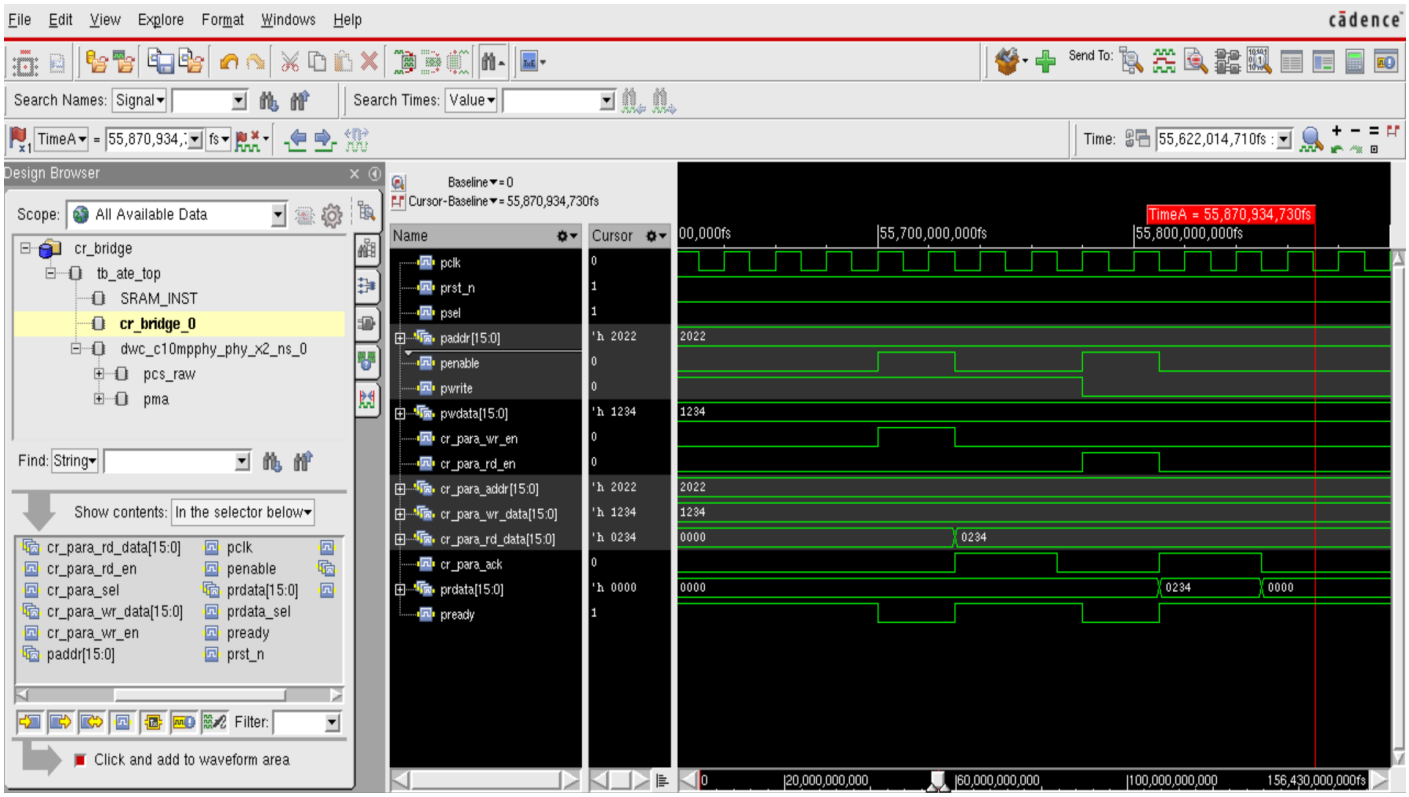


Fig 3.4 simulation waveform of the interface

In the waveform, we can see the paddr of 16'h2022 is being sent to the signal cr_para_addr. When pwrite is high, the cr_para_wr_en is high. When pwrite is low, cr_para_rd-en is high. A write cycle followed by a read cycle is performed and the results are obtained.

Conclusion :

If the timing diagram of the PHY which doesn't support APB protocol is known, then we can implement an interface and successfully access the registers without any timing issues.

Chapter 4

Introduction to "MAC interfaces with Ethernet PHY"

4.1 Introduction OSI Model

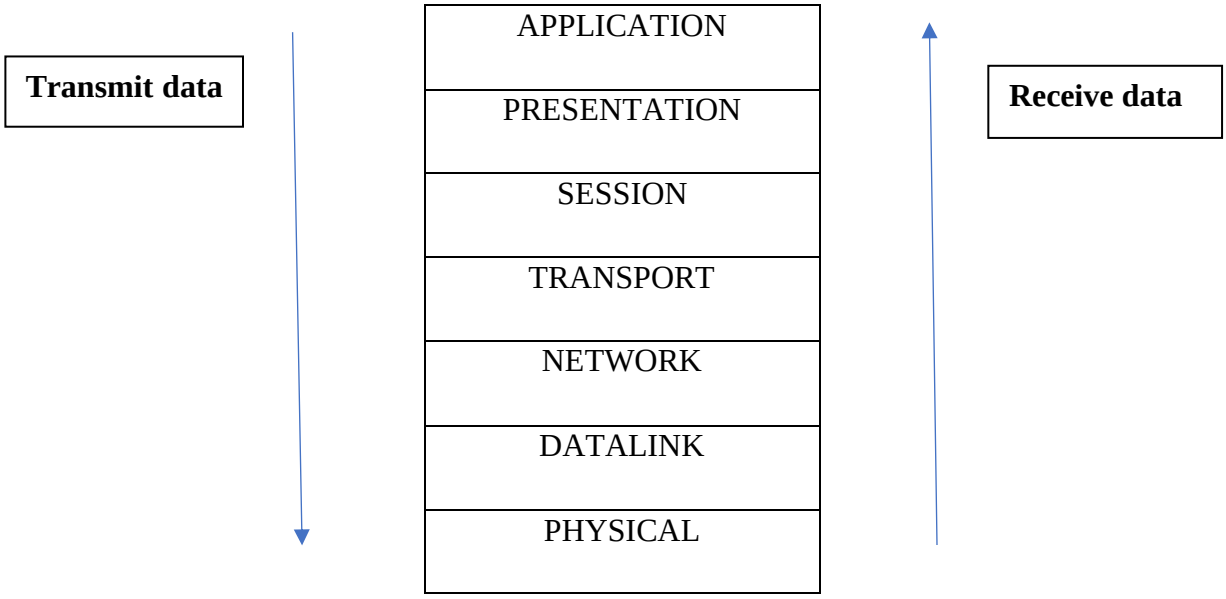


Fig 4.1 OSI model

The OSI model is Open systems Interconnection model which is standard for communication and networking. There are many communication systems available. This model helps in creating a bridge between all of them. It helps in interoperability among the networking systems.

Application layer 7:

- This layer is responsible for the user interface. Examples of application layer protocols are HTTP,SMTP,FTP etc.
- It deals with issues like network transparency and resource allocation

Presentation layer 6:

- This layer is responsible for data translation for a network. It deals with syntax of the data to be sent or received.
- Main activities are translation, compression, encryption

Session layer 5:

- Session layer is responsible for synchronization between the communicating devices
- It creates checkpoints between the devices and checks for errors

Transport layer 4:

- Transport layer is a very important layer as it takes care of the order in which data is sent and received .
- Data is securely sent without any duplication. It converts data into small segments.

Network layer 3:

- Network layer tracks the location of the communicating device over a network.
- It decides the best path for transferring data.

Data link layer 2:

- Data link layer decide the format in which data has to be transferred.
- It helps in identifying the correct device to make a transfer.
- It has two layers datalink control layer and media access control layer.

Physical layer 1:

- This is the physical layer which converts the bits into electrical or optical signals to transmit the data over a media.

4.2 MAC AND PHY layers :-

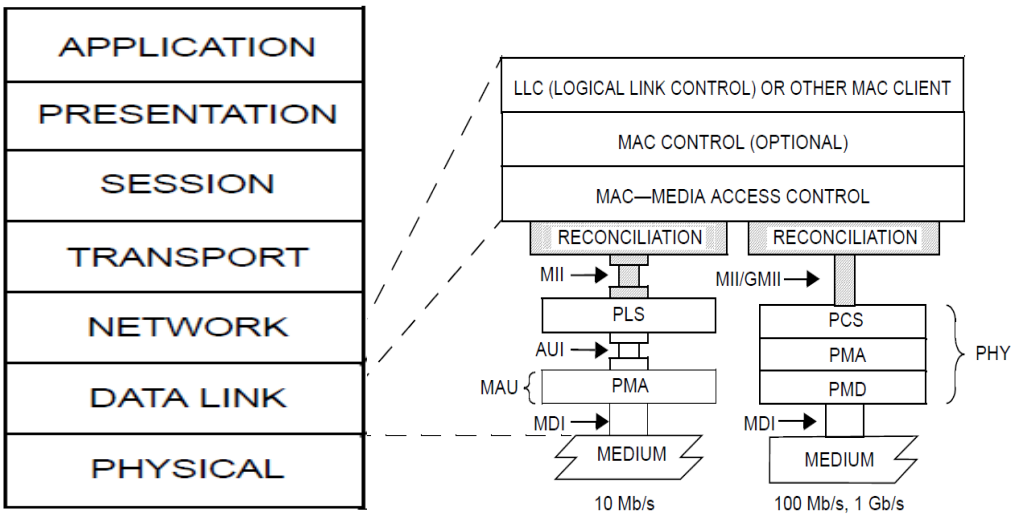


Fig 4.2 MAC and PHY layer

What is a Ethernet MAC?

The MAC is abbreviation for media access controller. The MAC layer is a part of Data link layer. It has two main functions. Data which is to be transferred is encapsulated using a head and a tail part. Data is converted into a frame and then transferred. Additional data is also added for Error detection reception. It can start a frame transmission and also resend a frame incase of a failure.

What is an Ethernet PHY?

The PHY is the abbreviation for physical interface transceiver. It is responsible for sending the data bits from one device to another over a networking media. It converts the bits into electrical or optical signal for transferring the data over the media. It can also reconvert the received frames from electrical to ones and zeros.

4.3 Media Independent Interface (MII)

The Media Independent Interface (MII) is a standard interface. This will establish a connection between Ethernet MAC and ethernet PHY. media independent term represents that different kinds of PHYs can be connected to the MAC devices without having to change the design.

The MII can support :

- Two specific data rates, 100 Mb/s and 10 Mb/s.
- 4bit interface clocked at 25 MHz for 100 Mbit/s, 2.5 MHz for 10 Mbit/s

Table 4.1.1 : MII TX ports

Signal name	Description
TX_CLK	Transmit clock
TXD0	Transmit data bit0
TXD1	Transmit data bit1
TXD2	Transmit data bit2
TXD3	Transmit data bit3
TX_EN	Transmit enable
TX_ER	Transmit error

Table 4.1.2: MII RX ports

signal name	Description
RX_CLK	Receive clock
RXD0	Receive data bit 0
RXD1	Receive data bit 1
RXD2	Receive data bit 2
RXD3	Receive data bit 3
RX_DV	Receive data valid
RX_ER	Receive error
CRS	Carrier sense
COL	Collision detect

Gigabit Media Independent Interface (GMII)

Gigabit Media Independent Interface (GMII) is also an interface between the Media Access Control (MAC) device and the physical layer (PHY).

The GMII can support :

- data speed up to 1Gbps ,
- 8 bit data interface with clockspeed of 125 MHz,

Table 4.2.1 : GMII TX ports

Signal name	Description
GTX_CLK	Clock signals for 1gbps signals
TXD[7..0]	Data to be transmitted
TX_EN	Transmitter enable
TX_ER	Transmitter error

Table 4.2.2 : GMII RX ports

Signal name	Description
RX_CLK	Received clock
RXD[7..0]	Received data
RX_DV	Received data is valid
RX_ER	Received data is corrupted
COL	Data collision
CS	Carrier sense

Reduced Gigabit Media Independent Interface (RGMII)

RGMII uses half the number of pins as used in the GMII interface. The number of pins are reduced and data is transferred at the two edges of the same clock. The positive edge and negative both are used for transferring the data. carrier-sense and collision-detection signals are removed.

The RGMII can support :

- data rates up to 1000 Mbit/s,
- 8 bit data interface clocked at 125 MHz.

Table 4.3 : RGMII ports

Signal name	Description
TXC	Clock signal
TXD[3..0]	Data to be transmitted
TX_CTL	transmitter enable and transmitter error are used in same pin
RXC	Received clock signal
RXD[3..0]	Received data
RX_CTL	Data valid and error detections signals are used in same pin by muxing

Serial Gigabit Media Independent Interface (SGMII)

The Serial Gigabit Media Independent Interface (SGMII) is an improved version of MII. It also connects the MAC and PHY layers. Reduced power and ports makes it better than GMII

- It operate at 1.25 Gbaud at clockspeed of 625 MHz. Double data rate interface is used.
- Differential pair signaling is used which will reduce the noise.

Table 4.4 SUMMARY OF MII VS GMII VS RGMII VS SGMII :

INTERFACE	SPEED(Mbps)	DUPLEX	No of Pins	Clock speed(Mhz)
MII	10/100	FULL/HALF	16	2.5/25
GMII	10/100/1000	FULL	24	125
RGMII	10/100/1000	FULL/HALF	12	125
SGMII	10/100/1000	FULL/HALF	10	625

4.5 IEEE 802.3 Ethernet Frame Format :-

Preamble	SFD	Destination MAC	Source MAC	Type	Data and Pad	FCS
7 Bytes	1 Byte	6 Bytes	6 Bytes	2 Bytes	46-1500 Bytes	4 Bytes

Fig 4.3. Ethernet frame format

Preamble :

- Ethernet frame begins with alternating zero's and one's.
- It helps the receiver to understand that a frame is about to come.

Start Frame Delimiter SFD :

- It is two bytes of 8'hd5.After the sfd, addresses begin.

Destination and source addresses :

- They are each 6 bytes. They contain the addresses of the destination MAC and source MAC address

Type/ length :

- It is 2 bytes. It can be either type of the payload data or the length of the data.

Payload :

- The main data to be transferred is called the payload
- It is of 46 to 1500 bytes. It can be future divided based on the type of data.

FCS : Frame check sequence

- This is 4 bytes of data which is used for error correction.

4.6 Cyclic Redundancy Check (CRC) :- The below method is taken from the reference [2]

Cyclic Redundancy Check (CRC) is method to detect errors in the received data. It comes from the branch of linear block code and is widely used in communication systems. An additional code is created using the data to be sent and it is added at the end of the data.

- Message bits as polynomial M(x) = $x^3 + x^2 + x + 1$
- Standard generating polynomial G(x) = $x^4 + x + 1$

$$\frac{\boldsymbol{M(x)*x^4}}{\mathrm{G(x)}} = \frac{x^7+x^6+x^5+x^4}{x^4+x+1} = x^3 + x^2 + x + \frac{x}{x^4+x+1}$$

The remainder is x. so the corresponding CRC-4 of 1111 is 0010. At the sending side, the bits 11110010 will be forwarded.

Extending the concept for CRC-32

$$\boldsymbol{x^{32} + x^{26}+x^{23}+x^{22}+x^{16} + x^{12} + x^{11} + x^{10} + x^8+x^7 + x^5 + x^4 + x^2 + x + 1}$$

The CRC encoding theory is based on polynomial manipulation using modulo2 arithmetic . The modulo2 arithmetic is realized by XOR gate. Therefore the coding of CRC means that transforming the information bits into the bits that can be divided by the generated multinomial. At the receiver side, we can use the generated multinomial G(x) to divide the received code. If the result is zero, there is no error, or else the error emerged.

The generated crc register is appended to the ethernet frame and sent to the checker. On the checker side, we can decode received fcs either by calculating fcs of the received pay load bits and comparing it with received FCS.

The CRC-32 polynomial is 04C1 1DB7h. In order to avoid the influence of extra zeros in front of the DPP, the initial value of CRC-32 is 0xFFFF FFFF. CRC-32 is calculated for all bytes of the Data Packet Payload. Avoiding the influence of extra zeros at the end of the DPP, complementing the remainder of CRC-32. The inversion of the CRC-32 remainder adds an offset of 0xFFFF FFFF that will create a constant CRC-32 residual of C704DD7Bh at the receiver side, called magic number.[2]

CONCLUSION :-

The best MII interface which can be taken is SGMII interface with 8 pin outputs. 32 bit Cyclic Redundancy checksum can be used to verify the received frames after transmitting over data paths.

CHAPTER 5

Design and Implementation of Built In Self-Test (BIST) Engine

5.1 Ethernet PHY Built In Self-Test Generator & Checker :

Built In Self-Test Generator & Checker enables the ethernet PHY to test its data path without MAC.

MAC layer sends ethernet frames to the PHY layer over media independent interface(MII) and these frames pass through the data path and reach MII receive side. To replicate this action, BIST Generator is used to generate the ethernet frame. On the other side of data path, BIST checker will receive this frame and checks for errors. If errors are found, then data path has to be checked for silicon bugs.

Below diagram depicts the general system level block diagram of presence of Bist engine:

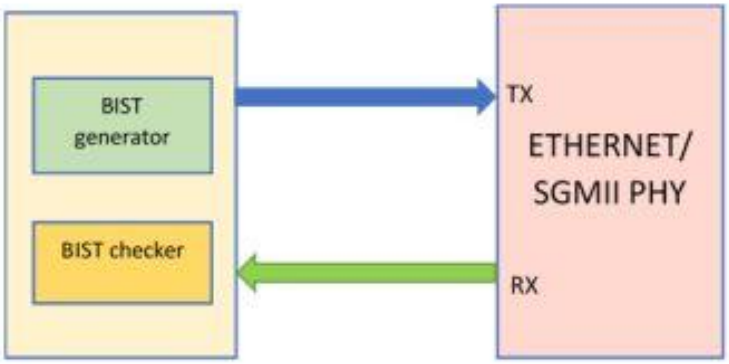


Fig 5.1 Typical diagram of BIST engine & ETHERNET/SGMII PHY

The block with BIST generator and BIST checker is called as BIST engine and it is connected through the data path line of the ETHERNET/ SGMII PHY. Data traverses from BIST generator through the data path and finally to the checker.

There are multiple ways to interface Bist engine with single PHY or with multiple PHYs'.

Block diagram with data flow shows the possible scenarios:

In the diagram, SGMII chip is connected to an ETHERNET PHY. Two sets of BIST generators and checkers are shown. SGMII BIST generator is the one connected to the SGMII PHY side and ETHERNET BIST generator is the one connected to the ETHERNET PHY side. Similarly SGMII BIST checker AND ETHERNET BIST checker are also used.

SGMII BIST generator and checker makes the SGMII BIST engine. And ETHERNET BIST generator and checker make the ETHERNET BIST engine.

ETHERNET traffic generator is an external generator used while performing post Si validation. It will also generate the ethernet frame and sends them over the data path same like BIST generator.

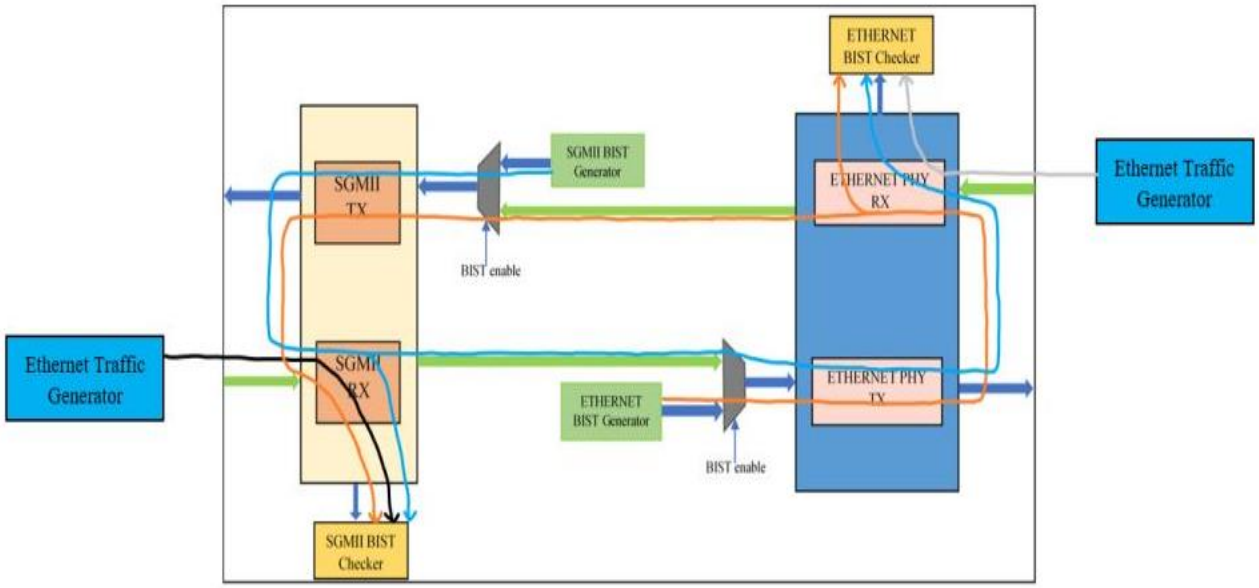


Fig 5.2 Data flow showing diagram of possible scenario

Here is the possible data flow:

- EPHY Bist generator - EPHY TX – EPHY RX – (EPHY Bist checker) - SGMII TX – SGMII RX – SGMII Bist checker (ORANGE PATH)
- SGMII Bist generator - SGMII TX – SGMII RX – (SGMII Bist checker) - EPHY TX – EPHY RX – EPHY Bist checker (CYAN PATH)
- External ethernet traffic generator - EPHY RX – EPHY Bist checker (GREY PATH)
- External ethernet traffic generator - SGMII RX – SGMII Bist checker (BLACK PATH)

Advantages of having two sets of BIST engine (BIST generator and BIST checker):

1. Simultaneously both PHY can be tested .
    - EPHY standalone and SGMII standalone.
    - Start from EPHY Bist generator end with SGMII Bist checker. If SGMII Bist checker results fail, then we know the issue is in EPHY or SGMII or EPHY to SGMII interface .
    - Start from SGMII Bist generator end with EPHY Bist checker. If SGMII Bist 36 checker results fail, then we know the issue is in EPHY or SGMII or EPHY to SGMII interface.
2. Risk involved in clock muxing and data muxing is avoided.
3. Design size is not bulky.
4. If wrapper is created including SGMII IP and Bist engine(apb slave, bist gen, bist checker), then wrapper can be seamlessly reused in any other projects (wrapper would have been verified).

Disadvantage:

1. User has to programme two BIST engine

5.2 Block diagram of BIST Engine:

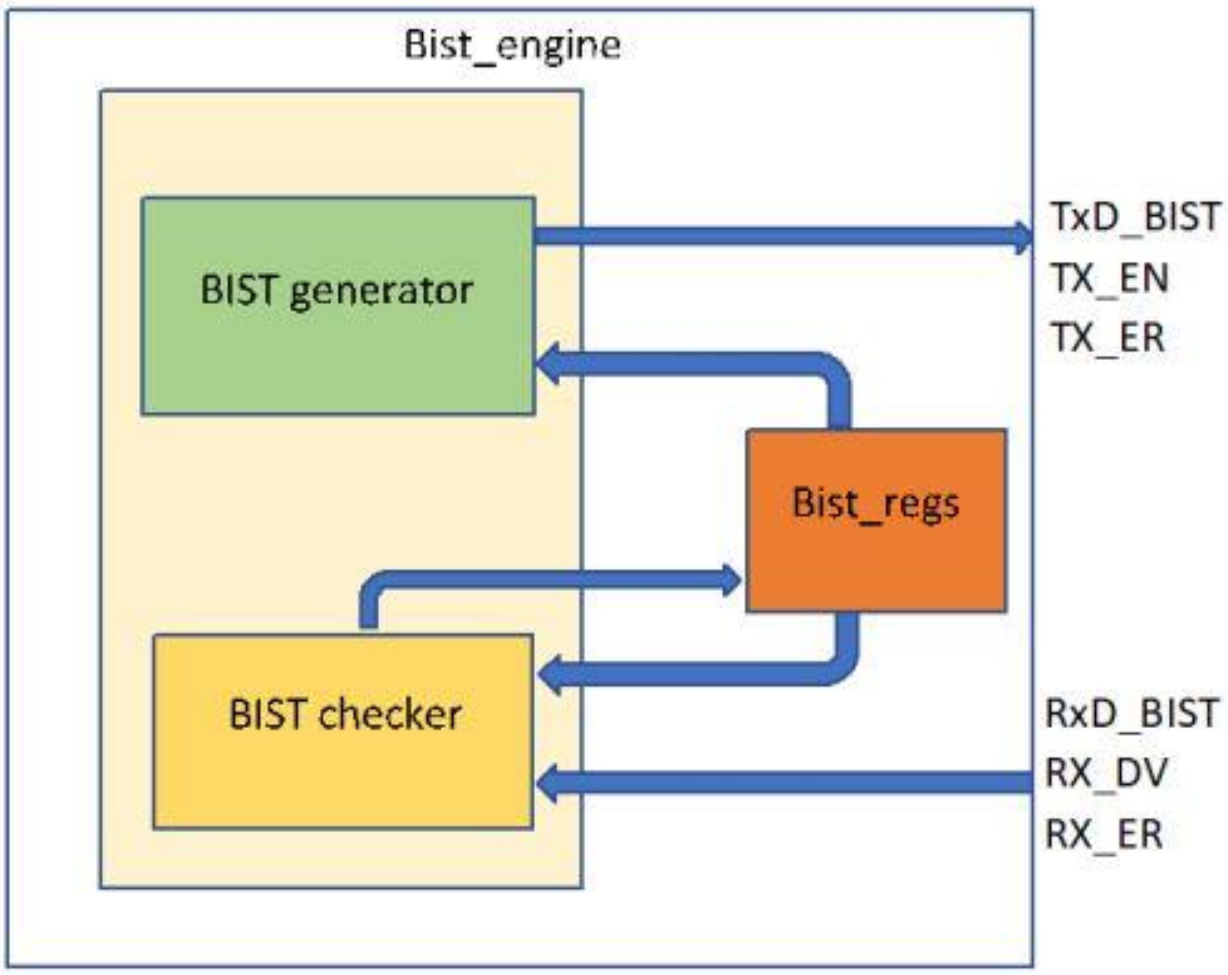


Fig 5.3: Block diagram of Bist engine

Bist generator and checker starts working when the respective enable bits are set. BIST generator generate the ethernet frames according to the frame length and other configuration bits. This frame is sent through a MUX to the ethernet PHY and is tapped by the BIST checker. BIST checker performs the CRC calculations and then set the status. The status registers (and statistic registers) can be analysed to find the issue, if any. There are two modes of operation They are validation mode and production mode.

Production mode:

This mode can be used by the production test. In this mode, programmable total number of frames are sent and expect the same total number of frames at the checker. Total number of frames can be a combination of programmable good frames and programmable error injected frames (FCS corrupt) or can be either one. In the BIST checker, after a BIST Done (after timeout), BIST status indicates the BIST is fail or pass. If the BIST is failed, then check the individual flag bits which indicates the actual reasons for fail.

This mode can be used during the validation stage as well

Validation mode:

This mode is very useful when the external ethernet tester is used to test the DUT. In this mode, external ethernet tester generates ethernet frames and the statistics are issued by the BIST checker. BIST checker issues a statistics of good frames received, bad frames received, number of frames in which rx_er is set when rxdv is high.

5.3 BIST Generator:

The BIST Generator is capable of generating the IEEE 802.3 ethernet frame format with varying frame length and different types of payload data. Once BIST generator is set, it starts generating the ethernet frame according to configuration. This module works for GMII interface and data converter for MII is outside the scope of BIST engine.

Complete custom frame :-

Preamble 7bytes	SFD 1 byte	DA 6 bytes	SA 6 bytes	T/L 2 bytes	Packet Data 46-1500 bytes	FCS 4bytes	IPG 12 bytes

PREAMBLE, SFD and IPG:-

Data length for preamble and IPG are programmable

DA & SA:

Destination and Source addresses are up to 6 bytes each and are configured using the registers BIST_DA and BIST_SA. Higher bytes of DA and SA are sent first .

T/L:

To make it easy and completeness, fixed some value is sent always (instead of indicating frame length count) and is not checked in BIST checker.

FRAME PAYLOAD LENGTH:-

BIST generator is capable of generating different payload sizes. Here is the supported types:

- It can generate fixed programmable number of bytes for every frame.
- It can start with programmable number of bytes and then keep incrementing by one byte for every subsequent frames
- Random data length is chosen with a minimum of 46 bytes.

Here is the supported payload data type:

- Ramp data pattern
- Fixed programmable value
- PRBS data (from LFSR)

Generated random sequence depends on the seed value. BIST generator can choose seed value to be either random number or a programmable number. If the programmable number is chosen, then the same value is loaded to initialize for every ethernet frame. If the random number type is chosen, then random seed value is used to initialize for every ethernet frame.

FCS :-

The Frame Check Sequence is calculated from DA to frame-data and appended to the frame payload. To process 8 bits per clock cycle, 8-bit parallel CRC-32 is used for FCS. BIST Generator also generates the GMII transmit interface control signals (TX_EN & TX_ER = 1'b0 always) along with the TXD.

ERROR INJECTION:-

BIST generator provides FCS (Frame Check Sequence) Error Insertion. FCS Error insertion allows sending frames with wrong FCS value. FCS Error insertion allows programmable error insertion, where programmed number of outgoing frames will have FCS error. This should be detected by BIST checker as an error frame.

FRAME COUNT:

Number of good frames and error frames is programmable. Hence total number of generated frames will be number of good frames plus number of error injected frames. In validation mode, BIST generator is not required, hence these are not required.

5.4 BIST Checker:

The BIST checker feature is to qualify the received ethernet frame and generate the status and various statistical results. Bist checker is enabled by programming a configuration bit. BIST checker is not tied with BIST generator and altogether can independently work.

FCS calculation:

BIST checker monitors the received bit stream for the SFD bit pattern. If pattern is detected, it will perform Cyclic Redundancy Check on received data pattern until the end of frame (RX_DV falling edge). If this result is equal to the magic number of the standard 8 bit CRC-32 (32'hC704_DD7B) then the received frame is good else the frame is termed as bad.

Good frame and Bad frame counters:

Good frame counter keep counting the good frame and bad frame counter keep counting the bad frame. These counters are available for host to read at any point of time (for both validation and production modes).

In validation mode, these counters are used as a statistical registers. If the clear bit is set, then these counters are resetted and starts counting from zero for any new good and bad frames.

Production mode BIST Fail status :

Following error status is reported when BIST done is set (BIST done is set when expected good frames and expected bad frames is received or timed out, which ever event happens first).

1. Good frame fail:
   If the expected number of good frames are not received, then bit flag is set.

2. Bad frame fail:
   If the expected number of bad frames are not received, then bit flag is set (Error is injected, but not received)

3. rx_dv is not detected:
   If RX_DV is not received by the BIST checker, then this status is set.

4. RX_ER detected:
   This status is set when rx_er is set with rx_dv is high

Note: BIST_CHECK_DONE and BIST Fail flags are not valid in validation mode

5.5 BIST Registers :

BIST generator control :- BIST_GEN_CTRL

This is a 16 bits register which controls the controls of the BIST generator. Its default value is 0x0004.

- BIST Generator enable : BIST_GEN_EN
  This bit controls the BIST generator's enable signals. When BIST_GEN_EN is zero, then BIST generator is disabled, else when it is one BIST generator is enabled.

- Payload data type :
  This is of two bits.
  - 00 - Programmable data is sent
  - 01 - Ramp data is sent
  - 10 - PRBS output is sent (see BIST_PRBS_SELECT)
  - 11 - Not used

- PRBS SELECT: BIST_PRBS_SELECT
  Different types of PRBS is selected
  - 1'b0 – PRBS7
  - 1'b1 – PRBS13

- LFSR RANDOM VALUE ENABLE : LFSR_RANDOM_EN

This bit selects the seed value of the LFSR.

- 1'b0: Seed value is taken from BIST_LFSR_SEED register
- 1'b1: Seed value is taken from random value

- GENERATOR MODE : BIST_GEN_MODE

  Controls whether BIST is operating in validation mode or production mode.
  - 1'b0 – Production mode. BIST generator generates number of programmed good frame and number of programmed bad frames.
  - 1'b1 – Validation mode. Bist generator is not used.

Preamble and IPG data length: PREAMBL_IPG_SIZE

This register is loaded with the preamble and IPG data length.

PREAMBLE_LENGTH: Length of preamble in bytes.

IPG_LENGTH: Length of IPG in bytes (96 bits as per IEEE standard).

BIST_DA_0: Lower 16 bits [15:0] of DA value .

BIST_DA_1: middle 16 bits [31:16] of DA value .

BIST_DA_2: higher 16 bits [47:32] of DA value .

BIST_SA_0: Lower 16 bits [15:0] of SA value.

BIST_SA_1: middle 16 bits [31:16] of SA value.

BIST_SA_2: higher 16 bits [47:32] of SA value .

Payload control register : PAYLOAD_CTRL

This register controls the payload data.

- Constant payload data: CONST_PAYLD_DATA

  When constant data type is chosen, then the value from this register is taken.
- Payload length select: PAYLD_LENGTH_SEL

  Indicates the increment in payload size :
  - 2'b00- No increment in payload size (Always PAYLOAD_LENGTH) .
  - 2'b01- Increment in payload size by 1 (Begin with PAYLOAD_LENGTH) .
  - 2'b10- Random number is taken as payload size with a minimum value of 46 bytes.
  - 2'b11- Not used.

Payload length:

This register is loaded with payload length.

PAYLOAD_LENGTH: Length of frame payload data in bytes.

Good Frame Count : BG_GOODFRAME_CNT

This register is loaded with Number of good frames to be generated by BIST generator.

Bad Frame Count : BG_BADFRAME_CNT

This register is loaded with Number of good frames to be generated by BIST generator.

BIST LFSR SEED value : BIST_LFSR_SEED

This register is loaded with Initial value to be loaded into the LSFR for PRBS data.

Generator Status : BIST_GEN_STATUS

This register denotes the status of the generator.

- BIST_GEN_DONE :

  This status flag is set when programmed number of good frame and bad frame are completely generated and sent.

Bist Checker Control Register :

This register controls the BIST checker.

- BIST_CHECKER_EN
  - Bist checker enable
    0 – Bist checker is disabled

    1 – Bist checker is enabled
- BIST_CHECK_MODE
  - Controls whether BIST is operating in validation mode or production mode.
    1'b0 – Production mode. Bist checker checks for BIST done and BIST fail status.

    1'b1 – Validation mode. Bist checker keep checking the good frames, bad frames and rx_er when rx_dv is set. Results are available in statistical registers.
- STATISTC_CNT_RST
  - If this bit is set, then resets all status – BIST_STATUS, GOOD_FRAME_CNT, BAD_FRAME_CNT.
  - This bit resets good frame counter, bad frame counter and rx_er detected frame counter.
  - After reset, all status registers hold fresh values.
  - This bit is self-clearing bit. This bit is used only when BIST is in validation mode.
  -

Bist Checker Good Frame Count :

- BC_GOODFRAME_CNT :

- Expected number of good frames generated by BIST generator.
- This register is used to indicated expected good frames is received or not.

Bist Checker Bad Frame Count :

- BC_BADFRAME_CNT :
  - Expected number of bad frames generated by BIST generator.
  - This register is used to indicated expected bad frames is received or not.

Bist Timer:

This register contains the time out value.

- BIST_WAIT_TIMER :
  - Timer timeout value for checking all status in production mode. This count begins when BIST checker is enabled. Hence this register should be programed before BIST checker is enabled.
  - Ideally this value should be- time taken to send all frames by the BIST generator plus latency from BIST generator to BIST checker plus additional time to program BIST generator and checker enable bit.
  - This is time out register and is useful when expected good number of frames and bad number of frames is not received. In this condition, time out asserts BIST_CHECK_DONE status.
  - This count is in terms of microseconds.

Bist Production Mode Status :

BIST status used when BIST is in production mode and NOT valid for validation mode.

- BIST_CHECK_DONE :
  - This bit indicates BIST status is ready for reading. This bit is set when expected good number of frames and bad number of frames is received or timed out.
- BIST_CHECK_FAIL
  - Indicates at least one of the following error status is set:
    1. Good frame fail: If the expected number of good frames are not received, then bit flag is set.
    2. Bad frame fail: If the expected number of bad frames are not received, then bit flag is set (Error is injected, but not received)
    3. rx_dv is not detected: If RX_DV is not received by the BIST checker, then this status is set.
    4. RX_ER detected: This status is set when rx_er is set with rx_dv is high.

- If BIST_CHECK_DONE is set and BIST_CHECK_FAIL is logic zero, then it is said to be PASSED. If BIST_CHECK_DONE is set and BIST_CHECK_FAIL is logic high, then it is said to be FAIL.

- GOOD_FRAME_FAIL:
  If the expected number of good frames are not received, then bit flag is set.

- BAD_FRAME_FAIL:
  If the expected number of bad frames are not received, then bit flag is set (Error is injected, but not received)

- RXDV_DETECT_ER:
  If RX_DV is not received by the BIST checker, then this status is set.

- RX_ER_DETECT_ER:
  This status is set when rx_er is set with rx_dv is high

Bist Received Good Frame Count :

- GOOD_FRAME_CNT:
  - Counts the number frames for which CRC magic number is matched.
  - This can be used in both production mode and validation mode.
  - This counter can be reset by programmable bit to count fresh good frame (in validation mode).

Bist Received Bad Frame Count :

- BAD_FRAME_CNT:
  - Counts the number frames for which CRC magic number is NOT matched.
  - This can be used in both production mode and validation mode.
  - This counter can be reset by programmable bit to count fresh bad frame (in validation mode).

Bist Error Frame Count:

This register stores the error frames for which RX_ER detected when rx_dv is set.

- RX_ER_FRAME_CNT:
  - Counts the number frames for which rx_er is detected when rx_dv is high.
  - This can be used in both production mode and validation mode.

5.1 Register Table:

These register are accessed using the APB SLAVE protocol as explained in chapter 2.

REGISTER NAME	ADDRESS	Power on default value	ACCESS
BIST_GEN_CTRL	0x0	POD: 0x0004	RW
PREAMBL_IPG_SIZE	0x1	POD: 0x0C07	RW
BIST_DA_0	0x2	POD: 0x0000	RW
BIST_DA_1	0x3	POD: 0x0000	RW
BIST_DA_2	0x4	POD: 0x0000	RW
BIST_SA_0	0x5	POD: 0x0000	RW
BIST_SA_1	0x6	POD: 0x0000	RW
BIST_SA_2	0x7	POD: 0x0000	RW
PAYLOAD_CTRL	0x8	POD: 0x0066	RW
PAYLOAD_LENGTH	0x9	POD: 0x0064	RW
BG_GOODFRAME_CNT	0xA	POD: 0x0064	RW
BG_BADFRAME_CNT	0xB	POD: 0x0000	RW
BIST_LFSR_SEED	0xC	POD: 0xFFFF	RW
BIST_GEN_STATUS	0xD	POD: 0x0000	RO

REGISTER NAME	ADDRESS	Power on default value	ACCESS
BIST_CHECK_CTRL	0x18	POD: 0x0000	RW
BC_GOODFRAME_CNT	0x19	POD: 0x0064	RW
BC_BADFRAME_CNT	0x1A	POD: 0x0000	RW
BIST_WAIT_TIMER	0x1B	POD: 0x00FF	RW
BIST_PROD_STATUS	0x1C	POD: 0x0000	RO
GOOD_FRAME_CNT	0x1D	POD: 0x0000	RO
BAD_FRAME_CNT	0x1E	POD: 0x0000	RO
RX_ER_FRAME_CNT	0x1F	POD: 0x0000	RO

Note: RW is read and write access. RO is read only.

5.6 BIST Programming sequence :

Production Mode:

1. Wait for Link up

2. Set BIST_GEN_MODE and BIST_GEN_MODE for production mode

3. Program all relevant registers which controls BIST generator to generate required frames (except BIST generator enable)

4. Program all relevant registers which controls BIST checker (except BIST checker enable)

5. Program BIST_WAIT_TIMER

6. Enable BIST checker

7. Enable BIST generator

8. Poll BIST_CHECK_DONE status bit.

9. If BIST_CHECK_DONE is set, then read BIST_CHECK_FAIL.

10. If BIST_CHECK_DONE is set and BIST_CHECK_FAIL is logic zero, then it is said to be PASSED. If BIST_CHECK_DONE is set and BIST_CHECK_FAIL is logic high, then it is said to be FAIL. Check further status flag to know the reason for failure.

11. If needed (as this mode can be enabled during validation stage too) read GOOD_FRAME_CNT, BAD_FRAME_CNT, RX_ER_FRAME_CNT

12. If wish to repeat the same step, disable both BIST generator and BIST checker. Follow step 2-11 or step 6-11 as per the requirement

Validation Mode:

1. Wait for Link up

2. Run external ethernet tester to generate the traffic

3. Set BIST_CHECK_MODE for validation mode

4. Enable BIST checker

5. Wait for some time to receive frames

6. Read GOOD_FRAME_CNT, BAD_FRAME_CNT, RX_ER_FRAME_CNT

7. Program STATISTC_CNT_RST bit to reset all statistic registers

8. If needed, follow step 4-5

Testbench :

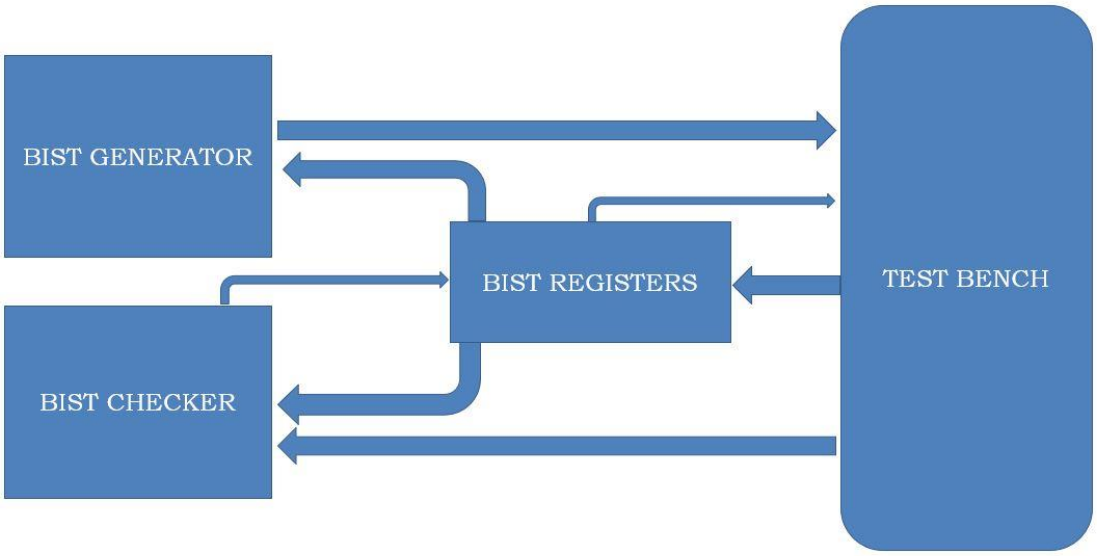


Fig 5.4 : Test Bench Driving BIST ENGINE

Above figure shows the flow of signals from testbench to the DUT and from DUT to the testbench.

Step 1 : load the BIST control register values.

Step 2: BIST registers will drive the BIST engine by giving enable signals.

- BIST generator and checker will now start and results will be store in status registers of BIST register bank

Step 3: Read the status register and analyse the results.

5.7 Simulation Results :

First step to run the BIST engine is to Load the Registers with desired values. All the results are divided based on validation mode and production mode

Accessing the registers :

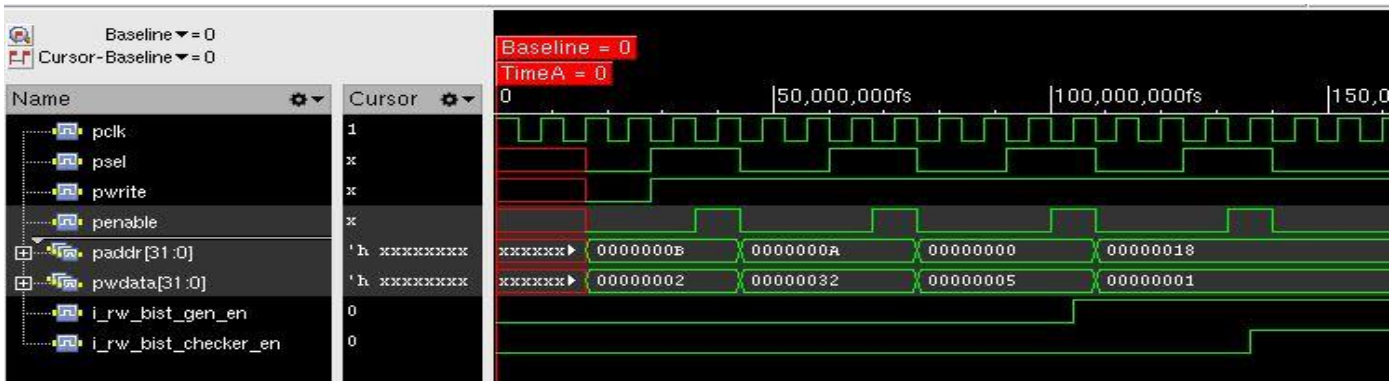


Fig 5.5 Accessing the registers

In the above simulation waveform, multiple write cycle are performed. At address 0x0000 and 0x0018 Enable signals are made high. And this is seen by bist_gen_en and bist_check_en going high after the cycle.

Generator Inputs :

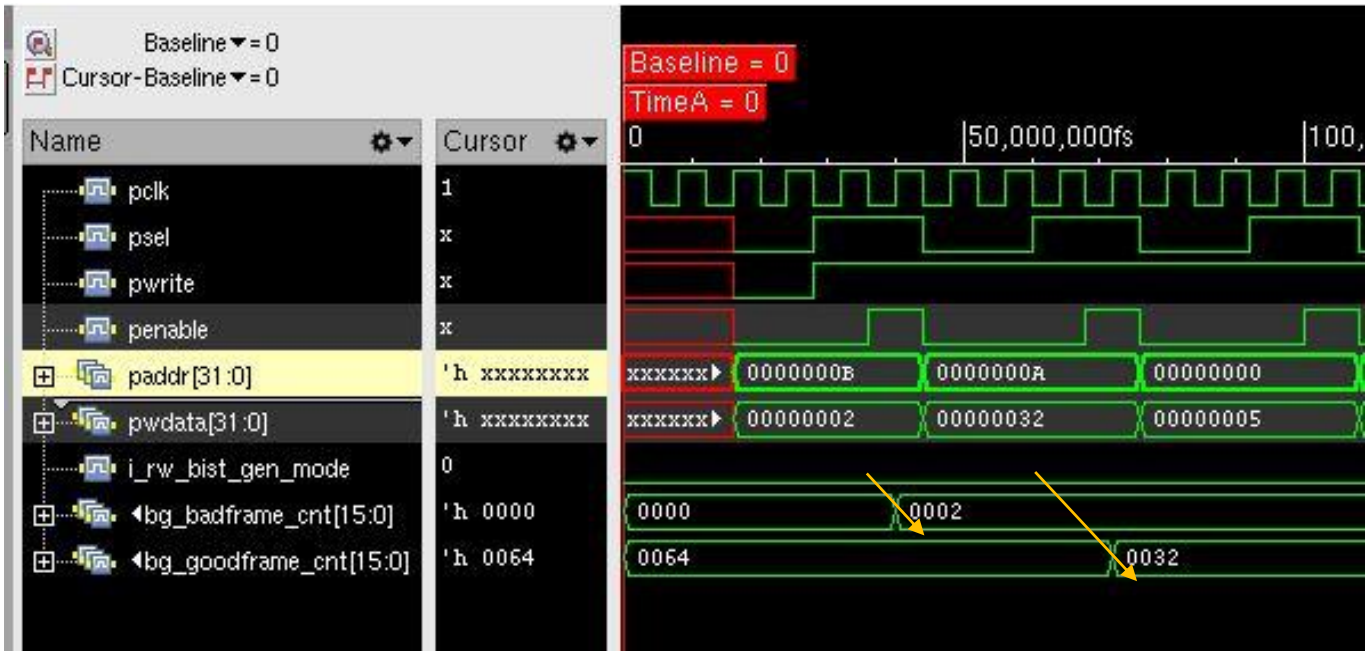


Fig 5.6 Waveform 1

In this waveform, generator mode is zero representing production mode. And number of good frames are 16'h0032 and number of bad frames are 16'h0002. These are the inputs taken for this test.

Destination and Source addresses:-

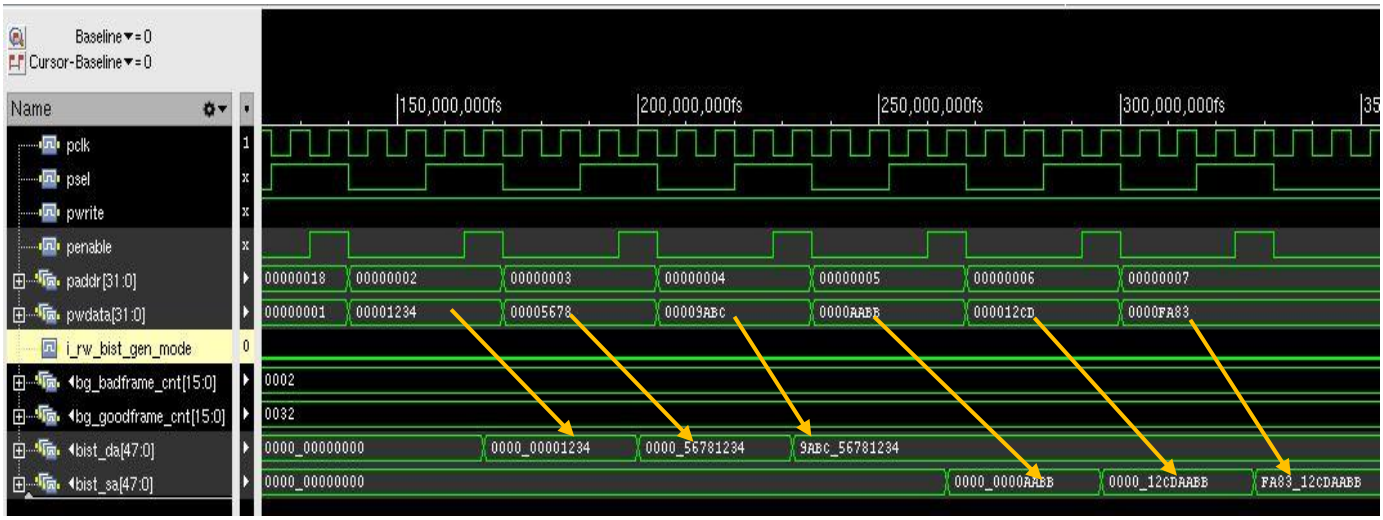


Fig 5.7 waveform 2

The waveform 2 shows how 16 bit address is converted to a 48 bit address in side a generator. 6 bytes of data represents the address in ethernet frame format.

Default Values :

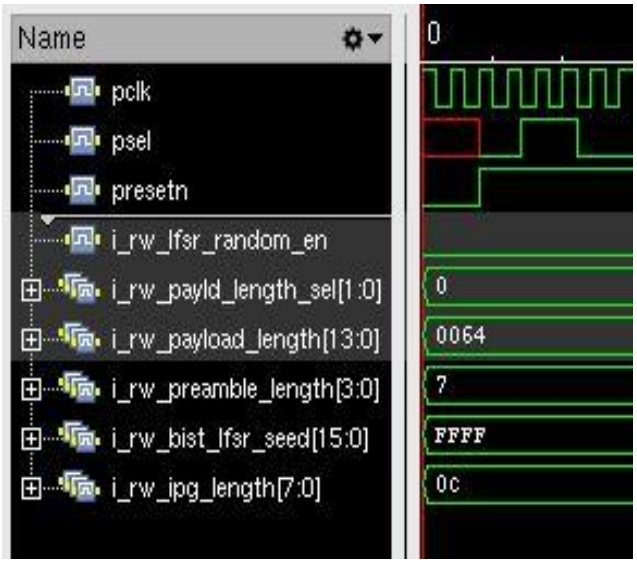


Fig 5.8 waveform 3

In the waveform3, we can see the power on default value(refer register table) are sent by the register bank when a reset signal is applied. Also when lfsr_random_en value is zero, default value 16'hFFFF is used as a seed value for PRBS.

Generator Outputs:

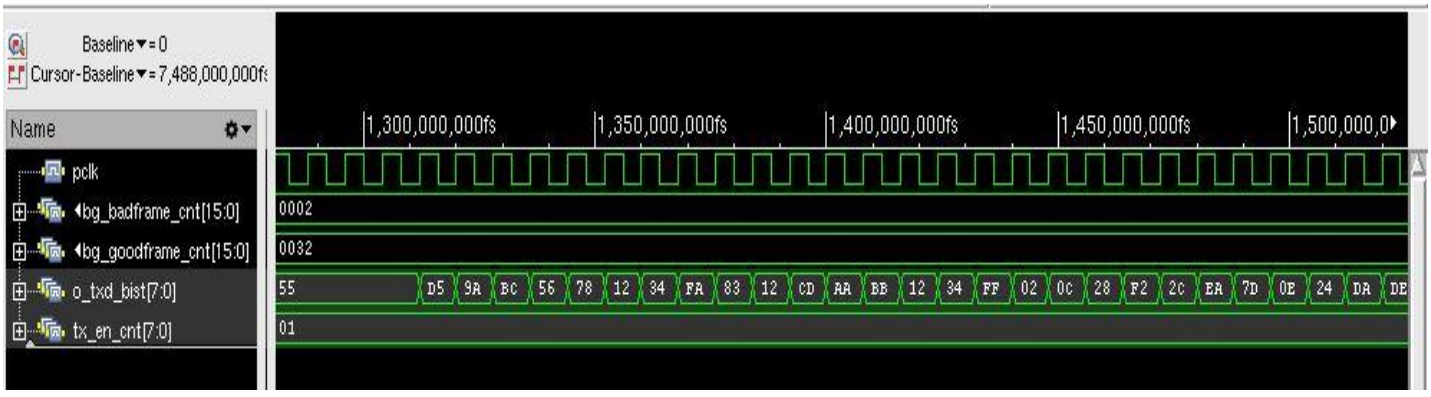


Fig 5.9 Waveform 4

The waveform 4 is showing the first generated frame. It started with 7 bytes of preamble 8'h55. Followed by SFD 8'Dd5. Then the address shown in waveform 2 are followed. Tx_en_cnt shows the frame number.

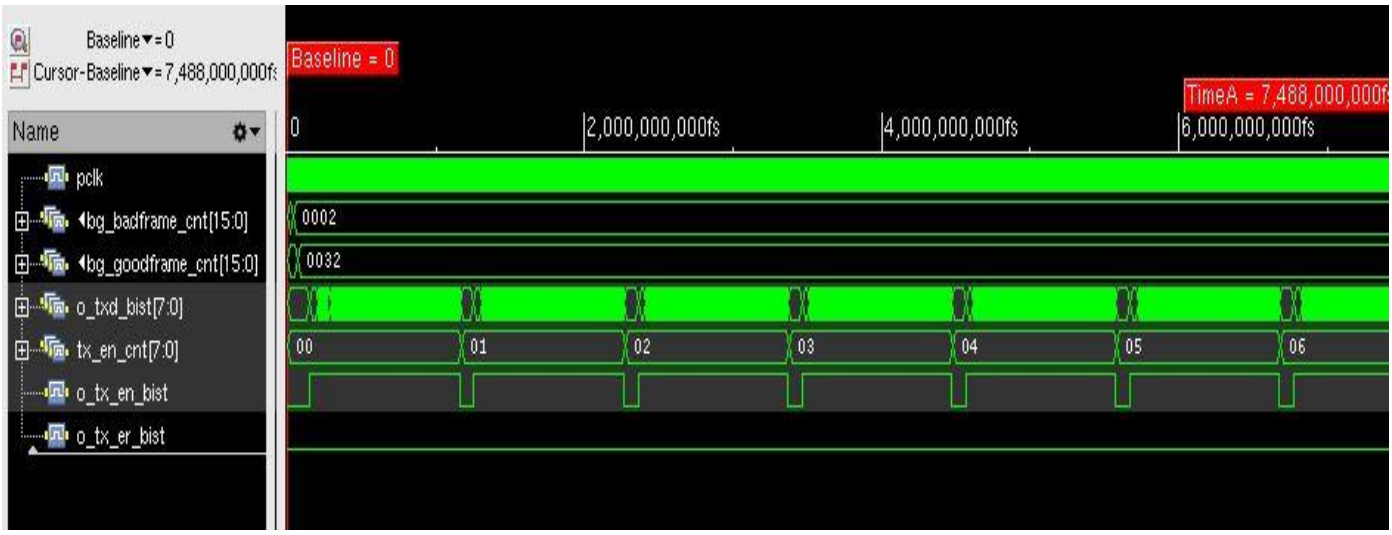


Fig 5.10 waveform 5

In waveform 5, we can see the outputs transmission enable and error signals. As the number of frames are sent, count keeps increasing. And frame is valid only if error is zero and enable is high.

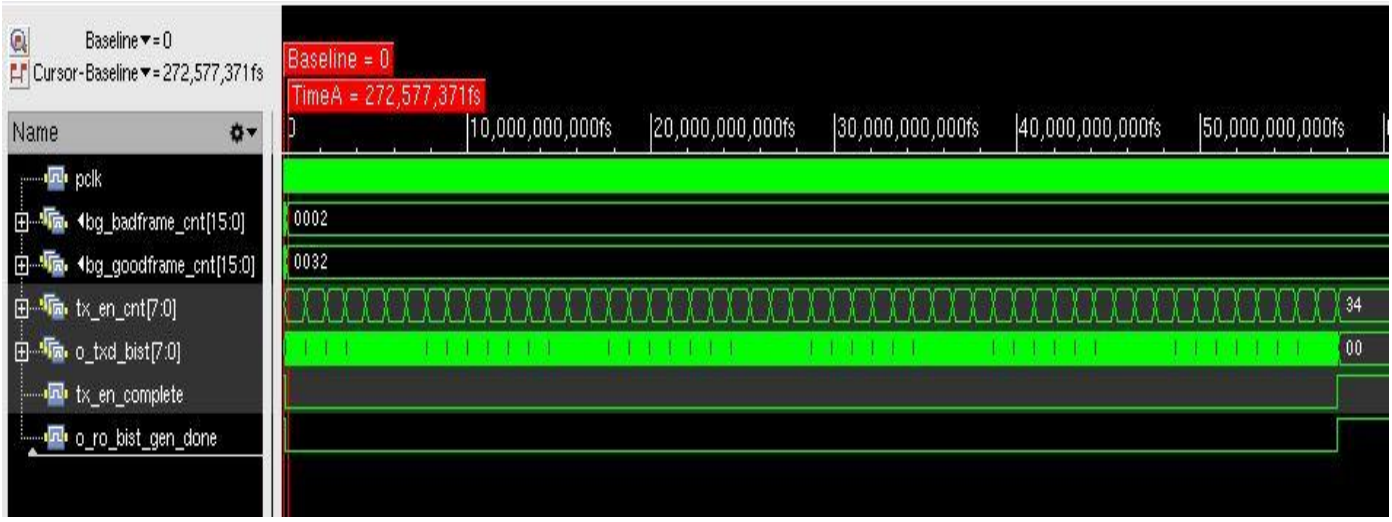


Fig 5.11 waveform 6

In waveform 6, when count reaches 16'h0034 (number of good frames + number of bad frames, refer waveform 1) transmission is completed and bist generator status is updated, bist_gen_done goes high.

Checker Inputs & Outputs:

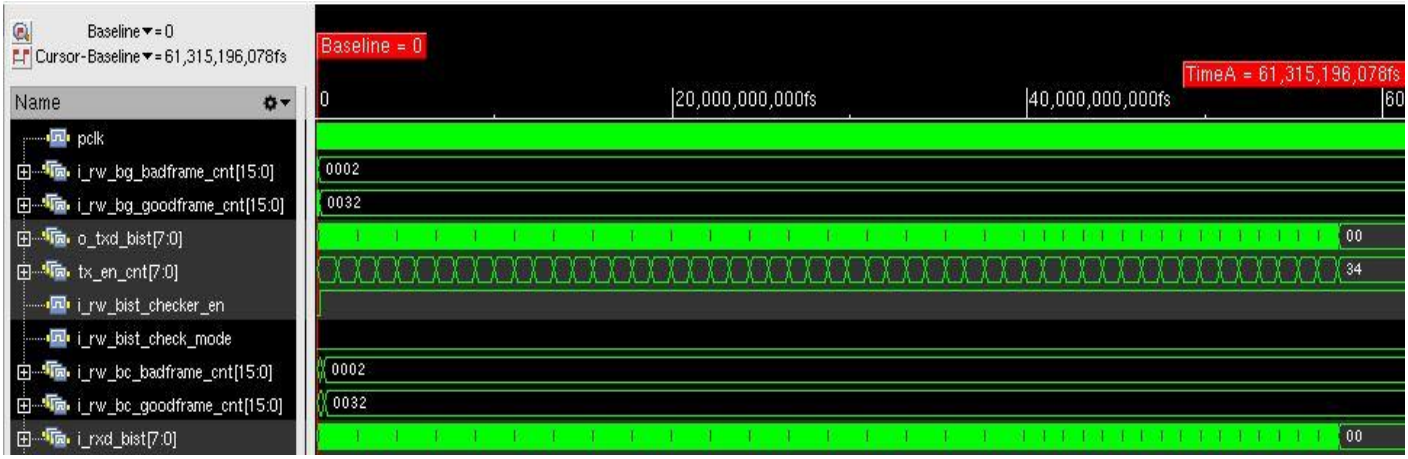


Fig 5.12 Waveform 7

In the waveform 7, checker enable can be seen high and it receives the generator output. Expected number of good frames and bad frames are also sent.
Detecting Good frames and Bad frames :

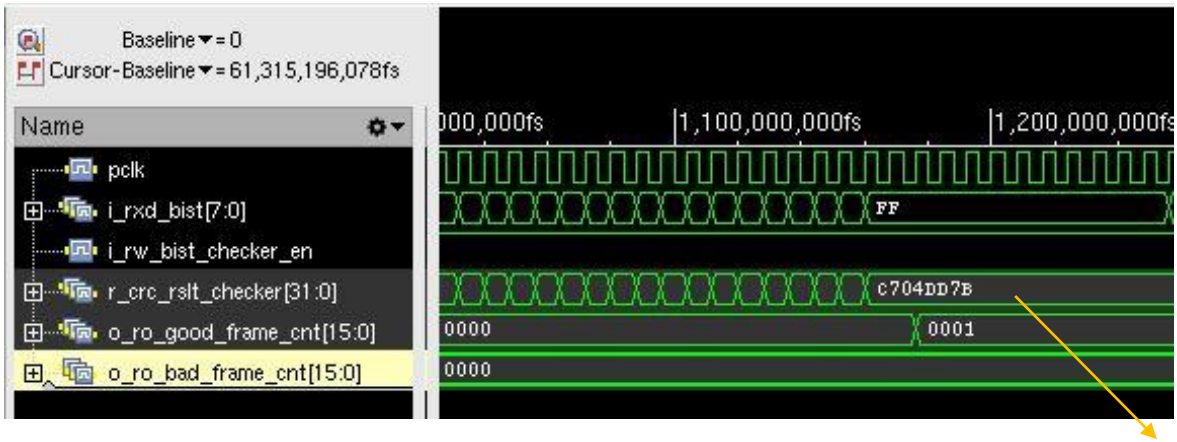


Fig 5.13 Waveform 8

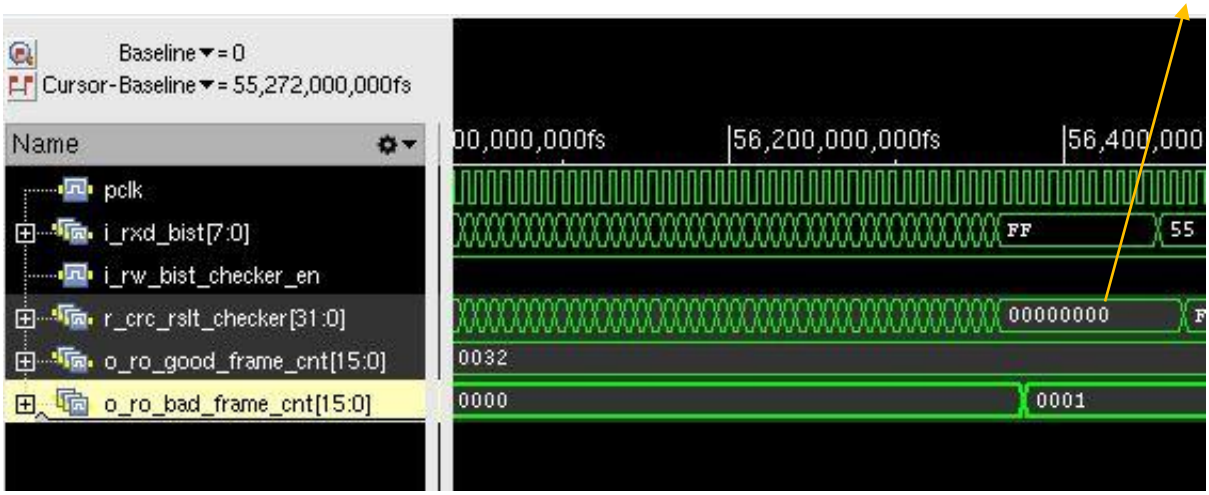


Fig 5.14 Waveform 9

In waveform 8, we can see the magic number (32'hC704DD7B, refer chapter 4). When this number is calculated then the received frame is good. In waveform 9, this magic number is not achieved so this frames comes under bad.

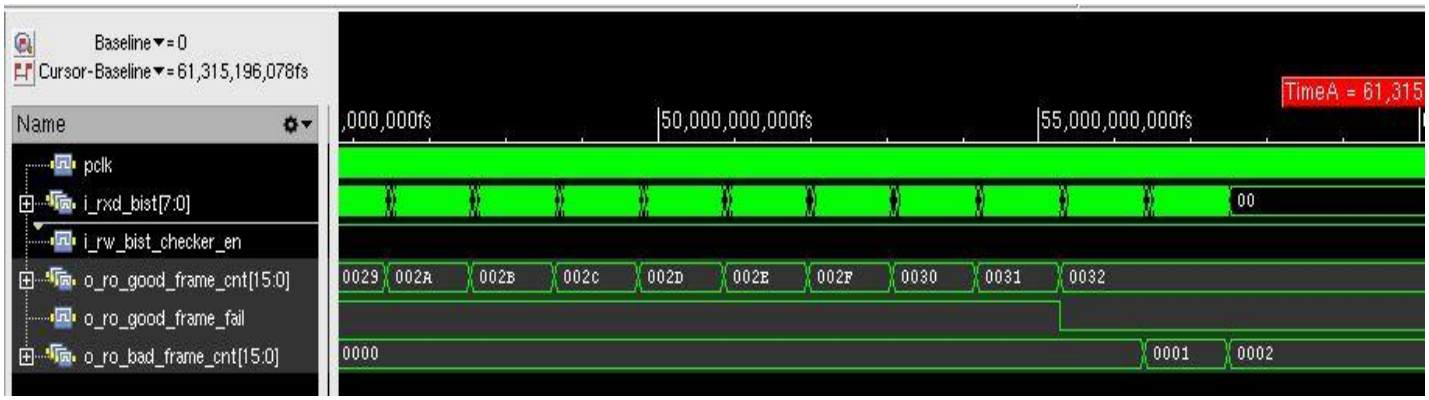


Fig 5.15 Waveform 10

In waveform 10, total number of good frames and bad frames calculated are shown. We can see that it matches with input values.

Status Registers

Fig 5.16 Waveform 11

In waveform 11, when bist checker done is high, we can see the different flag register results. As the good frame count is matched the good frame fail is low.

Similarly As the bad frame count is matched all the bad frames are properly detected by the checker so the bad_frame_fail is low.

Also, rx_dv and rx_er are also low indicating the received frames are valid with no errors.

Different data types :

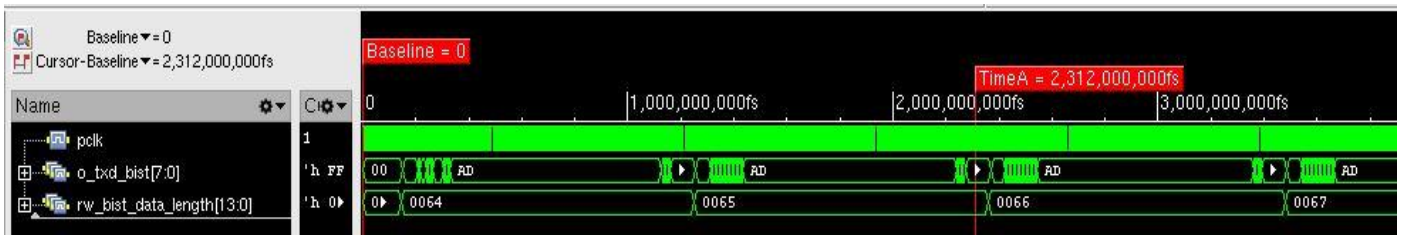


Fig 5.17 Waveform 12

In this test, constant data type is selected with incrementing data length. For every frame data length is incremented by 1. And in every frame, data is constant ie 8'h AD. We can generate different combination of such datatypes and data length and these can be verified at checker side.

Error cases :

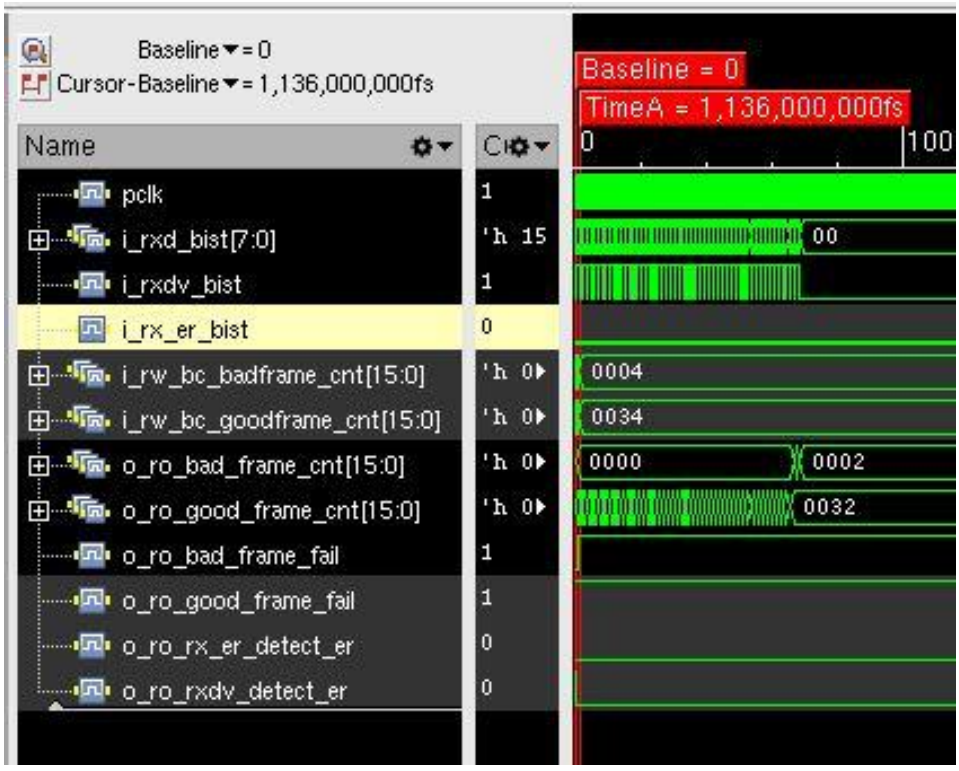


Fig 5.18 Waveform 13

In waveform 13, expected number of frames does not match with received number of frames causing an error in the test.

Bad_frame_fail and good_frame_fail goes high indicating this issue.

Validation Mode Simulation Results :

Until now, Production mode results are shown. Now the bist mode is changed to validation. Here frames are continuously sent and number of good frames and bad frames are captured.

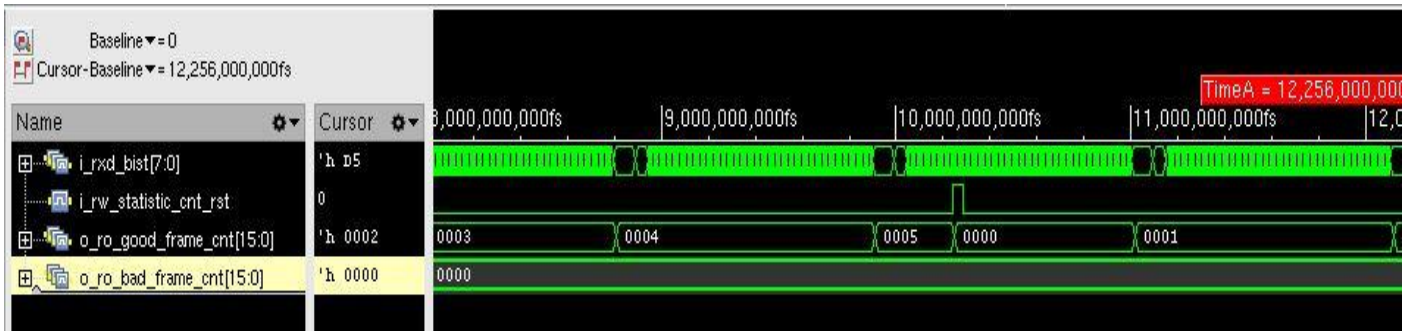


Fig 5.19 Waveform 14

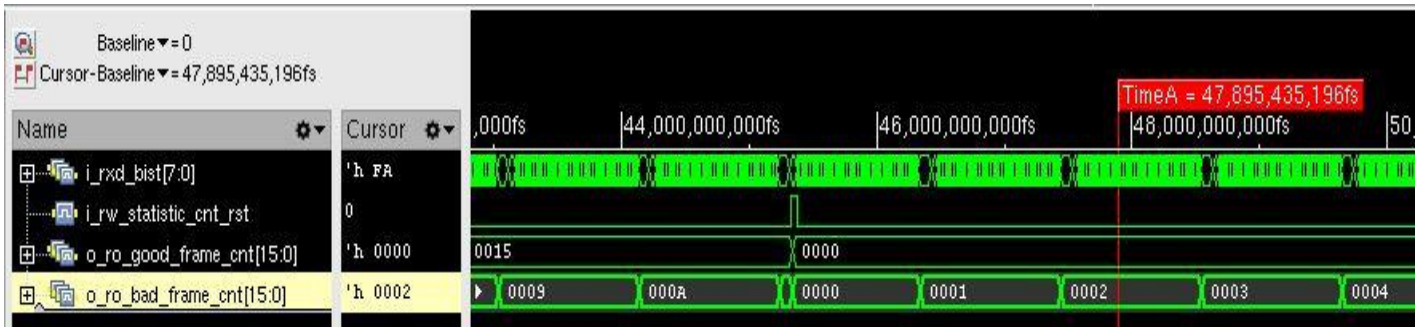


Fig 5.20 Waveform 15

In the waveforms 14 & 15, when read statistics signal is sent, then the number of good frames and bad frames until that instant are captured and count is made to zero. The counter keeps on increasing until a new read signal is sent. In validation mode only good frame and bad frame counters are checked.

Conclusion :

BIST Engine module is designed and the ethernet frames are successfully generated by the BIST generator which are verified by the BIST checker. Different status flags can be accessed from the register bank to understand which type of error is detected by the checker.

CHAPTER 6

Conclusion And Future Scope

AMBA APB protocol is clearly understood and implemented using Verilog HDL. AMBA bus architecture and APB slave working is explained in detailed. The simulated waveform has been compared with the specification mentioned in [6] and the results match. So this module can be used in any design whose registers are to be accessed by an AMBA architecture Processor or a master.

An interface is designed and it is successful in accessing the registers of the PHY chips which doesn't support the AMBA bus structure.

Different media independent interfaces are explained and understood. The best MII interface which can be taken is SGMII interface with 8 pin outputs. SGMII interface is by far the best interface in terms of clock speed and number of pins. 32 bit Cyclic Redundancy checksum can be used to verify the received frames after transmitting over data paths. A successful design is developed using Verilog RTL to find the silicon bugs using BIST engine module. Different types of frames are generated and sent over the data path where the checker verified the data.

In future, the BIST generator Module can be modified to create continuous frames which will be helpful in analysing the data during thermal testing. BIST engine can also be improved to work for multiple interfaces for 100Mbps and 1000Mbps by sampling the data. Single BIST engine can be made to work for 4 output ports and 8 output ports with a simple sampling logic which reduces the requirement of having separate modules for different speeds.

REFERENCES

[1] Design & Implementation of Advance Peripheral Bus Protocol. International Journal of Scientific Engineering and Applied Science (IJSEAS) - Volume-1, Issue-3, June 2015.

[2] The 8-bit parallel CRC-32 research and Implementation in USB 3.0, 2012 International Conference on Computer Science and Service System

[3] KOU Ke-nan, RU Xiao, SUN Zhi-yue, REN Guang-hui "Research on parallel CRC32 algorithm based on SATA interface", The Techniques of Automation &Application, vol. 29, pp. 40-43,August 2010.

[4] LI You-mou,"The design of the 8bits parallel CRC-32 soft-core in Ethernet",Journal of Xi'an University of Post and Telecommunication, vol.11, pp.32-35, sep. 2006

[5] International Journal of Computer Applications (0975 – 8887) Volume 95– No.21, June 2014 29 Design and Verification of AMBA APB Protocol

[6] ARM, "AMBA Specification Overview", http://www.arm.com/. .

[7] Samir Palnitkar, "Verilog HDL: A guide to Digital Design and Synthesis (2nd Edition), Pearson, 2008.

[8] Santhi Priya Sarekokku, K. Rajasekhar, "Design and Implementation of APB Bridge based on AMBA AXI 4.0," IJERT, Vol.1, Issue 9, Nov 2012.

[9] Design & Implementation of Advance Peripheral Bus Protocol International Journal of Scientific Engineering and Applied Science (IJSEAS) - Volume-1, Issue-3, June 2015 ISSN: 2395-3470 www.ijseas.com