System for Communication between SPI and AMBA AHB

[1]Gandhigude Lankesh, [2] Rahul Pal
[1]M.Tech Student, [2]Assistant Professor
Department of Electronics and Communication Engineering, Manipal University Jaipur, Rajasthan, India
[1]lankesh.lg@gmail.com, [2]rahul.pal@jaipur.manipal.edu

***Abstract* - Modern computer system rely more and more on - chip communication protocol to exchange data. System-on-chip (SoC) designs use bus protocols for high performance data transfer among the Intellectual Property (IP) cores. AMBA AHB protocol incorporate advanced features such as pipelining burst and split transfers. Serial Peripheral Interface or SPI bus is a synchronous serial data link, a de facto standard, named by Motorola, that operates in full duplex mode. It is used for short distance, single master communication**

I. INTRODUCTION

The Advanced Microcontroller Bus Architecture (AMBA) is used as the on-chip bus in system-on-a-chip (SoC) designs. Since its inception, the scope of AMBA has gone far beyond microcontroller devices, and is now widely used on a range of ASIC and SoC parts including applications processors used in modern portable mobile devices like Smartphone.

AMBA is a registered trademark of ARM Limited, and is an open standard, on-chip interconnect specification for the connection and management of functional blocks in a System-on-Chip (SoC). It facilitates right-first-time development of multi-processor designs with large numbers of controllers and peripherals.

AHB is a new generation of AMBA bus which is intended to address the requirements of high performance synthesizable designs. AMBA AHB is a new level of bus which implements the features required for high-performance, high clock frequency systems including:

- Burst transfers
- Split transactions
- Non-tristate implementation

The Serial Peripheral Interface or SPI bus is a synchronous serial data link, a de facto standard, named by Motorola, that operates in full duplex mode.

Devices communicate in master/slave mode where the master device initiates the data frame. Multiple slave devices are allowed with individual slave select lines. Sometimes SPI is called a *four-wire* serial bus, contrasting with three, two and one wire serial buses. SPI is often referred to as SSI (Synchronous Serial Interface).

Many peripheral devices use SPI protocol and AHB is a system protocol used by most SOCs. Hence the communication between these two protocols would be very important and useful across lot of chips. This project addresses that need.

The design of the system includes decoder and asynchronous FIFO. The design is done by using Verilog coding, environment is created using System Verilog code. The verification of the communication system is by QuestaSim tool.

II. METHODOLOGY

Block Diagram

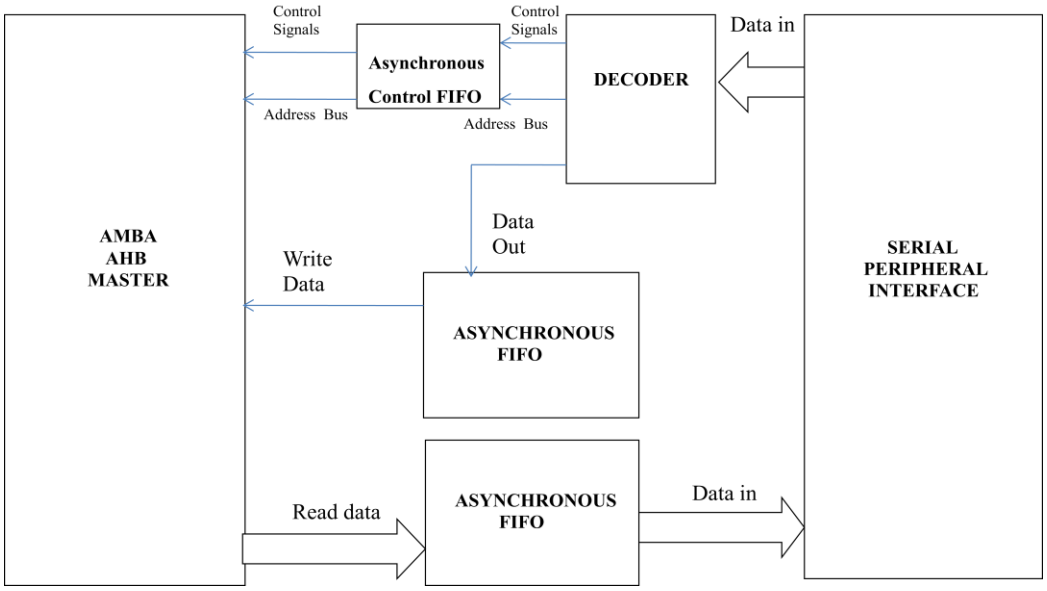

Fig 1 Block Diagram

AHB Master

An AHB bus master has the most complex bus interface in an AMBA system. Typically an AMBA system designer would use predesigned bus masters and therefore would not need to be concerned with the detail of the bus master interface.

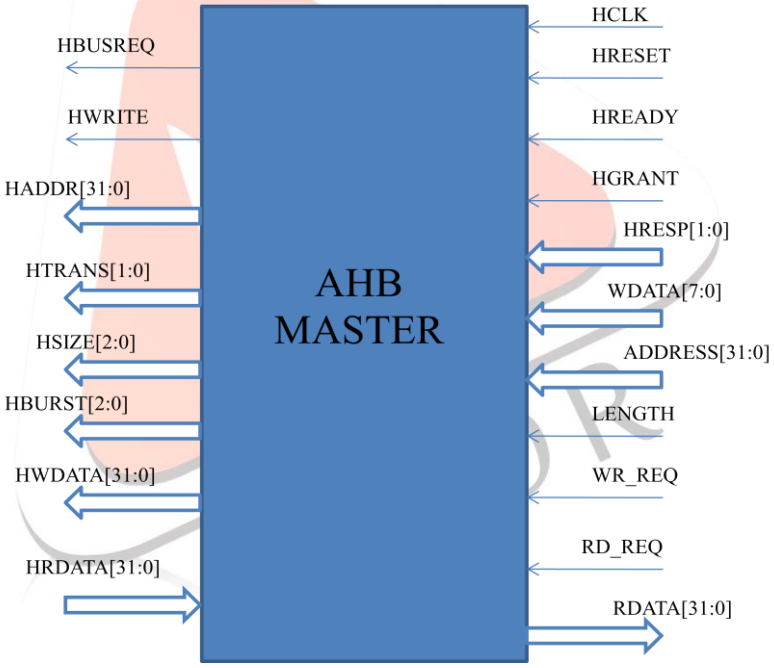

Fig 2 AHB Master

SPI Slave

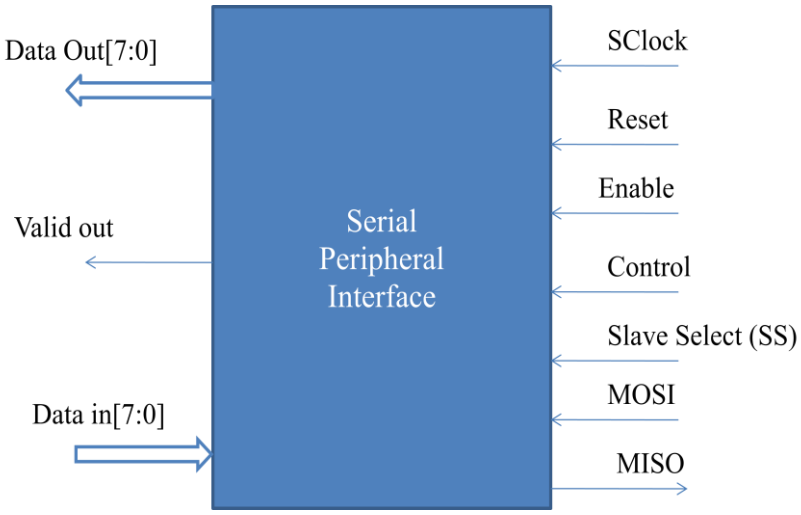

Fig 3 Block Diagram of SPI slave

The serial interface consists of slave select lines, serial clock lines, as well as input and output data lines. All transfers are full duplex transfers of a programmable number of bits per transfer (up to 64 bits). Compared to the SPI protocol, this core has some additional functionality.

It can drive data to the output data line in respect to the falling (SPI compliant) or rising edge of the serial clock, and it can latch data on an input data line on the rising (SPI compliant) or falling edge of a serial clock line.

It also can transmit (receive) the MSB first (SPI compliant) or the LSB first. It is important to know that the Rx and Tx registers share the same flip-flops, which means that what is received from the input data line in one transfer will be transmitted on the output data line in the next transfer if no write access to the Tx register is executed between the transfers.

SPI – Slave contains 4 signals

- MISO: Master In Slave Out
- MOSI: Master Out Slave In
- SCLK: Serial Clock
- SS: Slave Select

MISO: It is one of the two lines that transfer serial data in one direction with most significant bit sent first.
MOSI: It is used to display the output which is received from AHB master.
SCLK: It is used to synchronize data movement both in and out of the device through its MOSI and MISO lines. It is generated by master.
SS: It is used to select slave devices. It has to be low prior to data transaction and must stay low for the duration of the transaction.

Decoder

Decoders are common enough that we want to encapsulate them and treat them as an individual entity. A decoder is a device which does the reverse operation of an encoder, undoing the encoding so that the original information can be retrieved. It is a combinational circuit that converts binary information from n input lines to a maximum output lines.

First 8 bits are control signals, next 4 bytes are address signals and later input data will be decoded in the decoder. During 6th cycle in which we are transferring the input data a latency is been created by decoder. It is because to conform whether the command is read or write.

A decoder block provides abstraction:

- It makes diagrams simpler by hiding the internal circuitry.
- It simplifies hardware reuse. You don't have to keep rebuilding the decoder from scratch every time you need it.

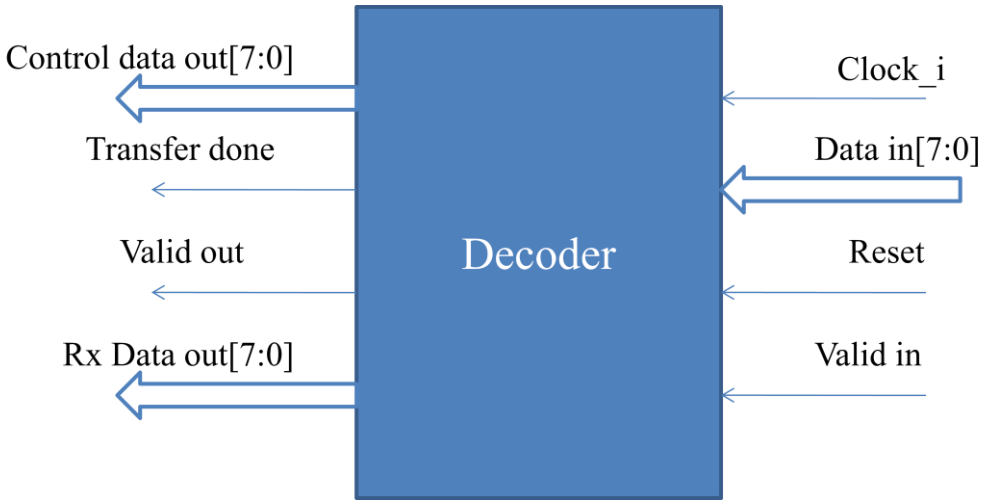

Fig 4 Block Diagram of Decoder

Asynchronous FIFO

The Asynchronous FIFO is a First-In-First-Out memory queue with control logic that performs management of the read and write pointer, generation of status flags, and optional handshake signals for interfacing with the user logic. The individual read and write ports are fully synchronous (all operations qualified by a rising clock edge), but this FIFO does not require the read and write clocks to be synchronized to each other.

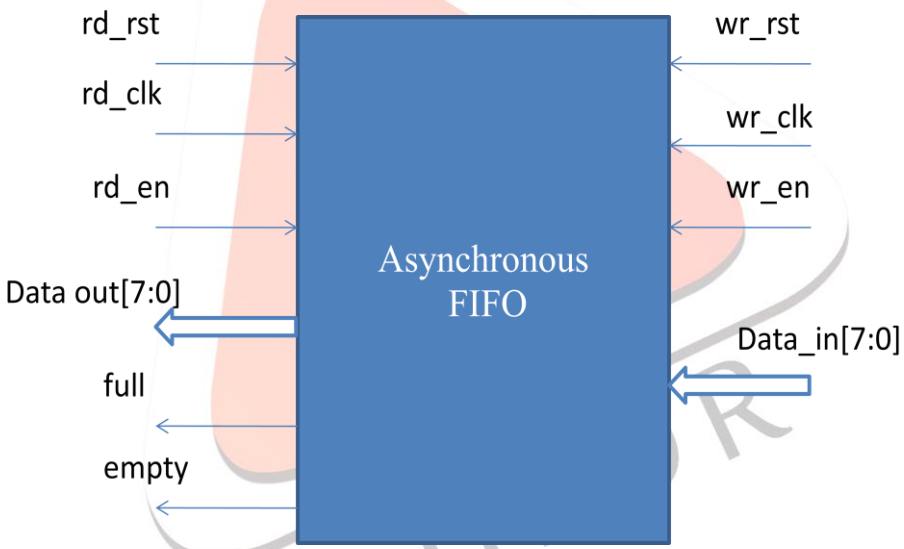

Fig 5 Block Diagram of Asynchronous FIFO

In order to understand FIFO design, one needs to understand how the FIFO pointers work. The write pointer always points to the next word to be written therefore, on reset, both pointers are set to zero, which also happens to be the next FIFO word location to be written. On a FIFO-write operation, the memory location that is pointed to by the write pointer is written, and then the write pointer is incremented to point to the next location to be written.

Similarly, the read pointer always points to the current FIFO word to be read. Again on reset, both pointers are reset to zero, the FIFO is empty and the read pointer is pointing to invalid data (because the FIFO is empty and the empty flag is asserted).

As soon as the first data word is written to the FIFO, the write pointer increments, the empty flag is cleared, and the read pointer that is still addressing the contents of the first FIFO memory word, immediately drives that first valid word onto the FIFO data output port, to be read by the receiver logic. The fact that the read pointer is always pointing to the next FIFO word to be read means that the receiver logic does not have to use two clock periods to read the data word. If the receiver first had to increment the read pointer before reading a FIFO data word, the receiver would clock once to output the data word from the FIFO, and clock a second time to capture the data word into the receiver. That would be needlessly inefficient.

III. RESULT ANALYSIS

AHB Master

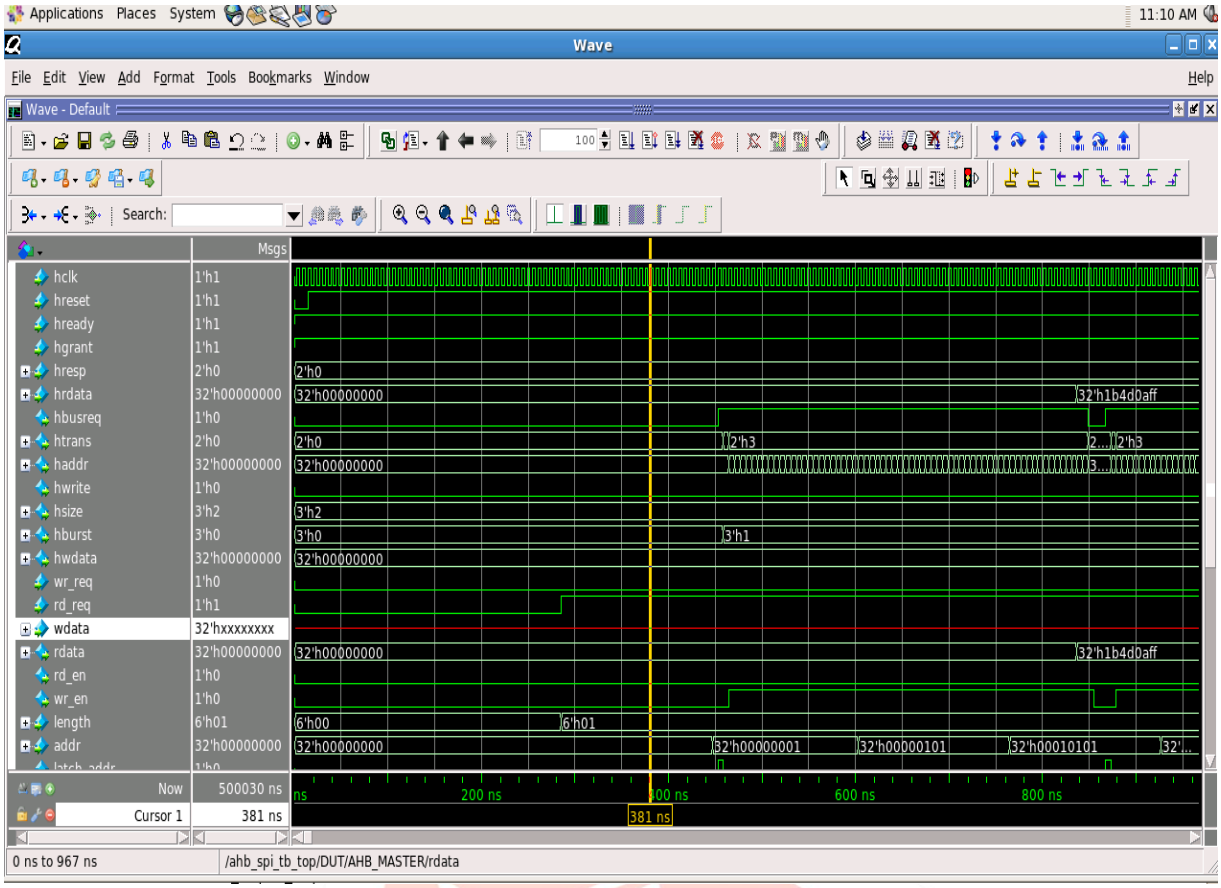

Fig 6 AHB Master Waveform

Serial Peripheral Interface

The serial interface consists of slave select lines, serial clock lines, as well as input and output data lines. All transfers are full duplex transfers of a programmable number of bits per transfer (up to 64 bits).

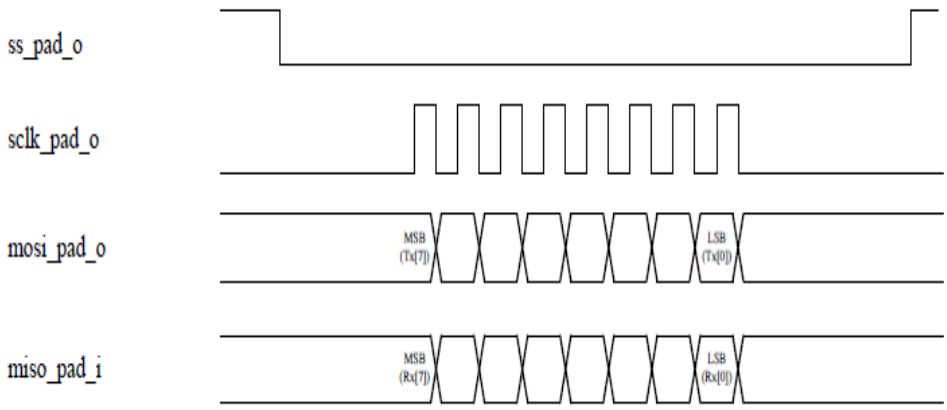

Fig 7 Wave form of SPI

The waveform for SPI slave design is shown below using QuestaSim with verilog coding.

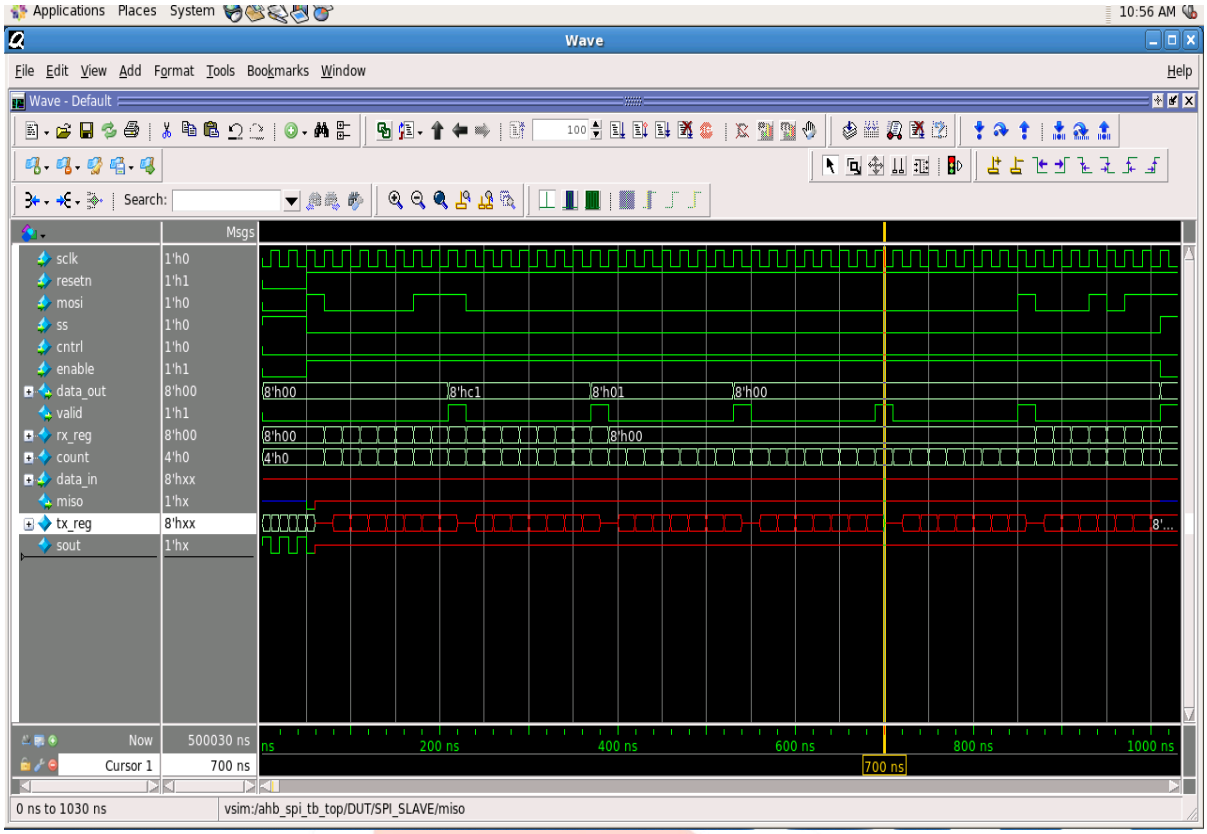

Fig 8 Wave form of SPI using QuestaSim

Decoder

The waveform for the decoder is shown below using verilog coding and QuestaSim

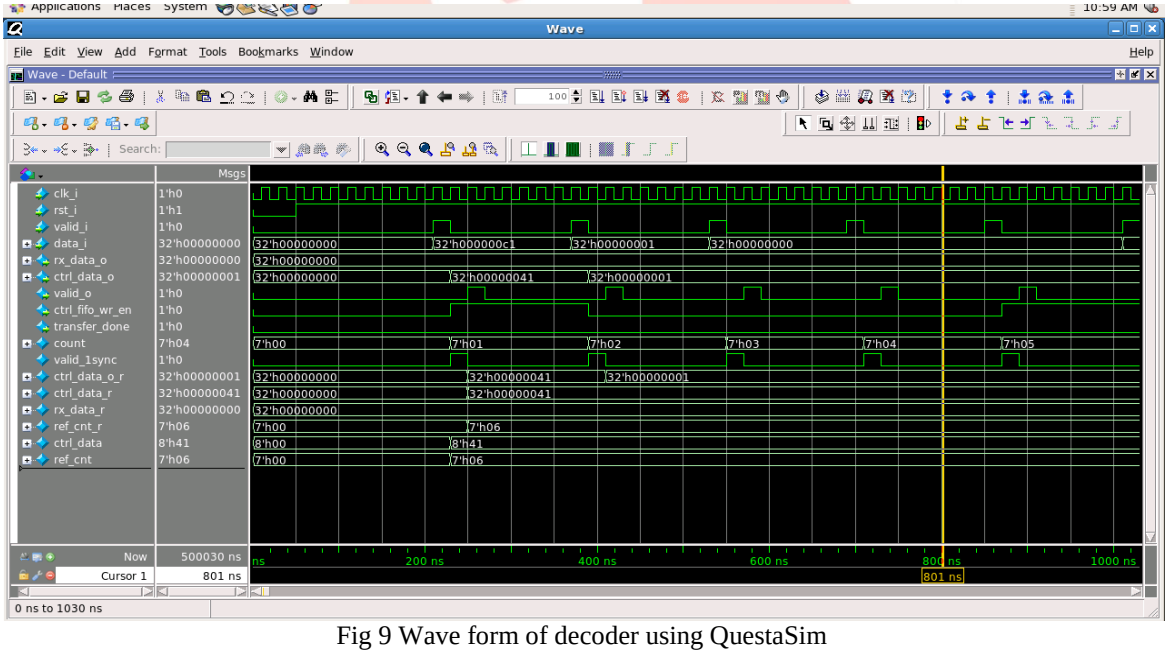

Fig 9 Wave form of decoder using QuestaSim

Asynchronous FIFO

The waveform for the asynchronous FIFO is shown below using verilog coding and QuestaSim

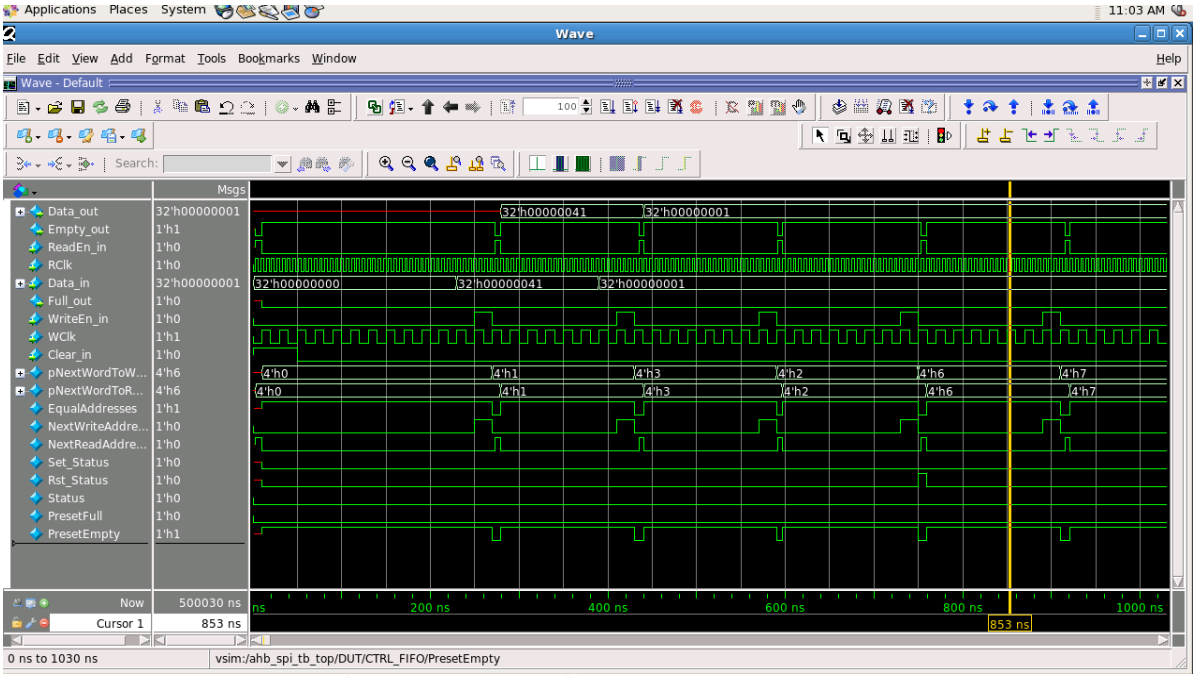

Fig 10 Asynchronous FIFO waveform using QuestaSim

IV. CONCLUSION

The modules are designed using Verilog coding and verified using System Verilog and Mentor Graphics tool QuestaSim. The proper, high performance and lossless communication is performed.

REFERENCES

Journal / Conference Papers

[1] Copyright ARM Limited 13th May 1999, "AMBA Specification", Volume- ARM IHI 0011A

[2] Copyright © 2001, 2006 ARM Limited. All rights reserved." AMBA 3 AHB-Lite Protocol"

[3] Copyright © 2009 ORSoC, "Versatile FIFO"

[4] Open Cores SPI Master Core Specification

[5] Clifford E. Cummings, "Synthesis and Scripting Techniques for Designing Multi-Asynchronous Clock Designs," SNUG 2001 (Synopsys Users Group Conference, San Jose, CA, 2001) User Papers, March 2001, Section MC1, 3rd paper. Also available at www.sunburst-design.com/papers

[6] Clifford E. Cummings and Don Mills, "Synchronous Resets? Asynchronous Resets? I am so confused! How will I ever know which to use?," SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers, March 2002, Section TB2, 1st paper. Also available at www.sunburst-design.com/papers

[7] Clifford E. Cummings and Peter Alfke, "Simulation and Synthesis Techniques for Asynchronous FIFO Design with Asynchronous Pointer Comparisons," SNUG 2002 (Synopsys Users Group Conference, San Jose, CA, 2002) User Papers, March 2002, Section TB2, 3rd paper. Also available at www.sunburst-design.com/papers

[8] Dinesh Tyagi, former CAE Manager for Synopsys Design Ware product, personal communication

[9] Edward Paluch, personal communication

[10] Frank Gray, "Pulse Code Communication." United States Patent Number 2,632,058. March 17, 1953.

[11] John O'Malley, *Introduction to the Digital Computer*, Holt, Rinehart and Winston, Inc., 1972, pg. 190.

[12] Steve Golson, personal communication

[13] Synopsys SolvNet, Doc Name: DesignWare-110.html, "Functional Bugs in Design Ware Components," Updated: 11/30/2000

Reference / Hand Books

[14] "Verilog HDL: A Guide to Digital Design and Synthesis 2nd Edition" by Samir Palnitkar

[15] SystemVerilog for Verification: A Guide to Teach the Testbench Language Features Third Edition"

Web

[16] "Asynchronous FIFO" http://www.orsoc.se

[17] "AMBA AHB" http://www.arm.com

[18] "Serial Peripheral Interface" http://www.opencores.com