Ms. Anusha Ranga , Mr. L. Hari Venkatesh , Mr.Venkanna / International Journal of Engineering Research and Applications (IJERA) ISSN: 2248-9622 www.ijera.com
Vol. 2, Issue4, July-August 2012, pp.1102-1106

Design And Implementation Of AMBA-AXI Protocol Using VHDL For Soc Integration

Ms. Anusha Ranga*, Mr. L. Hari Venkatesh, Mr.Venkanna*****

* (M.Tech II year (VLSI System Design), ECE Department, St. Mary's Institute of Tech. Hyderabad)
** (Field Application Engineer, Nu Horizons Electronics Asia pvt ltd)

*** (Professor and HOD, ECE Department, St. Mary's Institute of Tech., Hyderabad)

Abstract

System-on-a-Chip (SoC) design has become more and more complexly. Because difference functions components or IPs (Intellectual Property) will be integrated within a chip. The challenge of integration is "how to verify on-chip communication properties". Although traditional simulation-based on-chip bus protocol checking bus signals to obey bus transaction behavior or not, however, they are still lack of a chip-level dynamic verification to assist hardware debugging. We proposed a rule based synthesizable AMBA AXI protocol checker. The AXI protocol checker contains 44 rules to check on-chip communication properties accuracy. In the verification strategy, we use the Model sim to verify AXI protocol checker.

***Keywords:** AMBA AXI, Verilog, VLSI, FPGA*

I. INTRODUCTION

In recent years, the improvement of the semiconductor process technology and the market requirement increasing. More difference functions IPs are integrated within a chip. Maybe each IPs had completed design and verification. But the integration of all IPs could not work together. The more common problem is violation bus protocol or transaction error. The bus-based architecture has become the major integrated methodology for implementing a SoC. The on-chip communication specification provides a standard interface that facilitates IPs integration and easily communicates with each IPs in a SoC. To speed up SoC integration and promote IP reuse, several bus-based communication architecture standards have emerged over the past several years. Since the early 1990s, several on chip bus-based communication architecture standards have been proposed to handle the communication needs of emerging SoC design. Some of the popular standards include ARM Microcontroller Bus Architecture (AMBA) versions 2.0 and 3.0, IBM Core Connect, STMicroelectronics STBus, Sonics SMARRT Interconnect, Open Cores Wishbone, and Altera Avalon [1]. On the other hand, the designers just integrate their owned IPs with third party IPs into the SoC to significantly reduce design cycles. However, the main issue is that how to efficiently make sure the IP functionality, that works correctly after integrating to the corresponding bus architecture.

There are many verification works based on formal verification techniques [2]-[6]. Device under test (DUT) is modeled as finite-state transition and its properties are written by using computation tree logic (CTL) [7], and then using the verification tools is to verify DUT's behaviors [8]. Although formal verification can verify DUT's behaviors thoroughly, but here are still unpredictable bug in the chip level, which we want to verify them.

The benefits of using rule-based design include improving observability, reducing debug time, improving integration through correct usage checking, and improving communication through documentation. In the final purpose, increasing design quality while reducing the time-to-market and verification costs. We anticipate that the AMBA AXI protocol checking technique will be more and more important in the future. Hence, we propose a synthesizable AMBA AXI protocol checker with an efficient verification mechanism based on rule checking methodology. There are 44 rules to check the AMBA AXI protocol that provide AXI master, slave, and default slave protocol issues.

II.AMBA AXI architecture

AMBA AXI [3] supports data transfers up to 256 beats and unaligned data transfers using byte strobes. In AMBA AXI4 system 16 masters and 16 slaves are interfaced. Each master and slave has their own 4 bit ID tags. AMBA AXI system consists of master, slave and bus.

The system consists of five channels namely write address channel, write data channel, read data channel, read address channel, and write response

channel. The AXI4 protocol supports the following mechanisms:

- Unaligned data transfers and up-dated write response requirements.
- Variable-length bursts, from 1 to 16 data transfers per burst.
- A burst with a transfer size of 8, 16, 32, 64, 128, 256, 512 or 1024 bits wide is supported.
- Updated AWCACHE and ARCACHE signaling details

Each transaction is burst-based which 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. Table 1[3] gives the information of signals used in the complete design of the protocol.

The write operation process starts when the master sends an address and control information on the write address channel as shown in fig 1. The master then sends each item of write data over the write data channel. The master keeps the VALID signal low until the write data is available. The master sends the last data item, the WLAST signal goes HIGH.

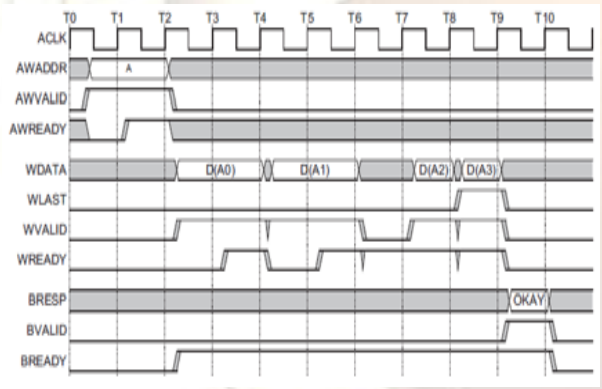

Fig 1: Write address and data burst.

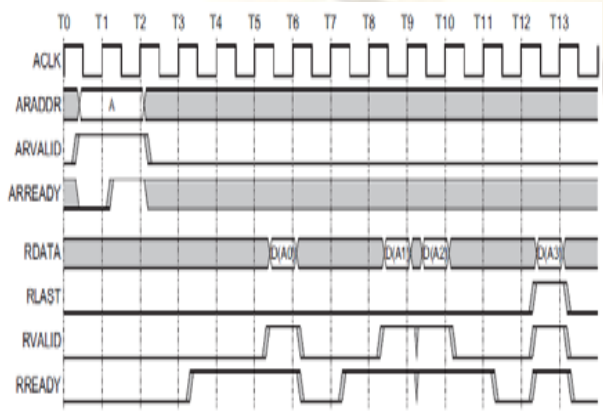

Fig 2: Read address and data burst.

When the slave has accepted all the data items, it drives a write response signal BRESP[1:0] back to the master to indicate that the write transaction is complete. This signal indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.

Table 1: Signal descriptions of AMBA AXI4 protocol.

Signal	Source: master/ slave	Input/ Output	Description
Aclk	Global	Input	Global clock signal.
AResetn	Global	Input	Global reset signal
AWID[3:0]	Master	Input	Write address ID.
AWADDR[31:0]	Master	Input	Write address.
AWLEN[3:0]	Master	Input	Write burst length.
AWSIZE[2:0]	Master	Input	Write burst size.
AWBURST[1:0]	Master	Input	Write burst type.
AWLOCK[1:0]	Master	Input	Write lock type.
AWCACHE[3:0]	Master	Input	Write cache type.
AWPROT[2:0]	Master	Input	Write protection type.
WDATA[31:0]	Master	Input	Write data.
ARID[3:0]	Master	Input	Read address ID.
ARADDR[31:0]	Master	Input	Read address.
ARLEN[3:0]	Master	Input	Read Burst length.
ARSIZE[2:0]	Master	Input	Read Burst size.
ARLOCK[1:0]	Master	Input	Read Lock type.
ARCACHE[3:0]	Master	Input	Read Cache type.
ARPROT[2:0]	Master	Input	Read Protection type.
RDATA[31:0]	Master	Input	Read data.
WLAST	Master	Input	Write last.
RLAST	Slave	Output	Read last.
AWVALID	Master	Output	Write address valid.
AWREADY	Slave	Output	Write address ready.
WVALID	Master	Output	Write valid.
RAVLID	Slave	Output	Read valid.
WREADY	Slave	Output	Write ready.
BID[3:0]	Slave	Output	Write Response ID.
RID[3:0]	Slave	Output	Read response ID.
BRESP[1:0]	Slave	Output	Write response.
RRESP[1:0]	Slave	Output	Read response.
BVALID	Slave	Output	Write response valid.
BREADY	Master	Output	Response ready.
RVALID	Slave	Output	Read valid.

After the read address appears on the address bus, the data transfer occurs on the read data channel as shown in fig. 2. The slave keeps the VALID signal

LOW until the read data is available. For the final data transfer of the burst, the slave asserts the RLAST signal to show that the last data item is being transferred. The RRESP[1:0] signal indicates the status of the read transfer. The allowable responses are OKAY, EXOKAY, SLVERR, and DECERR.

The protocol supports 16 outstanding transactions, so each read and write transactions have ARID[3:0] and AWID [3:0] tags respectively. Once the read and write operation gets completed the module produces a RID[3:0] and BID[3:0] tags. If both the ID tags match, it indicates that the module has responded to right operation of ID tags. ID tags are needed for any operation because for each transaction concatenated input values are passed to module

III. RELATED WORK

In a SoC, it houses many components and electronic modules; to interconnect these, a bus is necessary. There are many buses introduced in the due course some of them being AMBA [2] developed by ARM, CORE CONNECT [4] developed by IBM, WISHBONE [5] developed by Silicore Corporation, etc. Different buses have their own properties the designer selects the bus best suited for his application. The AMBA bus was introduced by ARM Ltd in 1996 which is a registered trademark of ARM Ltd. Later advanced system bus (ASB) and advanced peripheral bus (APB) were released in 1995, AHB in 1999, and AXI in 2003[6]. AMBA bus finds application in wide area. AMBA AXI bus is used to reduce the precharge time using dynamic SDRAM access scheduler (DSAS) [7]. Here the memory controller is capable of predicting future operations thus throughput is improved. Efficient Bus Interface (EBI) [8] is designed for mobile systems to reduce the required memory to be transferred to the IP, through AMBA3 AXI. The advantages of introducing Network-on-chip (NoC) within SoC such as quality of signal, dynamic routing, and communication links was discussed in [8]. To verify on-chip communication properties rule based synthesizable AMBA AXI protocol checker is used. The block diagram of the AXI Generic Mater Controller has the following blocks and is shown in the fig 3 :

1) Master
2) AMBA AXI4 Interconnect
3) Slave

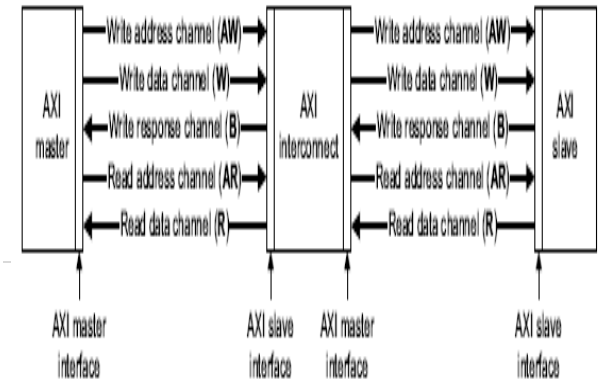

Fig 3: The Block Diagram of AXI Generic Master

Controller

The master is connected to the interconnect using a slave interface and the slave is connected to the interconnect using a master interface as shown in fig 4. The AXI4 master gets connected to the AXI4 slave interface port of the interconnect and the AXI slave gets connected to the AXI4 Master Interface port of the interconnect. The parallel capability of this interconnects enables master M1 to access one slave at the same as master M0 is accessing the other.

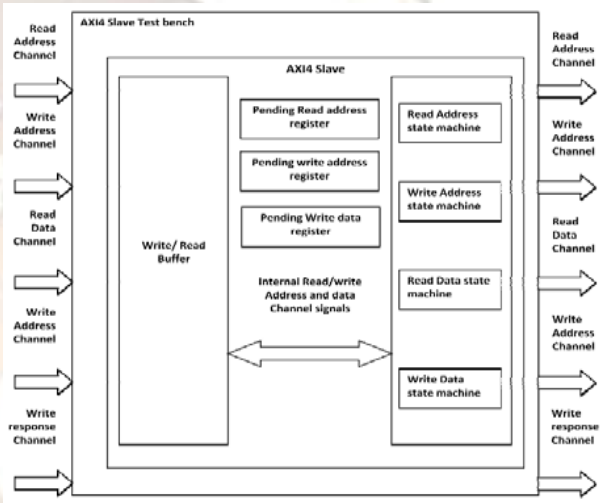

Fig 4: AMBA AXI slave Read/Write block Diagram

IV. SIMULATION

Simulation is being carried out on ModelSim Quartus II which is trademark of Menter Graphics, using Verilog as programming language. The test case is run for multiple operations and the waveforms are visible in discovery visualization environment

IV.i. Simulation inputs

To perform multiple write and read operations, the concatenated input format and their values passed to invoke a function is shown in the fig 6 and 7 respectively. Here the normal type of the burst is passed to module. Internal_lock value is 0, internal_burst value is 1 and internal_prot value is 1, for both read and write operations, which indicate that the burst is of normal type. For write operation address locations passed to module are 40, 12, 35, 42 and 102; for read operations 45, 12, 67 and 98.

IV.ii. Simulation outputs

The simulation output signals generated are as follows:

- From input side the validating signals AWVALID/ARVALID signals are generated by interconnect which gives the information about valid address and ID tags.
- For write operations BRESP[1:0] response signal generated from slave indicates the status of the write transaction. The allowable responses are OKAY, EXOKAY, SLERR, and DECERR.
- For read operations RLAST signal is raised by slave for every transaction which indicates the completion of operation

V. RESULTS

Simulation is carried out in Modelsim tool and Verilog is used as programming language.

V.i. **Simulation result for write operation**

The AResetn signal is active low. Master drives the address, and the slave accepts it one cycle later. The write address values passed to module are 40, 12, 35, 42 and 102 as shown in fig 5 and the simulated result for single write data operation is shown in fig 6. Input AWID[3:0] value is 11 for 40 address location, which is same as the BID[3:0] signal for 40 address location which is identification tag of the write response. The BID[3:0] value is matching with the AWID[3:0] value of the write transaction which indicates the slave is responding correctly. BRESP[1:0] signal that is write response signal from slave is 0 which indicates OKAY. Simulation result of slave for multiple write data operation is shown in fig 7.

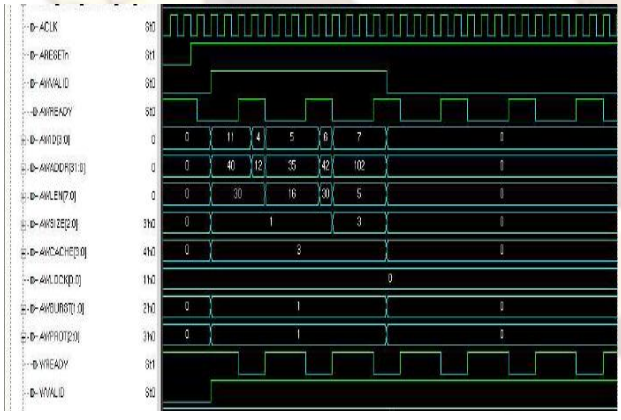

Fig 5: Simulation result of slave for write address operation

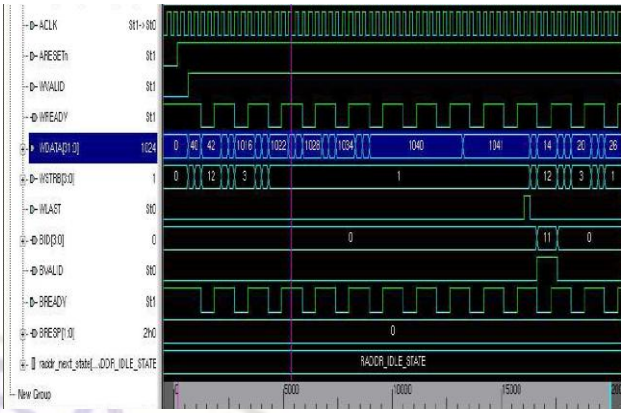

Fig 6: Simulation result of slave for single write data operation

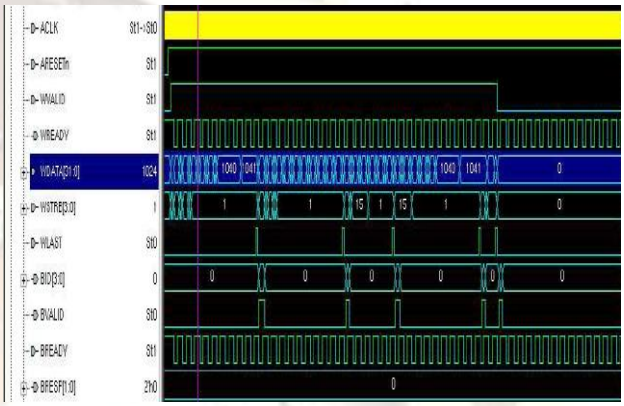

Fig 7: Simulation result of slave for multiple write data operation

V.ii. **Simulation result for read operation**

The read address values passed to module are 45, 12, 67, 98 as shown in fig 8 and the simulated result for single read data operation is shown in fig 9.

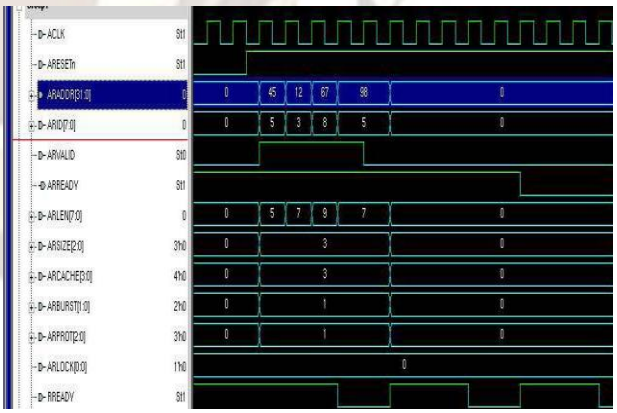

Fig 8: Simulation result of slave for read address operation

Input ARID[3:0] value is 3 for 12 address location, which is same as the RID[3:0] signal for 12 address location which is identification tag of the write

response. The RID[3:0] and ARID[3:0] values are matching, which indicates slave has responded properly.

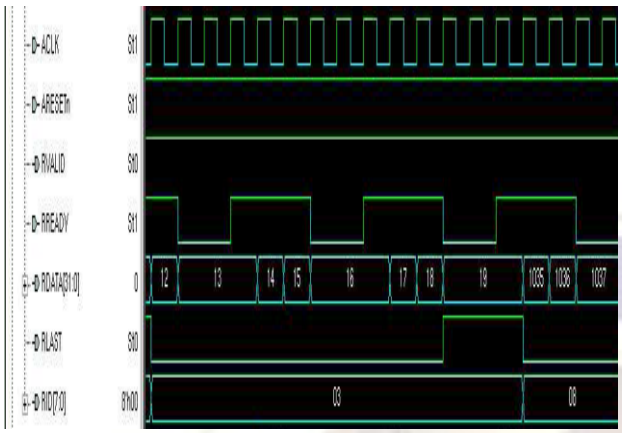

Fig 9: Simulation result of slave for single read data operation

VI. CONCLUSION AND FUTURE SCOPE

VI.i. **Future scope**

The AMBA AXI4 has limitations with respect to the burst data and beats of information to be transferred. The burst must not cross the 4k boundary. Bursts longer than 16 beats are only supported for the INCR burst type. Both WRAP and FIXED burst types remain constrained to a maximum burst length of 16 beats. These are the drawbacks of AMBA AXI system which need to be overcome.

VI.ii**. Conclusion**

AMBA AXI4 is a plug and play IP protocol released by ARM, defines both bus specification and a technology independent methodology for designing, implementing and testing customized high-integration embedded interfaces. The data to be read or written to the slave is assumed to be given by the master and is read or written to a particular address location of slave through decoder. In this work, slave was modeled in Verilog with operating frequency of 100MHz and simulation results were shown in Modelsim tool. To perform single read operation it consumed 160ns and for single write operation 565ns.

REFERENCES

[1] Shaila S Math, Manjula R B, "*Survey of system on chip buses based on industry standards*", Conference on Evolutionary Trends in Information Technology(CETIT), Bekgaum,Karnataka, India, pp. 52, May 2011

[2] ARM, AMBA Specifications ev2.0). [Online]. Available at http://www.arm.com,

[3] ARM, AMBA AXI Protocol Specification (Rev 2.0).

[4] IBM, Core connect bus architecture. IBM Microelectronics.[Online].Available: http://www.ibm.com/chips/products/coreconnect

[5] Silicore Corporation, Wishbone system-on-chip (soc) interconnection architecture for portable ip cores,

[6] ARM, AMBA AXI protocol specifications, Available at, http://www.arm.com, 2003

[7] Jun Zheng, Kang Sun , Xuezeng Pan, and Lingdi Ping "*Design of a Dynamic Memory Access Scheduler*", IEEE transl, Vol 7, pp. 20-23, 2007

[8] Na Ra Yang, Gilsang Yoon, Jeonghwan Lee, Intae Hwang, Cheol Hong Kim, Sung Woo Chung and Jong Myon Kim, "*Improving the System-on-a-Chip Performance for Mobile Systems by Using Efficient Bus Interface*", IEEE transl, International Conference on Communications and Mobile Computing, Vol 4, pp. 606-608, March 2009

AUTHORS

Ms. Anusha Ranga Pursuing M.Tech in VLSI systems from St. Mary's institute of technology and science, her area of interest is VLSI design and Embedded systems.

Mr.L.Hari Venkatesh, M.Tech in Microelectronics & VLSI Design from NIT Calicut and Working as Field Application Engineer in Nu Horizons Electronics pvt ltd, Hyderabad and his area of interest is Digital electronics and VlSI design.

Mr. Venkanna Professor and Head of the department in St. Mary's institute of technology and management. His area of interest is VLSI systems and communication theory.