ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology
Volume 1, Issue 3, May 2012

DESIGN OF AXI BUS FOR 32-BIT PROCESSOR USING BLUESPEC

T.Ananth kumar, DR.S.Saraswathi Janaki

***Abstract* — For high frequency on-chip communication architecture design, AXI bus is proposed. With the need of application and high performance, chip with a single processor can't meet the need of more and more complex computational task. We are able to integrate multiple processors on a chip, thanks to the development of integrated circuit manufacturing technology, 32 bit RISC processor which gives a solution to this requires efficient on-chip communication architectures to support high data bandwidth and increase parallelism. The traditional form of interconnection between multiple cores usually is on-chip bus (such as AHB and Avalon), which determiners the Performance of 32 bit RISC processor. However, these buses can connect from one master to another at one time, which restricts the performance of the whole system badly. The performance of 32 bit RISC processor is determined not only by the capacity of processor, but also by on-chip communication architecture. Now the speed of processor is becoming higher and higher, the performance is mainly limited by the communication architecture. In AXI bus protocol, there are five independent data transfer channels, making it translate data in high speed and efficient. As well as the data transfer protocol, the AXI protocol includes optional extensions that cover signaling for low-power operation. For developing this AXI protocol, BlueSpec System Verilog is being used for describing the hardware. For simulating and synthesizing the protocol, Xilinx and I Verilog is being used.**

***Index Terms*— Axi, Bluspec, NOC, AMBA, ARM**

I. INTRODUCTION

With the need of application, chip with a single processor cannot meet the need of more and more complex computational task. The Processor requires efficient on-chip communication architectures to support high data bandwidth and increase parallelism. The traditional form of interconnection between multiple cores usually is on-chip bus (such as AHB and Avalon), which determiners the performance of 32 bit RISC processor. However, traditional buses only allow one master to access one slave at one time, which badly restricts the performance of the whole system.

The performance of 32 bit RISC processor is determined not only by the capacity of processor, but also by on-chip communication architecture. Now the speed of processor is becoming higher and higher, the performance is mainly limited by the communication architecture. However, the significantly increased design complexity of the 32 bit RISC processor causes the unacceptable simulation time with the traditional simulation methods. Hence, it is necessary to develop an FPGA prototype, which can provide accurate performance evaluation under real application and prediction for future ASIC design. This paper focuses on the design and implementation of AXI bus based 32 bit RISC processor, which translates data in burst, maximal length of which is up to 16 transactions. Besides, it only needs to translate the head address of the burst in this transaction. Due to that feature, multiple masters accessing multiple slaves at one time becomes possible in sharing address bus architecture. Moreover, AXI protocol has five independent data transfer channels, making it translate data in high speed and efficient. As well as the data transfer protocol, the AXI protocol includes optional extensions that cover signaling for low-power operation.

II. LITERATURE SURVEY

2. AMBA Bus

The AMBA protocol is an on-chip interconnect specification which is used in functional and management blocks in System On Chip. It is an open standard protocol. Due to the following specifications - well documented and can be used without royalties, AMBA protocol is the de-facto standard for 32-bit embedded processors[6].

The objective of the AMBA specification is to facilitate right-first-time development of embedded microcontroller products with one or more CPUs, GPUs or signal processors, be technology independent, to allow reuse of IP cores, peripheral and system macro-cells across diverse IC processes encourage modular system design to improve processor independence, and the development of reusable peripheral and system IP libraries.

2.1 AMBA Protocol Specifications

The AMBA specification defines an on-chip communications standard for designing high-performance embedded microcontrollers. It is supported by the ARM Limited Corporation with wide cross-industry participation.

2.1.1. Advanced Peripheral Bus (APB)

The APB Bus is the lowest performance bus in the AMBA family[7]. The APB is designed for low bandwidth control accesses, for example register interfaces on system peripherals. There are separate address (PADDR), write data (PWDATA), and read data (PRDATA) buses, up to 32-bits each. With 7 additional control signals, there can be up to 103 I/O for each APB slave. There is one APB master, usually the bridge from a higher performance bus, that begins a transfer by asserting the appropriate PSELn signal with PADDR. PWRITE is active for a write and inactive on a read. PENABLE is asserted in the second clock, and is held active until PREADY is returned by the slave. The minimum transfer, read or write, is two clocks. APB slaves also have

the option of inserting wait states for reads or writes by withholding PREADY.

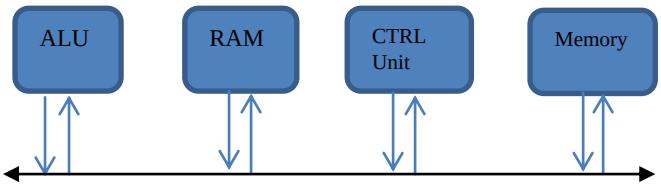

Fig 1 – APB Bus

The APB fits squarely within the low-performance bus definition. It follows a simple protocol with no burst transfers, data pipelining, or byte enables. Fig 1 shows the basic architecture of APB. There is only one master and the bridge from a higher level bus and it has only one bus width per implementation. It has no timing specifications, so that timing at an SoC level at higher frequencies will be more challenging.

2.1.2. Advanced High Performance Bus (AHB)

AHB is the mid-performance bus in the AMBA family. The protocol defines a 32-bit address bus (HADDR), but this has been extended in some implementations. The read and write data buses (HRDATA and HWDATA) may be defined under the specification as 2 n bits wide, from 8-bit to 1024-bit, but the most common implementation has been 32-bit. Compared to APB bus, it is improved in single edge clock protocol and it has split transactions. Fig-2 shows that it has several bus masters and burst transfers. It has pipelined operations to increase the speed of data transfer.

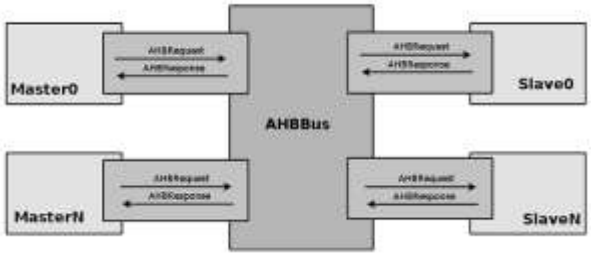

Fig-2 AHB bus

AHB offers the ability for slaves to act on a declared burst length, supports delayed reads, and can reach mid-performance bandwidth through increases in frequency and an expandable data bus width. The limitations of AHB are technical in nature, and prevent it from being considered as a high-performance bus. Notably absent from AHB are data pipelining, simultaneous read and write data.

2.1.3. Advanced eXtensible Interface (AXI)

AXI is the third generation of AMBA interface defined in the AMBA 3 specification[3]. It is targeted to achieve high performance, high clock frequency system designs and it is very suitable for high speed sub-micrometer interconnect.

The features are:

- Separate address/control and data phases.
- Support for unaligned data transfers using byte strobes.
- Burst based transactions with only start address issued issuing of multiple outstanding addresses.
- Easy addition of register stages to provide timing closure.

III. WORKING MODEL

AXI is the high-performance bus in the AMBA family. The architecture defines three write channels and two read channels. The write channels are address, write data, and response. The read channels are address and read data[7].

AXI protocol defines five independent channels as shown in Fig.4. Both write and read address channels have their own addresses to transfer, as well as the control information that describers the nature of the data to be transferred. AXI bus uses a write data channel to transfer data from master to slave and a read data channel to transfer data from slave to master. In write transaction, there is an additional write response channel to indicate the state of the transaction[2].

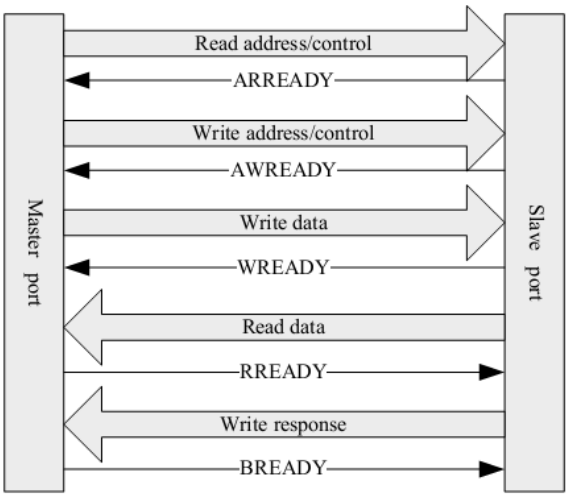

Fig-3 AXI Protocol

3.1 Read and write address channel

Read and write transactions each have their own address channel which carries all of the required address and control information for a transaction.It supports variable-length bursts, from 1 to 16 data transfers per burst (AxLEN signal) and bursts with a transfer size of 8-1024 bits (AxSIZE signal). It has wrapping, incrementing, and non-incrementing bursts (AxBURST signal). AXI mainly performs atomic operations, using exclusive or locked accesses (AxLOCK signal).

3.2 Read data channel

The read data channel conveys both the read data and read response information from the slave back to the master. Fig-4 shows how a read transaction uses the read address and read data channel.

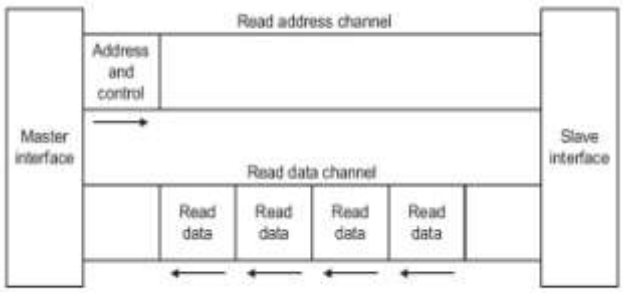

Fig – 4 AXI Read channel

It includes the data bus, which can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide (RDATA). It has a read response indicating the completion status of the read transaction.In read data channel, data and response group signals are maintained until the RReady signal is asserted.

3.3 Write data channel

The write data channel conveys the write data from the master to the slave . Fig-5 shows how a write transaction uses the write address, write data, and write response channels.

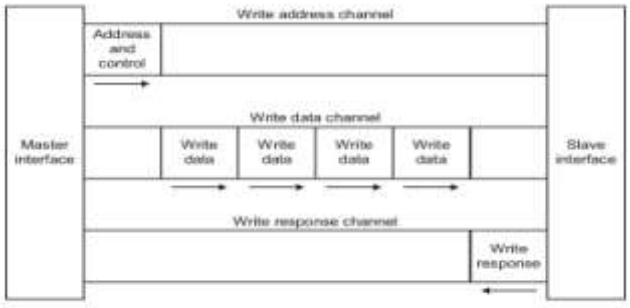

Fig-5 AXI Write Channel

It includes the data bus, which can be 8, 16, 32, 64, 128, 256, 512, or 1024 bits wide (WDATA). Here, one byte lane strobe for every byte, indicating which bytes of the data bus are valid (WSTRB).The last transfer inside a burst must be signaled through the WLAST signal. In write data channel, data, strobe and wlast information are maintained until the WReady signal is asserted.

3.4 Write Response channel

The write response channel provides a way for the slave to respond to write transactions. All write transactions use completion signaling. The completion signal occurs once for each burst, not for each individual data transfer within the burst. Response group signals are maintained until the BReady signal is asserted.

3.5 Interface and Interconnect

A typical system consists of a number of master and slave devices connected together through some form of interconnect, as shown in Fig 6.

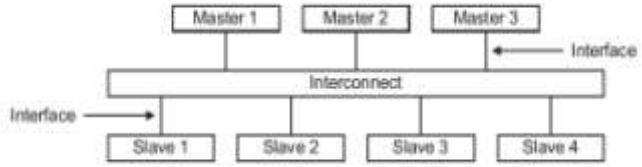

Fig-6 Interface and Interconnect

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

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

The interface definition enables a variety of different interconnect implementations[4]. The Interconnect between devices is equivalent to another device with symmetrical master and slave ports to which real master and slave devices can be connected. In most systems, the address channel bandwidth requirement is significantly less than the data channel bandwidth requirement. Such systems can achieve a good balance between system Performance and interconnect complexity by using a shared address bus with multiple data buses to enable parallel data transfers.

The AXI protocol is burst-based. Every transaction has address and control information on the address channel that describes the nature of the data to be transferred. The address channels include 32-bit address buses, AWADDR and ARADDR, but this could be extended in some implementations. The write and read data buses (WDATA and RDATA) may be defined under the specification as any 2n number, from 8-bit to 1024-bit. AXI masters and slaves are connected together through a central interconnect, which routes master requests and write data to the proper slave, and returning read data to the requesting master. The interconnect also maintains ordering based on tags.

An AXI master begins a read transfer by driving an address, ARADDR, and other transfer qualifiers with ARVALID. In high frequency implementations, the interconnect latches and drives the same signals to the slave, which responds with ARREADY. The slave drives read data, RDATA, with RVALID, and the transfer is made when RVALID and RREADY are sampled active. Again, in high frequency implementations, it is common for the interconnect to latch and drive the read data back to the requesting master. The last beat of read data is driven with RLAST. For a write transfer, the AXI master begins by driving an address, AWADDR, and other transfer qualifiers with AWVALID. An interconnect latches and drives the same signals to the slave in high frequency implementations, and the slave responds with AWREADY. When WVALID is active, the first beat of write data is valid. WVALID may be driven with valid data even before or after the address that relates to it. The transfer is made when WVALID and WREADY are sampled active. The AXI master drives the last beat of write data with WLAST. The slave then drives a write response, BRESP, with BVALID back to the master to indicate when the write transfer is complete, along with any applicable error information. Like PLB4, AXI splits the address and data phases, so data pipelining is supported. Thus, AXI can achieve full bus utilization, even when slaves with relatively high latency are attached.

3.6 Basic transactions

Transfer of either address information or data occurs when both the VALID and READY signals are HIGH. Two Operations are performed using AXI bus. They are Read Burst and Write Burst[5].

3.6.1. Read Burst

Fig 7 shows a read burst of four transfers. Here, the master drives the address, and the slave accepts it one cycle later. After the address appears on the address bus, the data transfer occurs on the read data channel.

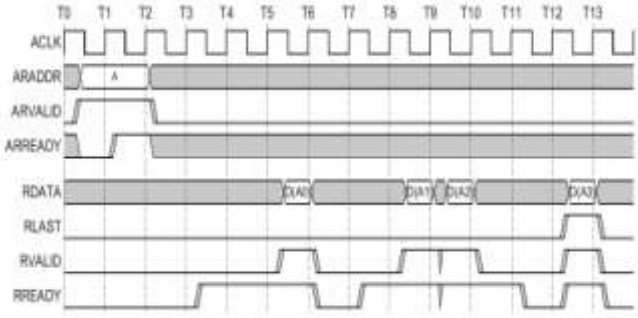

Fig 7- Read Burst

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.

3.6.2 Write Burst

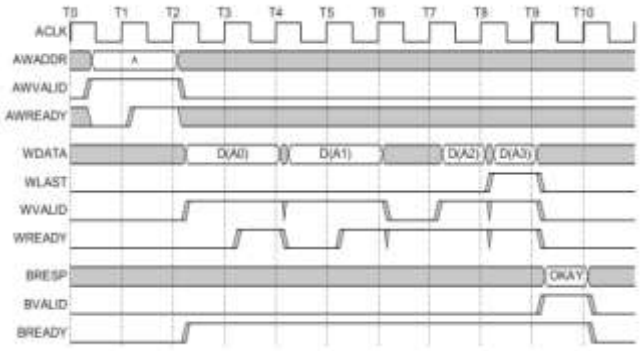

Fig -8 Write Burst

Fig 8 shows a write transaction. The process starts when the master sends an address and control information on the write address channel. The master then sends each item of write data over the write data channel. When the master sends the last data item, the **WLAST** signal goes HIGH. When the slave has accepted all the data items, it drives a write response back to the master to indicate that the write transaction is complete.

For designing this AXI protocol, Bluespec System Verilog (BSV) is used. BSV is based on a synthesizable subset of System Verilog, including System Verilog types, modules, module instantiation, interfaces, interface instantiation, parameterization, static elaboration, and generate elaboration. BSV allows a designer to develop a high-level, behavioral, hardware design utilizing atomic rules, which can be compiled to a Verilog RTL design[6].

Features of Bluespec:

- It has transparent control over architecture unlike C, C++,System C.
- Extensive libraries and utilities to accelerate coding.
- Accelerate the speed of verification by 50%.
- Atomic Methods instead of Verilog 'port lists' and System C 'methods'.

IV. SIMULATION RESULTS

Verification is a very important phase in VLSI. It is achieved by building test benches. The test benches are written as Bluespec System Verilog Code.

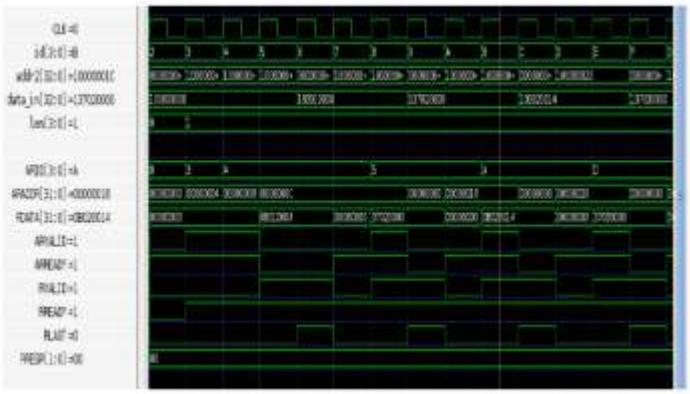

Fig 9 – AXI Read transactions

Here, Fig-9 explores the transaction signals of AXI read bus. The read bus depends on ARVALID and RVALID signal which will pass an handshake to the RREADY signal. For each transaction, it will create a Tag Id.

In fig-10, AXI Write Bus transactions are explored. Here, Tag ID is verified and it checks for response signaling. AWVALID signal depends on WWREADY/AWREADY signal which lies on BREADY/BVALID signal. AXI Read and Write bus are merged in fig-11. It has read as well as write transactions which are issued continuously. ID matching is being checked here. It mainly lies on response signals.

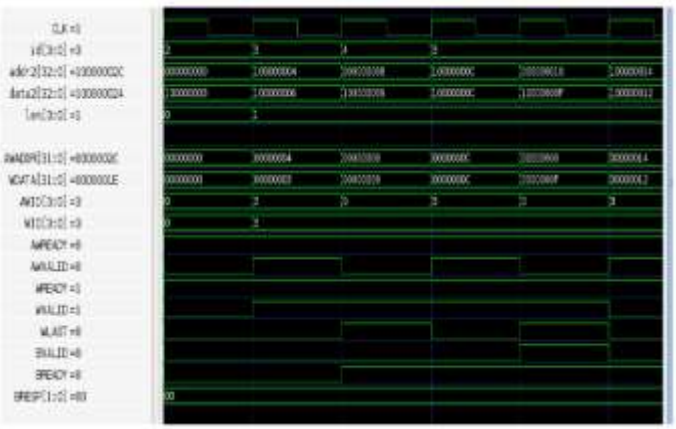

Fig – 10 AXI Write transactions

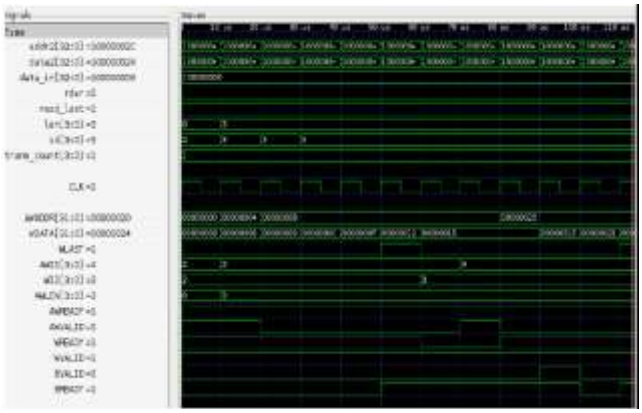

Fig – 11 AXI Read/Write Transactions

V. CONCLUSION

The AMBA AXI specification represents a major evolutionary step in interconnect technology for on-chip system design. The result is an interconnect architecture suitable for FPGAs and ASICs. We have designed an AXI bus which offers high-lever service at high frequency. We can implement this using Cyclone-III FPGA with the help of Mimic processor and the interface such as USB, peripheral devices. The technical co-ordination doesn`t supports the present structure of protocol design in valid AXI architecture. Further it can be evolved in such a way that the protocol structure can be modified for USB interface.

ACKNOWLEDGMENT

I would like to express my sincere thanks to Dr.Saraswathi Janaki, HOD, ECE department, Easwari Engineering College for her continuous support and guidance towards the concept. Her continuous feedback has always been the strongest motivation.

REFERENCES

[1] http://www.arm.com
[2] "Design of AXI Bus Based MPSoC on FPGA"Fu-ming Xiao, Dong-sheng Li, Gao-ming Du, Yu-kun Song, Duo-li Zhang, Ming-lun
[3] Ying-Ze Liao, "system design and implementation of AXI bus,"
[4] Jiang Zhou-liang, Quan Jin-guo, Lin Xiao-kang, "Analysis and Application of New Generation AMBA 3 AXI Protocol."
[5] www.xilinx.com/ipcenter/axi3.htm
[6] www.bluespec.com
[7] www.opencores.org – AMBA bus Specification

Dr. Saraswathi Janaki obtained her Ph.D degree from Anna University, Chennai, Tamilnadu, India. She is currently working as a professor and Head in the department Electronics and communication Engineering in Easwari Engineering college, Chennai, Tamilnadu, India. Her areas of interest are signal and Image processing, Electronic circuits. She is a m ember of IEEE, ISTE, IACST.

T.Ananth kumar, pursuing his master degree in VLSI Design in the department of Electronics and communication Engineering in Easwari Engineering college, Chennai, Tamilnadu, India..