International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395 -0056
Volume: 03 Issue: 07 | July-2016 www.irjet.net p-ISSN: 2395-0072

AMBA Compliant Programmable Interrupt Controller

V Bhagyalakshmi[1], Chandan G N [2]

[1]*Assistant Professor, Dept. Of EC Engineering, GSSSIETW, Mysuru, Karnataka, India*
[2]*Assistant Professor, Dept. Of EC Engineering, ATME, Mysuru, Karnataka, India*

Abstract - *Advanced Microcontroller Bus Architecture (AMBA) specification defines an on-chip communication standard for designing high performance embedded microcontrollers. The AMBA specification includes three distinct buses, namely AHB (Advanced High-performance Bus), ASB (Advanced System Bus) and APB (Advanced Peripheral Bus).As our design involves PIC (Programmable Interrupt Controller) which is low-power peripheral device and do not require the high performance of pipelined bus Interface. Hence we make use of APB which is optimized for minimal power consumption and reduced interface complexity to interface with PIC. APB interface is basically a communication protocol between processor and peripheral devices. In this project the design of the protocol is implemented for the Processor PIC interface. AMBA was designed for use in System-On-Chip (SOC) designs. The main advantage of AMBA on-chip specification (version 2.0) is that the standard is well documented and is exempted from royalties. The AMBA-AHB is for high-frequency system modules like processors and memories. The AMBA-APB is for low-power peripheral devices. One more main advantage of this design is that the controller is programmable so that the address and data widths can be programmed. This makes the design compatible with any processor. AMBA-based SOC's maximize the efficiency of data movement and storage, thus delivering the performance they need at the lowest power and cost. This design can be used in generic mobile phones, PDA's (personal digital assistant) mobiles System-On-Chip Architecture.*

Key Words: ASIC, EDA, Interrupt, FPGA, Verilog .

1. INTRODUCTION

The APB is part of the AMBA 3 protocol family. It provides a low-cost interface that is optimized for minimal power consumption and reduced interface complexity .The APB interfaces to any peripherals that are low-bandwidth and do not require the high performance of a pipelined bus interface. The APB has un pipelined protocol. Signal Transitions are only related to the rising edge of the clock to enable the. Every transfer takes at least two cycles. The APB can interface with the AMBA Advanced High performance Bus Lite (AHB-Lite) and AMBA Advanced Extensible Interface (AXI). We can use it to provide access to the programmable control registers of peripheral devices. The APB bus is used to interface to any peripheral device which are low bandwidth and do not require the high performance of a pipelined bus interface. The APB slave interface acts as a bridge between the APB bus and the peripheral device to which the bus is connected. It receives the APB bus signals and converts them to a form in which is understood by the connected peripheral device. Most common applications of the APB interface for read and write registers of the connected device. The Peripheral devices connected to the APB bus could be UART, Timer, Keypad, etc. Compatible with 8-bit as well as 16-bit processors. Manage 8 interrupts according to the instructions written into the control registers can mask each interrupt request individually. Read the status of pending interrupts, in-service interrupts and masked interrupts. Accept either the level triggered or the edge triggered interrupt request The Advanced Microcontroller Bus Architecture (AMBA) specification defines an On chip communications standard for designing high-performance embedded microcontrollers.

Three distinct buses are defined within the AMBA specification:

- The Advanced High-performance Bus (AHB)
- The Advanced System Bus (ASB)
- The Advanced Peripheral Bus (APB)

Our design can be used in any embedded systems with AMBA bus architecture to serve the hardware interrupts. This is mainly used in mobile phones, laptops, PDA, touch screen controller etc. This design can be part of SOC design to serve hardware interrupts.

1.1 Literature Survey

APB stands for Advanced Peripheral Bus which provides the communication between the processor and peripheral devices. The APB is part of the AMBA hierarchy of buses and is optimized for minimal power consumption and reduced interface complexity. The AMBA APB appears as a local secondary bus that is encapsulated as a single AHB or ASB slave device. APB provides a low-power extension to the system bus which builds on AHB or ASB signals directly. The APB Bridge appears as a slave module which handles the bus handshake and control signal retiming on behalf of the local peripheral bus. The AMBA APB should be used to interface to any peripherals which are low bandwidth and do not require the high performance of a pipelined bus interface. The latest revision of the APB is specified so that all signal transitions are only related to the rising edge of the clock. This improvement ensures the APB peripherals can be integrated easily into any design flow, with the following advantages:

High-frequency operation is easier to achieve.

a) Performance is independent of the mark-space ratio of the clock.
b) Static timing analysis is simplified by the use of a single clock edge.
c) No special considerations are required for automatic test insertion.
d) Many Application Specific Integrated Circuit (ASIC) libraries have a better selection of rising edge registers

These changes to the APB also make it simpler to interface it to the new AHB. An AMBA APB implementation typically contains a single APB bridge which is required to convert AHB or ASB transfers into a suitable format for the slave devices on the APB. The bridge provides latching of all address, data and control signals, as well as providing a second level of decoding to generate slave select signals for the APB peripherals. All other modules on the APB are APB slaves. The APB slaves have the following interface specification:

• Address and control valid throughout the access (unpipelined)

• Zero-power interface during non-peripheral bus activity (peripheral bus is static when not in use)

• Timing can be provided by decode with strobe timing (unlocked interface)

•Write data valid for the whole access (allowing glitch-free transparent latch implementations)

The AMBA AHB is for high-performance, high clock frequency system modules. The AHB acts as the high-performance system backbone bus. AHB supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces with low-power peripheral macro cell functions. AHB is also specified to ensure ease of use in an efficient design flow using synthesis and automated test techniques.

The AMBA ASB is for high-performance system modules. AMBA ASB is an alternative system bus suitable for use where the high-performance features of AHB are not required. ASB also supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces with low-power peripheral macro cell functions.

The AMBA APB is for low-power peripherals. AMBA APB is optimized for minimal power consumption and reduced interface complexity to support peripheral functions. APB can be used in conjunction with either version of the system bus.

1.2 DESIGN METHODOLOGY

Figure 1 gives the overall view of the project. It consists of APB controller, logic, programmable interrupt controller logic, registers, bridge, and APB bus. The APB bus is used to interface to any peripheral device which are low bandwidth and do not require the high performance. We require a bridge to convert all the AHB signals to APB signals so that it can be compatible. The bridge buffers address and controls the data from the AHB, drives the APB peripherals and returns data and response signals to the AHB. Interface operates when the APB and AHB clocks have the same frequency and phase. Pclk, Pwrite, Penable, Psel, Preset, Paddr, Pwdata are the input to the APB controller logic and Prdata is the output of APB controller logic. The rising edge of Pclk (bus clock) is used to time all transfers on the APB. When Pwrite is high APB indicates write access and when low read access. Enable signal is used to indicate the second cycle of an APB transfer. The rising edge of the Penable occurs in the middle of the APB transfer. Psel acts as enable for selecting the particular device (similar to chip select). The APB bus reset signal (preset) is active low and this will normally be connected directly to the system bus reset signal. Paddr [31:0] is the APB address bus, which may be up to 32 bits wide and is driven by the peripheral bus bridge unit. Pwdata is an input data bus from APB bus. Pr data is the output data bus to the APB.

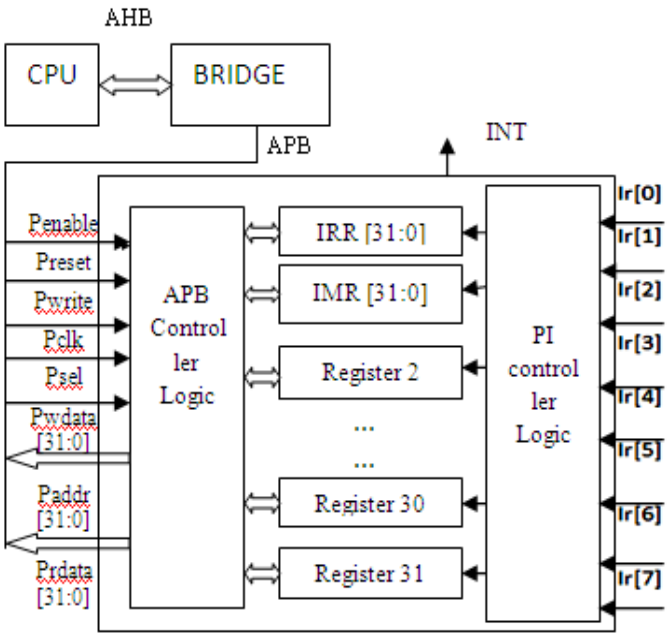

Fig. 1: Block Diagram AMBA

In this system there are 32 registers each of width 32 bits, Register0 is considered as IRR(interrupt request register) in which interrupt will be stored if any of the interrupt pin goes high. Register1 is considered as IMR (interrupt mask register), if we need to mask any interrupt we need to store

that value in the IMR accordingly those interrupts will not be served. But software interrupts cannot be masked and has to be served. We assign the priority for serving the interrupt highest priority interrupt will be served first. Depending upon the value of IMR and IRR the INT signal will change. If any of the interrupt pin is high and if the interrupt is not masked then INT signal goes high. If the corresponding bit in IMR is masked then the INT signal will not go high even when the interrupt bit is high.

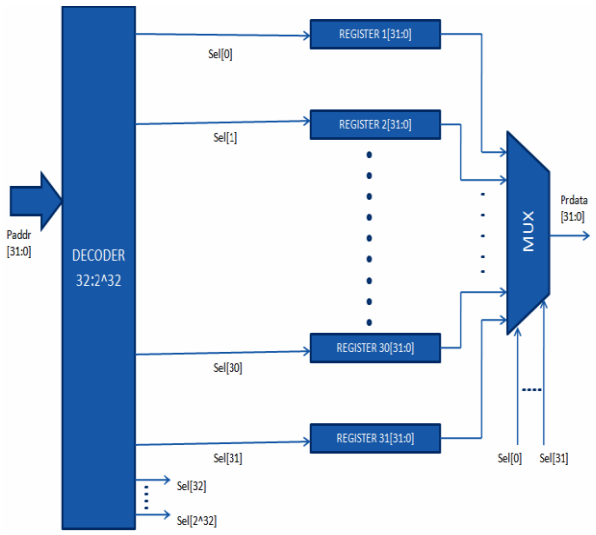

Fig 2 - General block diagram

The Fig 2 shows the general block diagram. It consists of decoder, multiplexer and registers. A decoder is a device which does the reverse of an encoder, undoing the encoding so that the original information can be retrieved. The same method used to encode is usually just reversed in order to decode. In digital electronics, a decoder can take the form of a multiple-input, multiple output logic circuit that converts coded inputs into coded outputs, where the input and output codes are different. For example: n to 2n decoder, binary-coded decimal decoders. Enable inputs must be on for the decoder to function, otherwise its outputs assume a single "disabled" output code word. Decoding is necessary in applications such as data multiplexing, 7 segment display and memory address decoding.

If there is any interrupt occurring, it will be captured in the IRR register. If we need to mask any interrupt the corresponding bit in the IMR register is made high. The corresponding bit in IRR and IMR are given to the AND gate, i.e. IRR[0] and IMR[0] are given to the AND[0] gate, this repeats for all the eight interrupts. All the eight AND gates output signals are given to the OR gate. If there is any interrupt, the output of OR gate i.e. INT signal goes high. AMBA Compliant Programmable Interrupt Controller 30Department of E & C Engineering, NHCE 2009-10Consider a case where interrupt request ir [2] is gone high, then the value in IRR register will be 00000100 and suppose IMR content is 00000100 then AND[2]= IRR[2]& ~IMR[2] , which will be zero in this case. Since all other bits are zero the input to the OR gate are 00000000, therefore output of OR gate is low, which means the INT signal will not go high. It seems that even though the interrupt has occurred the INT signal is not high, since the interrupt is masked by writing the appropriate value in the mask register.

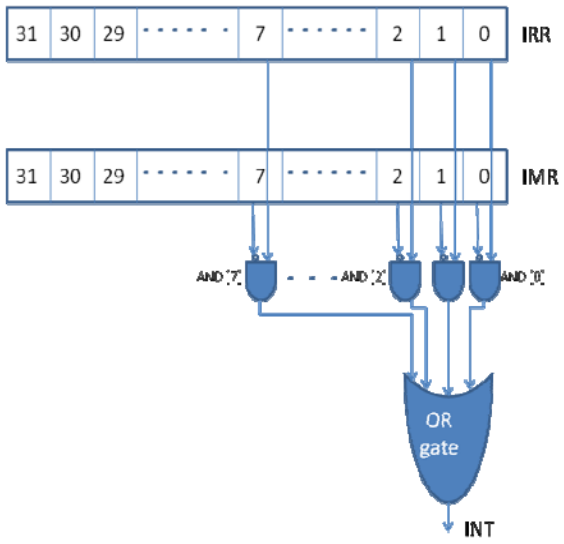

Fig 3 - INT signal generation

Let us consider an another case where interrupt requests ir[0] and ir[3] goes high, then IRR will have 00001001 and suppose IMR will have all zeros then AND[0] =IRR[0] & ~ IMR[0] = 1, AND[3] = IRR[3] & ~IMR[3]= 1, rest all AND output are zero. It is given to the OR gate and hence the output goes high, indicating that interrupt has occurred and needs to be served. In this case as there are two interrupts occurring simultaneously the priority in which the interrupts has to be served is decided by the processor as per user requirements.

Step 1: If the behavioral description of the system is available go to step 3; otherwise, formulate a flowchart for the behavior of the system.
Step 2: Use the flowchart to write a behavioral description of the system. We should make sure to review the instructions of the synthesis tools to see if there are constraints on any of the behavioral statements that are used.
Step 3: Simulate the behavioral code and verify that the simulation correctly describes the system are acceptable
Step 4: Map the behavioral statements into components or logic gates. We should make sure that the components used are acceptable to our synthesizer.
Step 5: Write a structural or gate-level description of the components and logic gates of step3. Simulate the structural

description, and verify that this simulation is similar to that of step 3.

Step 6: Use the CAD tools to download the gates and components of step 4 into the electronic chip usually an FPGA chip.

Step 7: Test the chip by giving signals to the input pins of the chip, and observe the output from the output pins.

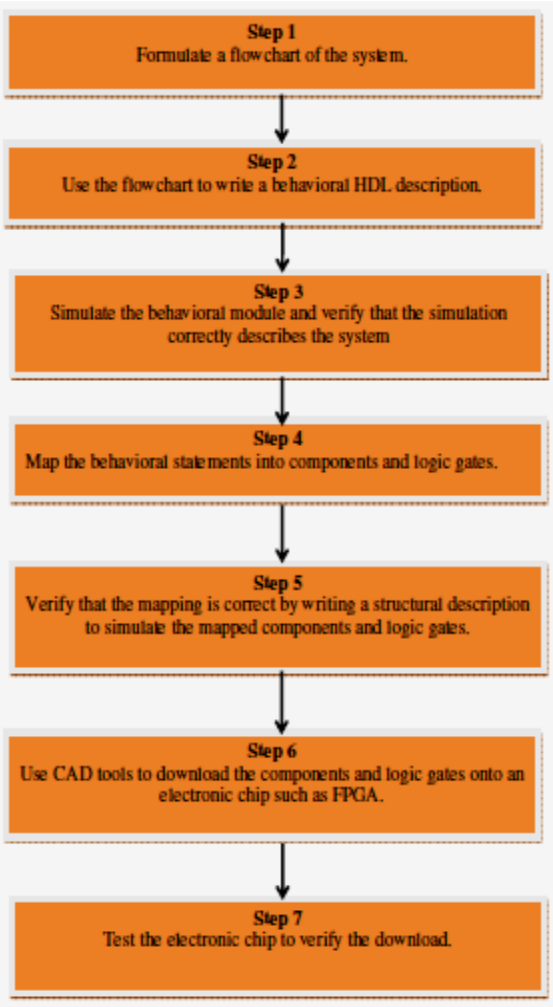

Fig 4 - Synthesis steps

Synthesis process is shown in Fig 4. The decoder used here is 32:232. Input to the decoder is 32 bits (Paddr) and output is 232 bits (sel [31:0]). Here out of 232 combinations we are making use of only 32 bits. A multiplexer, sometimes referred to as a "multiplexor" or simply "mux", is a device that selects between a number of input signals. In its simplest form, a multiplexer will have two signal inputs, one control input, and one output. An everyday example of an analog multiplexer is the source selection control on a home stereo unit Multiplexers are used in building digital semiconductors such as CPUs and graphics controllers. In these applications, the number of inputs is generally a multiple of2 (2, 4, 8, 16, etc.), the number of outputs is either 1 or relatively small multiple of 2, and the number of control signals is related to the combined number of inputs and outputs.

For example, a 2-input, 1-output mux requires only 1 control signal to select the input, while a 16-input, 4-output mux requires 4 control signals. The MUX used here has 232 inputs and one output and 32 control signals. Here out of 232 inputs we are just making use of 32 inputs. Depending upon the control signal we are getting the output.

The input to the decoder is a Paddr. Depending upon the Paddr the sel line is chosen, if the Paddr is 32'b 0 then sel [0] is high, then depending up on the signals Pwrite, Penable, Psel, Preset the data is written into or read from the register through APB.

In computer architecture, a processor register (or general purpose register) is a small amount of storage available on the CPU whose contents can be accessed more quickly than storage available elsewhere. Typically, this specialized storage is not considered part of the normal memory range for the machine. Most, but not all, modern computers adopt the so-called load-store architecture. Under this paradigm data is 'shuffled' from subordinated memory be it L1, L2 cache or RAM into registers, 'crunched' therein by running instructions from the instruction set, then transferred out.

A common property of computer programs is locality of reference: the same values are often accessed repeatedly; and holding these frequently used values in registers improves program execution performance. Processor registers are at the top of the memory hierarchy, and provide the fastest way for a CPU to access data. Here we are using 32 registers each of size 32 bits.

2. RESULTS

The verification process consists of static/structural and dynamic/behavioral aspects. E.g., for a software product one can inspect the source code (static) and run against specific test cases (dynamic). Establishing properties of hardware or software designs using logic, rather than testing or informal arguments. This involves formal specification of the requirement, formal modeling of the implementation, and precise rules of inference to prove, that the implementation satisfies the specification.

Formal verification can be helpful in proving the correctness of systems such as: cryptographic protocols, combinational circuits, digital circuits with internal memory, and software expressed as source code. The verification of these systems is done by providing a formal proof on an abstract mathematical model of the system, the Correspondence between the mathematical model and the nature of system being otherwise known by construction.

Examples of mathematical objects often used to model systems are: finite state machines, labeled transition systems,

Petri nets, timed automata, hybrid automata, process algebra, formal semantics of programming languages such as operational semantics, denotation semantics, axiomatic semantics and Hoare logic.

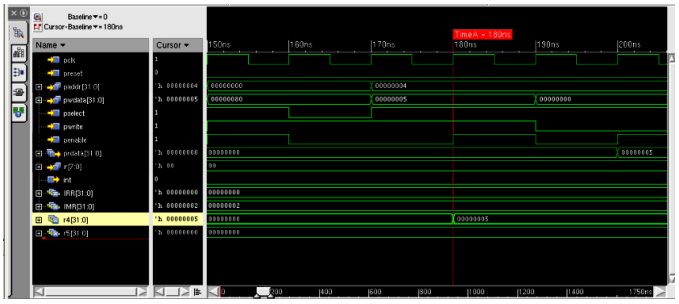

Fig 5- write transfer

In the above Fig 5, waveform shows the write operation for the example considered. We can observe that the required data is written into the required register.

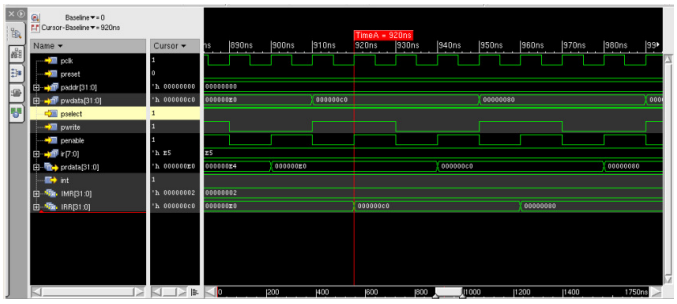

Fig 6- Read Transfer

In the above Fig 6, waveform shows the write operation for the example considered. We can observe that the data in IRR register is read through prdata.

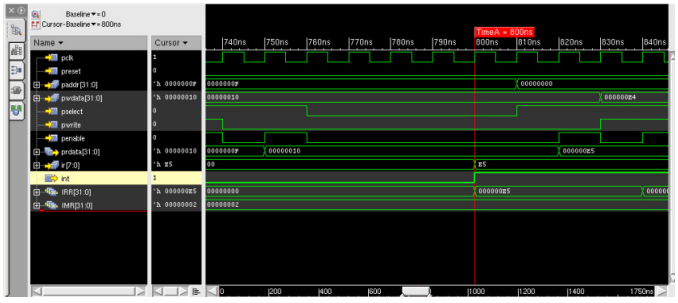

Fig 7-Int signal generation

In the above Fig 7, the waveform shows the interrupt signal generation. Since int is generated the interrupts will be served as per priorities. In our design ir[0] has the highest priority and ir[7] has least priority. Hence ir[0] will be served first then ir[2] and so on.

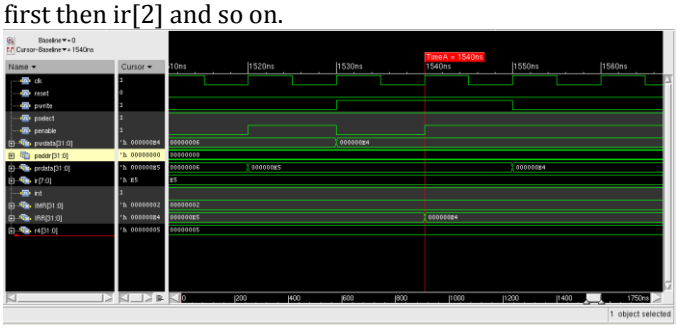

Fig 8-Interrupt servicing

In Fig 8, the waveform shows that interrupt ir[0] has been served and the corresponding bit is resetted.

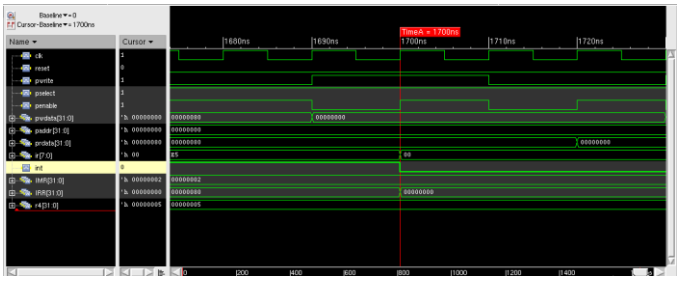

Fig 9- All interrupts serviced and resetted.

In Fig 9, the waveform shows the reset condition after serving all the interrupts.

Table -1: Synthesis Results

Clk period (ns)	Timing slack (ps)	Area (µm)	Number of cells
2	-6	24855	2048
2.1	5	24722	2037
2.2	1	24735	2045
2.4	33	24703	2041
2.6	70	24664	2031
2.8	239	24667	2031
3	145	24664	2031
4	1266	24632	2031

The optimal output is obtained for clock period of 2.2ns which is shown in Table 1.

3. CONCLUSIONS

Design of programmable interrupt controller used in cell phones, laptops is done and coded in Verilog HDL maintaining industry standard coding guidelines. After coding the design, a verification plan describing verification strategy for the design is developed. This is followed by verification environment development and simulation of

design. Once design is cleared through functional verification, synthesis strategy is developed. Synthesis scripts are added and design is synthesized on 180nmtechnology and timing checks are performed. Synthesis report with final working frequency, gate count and chip area for different design constraints are implemented.

a) Test bench can be more modular.
b) If the full system is available, system level testing environment can be generated.
c) Capability to extend to 64-bit and 128-bit interrupts.
d) Internal priority rotation schemes can be added.

The designed controller is generic and compliant with AMBA; hence it can be used in any embedded system employing AMBA bus architecture. In this project we are configuring 32-bit register of a QAM chip, but depending on the requirement we can extend this to configure any 8 bit, 16- bit or 64-bit register modes of any chip. Future implementation of the design on FPGA can be carried and tested for real time communication.

REFERENCES

[1] VLSI Design Techniques for Analog and Digital Circuits, R. L. Geiger, P. E. Allen, and N. R. Strader, McGraw-Hill, 1990.

[2] [2]. Analysis and Design of Digital Integrated Circuits, Third Edition, David A. Hodges, Horace G. Jackson, and Resve A. Saleh, McGraw-Hill, 2004.

[3] [3]. Basic VLSI Design, Douglas Pucknell &Eshragian, PHI, 3rd Edition, 2009.

[4] [4]. Priyanka Gandhani, Charu Patel " Moving from AMBA AHB to AXI Bus in SoC Designs: A Comparative Study" Int. J Comp Sci. Emerging Tech Vol-2 No 4 ,pp.476-479 August, 2011.

[5] [5]. Fundamentals of Modern VLSI Devices,Yuan Taun Tak H Ning Cambridge Press, South Asia Edition 2003,

[6] [6]. ModernVLSI Design, Wayne wolf, Pearson Education Inc. 3rd edition, 2003.

[7] [7]. Introduction to CMOS VLSI Design-A Circuits and Systems Perspective, Neil Weste,Pearson Education.3rd Edition.2015.

[8] [8]. K. Goossens, O. P. Gangwal, J. Rover, and A. P. Niranjan, "Interconnect and memory organization in SOCs for advanced settop boxes and tv-evolution, analysis, and trends", in J. Nurmi, H. Tenhunen, J. Isoaho, and A. Jantsch, Editors, Interconnect-Centric Design for Advanced SoC and NoC, Chapter 15, pp. 399-423, Kluwer, 2004.

[9] [9]. Y. Hu and B. Yang, "Building an amba ahb compliant memory controller", Proceedings of the Third International Conference on Measuring Technology and Mechatronics Automation, Vol. 01, 2011, pp. 658-661.

[10] [10].YOO,S.J.; NICOLESCU,G., LYONNARD,D., BAGHDADI, A.,JERRAYA,A.A.,:'A generic wrapper architecture for multi-processor SOC co-simulation and design', Hardware/Softerware Codesign, 2001.CODES 2001.Proceedings of the Ninth International Symposium on,2001 pp. 195-200.